MD5 Preimages from Multiple Outputs with Known Input Differentials

Klaudius Ignazius Rschju

Bavarian University for Low-Level Systems, Hardware, and Internet of Things
(BULLSHIT)

`contact@hxp.io`

Abstract. We demonstrate a preimage attack against the MD5 hash function when multiple hash values are given whose corresponding input strings are unknown, but guaranteed to be short and related by known input differentials. The computational complexity of our attack is roughly equivalent to $2^{128/(n-\pi/4)^2}$ MD5 compressions, where n is the number of given distinct outputs. This attack marks the first practical preimage attack on MD5 in a specialized setting; by contrast, devastating attacks on the collusion resistance of MD5 have been known since 2004 [2].

1 Introduction

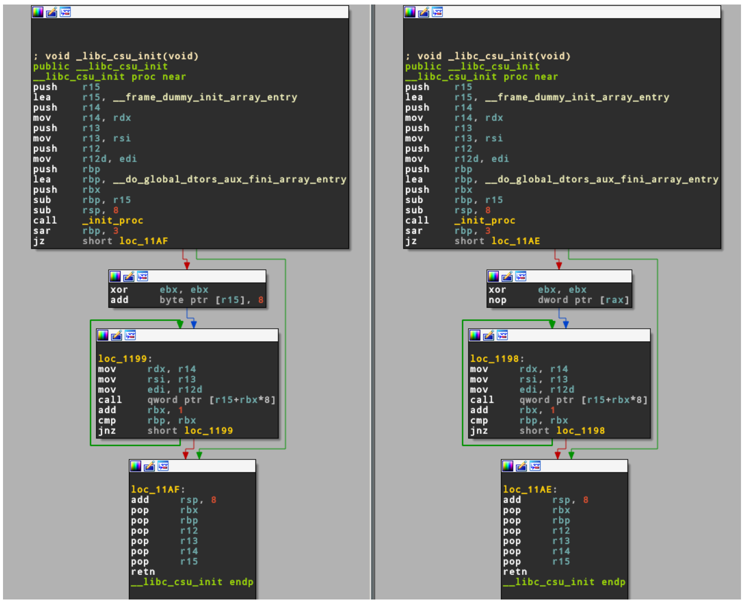
It is well-known that preimage attacks are pretty hard, thus this is clearly not the solution — especially for a `rev` challenge rated “medium”. In fact, the paper excerpt in the challenge description was just there for general trolling purposes and not intended as a hint for the challenge (read: we have *absolutely no idea* how to achieve the attack complexity claimed in the abstract). Therefore, we backdoored the implementation (`https://github.com/krisprice/simd_md5`).

2 Methodology

2.1 Single-Instruction Backdoor

The construction of the backdoor used in this paper relies on slight changes to code that commonly gets linked into program images as part of the `crtstuff.o` object when compiling C programs using the GNU C compiler (GCC). Such code is typically responsible for managing low-level functions of the C runtime environment such as constructors and destructors, and setting up monitoring components such as GNU `gperf`. As those functions are usually present in ELF program images regardless of their actual functionality, analysts tend to ignore them, or at least exercise less care during early functionality examination. Such functions therefore constitute perfect targets for dispatching hidden functionality.

Figure 1 shows the `__libc_csu_init` function in both the backdoored and the vanilla form. Careful examination shows that the backdoor version increments



```
; void _libc_csu_init(void)
public __libc_csu_init
__libc_csu_init proc near
push    r15
lea     r15, __frame_dummy_init_array_entry
push    r14
mov     r14, rdx
push    r13
mov     r13, rsi
push    r12
mov     r12d, edi
push    rbp
lea     rbp, __do_global_dtors_aux_fini_array_entry
push    rbx
sub     rbp, r15
sub     rsp, 8
call    _init_proc
sar     rbp, 3
jz      short loc_11AF

xor     ebx, ebx
add     byte ptr [r15], 8

loc_1199:
mov     rdx, r14
mov     rsi, r13
mov     edi, r12d
call    qword ptr [r15+rbx*8]
add     rbx, 1
cmp     rbp, rbx
jnz     short loc_1199

loc_11AF:
add     rsp, 8
pop     rbx
pop     rbp
pop     r12
pop     r13
pop     r14
pop     r15
retn
__libc_csu_init endp
```

```
; void _libc_csu_init(void)
public __libc_csu_init
__libc_csu_init proc near
push    r15
lea     r15, __frame_dummy_init_array_entry
push    r14
mov     r14, rdx
push    r13
mov     r13, rsi
push    r12
mov     r12d, edi
push    rbp
lea     rbp, __do_global_dtors_aux_fini_array_entry
push    rbx
sub     rbp, r15
sub     rsp, 8
call    _init_proc
sar     rbp, 3
jz      short loc_11AE

xor     ebx, ebx
nop     dword ptr [rax]

loc_1198:
mov     rdx, r14
mov     rsi, r13
mov     edi, r12d
call    qword ptr [r15+rbx*8]
add     rbx, 1
cmp     rbp, rbx
jnz     short loc_1198

loc_11AE:
add     rsp, 8
pop     rbx
pop     rbp
pop     r12
pop     r13
pop     r14
pop     r15
retn
__libc_csu_init endp
```

Fig. 1: The backdoored (left) and vanilla (right) versions of the constructor handling `__libc_csu_init` function usually linked into ELF binaries regardless of their functionality.

the target of the pointer contained in register `r15` by 8. As this function is responsible for dispatching constructors during the early program startup phase, this patch effectively shifts the control flow from the actual constructor location to an attacker-controlled location. The indirect call therefore ends up calling a location eight bytes behind the original default constructor's (`frame_dummy`) location. As GCC adds padding between functions, there is a small code cave located directly behind the `frame_dummy` function that allows hiding a jump beyond the `p_memsz` of the segment. Figure 2 shows this setup.

```
.text:0000000000001150
.text:0000000000001150 ; =============== S U B R O U T I N E ========
.text:0000000000001150
.text:0000000000001150
.text:0000000000001150 frame_dummy     proc near               ; DATA X
.text:0000000000001150                 jmp     register_tm_clones
.text:0000000000001150 frame_dummy     endp
.text:0000000000001150
.text:0000000000001150 ; -----------------------------------------------
.text:0000000000001155                 align 20h
.text:0000000000001160
```

```
.text:0000000000001150
.text:0000000000001150 ; =============== S U B R O U T I N E ========
.text:0000000000001150
.text:0000000000001150
.text:0000000000001150 frame_dummy     proc near               ; DATA
.text:0000000000001150                 jmp     register_tm_clones
.text:0000000000001150 frame_dummy     endp
.text:0000000000001150
.text:0000000000001150 ; -----------------------------------------------
.text:0000000000001155                 align 20h
.text:0000000000001160
```

```
.text:0000000000001150
.text:0000000000001150 ; =============== S U B R O U T I N E ========
.text:0000000000001150
.text:0000000000001150
.text:0000000000001150 frame_dummy     proc near               ; DATA X
.text:0000000000001150                 jmp     register_tm_clones
.text:0000000000001150 frame_dummy     endp
.text:0000000000001150
.text:0000000000001155 ; -----------------------------------------------
.text:0000000000001155                 nop     dword ptr [rax+0A3E9h]
.text:0000000000001155 ; -----------------------------------------------
.text:000000000000115C                 db    0
.text:000000000000115D                 db    0
.text:000000000000115E                 db    0
.text:000000000000115F                 db  90h ; É
.text:0000000000001160
```

```
.text:0000000000001150
.text:0000000000001150 ; =============== S U B R O U T I N E ========
.text:0000000000001150
.text:0000000000001150
.text:0000000000001150 frame_dummy     proc near               ; DATA
.text:0000000000001150                 jmp     register_tm_clones
.text:0000000000001150 frame_dummy     endp
.text:0000000000001150
.text:0000000000001155 ; -----------------------------------------------
.text:0000000000001155                 db      2Eh
.text:0000000000001155                 nop     word ptr [rax+rax+00000
.text:000000000000115F                 nop
.text:0000000000001160
.text:0000000000001160 ; =============== S U B R O U T I N E ========
.text:0000000000001160
.text:0000000000001160
```

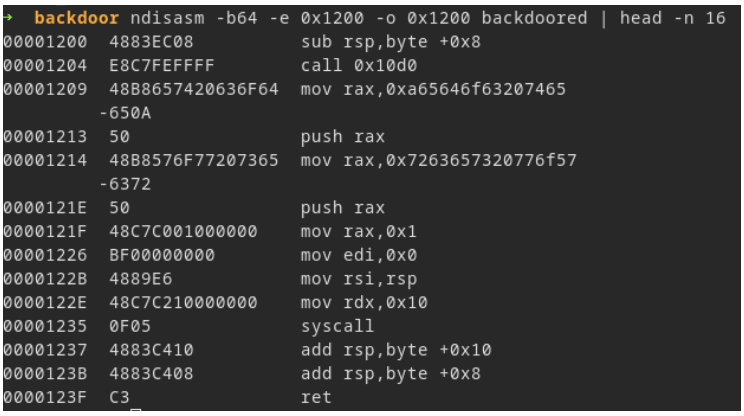
Fig. 2: Hiding the jump in a dummy application (backdoored version on the left)

2.2 Discrepancies in ELF Loading

When loading program segments into memory, the dynamic loader operates at page granularity. This is counter-intuitive to the semantics of the `p_memsz` field in the program segment headers of ELF files. To exacerbate the situation, common off-the-shelf analysis tools like `objdump` and the Interactive Disassembler (IDA) Pro[1] actually honor the `p_memsz` field and *omit code outside of the defined segment limits*[2] altogether.

This discrepancy can be used to hide the backdoor inserted in section 2.1 and thereby construct a stealthily-backdoored executable that might fool even experienced binary analysts.

Figure 4 shows the analysis results on a dummy application in several common analysis tools (IDA Pro, `objdump`, and Radare 2) — only manual disassembly of the binary (e.g. by using `ndisasm` as shown in Figure 3) or dynamic analysis (e.g. by single-stepping in the GNU Debugger (GDB) with the `stepi` command) reveals the jump beyond the segment boundaries.



```
→  backdoor ndisasm -b64 -e 0x1200 -o 0x1200 backdoored | head -n 16
00001200  4883EC08          sub rsp,byte +0x8
00001204  E8C7FEFFFF        call 0x10d0
00001209  48B8657420636F64  mov rax,0xa65646f63207465
         -650A
00001213  50                push rax
00001214  48B8576F77207365  mov rax,0x7263657320776f57
         -6372
0000121E  50                push rax
0000121F  48C7C001000000    mov rax,0x1
00001226  BF00000000        mov edi,0x0
0000122B  4889E6            mov rsi,rsp
0000122E  48C7C210000000    mov rdx,0x10
00001235  0F05              syscall
00001237  4883C410          add rsp,byte +0x10
0000123B  4883C408          add rsp,byte +0x8
0000123F  C3                ret
```

Fig. 3: Revealing the hidden backdoor with `ndisasm`

---

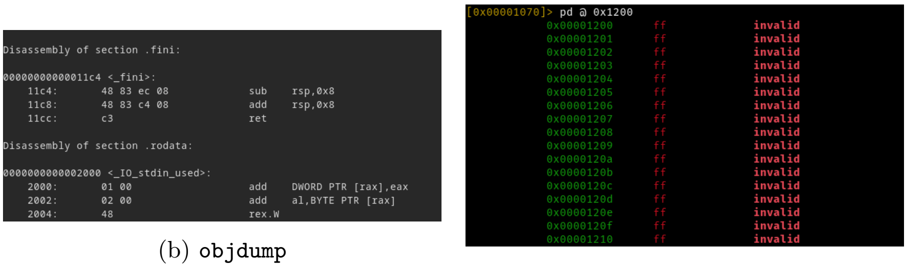
[1] https://www.hex-rays.com/

[2] Note that this is even true for `objdump` when using the "disassemble everything" switch `-D`

```
.fini:00000000000011C4
.fini:00000000000011C4 ; =============== S U B R O U T I N E =========
.fini:00000000000011C4
.fini:00000000000011C4
.fini:00000000000011C4                 public _term_proc
.fini:00000000000011C4 _term_proc      proc near
.fini:00000000000011C4                 sub     rsp, 8          ; _fini
.fini:00000000000011C8                 add     rsp, 8
.fini:00000000000011CC                 retn
.fini:00000000000011CC _term_proc      endp
.fini:00000000000011CC
.fini:00000000000011CC _fini           ends
.fini:00000000000011CC
.rodata:0000000000002000 ; ==========================================
.rodata:0000000000002000
.rodata:0000000000002000 ; Segment type: Pure data
.rodata:0000000000002000 ; Segment permissions: Read
.rodata:0000000000002000 _rodata          segment dword public 'CONST'
.rodata:0000000000002000                  assume cs:_rodata
.rodata:0000000000002000                  ;org 2000h
.rodata:0000000000002000                  public _IO_stdin_used
.rodata:0000000000002000 _IO_stdin_used  db    1
.rodata:0000000000002001                  db    0
.rodata:0000000000002002                  db    2
.rodata:0000000000002003                  db    0
.rodata:0000000000002004 ; char s[]
.rodata:0000000000002004 s                db 'Hello world!',0     ; DAT
.rodata:0000000000002004 _rodata          ends
```

```
.fini:00000000000011C4
.fini:00000000000011C4 ; =============== S U B R O U T I N E =========
.fini:00000000000011C4
.fini:00000000000011C4
.fini:00000000000011C4                 public _term_proc
.fini:00000000000011C4 _term_proc      proc near
.fini:00000000000011C4                 sub     rsp, 8          ; _fini
.fini:00000000000011C8                 add     rsp, 8
.fini:00000000000011CC                 retn
.fini:00000000000011CC _term_proc      endp
.fini:00000000000011CC
.fini:00000000000011CC _fini           ends
.fini:00000000000011CC
.rodata:0000000000002000 ; ==========================================
.rodata:0000000000002000
.rodata:0000000000002000 ; Segment type: Pure data
.rodata:0000000000002000 ; Segment permissions: Read
.rodata:0000000000002000 _rodata          segment dword public 'CONST'
.rodata:0000000000002000                  assume cs:_rodata
.rodata:0000000000002000                  ;org 2000h
.rodata:0000000000002000                  public _IO_stdin_used
.rodata:0000000000002000 _IO_stdin_used  db    1
.rodata:0000000000002001                  db    0
.rodata:0000000000002002                  db    2
.rodata:0000000000002003                  db    0
.rodata:0000000000002004 ; char s[]
.rodata:0000000000002004 s                db 'Hello world!',0     ; DAT
.rodata:0000000000002004 _rodata          ends
```

(a) IDA Pro



```

Disassembly of section .fini:

00000000000011c4 <_fini>:
    11c4:	48 83 ec 08          	sub    rsp,0x8
    11c8:	48 83 c4 08          	add    rsp,0x8
    11cc:	c3                   	ret

Disassembly of section .rodata:

0000000000002000 <_IO_stdin_used>:
    2000:	01 00                	add    DWORD PTR [rax],eax
    2002:	02 00                	add    al,BYTE PTR [rax]
    2004:	48                   	rex.W
```

(b) `objdump`

```
[0x00001070]> pd @ 0x1200
            0x00001200      ff             invalid
            0x00001201      ff             invalid
            0x00001202      ff             invalid
            0x00001203      ff             invalid
            0x00001204      ff             invalid
            0x00001205      ff             invalid
            0x00001206      ff             invalid
            0x00001207      ff             invalid
            0x00001208      ff             invalid
            0x00001209      ff             invalid
            0x0000120a      ff             invalid
            0x0000120b      ff             invalid
            0x0000120c      ff             invalid
            0x0000120d      ff             invalid
            0x0000120e      ff             invalid
            0x0000120f      ff             invalid
            0x00001210      ff             invalid
```

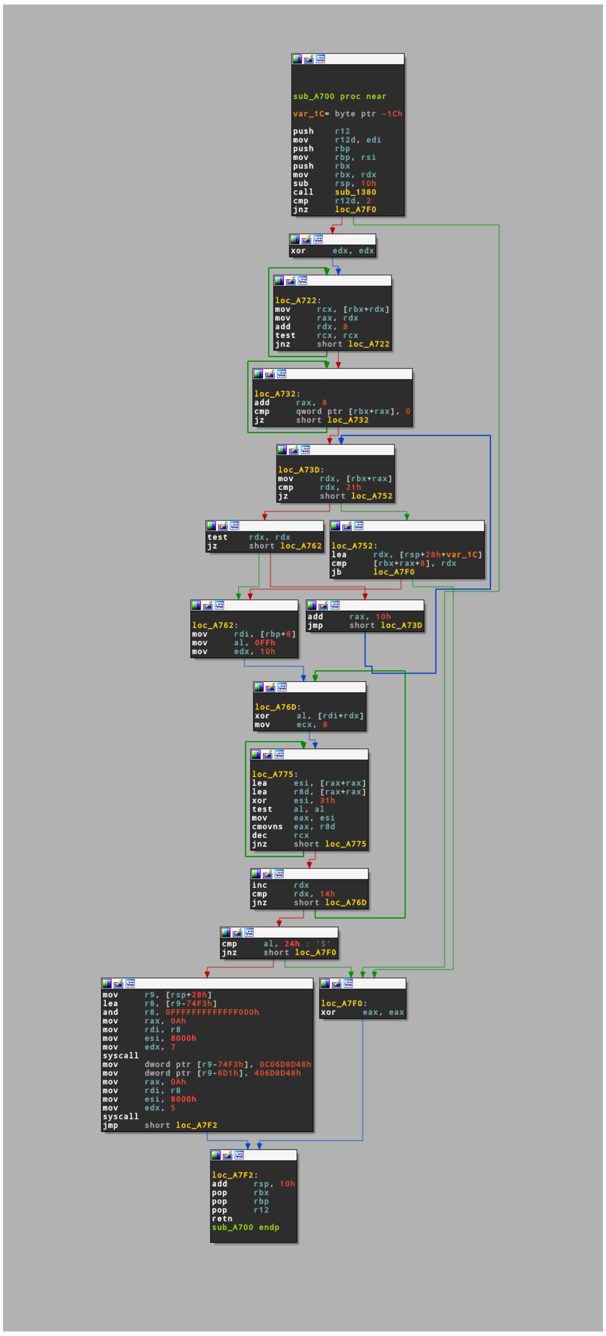
(c) Radare 2

Fig. 4: The backdoor in several common analysis tools

3 Application

At this point, one can freely specify backdoor functionality. In context of this work, desirable functionality might be the slight modification of a well-known cryptographic algorithm contained within the binary. A good example of such an algorithm might be a standard MD5 implementation (`https://github.com/krisprice/simd_md5`) that gets patched in such a way that it implements a reduced-round (12) version of MD5 instead. One feasible way of achieving this is moving the base pointer (`rbp`) to a stack location such that all further calculations on the internal MD5 state get discarded after returning from the function context.

Reducing the number of rounds used in MD5 from 64 to 12 breaks the hash's preimage resistance entirely, and allows reducing the hash to an SMT problem. Giving enough related inputs, the problem is constrained enough to be solved relatively quickly by an SMT solver such as Microsoft's `z3` (Listing 1.1).



```
sub_A700 proc near

var_1C= byte ptr -1Ch

push    r12
mov     r12d, edi
push    rbp
mov     rbp, rsi
push    rbx
mov     rbx, rdx
sub     rsp, 10h
call    sub_1380
cmp     r12d, 2
jnz     loc_A7F0
```

```
xor     edx, edx
```

```
loc_A722:
mov     rcx, [rbx+rdx]
mov     rax, rdx
add     rdx, 8
test    rcx, rcx
jnz     short loc_A722
```

```
loc_A732:
add     rax, 8
cmp     qword ptr [rbx+rax], 0
jz      short loc_A732
```

```
loc_A73D:
mov     rdx, [rbx+rax]
cmp     rdx, 21h
jz      short loc_A752
```

```
test    rdx, rdx
jz      short loc_A762
```

```
loc_A752:
lea     rdx, [rsp+28h+var_1C]
cmp     [rbx+rax+8], rdx
jb      loc_A7F0
```

```
loc_A762:
mov     rdi, [rbp+8]
mov     al, 0FFh
mov     edx, 10h
```

```
add     rax, 10h
jmp     short loc_A73D
```

```
loc_A76D:
xor     al, [rdi+rdx]
mov     ecx, 8
```

```
loc_A775:
lea     esi, [rax+rax]
lea     r8d, [rax+rax]
xor     esi, 31h
test    al, al
mov     eax, esi
cmovns  eax, r8d
dec     rcx
jnz     short loc_A775
```

```
inc     rdx
cmp     rdx, 14h
jnz     short loc_A76D
```

```
cmp     al, 24h ; '$'
jnz     short loc_A7F0
```

```
mov     r9, [rsp+28h]
lea     r8, [r9-74F3h]
and     r8, 0FFFFFFFFFFFFF000h
mov     rax, 0Ah
mov     rdi, r8
mov     esi, 8000h
mov     edx, 7
syscall
mov     dword ptr [r9-74F3h], 0C06D8D48h
mov     dword ptr [r9-6D1h], 406D8D48h
mov     rax, 0Ah
mov     rdi, r8
mov     esi, 8000h
mov     edx, 5
syscall
jmp     short loc_A7F2
```

```
loc_A7F0:
xor     eax, eax
```

```
loc_A7F2:
add     rsp, 10h
pop     rbx
pop     rbp
pop     r12
retn
sub_A700 endp
```

Fig. 5: Backdooring an implementation of MD5 (the mov operations in the penultimate basic block constitute the actual patch of the program image)

```
import struct, z3

h = bytes.fromhex(’3ed50eac373185348499454857b06fd3’) # md5(flag ^ ’h’)
x = bytes.fromhex(’448582faa78b404a898d0532542d327b’) # md5(flag ^ ’x’)
p = bytes.fromhex(’9973f05fde3fe6320be04a918c5b50ab’) # md5(flag ^ ’p’)

a0, b0, c0, d0 = 0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476
ah, bh, ch, dh = struct.unpack(’IIII’, h)
ax, bx, cx, dx = struct.unpack(’IIII’, x)
ap, bp, cp, dp = struct.unpack(’IIII’, p)

unknown_words = z3.BitVecs(’f0 f1 f2 f3’, 32)
remaining_words = struct.unpack(’I’ * 12, b’\x80’ + b’\x00’ * 39 + struct.
    pack(’Q’, 128))

h_flag = [w ^ 0x68686868 for w in unknown_words] + list(remaining_words)
x_flag = [w ^ 0x78787878 for w in unknown_words] + list(remaining_words)
p_flag = [w ^ 0x70707070 for w in unknown_words] + list(remaining_words)

K = [0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
     0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
     0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be]
S = [7, 12, 17, 22] * 3
def FF(b, c, d):
    return (b & c) | ((~b) & d)
def u32(value):
    return (value + (1 << 32)) % (1 << 32) if isinstance(value, int) else
        value
def ror(value, shift):
    if isinstance(value, int):
        shift %= 32
        shifted = u32(value) >> shift
        excess = value & ((1 << shift) - 1)
        return shifted | (excess << (32 - shift))
    return z3.RotateRight(value, shift)
def invert_md5(a, b, c, d, values):
    a -= a0
    b -= b0
    c -= c0
    d -= d0
    for r in range(11, -1, -1):
        B, C, D = c, d, a
        a_t1 = u32(ror(b - c, S[r]) - K[r])
        a_t2 = u32(a_t1 - values[r])
        A = u32(a_t2 - FF(B, C, D))
        a, b, c, d = A, B, C, D
    print(a, b, c, d)
    return z3.And(a == a0, b == b0, c == c0, d == d0)

ss = z3.Solver()
print(’Adding H flag’)
ss.add(invert_md5(ah, bh, ch, dh, h_flag))
print(’Adding X flag’)
ss.add(invert_md5(ax, bx, cx, dx, x_flag))
print(’Adding P flag’)
ss.add(invert_md5(ap, bp, cp, dp, p_flag))
print(’Solving’)
print(ss.check())

m = ss.model()
result = struct.pack(’IIII’, *[int(str(m.evaluate(w))) for w in
    unknown_words])
print(’hxp{’ + result.decode() + ’}’)
```

Listing 1.1: Breaking 12 rounds of MD5 with an SMT solver

4 Related Work

Similar work on ELF backdoors was presented in [1] (including hiding backdoor code in code caves such as the padding behind the `frame_dummy` function), but they do not use the flaws in analysis tools to further obfuscate the presence of such a backdoor.

5 Conclusion

This paper presents both a novel method of inserting a backdoor into ELF executables and of hiding the presence of such a backdoor from most common analysis tools. Future research should be performed to identify other edge cases in which analysis tools (and analysts) incorrectly assume that no backdoor is present, and automated means of detecting such backdoors should be identified.

Bibliography

[1] Aymeric Mouillard Pierre Graux and Mounir Saoud. Backdooring ELF using unused code. pages 1–6, 2016.

[2] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.