black hat ASIA 2022

Non-Intrusive Vulnerability Localization and Hotpatching for Industrial Control Systems

Prashant Hari Narayan Rajput

Michail Maniatakos

black hat®
ASIA 2022

Non-Intrusive Vulnerability Localization and Hotpatching for Industrial Control Systems

Prashant Hari Narayan Rajput

Michail Maniatakos

Michail Maniatakos (@realMoMAlab) received the B.Sc. and M.Sc. degrees in Computer Science and Embedded Systems from the University of Piraeus, Greece, and the Ph.D. degree in Electrical Engineering and the M.Sc. and M.Phil. degrees from Yale University, New Haven, CT, USA. He is currently an Associate Professor of Electrical and Computer Engineering with New York University (NYU) Abu Dhabi, Abu Dhabi, UAE, and a Research Associate Professor with the NYU Tandon School of Engineering, New York, NY, USA. He is also the Director of the MoMA Laboratory, NYU Abu Dhabi. His research interests, funded by industrial partners, the US Government, and the UAE Government, include robust microprocessor architectures, privacy-preserving computation, smart cities, as well as industrial control systems security. He has authored several publications in IEEE transactions and conferences, holds patents on privacy-preserving data processing, and also serves in the technical program committee for various international conferences.

black hat ASIA 2022

Non-Intrusive Vulnerability Localization and Hotpatching for Industrial Control Systems

Prashant Hari Narayan Rajput

Prashant Hari Narayan Rajput received the B.E. degree in Computer Engineering from Savitribai Phule Pune University and the M.S. in Computer Science degree from the University of California Los Angeles. He is currently pursuing a Ph.D. degree in Computer Science from New York University Tandon School of Engineering, Brooklyn, NY, USA. His research focuses on malware detection and vulnerability patching for embedded systems focusing on Industrial Control Systems while maintaining non-intrusiveness on the target device.

Michail Maniatakos

Michail Maniatakos (@realMoMAlab) received the B.Sc. and M.Sc. degrees in Computer Science and Embedded Systems from the University of Piraeus, Greece, and the Ph.D. degree in Electrical Engineering and the M.Sc. and M.Phil. degrees from Yale University, New Haven, CT, USA. He is currently an Associate Professor of Electrical and Computer Engineering with New York University (NYU) Abu Dhabi, Abu Dhabi, UAE, and a Research Associate Professor with the NYU Tandon School of Engineering, New York, NY, USA. He is also the Director of the MoMA Laboratory, NYU Abu Dhabi. His research interests, funded by industrial partners, the US Government, and the UAE Government, include robust microprocessor architectures, privacy-preserving computation, smart cities, as well as industrial control systems security. He has authored several publications in IEEE transactions and conferences, holds patents on privacy-preserving data processing, and also serves in the technical program committee for various international conferences.

Industrial Control Systems

- Industrial Control Systems
  - Control systems and associated instrumentation
  - Continuous deployed operation
  - Ex: PLCs, IEDs, SCADA, etc.

Industrial Control Systems

- Industrial Control Systems
  - Control systems and associated instrumentation
  - Continuous deployed operation
  - Ex: PLCs, IEDs, SCADA, etc.

- Critical Infrastructure
  - Vital assets
  - Desalination plant, Power grids, etc.

Industrial Control Systems

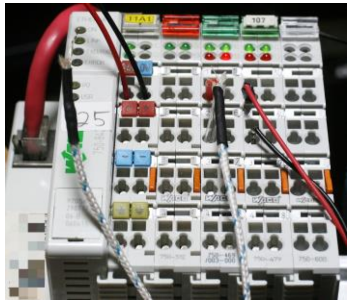
- Industrial Control Systems
  - Control systems and associated instrumentation
  - Continuous deployed operation
  - Ex: PLCs, IEDs, SCADA, etc.

- Critical Infrastructure
  - Vital assets
  - Desalination plant, Power grids, etc.

- Reliable uninterrupted operation
  - Scheduled maintenance – 5 years

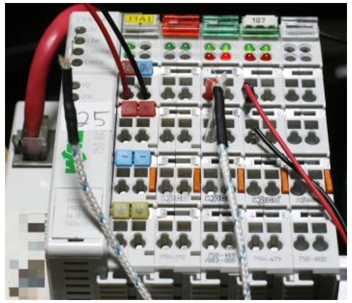
Industrial Control Systems

- Digitization Trend
  - Integrating automation and data exchange
- Modern ICS
  - General-purpose OS (Linux)
  - Additional functionality



Industrial Control Systems

- Digitization Trend
  - Integrating automation and data exchange
- Modern ICS
  - General-purpose OS (Linux)
  - Additional functionality



- IT threats into OT
  - Runtime relies on standard libraries
  - Out-of-bounds write/read, OS command injection, etc

Industrial Control Systems

- Long operation life cycle
  - Outdated OS/firmware + IT threats
  - Unpatched exploitable vulnerabilities

Industrial Control Systems

- Long operation life cycle
  - Outdated OS/firmware + IT threats
  - Unpatched exploitable vulnerabilities

- ICS is not designed for security
  - Vulnerabilities in process logic can impact the runtime

Industrial Control Systems

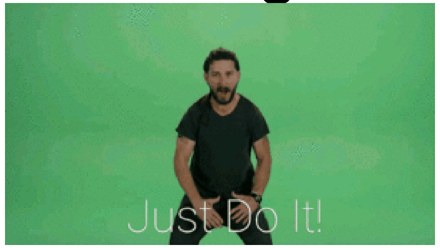
- Long operation life cycle
  - Outdated OS/firmware + IT threats
  - Unpatched exploitable vulnerabilities

- ICS is not designed for security
  - Vulnerabilities in process logic can impact the runtime

- Limited computation power

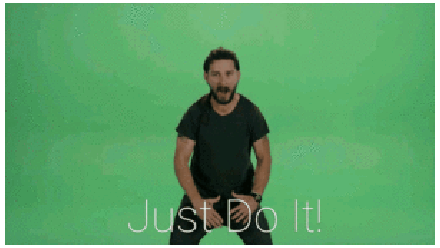
Industrial Control Systems

- How can one protect against an unpatched vulnerability in an ICS device?
  - Just patch it



Industrial Control Systems

- How can one protect against an unpatched vulnerability in an ICS device?
  - Just patch it

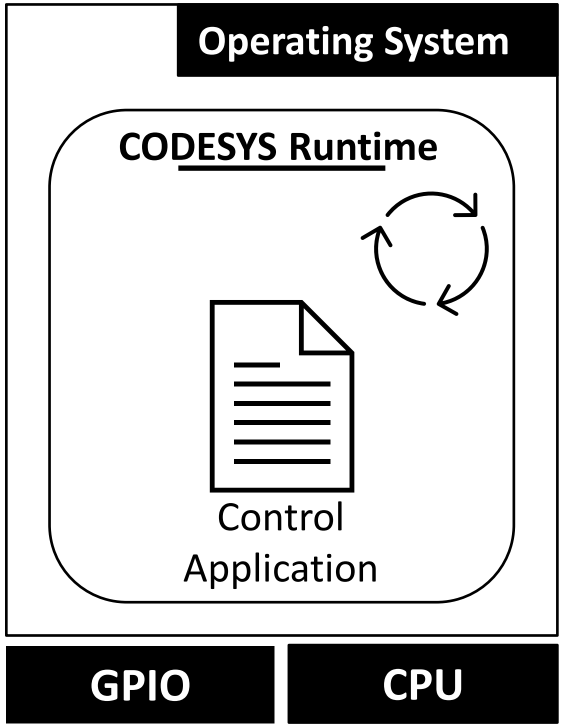


- **This talk:** Hot patching vulnerabilities in control application
  - **Step 1:** Extract process memory hexdumps
  - **Step 2:** Initialize Angr with hexdumps and violation rules
  - **Step 3:** Create patch based on the Angr instance
  - **Step 4:** Get live base addresses from deployed PLC and patch

- Utilize LKMs for non-intrusive patching

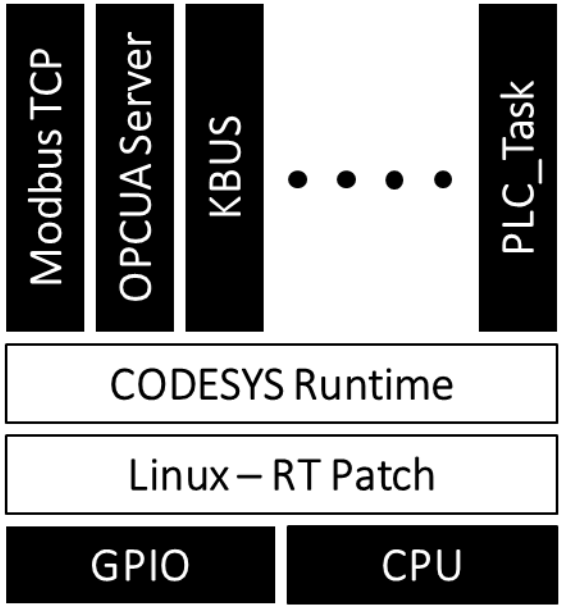
Codesys Runtime

- Programmable Logic Controllers (PLCs)
  - A rugged industrial computer
- Codesys Runtime
  - Collection of components necessary for proper execution of the application binary.
- Scan Cycle
  - Continuously scan program, input scan, execute program, output scan



Codesys Runtime

- IEC Application
  - Executes in PLC_TASK thread
  - Shares process memory with the runtime



Demo – The Codesys Stack

Experiment Setup

Demo – The Codesys Stack

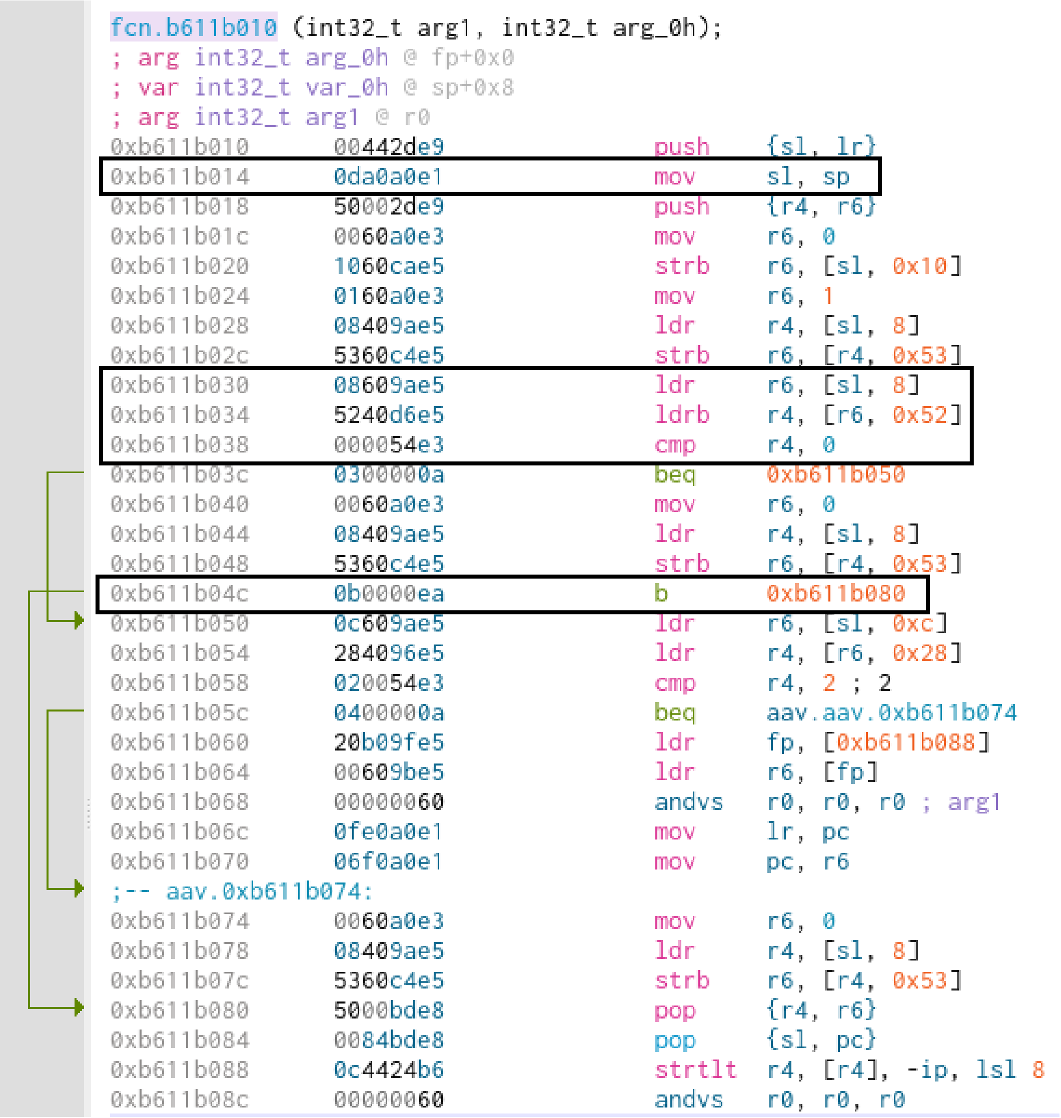
- What happened?
  - Overwrote an important location on the Codesys runtime stack

- How?
  - Using pointers in Structured Text

- Impact
  - Runtime skips the execution of the IEC application

Demo – The Codesys Stack

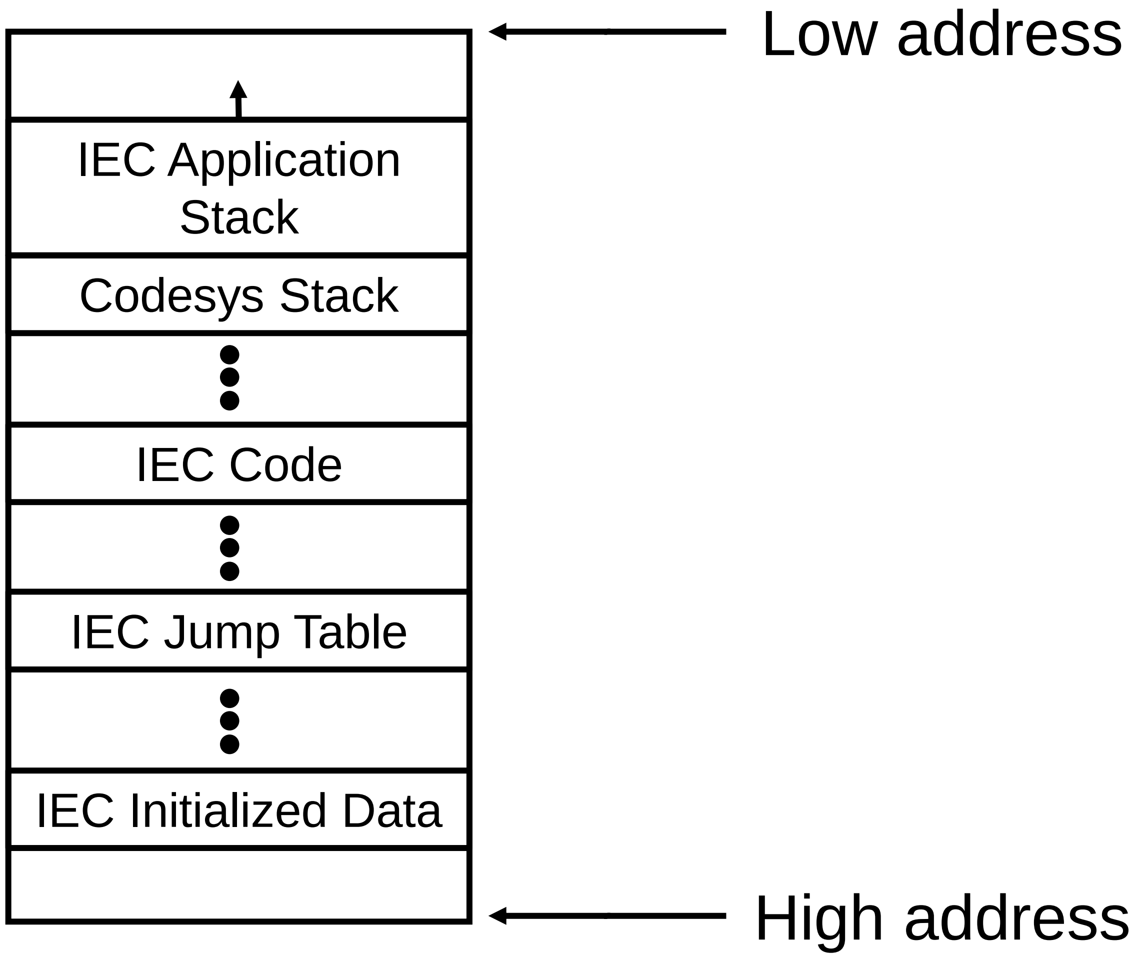
- Why?



```
fcn.b611b010 (int32_t arg1, int32_t arg_0h);
; arg int32_t arg_0h @ fp+0x0
; var int32_t var_0h @ sp+0x8
; arg int32_t arg1 @ r0
0xb611b010      00442de9        push    {sl, lr}
0xb611b014      0da0a0e1        mov     sl, sp
0xb611b018      50002de9        push    {r4, r6}
0xb611b01c      0060a0e3        mov     r6, 0
0xb611b020      1060cae5        strb    r6, [sl, 0x10]
0xb611b024      0160a0e3        mov     r6, 1
0xb611b028      08409ae5        ldr     r4, [sl, 8]
0xb611b02c      5360c4e5        strb    r6, [r4, 0x53]
0xb611b030      08609ae5        ldr     r6, [sl, 8]
0xb611b034      5240d6e5        ldrb    r4, [r6, 0x52]
0xb611b038      000054e3        cmp     r4, 0
0xb611b03c      0300000a        beq     0xb611b050
0xb611b040      0060a0e3        mov     r6, 0
0xb611b044      08409ae5        ldr     r4, [sl, 8]
0xb611b048      5360c4e5        strb    r6, [r4, 0x53]
0xb611b04c      0b0000ea        b       0xb611b080
0xb611b050      0c609ae5        ldr     r6, [sl, 0xc]
0xb611b054      284096e5        ldr     r4, [r6, 0x28]
0xb611b058      020054e3        cmp     r4, 2 ; 2
0xb611b05c      0400000a        beq     aav.aav.0xb611b074
0xb611b060      20b09fe5        ldr     fp, [0xb611b088]
0xb611b064      00609be5        ldr     r6, [fp]
0xb611b068      00000060        andvs   r0, r0, r0 ; arg1
0xb611b06c      0fe0a0e1        mov     lr, pc
0xb611b070      06f0a0e1        mov     pc, r6
;-- aav.0xb611b074:
0xb611b074      0060a0e3        mov     r6, 0
0xb611b078      08409ae5        ldr     r4, [sl, 8]
0xb611b07c      5360c4e5        strb    r6, [r4, 0x53]
0xb611b080      5000bde8        pop     {r4, r6}
0xb611b084      0084bde8        pop     {sl, pc}
0xb611b088      0c4424b6        strtlt  r4, [r4], -ip, lsl 8
0xb611b08c      00000060        andvs   r0, r0, r0
```

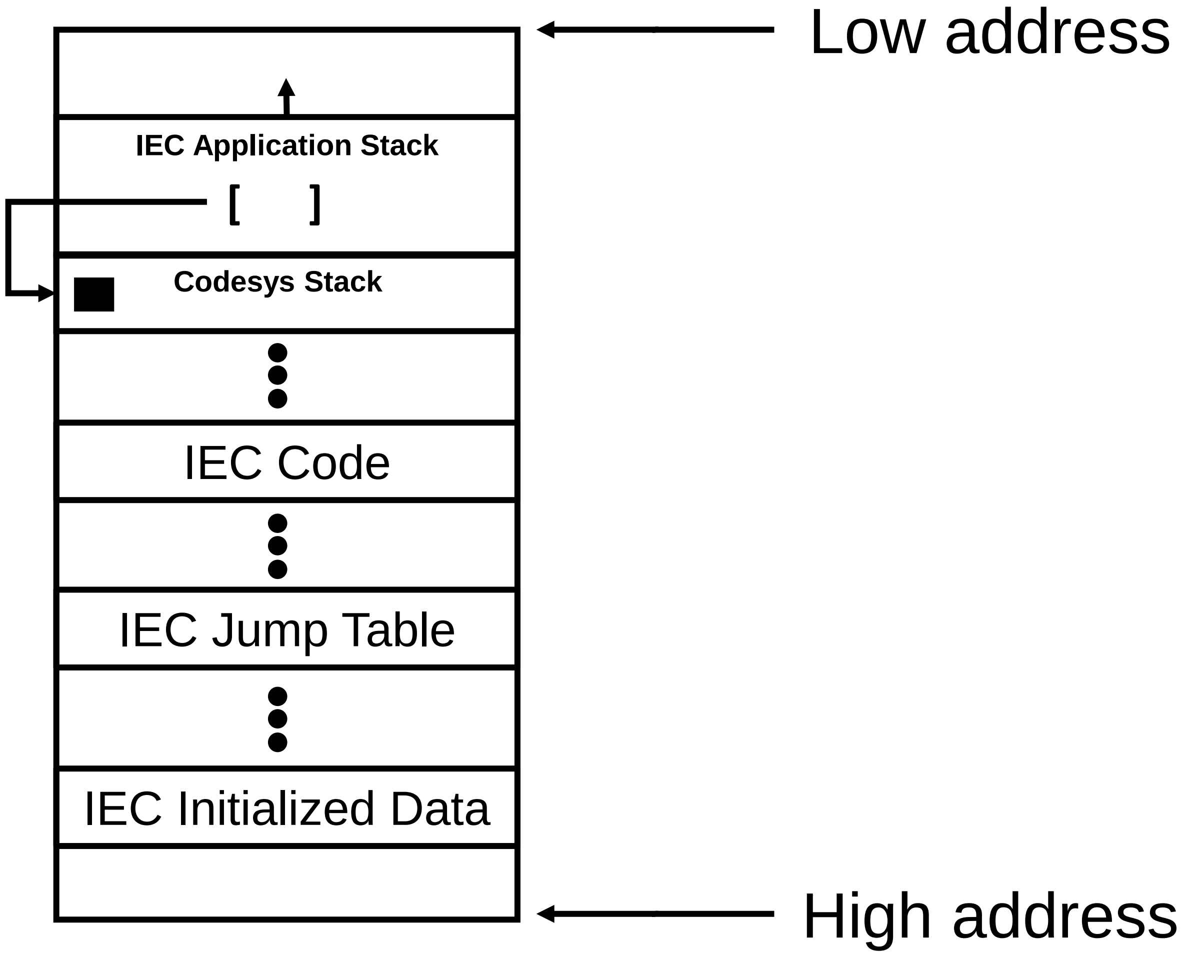
Demo – The Codesys Stack

- Why?
  - Shared stack



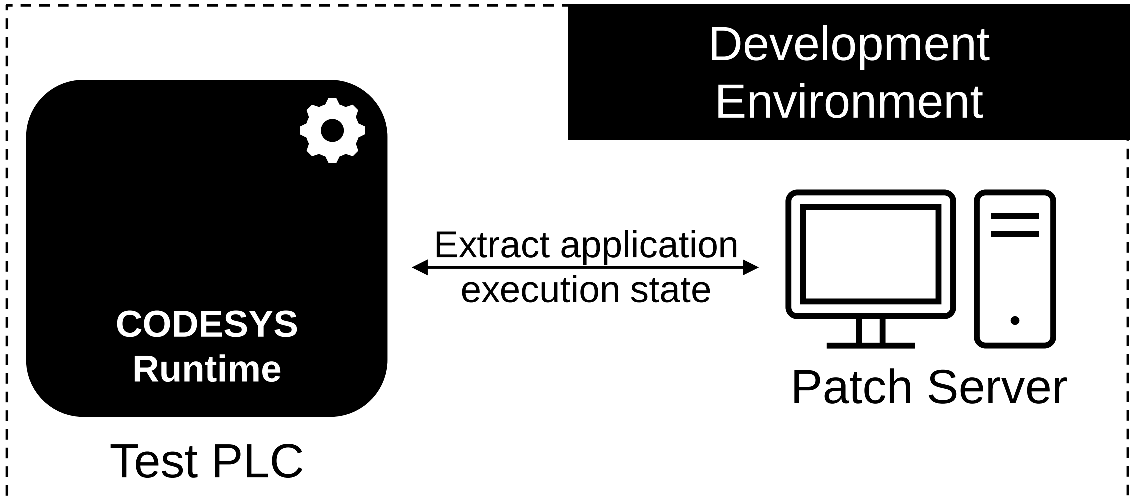
Demo – The Codesys Stack

- Why?
  - Shared stack
- Control the runtime state from the IEC Application
- Vulnerabilities in the IEC application require hotpatching



Hotpatching

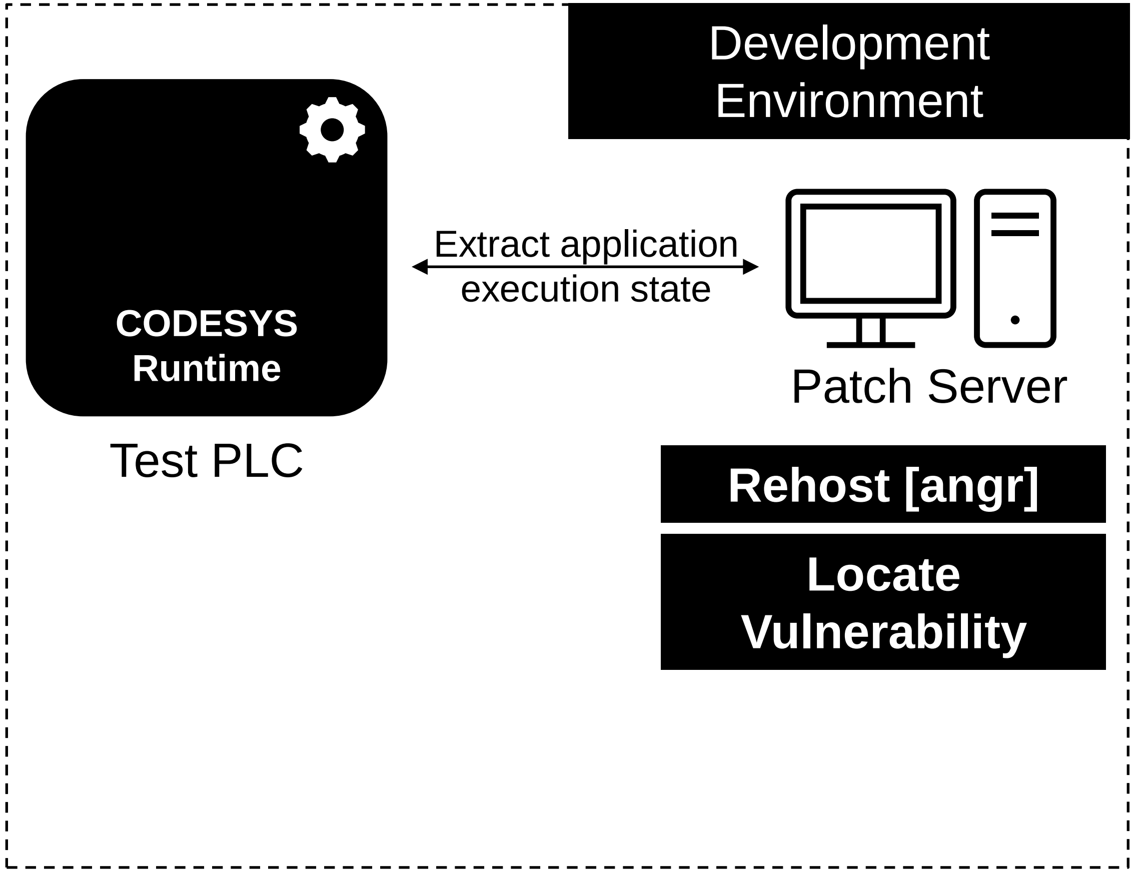
- **Step 1:** Extract execution state from Codesys runtime



- **Hexdumps**
  - Codesys Runtime
  - IEC application
  - Shared library: Libc, Libm
- **Register estimates**
  - PC
  - SP

Hotpatching

- **Step 2:** Rehost in Angr and execute the IEC application



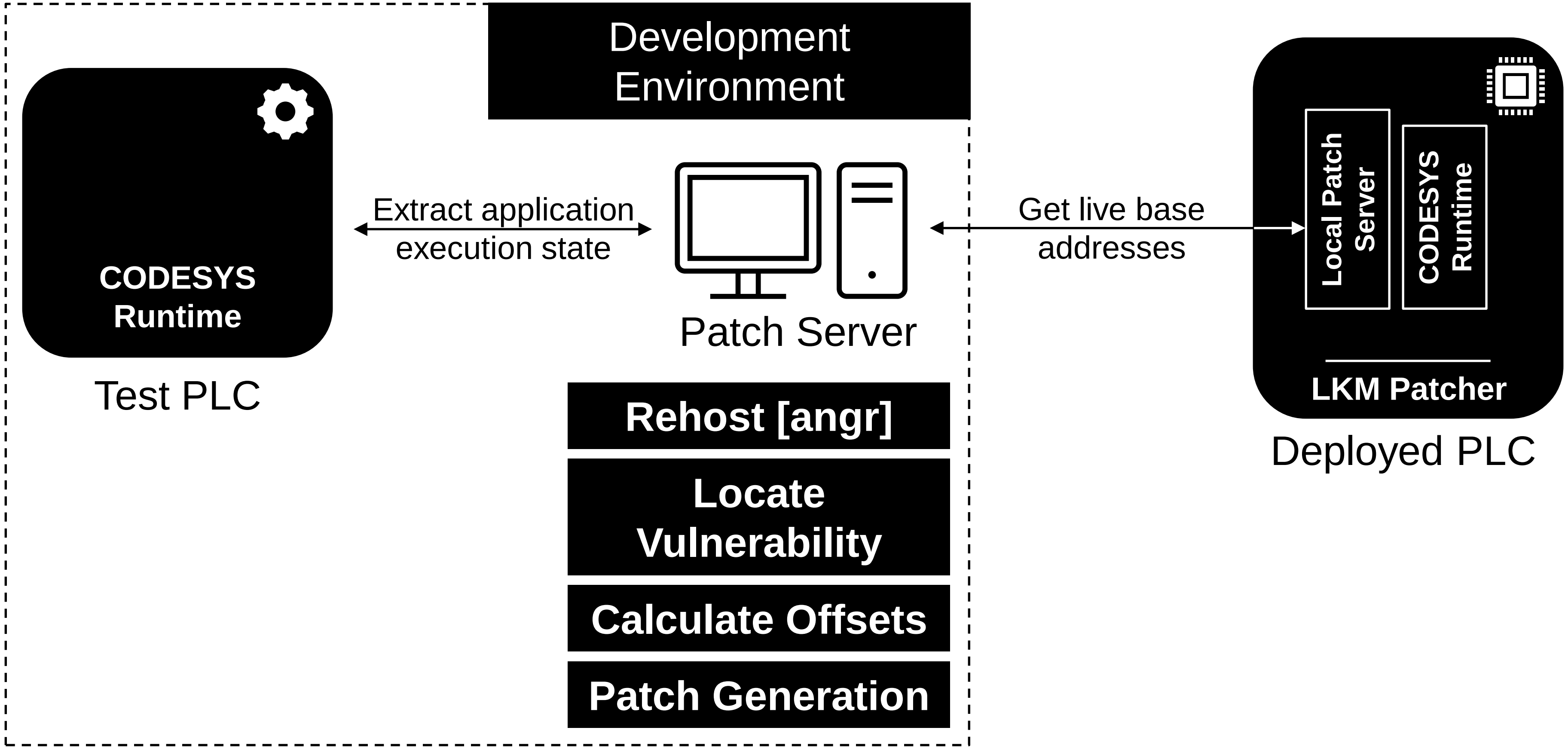
Development Environment

Rehost [angr]

Locate Vulnerability

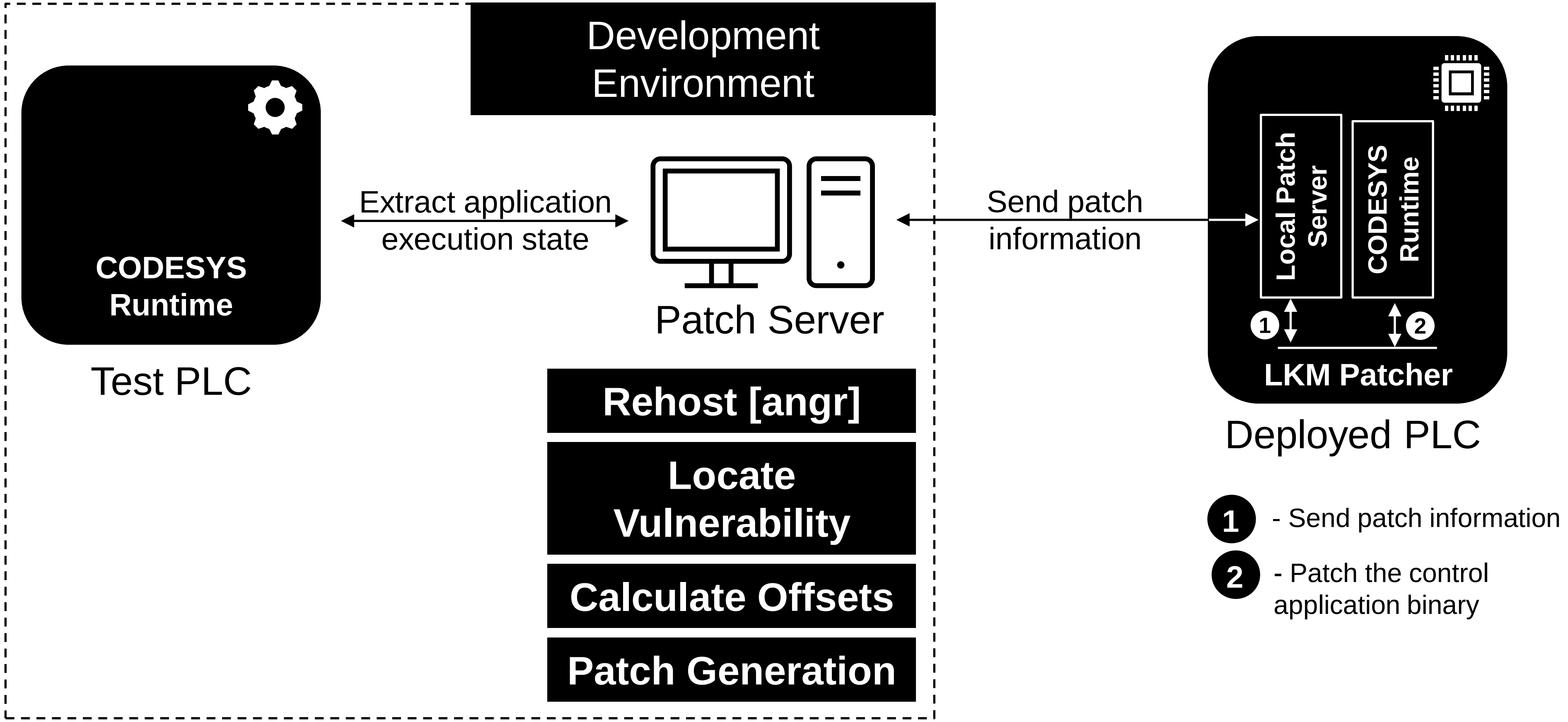
Hotpatching

- **Step 3:** Create a patch based on the Angr execution instance



Hotpatching

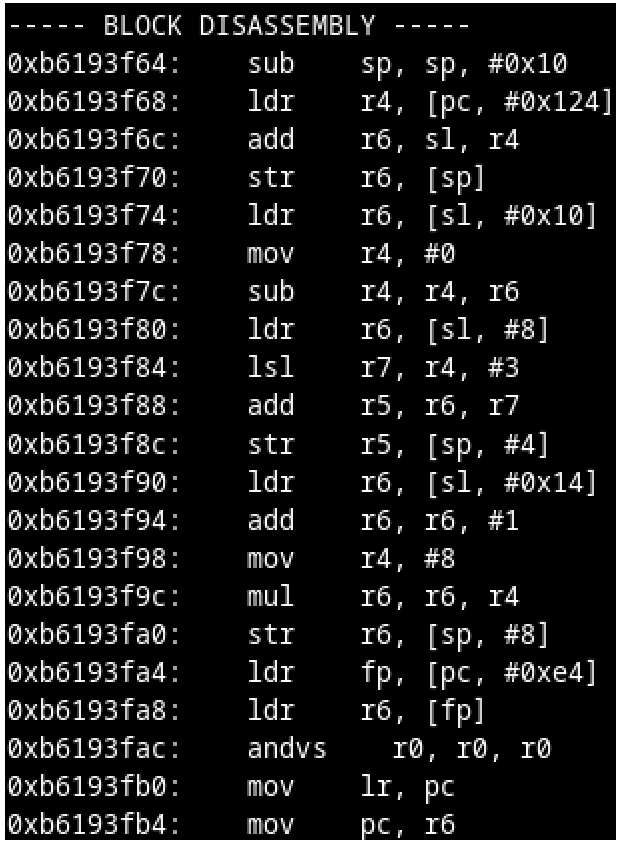
- **Step 4:** Get live base addresses and patch



Hotpatching Specifics

- Where to patch?
  - Vulnerability Localization with DDG*

IEC Application

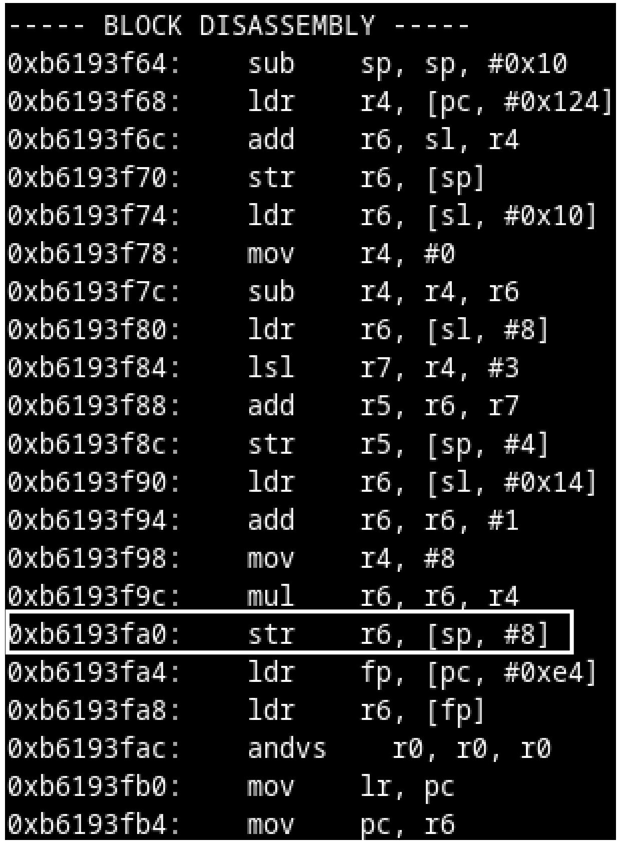


```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

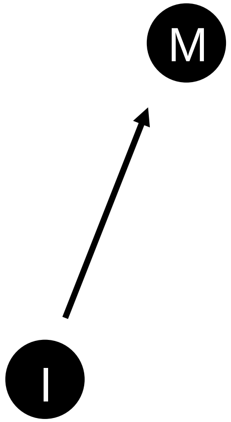
Hotpatching Specifics

- Where to patch?
  - Vulnerability Localization with DDG*

IEC Application

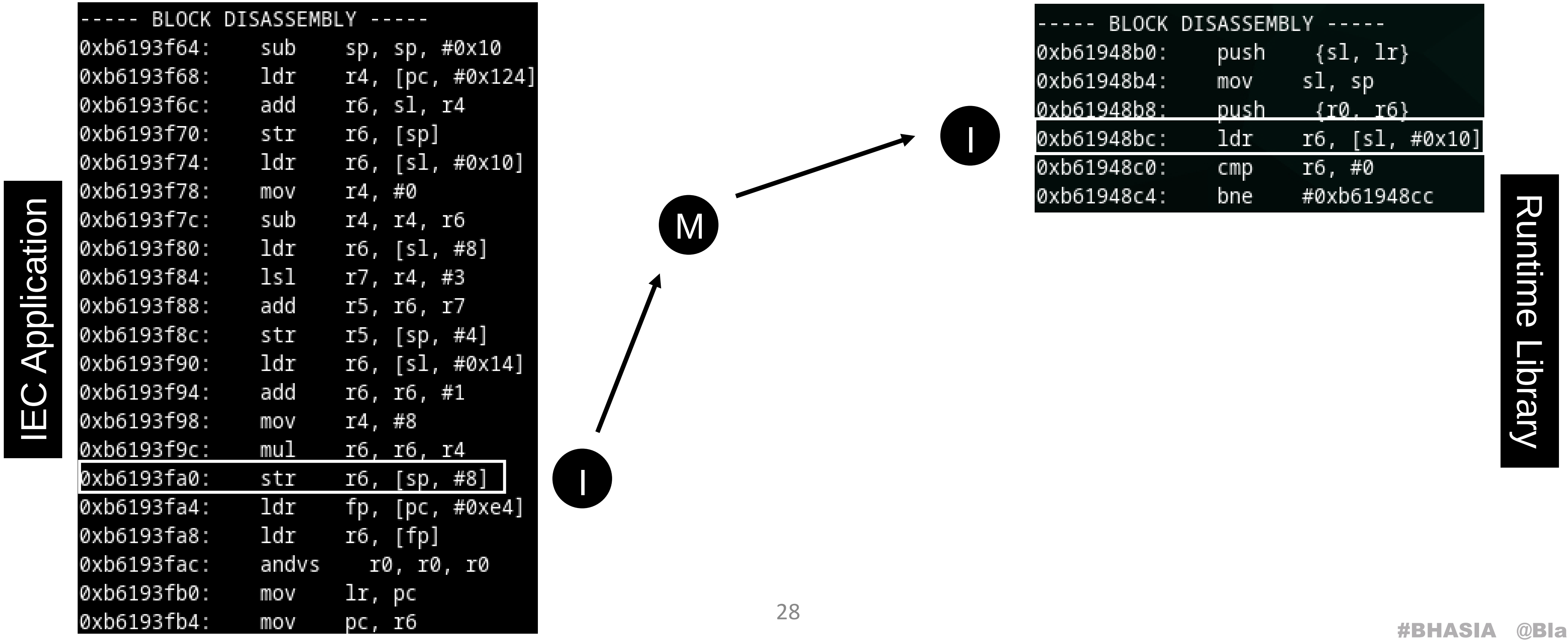


```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```



Hotpatching Specifics

- Where to patch?
  - Vulnerability Localization with DDG*



IEC Application

```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

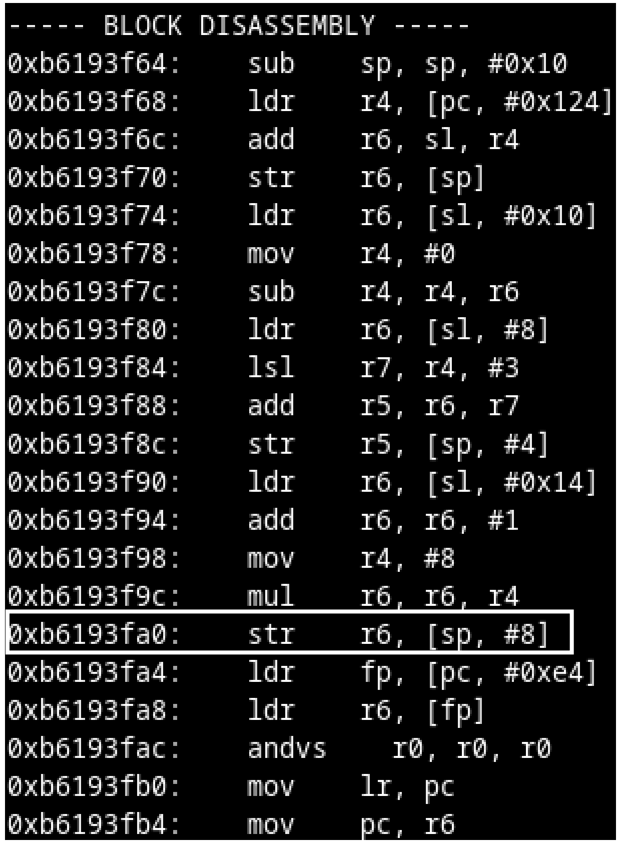
```
----- BLOCK DISASSEMBLY -----
0xb61948b0:    push    {sl, lr}
0xb61948b4:    mov    sl, sp
0xb61948b8:    push    {r0, r6}
0xb61948bc:    ldr    r6, [sl, #0x10]
0xb61948c0:    cmp    r6, #0
0xb61948c4:    bne    #0xb61948cc
```

Runtime Library

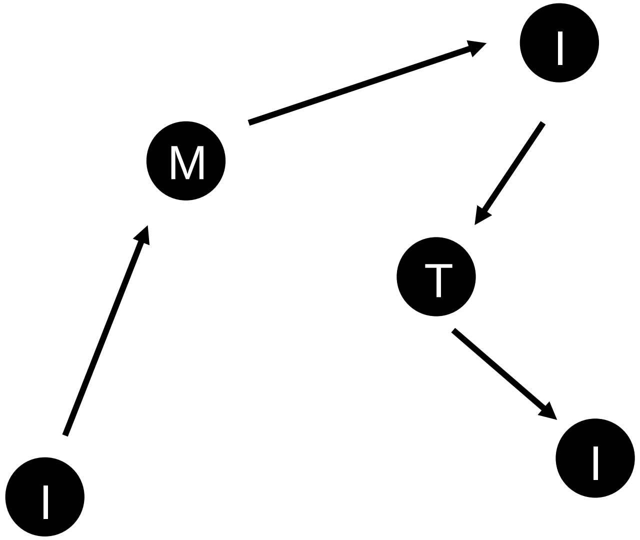
Hotpatching Specifics

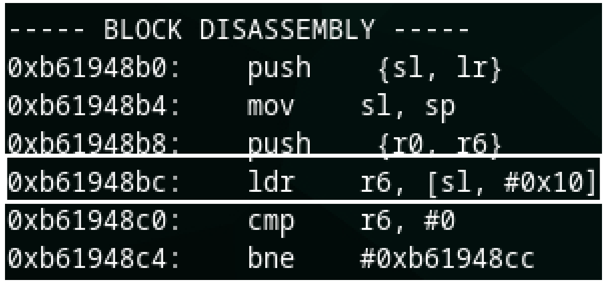
- Where to patch?
  - Vulnerability Localization with DDG*

IEC Application

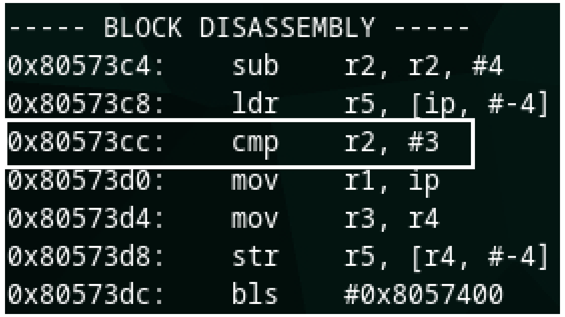


```
----- BLOCK DISASSEMBLY -----
0xb6193f64:     sub     sp, sp, #0x10
0xb6193f68:     ldr     r4, [pc, #0x124]
0xb6193f6c:     add     r6, sl, r4
0xb6193f70:     str     r6, [sp]
0xb6193f74:     ldr     r6, [sl, #0x10]
0xb6193f78:     mov     r4, #0
0xb6193f7c:     sub     r4, r4, r6
0xb6193f80:     ldr     r6, [sl, #8]
0xb6193f84:     lsl     r7, r4, #3
0xb6193f88:     add     r5, r6, r7
0xb6193f8c:     str     r5, [sp, #4]
0xb6193f90:     ldr     r6, [sl, #0x14]
0xb6193f94:     add     r6, r6, #1
0xb6193f98:     mov     r4, #8
0xb6193f9c:     mul     r6, r6, r4
0xb6193fa0:     str     r6, [sp, #8]
0xb6193fa4:     ldr     fp, [pc, #0xe4]
0xb6193fa8:     ldr     r6, [fp]
0xb6193fac:     andvs     r0, r0, r0
0xb6193fb0:     mov     lr, pc
0xb6193fb4:     mov     pc, r6
```





```
----- BLOCK DISASSEMBLY -----
0xb61948b0:     push     {sl, lr}
0xb61948b4:     mov     sl, sp
0xb61948b8:     push     {r0, r6}
0xb61948bc:     ldr     r6, [sl, #0x10]
0xb61948c0:     cmp     r6, #0
0xb61948c4:     bne     #0xb61948cc
```

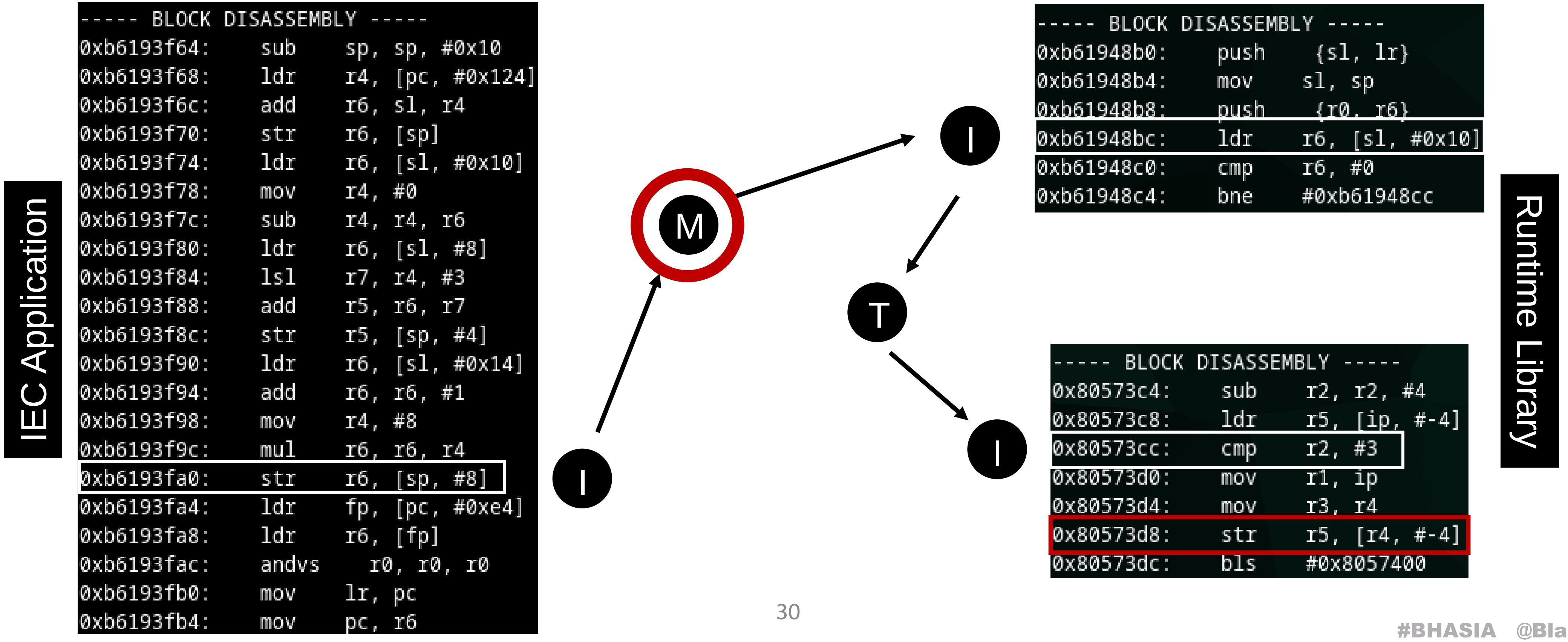


```
----- BLOCK DISASSEMBLY -----
0x80573c4:     sub     r2, r2, #4
0x80573c8:     ldr     r5, [ip, #-4]
0x80573cc:     cmp     r2, #3
0x80573d0:     mov     r1, ip
0x80573d4:     mov     r3, r4
0x80573d8:     str     r5, [r4, #-4]
0x80573dc:     bls     #0x8057400
```

Runtime Library

Hotpatching Specifics

- Where to patch?
  - Vulnerability Localization with DDG*



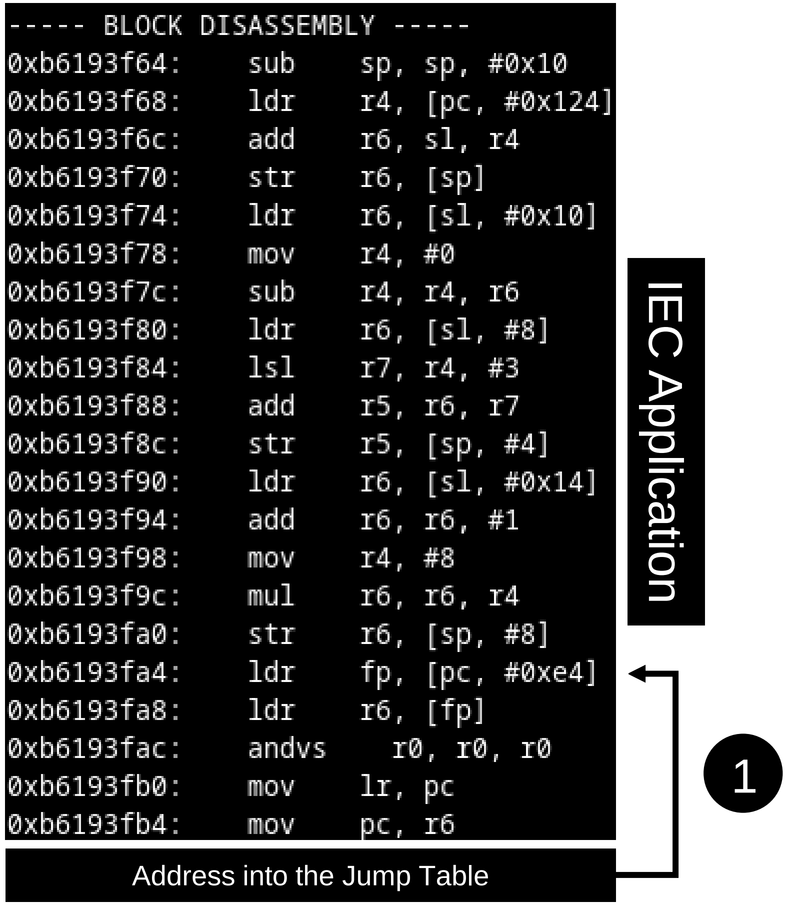
```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub     sp, sp, #0x10
0xb6193f68:    ldr     r4, [pc, #0x124]
0xb6193f6c:    add     r6, sl, r4
0xb6193f70:    str     r6, [sp]
0xb6193f74:    ldr     r6, [sl, #0x10]
0xb6193f78:    mov     r4, #0
0xb6193f7c:    sub     r4, r4, r6
0xb6193f80:    ldr     r6, [sl, #8]
0xb6193f84:    lsl     r7, r4, #3
0xb6193f88:    add     r5, r6, r7
0xb6193f8c:    str     r5, [sp, #4]
0xb6193f90:    ldr     r6, [sl, #0x14]
0xb6193f94:    add     r6, r6, #1
0xb6193f98:    mov     r4, #8
0xb6193f9c:    mul     r6, r6, r4
0xb6193fa0:    str     r6, [sp, #8]
0xb6193fa4:    ldr     fp, [pc, #0xe4]
0xb6193fa8:    ldr     r6, [fp]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov     lr, pc
0xb6193fb4:    mov     pc, r6
```

```
----- BLOCK DISASSEMBLY -----
0xb61948b0:    push    {sl, lr}
0xb61948b4:    mov     sl, sp
0xb61948b8:    push     {r0, r6}
0xb61948bc:    ldr     r6, [sl, #0x10]
0xb61948c0:    cmp     r6, #0
0xb61948c4:    bne     #0xb61948cc
```

```
----- BLOCK DISASSEMBLY -----
0x80573c4:     sub     r2, r2, #4
0x80573c8:     ldr     r5, [ip, #-4]
0x80573cc:     cmp     r2, #3
0x80573d0:     mov     r1, ip
0x80573d4:     mov     r3, r4
0x80573d8:     str     r5, [r4, #-4]
0x80573dc:     bls     #0x8057400
```

Hotpatching Specifics

- How to patch?
  - First, look at branching



```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

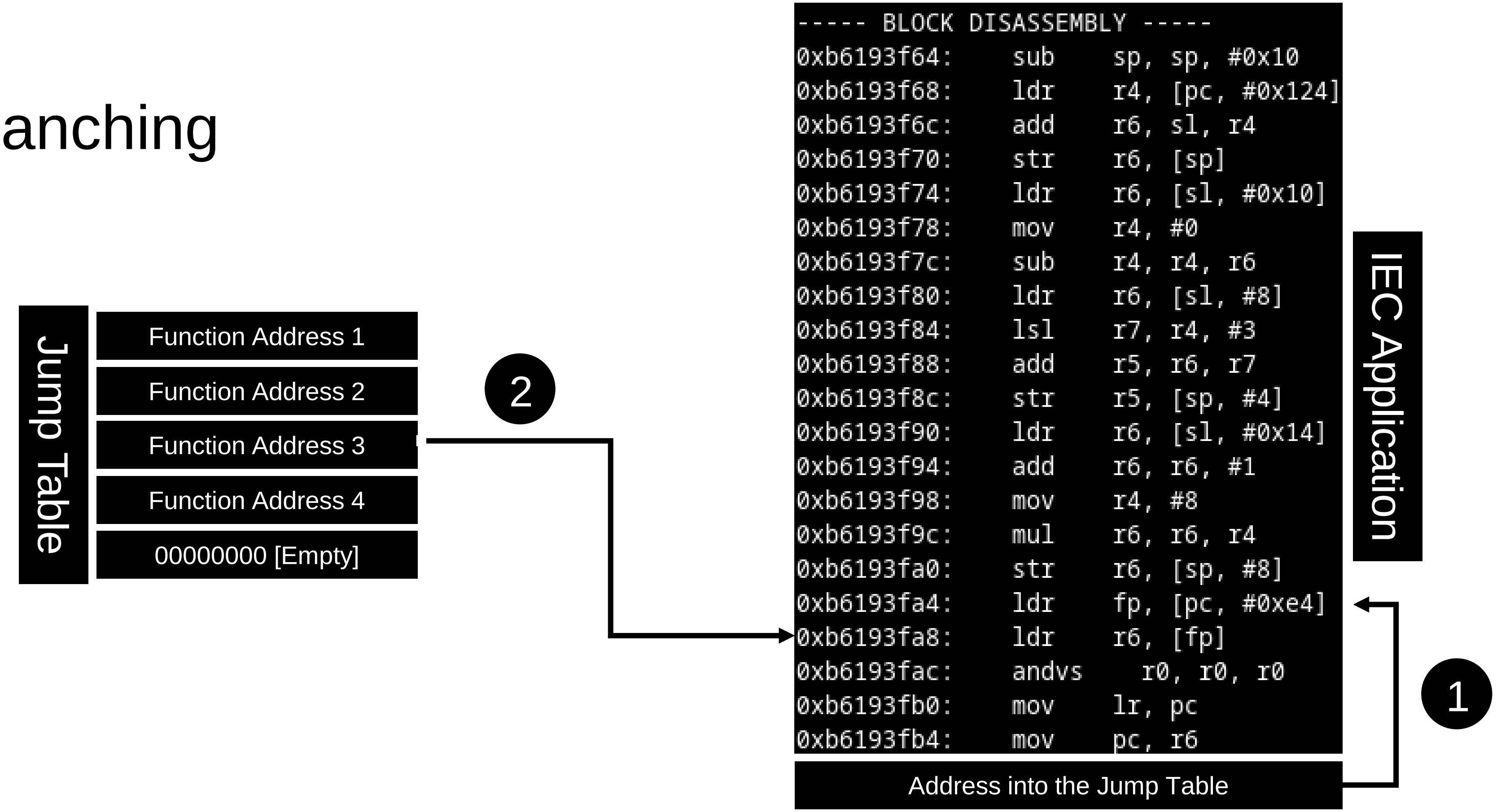
IEC Application

1

Address into the Jump Table

Hotpatching Specifics

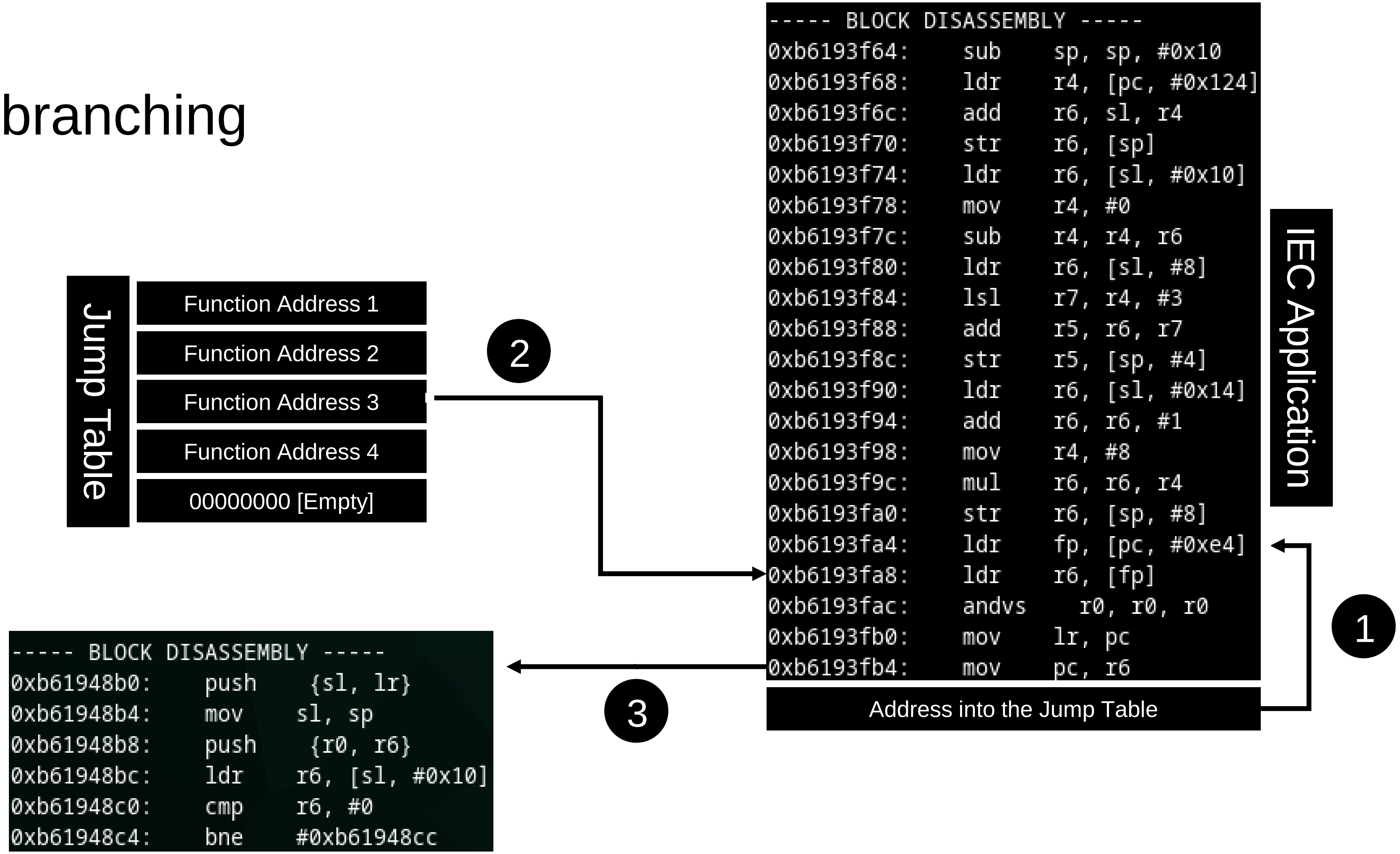
- How to patch?
  - First, look at branching



```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

Hotpatching Specifics

- How to patch?
    - First, look at branching

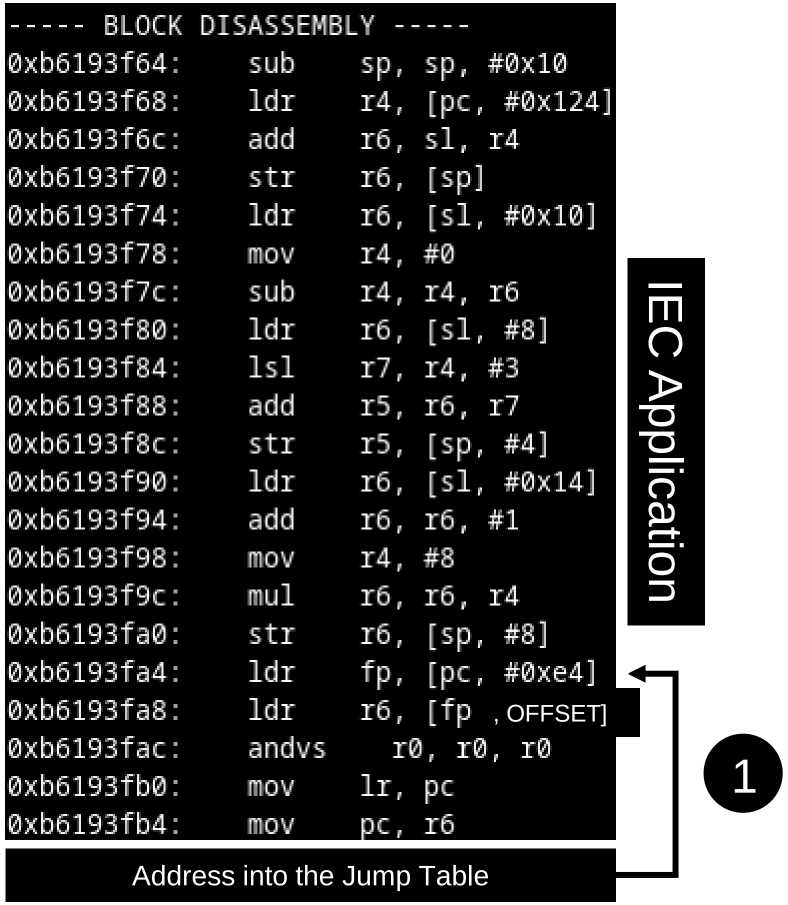


```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

```
----- BLOCK DISASSEMBLY -----
0xb61948b0:    push    {sl, lr}
0xb61948b4:    mov    sl, sp
0xb61948b8:    push    {r0, r6}
0xb61948bc:    ldr    r6, [sl, #0x10]
0xb61948c0:    cmp    r6, #0
0xb61948c4:    bne    #0xb61948cc
```

Hotpatching Specifics

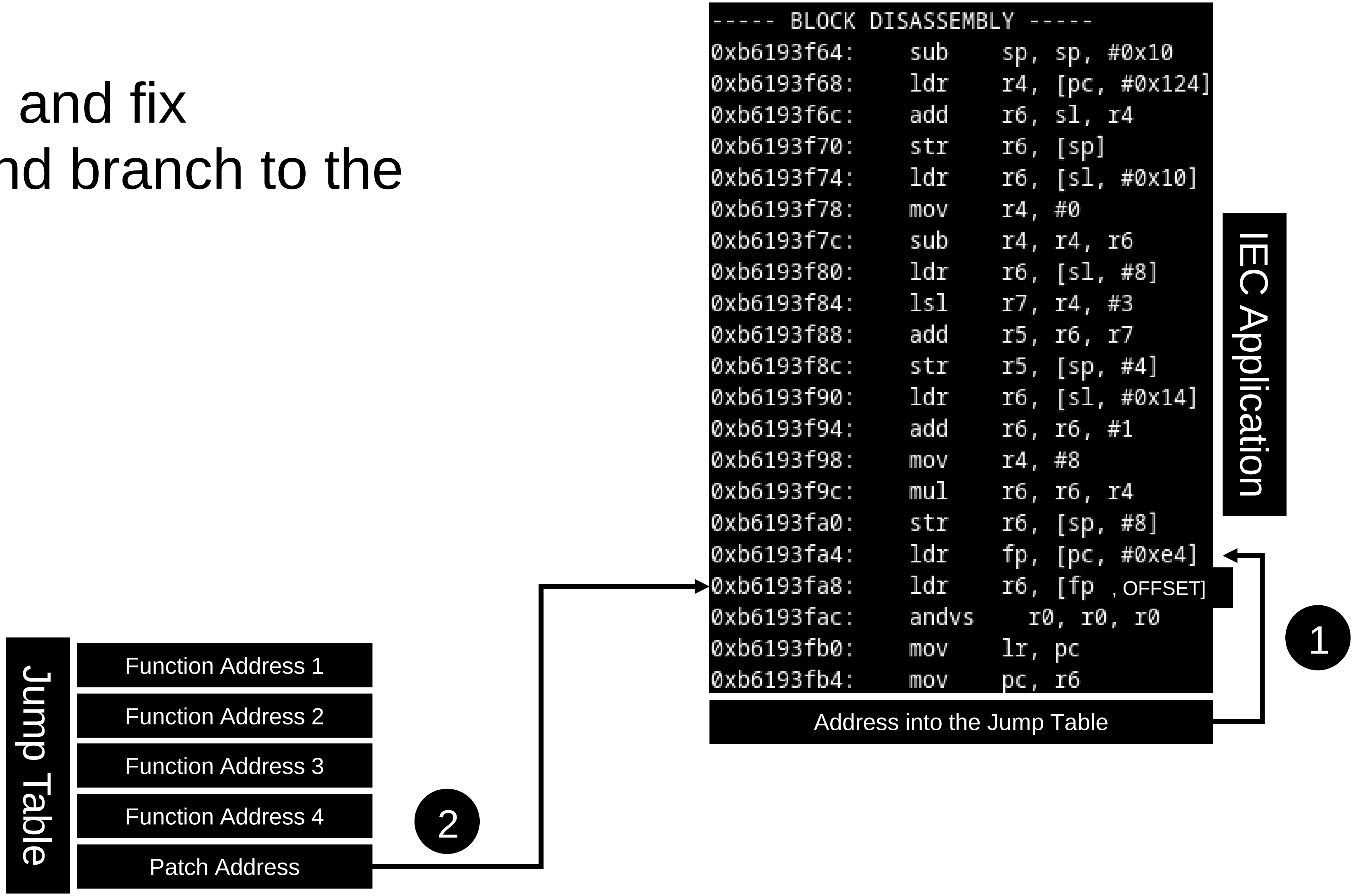
- How to patch?
  - Branch to patch and fix
  - Restore state and branch to the original function



```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp , OFFSET]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

Hotpatching Specifics

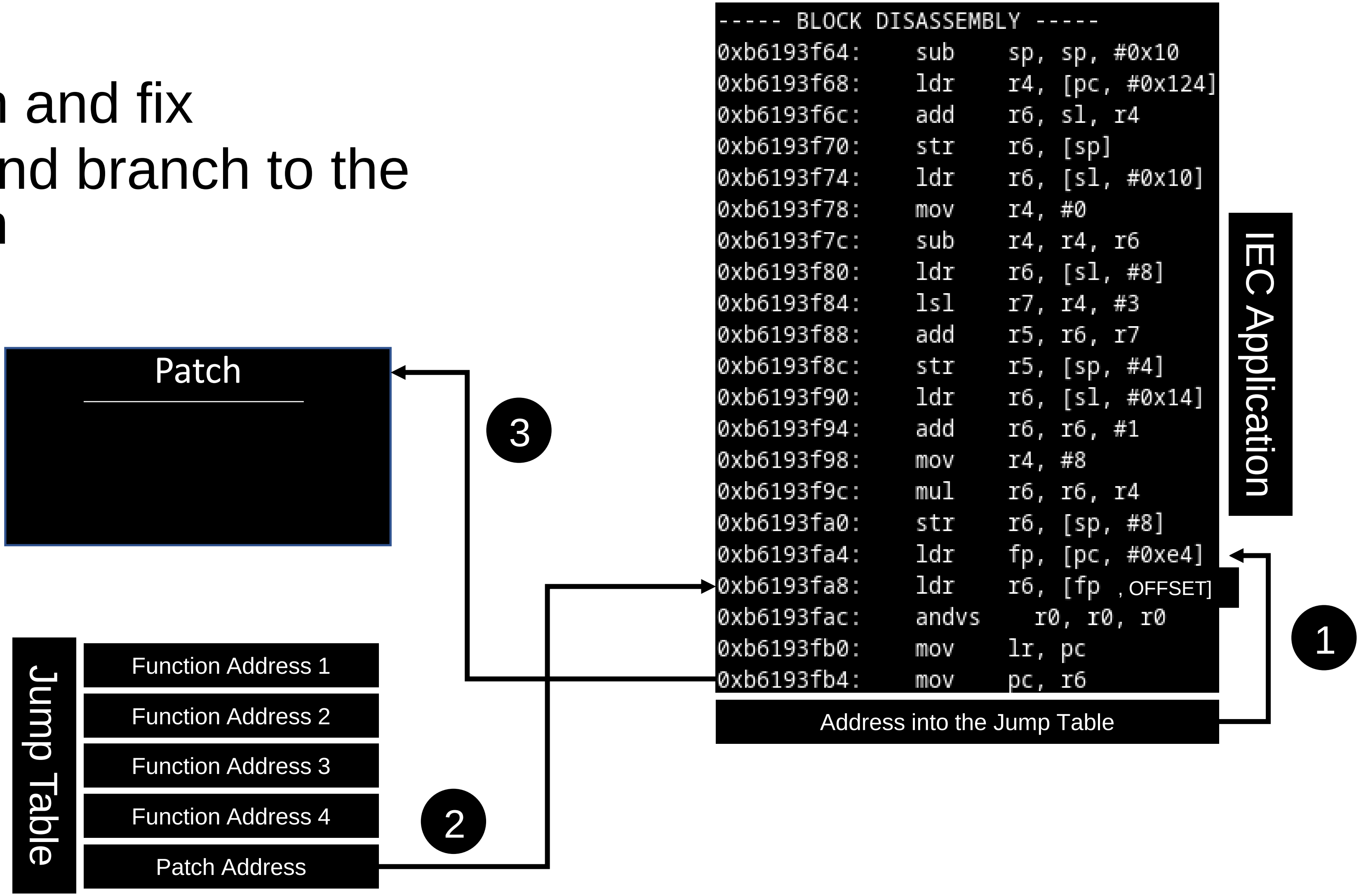
- How to patch?
  - Branch to patch and fix
  - Restore state and branch to the original function



```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp , OFFSET]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

Hotpatching Specifics

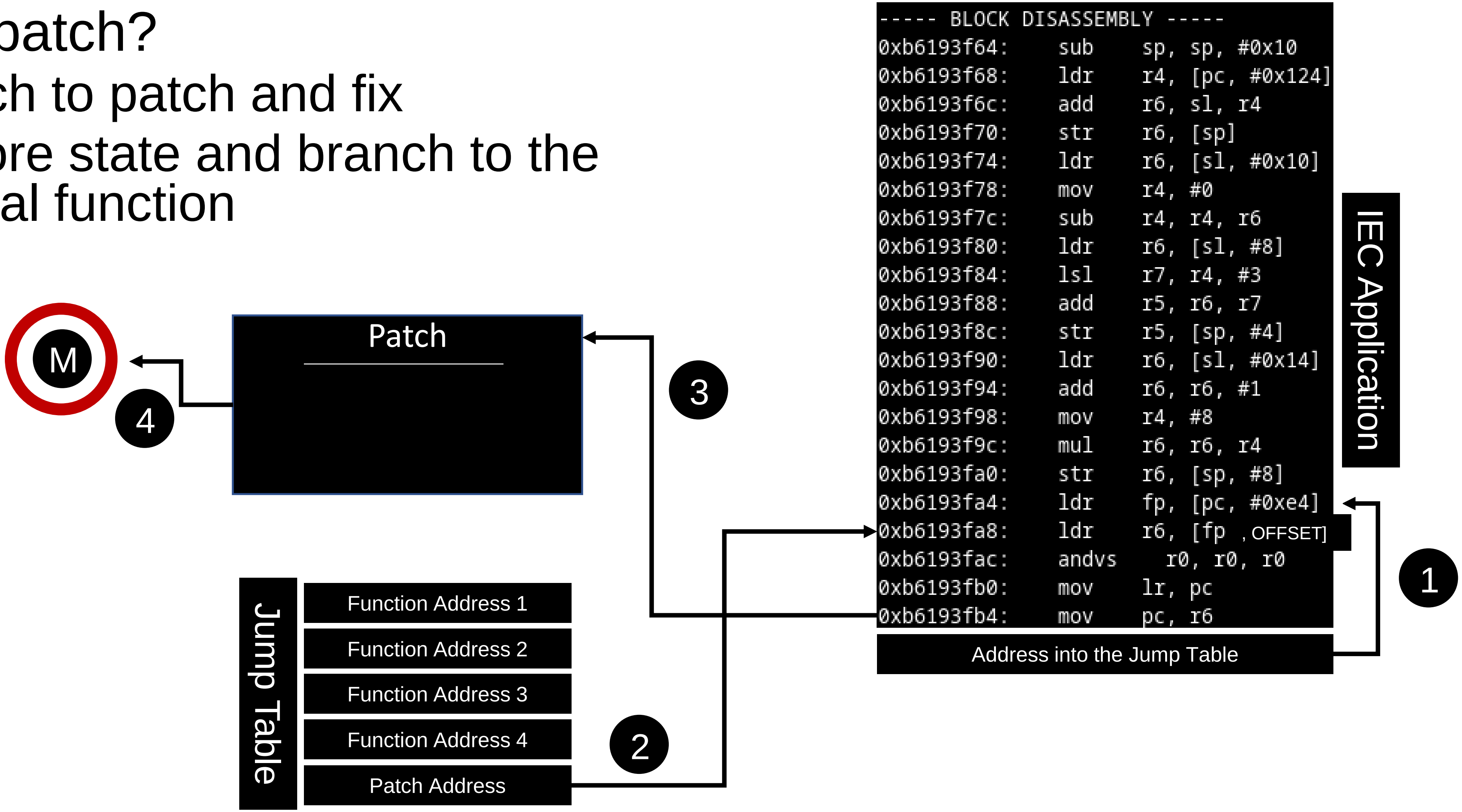
- How to patch?
  - Branch to patch and fix
  - Restore state and branch to the original function



```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp , OFFSET]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

Hotpatching Specifics

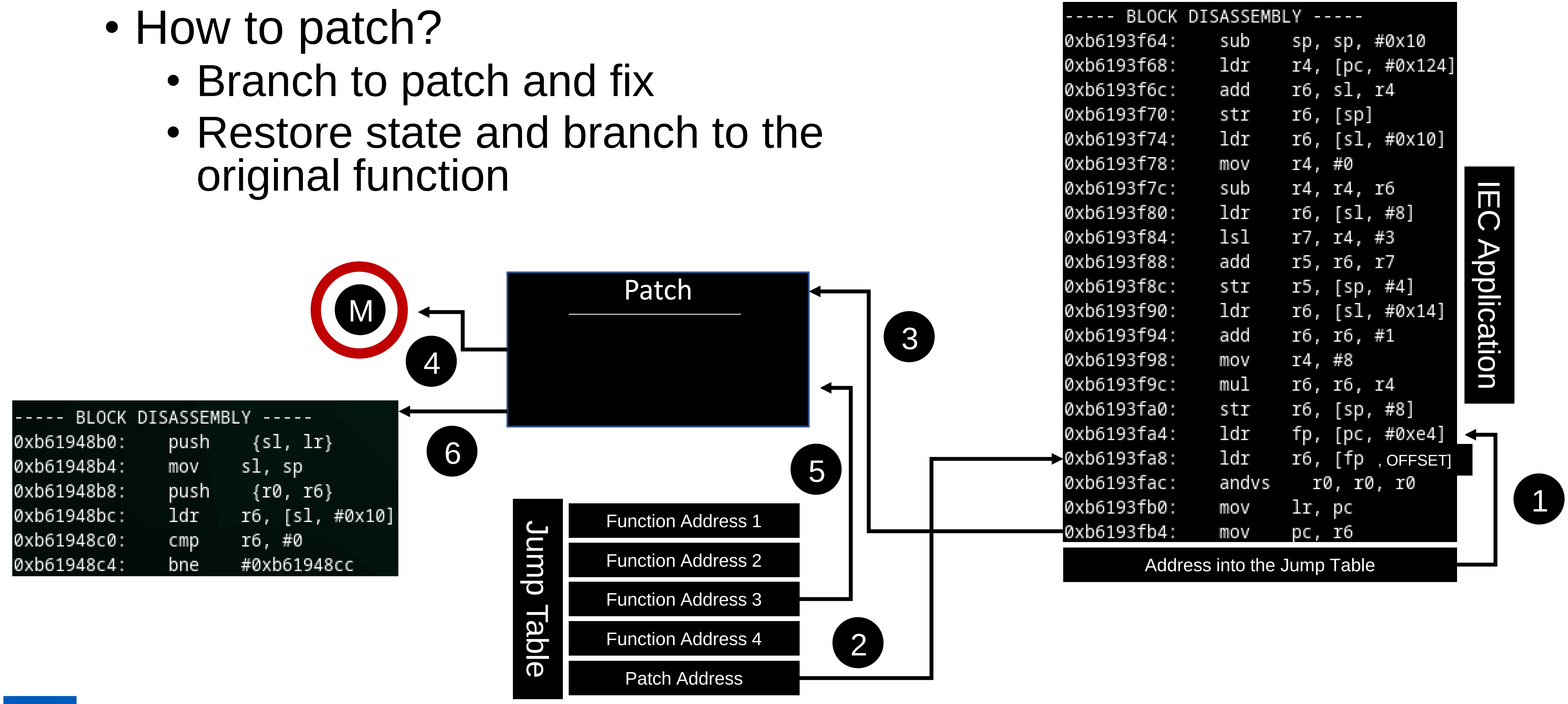
- How to patch?
  - Branch to patch and fix
  - Restore state and branch to the original function



```
----- BLOCK DISASSEMBLY -----
0xb6193f64:   sub    sp, sp, #0x10
0xb6193f68:   ldr    r4, [pc, #0x124]
0xb6193f6c:   add    r6, sl, r4
0xb6193f70:   str    r6, [sp]
0xb6193f74:   ldr    r6, [sl, #0x10]
0xb6193f78:   mov    r4, #0
0xb6193f7c:   sub    r4, r4, r6
0xb6193f80:   ldr    r6, [sl, #8]
0xb6193f84:   lsl    r7, r4, #3
0xb6193f88:   add    r5, r6, r7
0xb6193f8c:   str    r5, [sp, #4]
0xb6193f90:   ldr    r6, [sl, #0x14]
0xb6193f94:   add    r6, r6, #1
0xb6193f98:   mov    r4, #8
0xb6193f9c:   mul    r6, r6, r4
0xb6193fa0:   str    r6, [sp, #8]
0xb6193fa4:   ldr    fp, [pc, #0xe4]
0xb6193fa8:   ldr    r6, [fp , OFFSET]
0xb6193fac:   andvs    r0, r0, r0
0xb6193fb0:   mov    lr, pc
0xb6193fb4:   mov    pc, r6
```

Hotpatching Specifics

- How to patch?
  - Branch to patch and fix
  - Restore state and branch to the original function



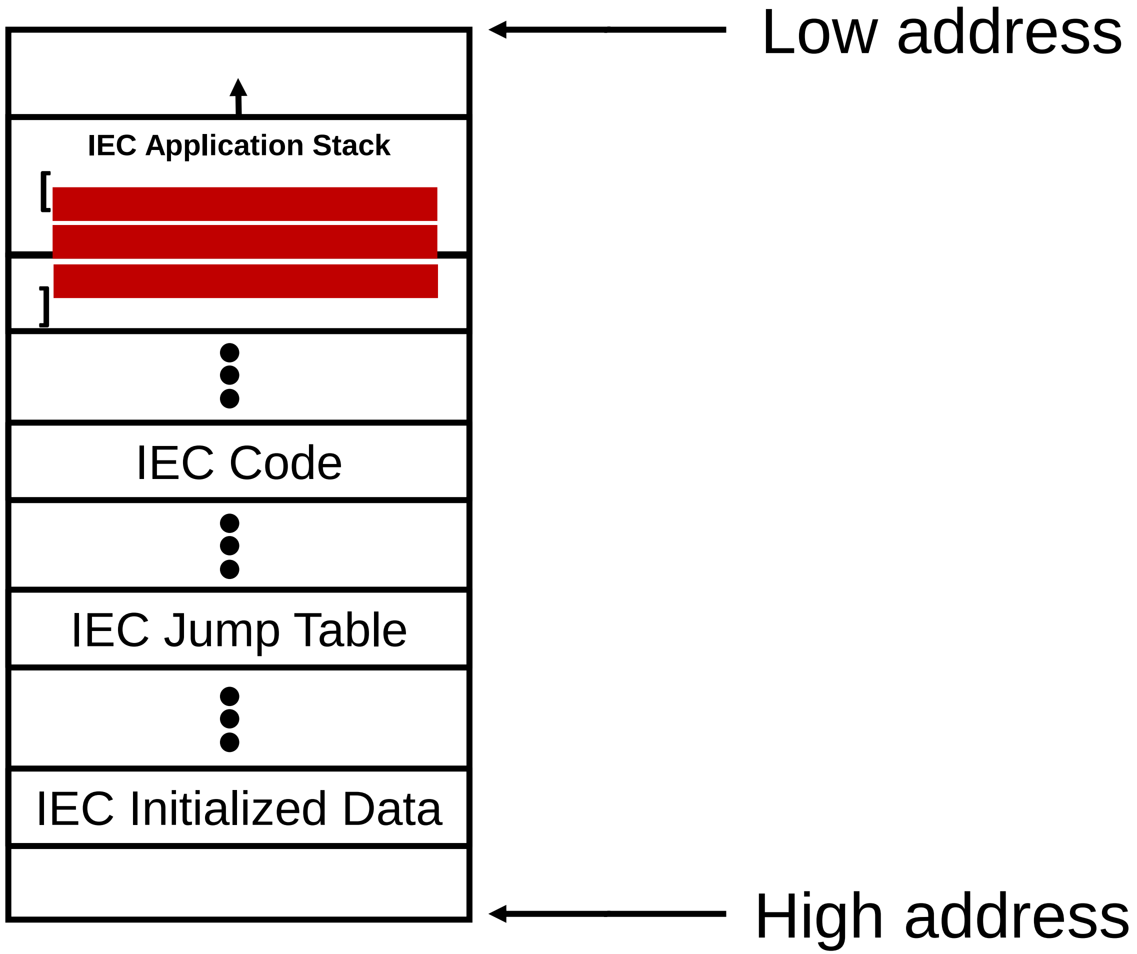
```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub     sp, sp, #0x10
0xb6193f68:    ldr     r4, [pc, #0x124]
0xb6193f6c:    add     r6, sl, r4
0xb6193f70:    str     r6, [sp]
0xb6193f74:    ldr     r6, [sl, #0x10]
0xb6193f78:    mov     r4, #0
0xb6193f7c:    sub     r4, r4, r6
0xb6193f80:    ldr     r6, [sl, #8]
0xb6193f84:    lsl     r7, r4, #3
0xb6193f88:    add     r5, r6, r7
0xb6193f8c:    str     r5, [sp, #4]
0xb6193f90:    ldr     r6, [sl, #0x14]
0xb6193f94:    add     r6, r6, #1
0xb6193f98:    mov     r4, #8
0xb6193f9c:    mul     r6, r6, r4
0xb6193fa0:    str     r6, [sp, #8]
0xb6193fa4:    ldr     fp, [pc, #0xe4]
0xb6193fa8:    ldr     r6, [fp , OFFSET]
0xb6193fac:    andvs     r0, r0, r0
0xb6193fb0:    mov     lr, pc
0xb6193fb4:    mov     pc, r6
```

```
----- BLOCK DISASSEMBLY -----
0xb61948b0:    push    {sl, lr}
0xb61948b4:    mov     sl, sp
0xb61948b8:    push    {r0, r6}
0xb61948bc:    ldr     r6, [sl, #0x10]
0xb61948c0:    cmp     r6, #0
0xb61948c4:    bne     #0xb61948cc
```

Out-of-Bounds Write

- How?
  - Missing bound check
  - Overwrite small buffer
  - Overwrites critical runtime return addresses

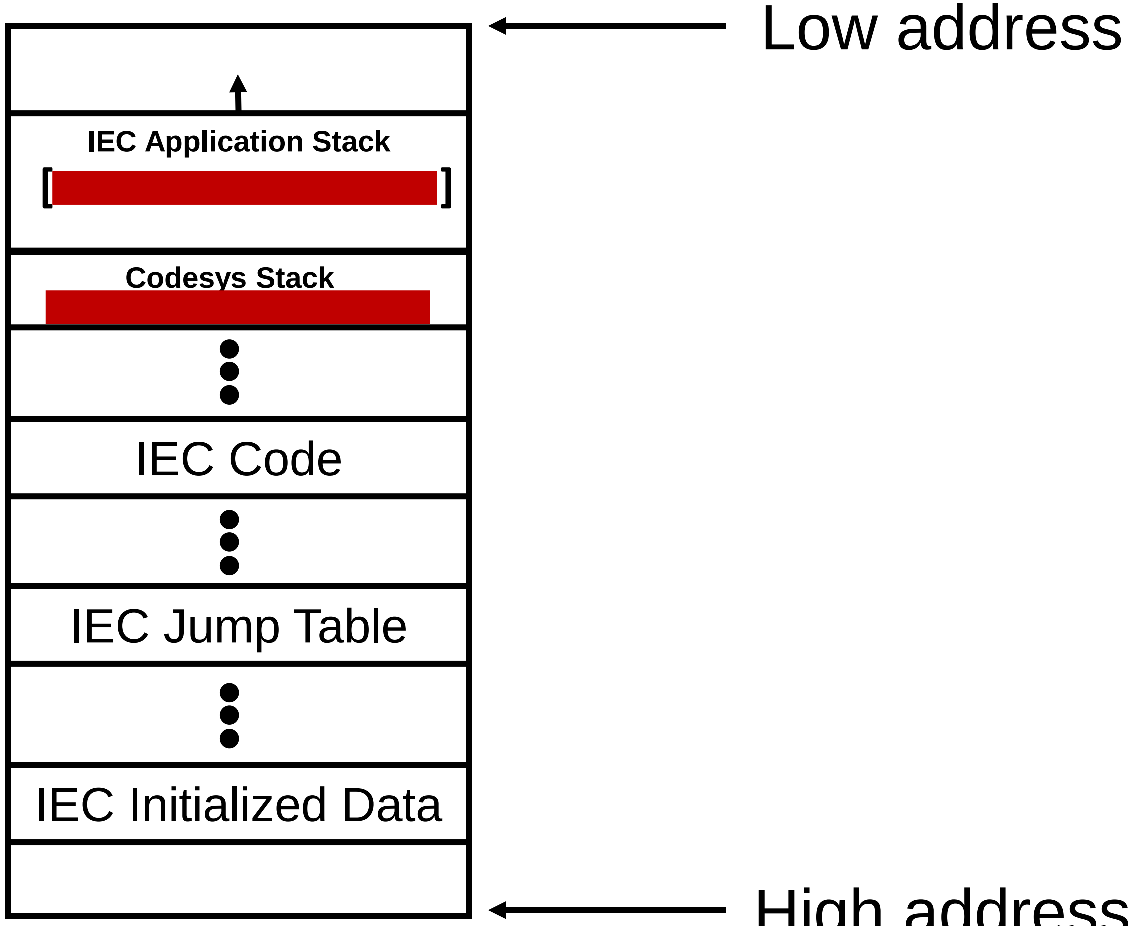
- Impact
  - Messes with the control flow
  - DoS
  - Requires runtime reboot



Out-of-Bounds Read

- How?
  - Missing bound check
  - Reads from the shared stack and code

- Impact
  - Can leak secrets
  - Does not crash the runtime



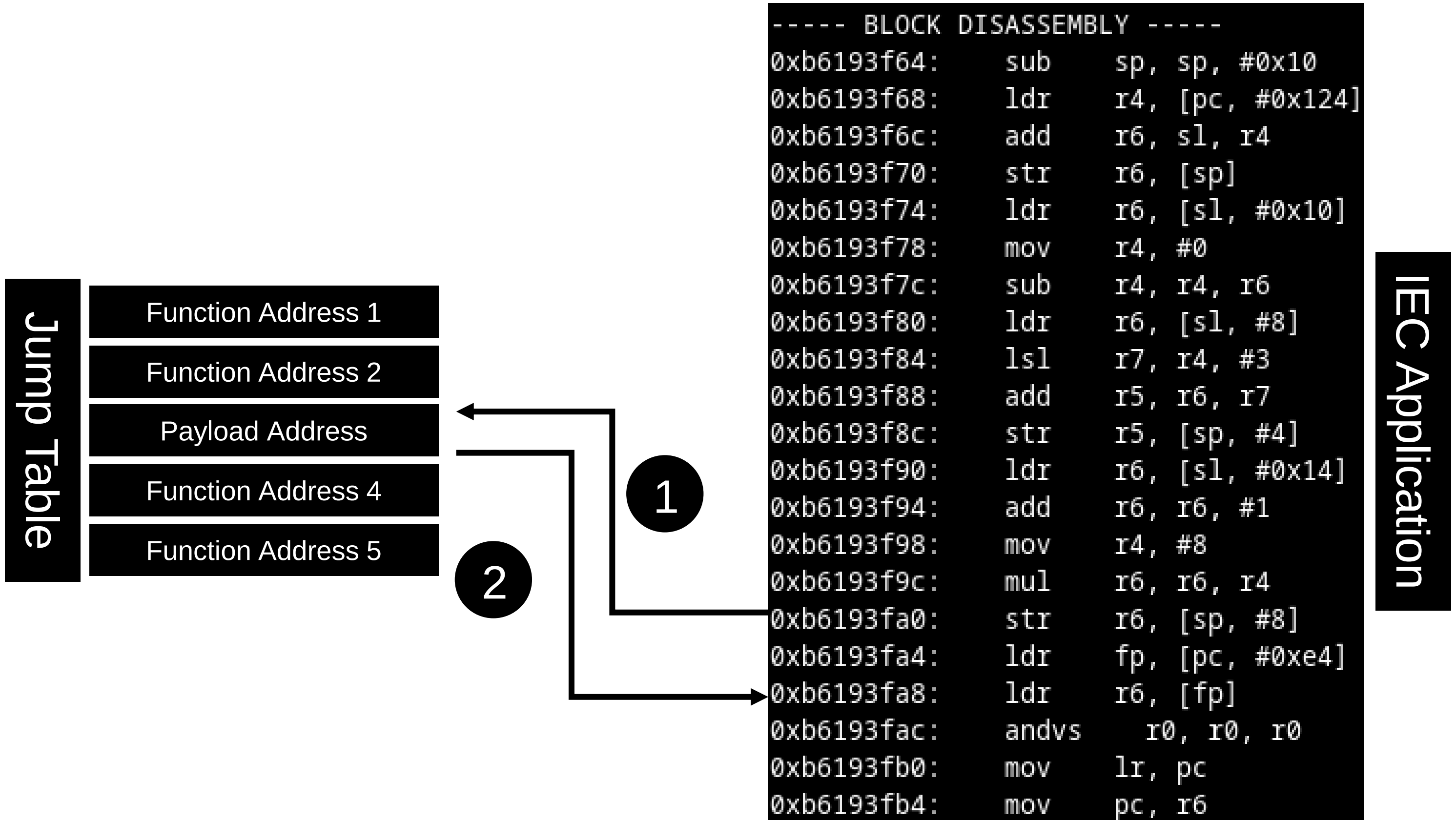
Demo – Patching OOB Read

Experiment Setup

MoMA LAB

OS Command Injection

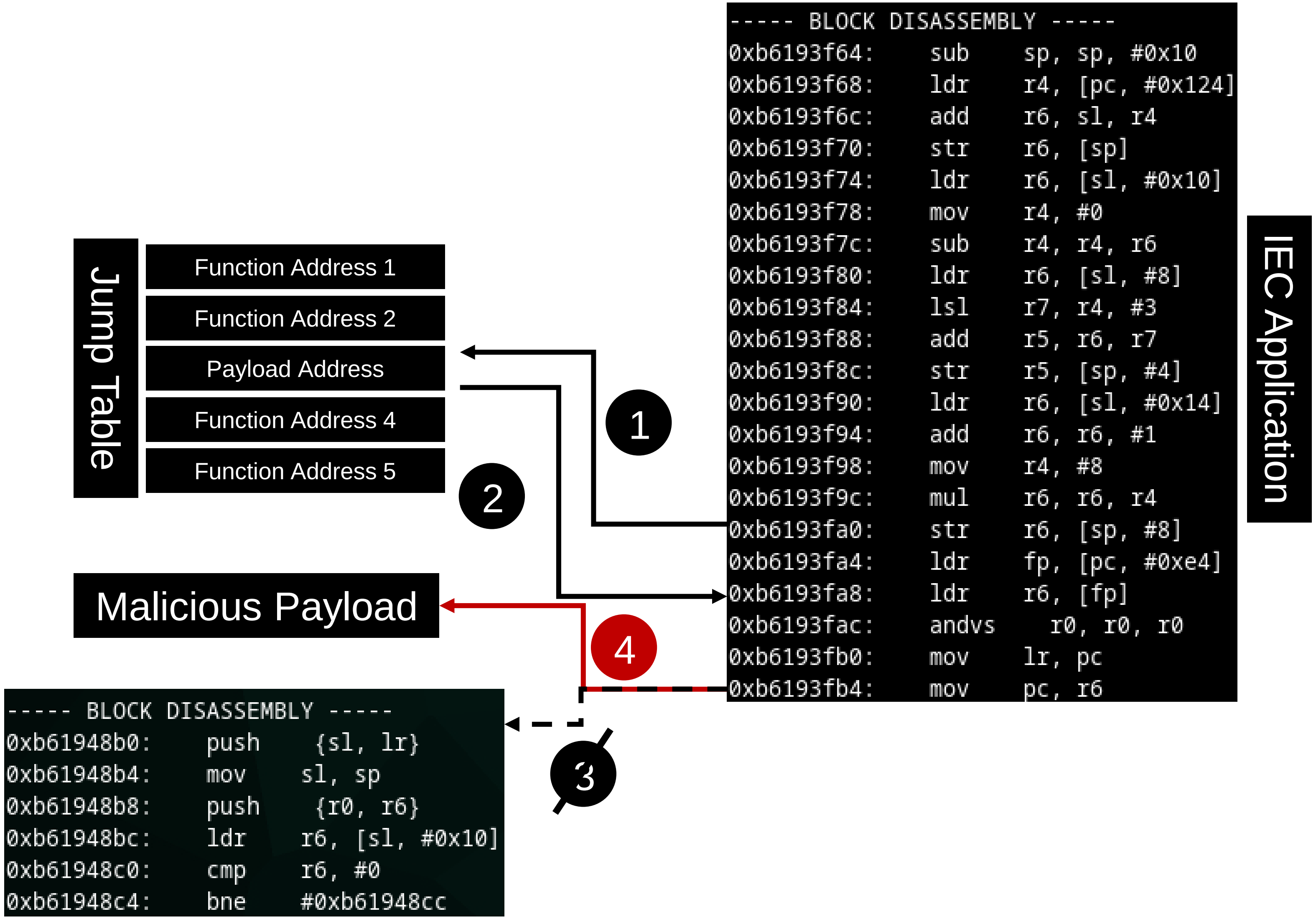
- How?
  - Overwrite Jump table address / return address on the stack



```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

OS Command Injection

- How?
  - Overwrite Jump table address / return address on the stack



```
----- BLOCK DISASSEMBLY -----
0xb6193f64:    sub    sp, sp, #0x10
0xb6193f68:    ldr    r4, [pc, #0x124]
0xb6193f6c:    add    r6, sl, r4
0xb6193f70:    str    r6, [sp]
0xb6193f74:    ldr    r6, [sl, #0x10]
0xb6193f78:    mov    r4, #0
0xb6193f7c:    sub    r4, r4, r6
0xb6193f80:    ldr    r6, [sl, #8]
0xb6193f84:    lsl    r7, r4, #3
0xb6193f88:    add    r5, r6, r7
0xb6193f8c:    str    r5, [sp, #4]
0xb6193f90:    ldr    r6, [sl, #0x14]
0xb6193f94:    add    r6, r6, #1
0xb6193f98:    mov    r4, #8
0xb6193f9c:    mul    r6, r6, r4
0xb6193fa0:    str    r6, [sp, #8]
0xb6193fa4:    ldr    fp, [pc, #0xe4]
0xb6193fa8:    ldr    r6, [fp]
0xb6193fac:    andvs    r0, r0, r0
0xb6193fb0:    mov    lr, pc
0xb6193fb4:    mov    pc, r6
```

```
----- BLOCK DISASSEMBLY -----
0xb61948b0:    push    {sl, lr}
0xb61948b4:    mov    sl, sp
0xb61948b8:    push    {r0, r6}
0xb61948bc:    ldr    r6, [sl, #0x10]
0xb61948c0:    cmp    r6, #0
0xb61948c4:    bne    #0xb61948cc
```

Demo – Patching OS Cmd Injection

Experiment Setup

Closing Remarks

- Limitations
  - Only tested on WAGO and BBB [4.0.0.0]
  - Requires modification to the DDG for supporting more devices
  - Slight change in patch based on the platform
  - Offsets change on different devices

Closing Remarks

- Limitations
  - Only tested on WAGO and BBB [4.0.0.0]
  - Requires modification to the DDG for supporting more devices
  - Slight change in patch based on the platform
  - Offsets change on different devices

- Shared stack allows control over the runtime from a vulnerable IEC application

- We successfully hotpatch IEC applications running in Codesys runtime using an LKM patcher.