

Non-Intrusive Vulnerability Localization and Hotpatching for Industrial Control Systems

Prashant Hari Narayan Rajput

Michail Maniatakos

#BHASIA @BlackHatEvents

blackhat 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.

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







- 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
 - 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











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









- 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









blackhat Industrial Control Systems

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 - Outdated OS/firmware + IT threats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





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 - Outdated OS/firmware + IT threats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



• Vulnerabilities in process logic can impact the runtime





- 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





blackhat Industrial Control Systems

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





- 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



- 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





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





Experiment Setup



HatEvents



- 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

blackhat Demo – The Codesys Stack

• Why?

	fcn.b611b010	(int32_t arg1, int	32_t arg_0h);	
	; arg int32_t	arg_0h @ tp+0x0		
	; var int32_t	. var_0n @ sp+0x8		
	; arg int32_t	argi @ r0		(1))
	0xb611b010	00442de9	push	{s1, 1r}
	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]
<u> </u>	0xb611b04c	0 b0000ea	b	0xb611b080
ц.	0xb611b050	0c609ae5	ldr	r6, [sl, 0xc]
	0xb611b054	284096e5	ldr	r4, [r6, <mark>0x28</mark>]
	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
L	; aav.0xb61	1b074:		
	0xb611b074	0060a0e3	mov	r6, 0
	0xb611b078	08409ae5	ldr	r4, [sl, 8]
	0xb611b07c	5360c4e5	strb	r6, [r4, 0x53]
	0xb611b080	5000bde8	DOD	{r4, r6}
	0xb611b084	0084bde8	DOD	{s1, pc}
	0xb611b088	0c4424b6	strtlt	r4, [r4], -ip, lsl 8
	0xb611b08c	00000060	andvs	r0, r0, r0



- Why?
 - Shared stack



blackhat Demo – The Codesys Stack

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





• Step 1: Extract execution state from Codesys runtime





• Step 2: Rehost in Angr and execute the IEC application





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





• Step 4: Get live base addresses and patch





- Where to patch?
 - Vulnerability Localization with DDG*

	BLOCK	DISASSEMB	LY
	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
OL	0xb6193f7c:	sub	r4, r4, r6
ati	0xb6193f80:	ldr	r6, [sl, #8]
^{co}	0xb6193f84:	lsl	r7, r4, #3
jli	0xb6193f88:	add	r5, r6, r7
bd	0xb6193f8c:	str	r5, [sp, #4]
A	0xb6193f90:	ldr	r6, [sl, #0x14]
\odot	0xb6193f94:	add	r6, r6, #1
Ш	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



- Where to patch?
 - Vulnerability Localization with DDG*

	BLOCK	DISASSEMB	LY
	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
oL	0xb6193f7c:	sub	r4, r4, r6
a ti	0xb6193f80:	ldr	r6, [sl, #8]
S	0xb6193f84:	lsl	r7, r4, #3
Jli	0xb6193f88:	add	r5, r6, r7
bd	0xb6193f8c:	str	r5, [sp, #4]
\forall	0xb6193f90:	ldr	r6, [sl, #0x14]
\bigcirc	0xb6193f94:	add	r6, r6, #1
Ш	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

M



- Where to patch?
 - Vulnerability Localization with DDG*

	BLOCK	DISASSEMB	LY	
	0xb6193f64:	sub	sp, sp, #0x10	
	0xb6193f68:	ldr	r4, [pc, #0x124]	
	0xb6193f6c:	add	r6, sl, r4	
	0xb6193f70:	str	rб, [sp]	
	0xb6193f74:	ldr	r6, [sl, #0x10]	
	0xb6193f78:	mov	r4, #0	
	0xb6193f7c:	sub	r4, r4, r6	
	0xb6193f80:	ldr	r6, [sl, #8]	
CV CV	0xb6193f84:	lsl	r7, r4, #3	
	0xb6193f88:	add	r5, r6, r7	
d	0xb6193f8c:	str	r5, [sp, #4]	
A	0xb6193f90:	ldr	r6, [sl, #0x14]	
\mathbf{C}	0xb6193f94:	add	r6, r6, #1	
	0xb6193f98:	mov	r4, #8	/
	0xb6193f9c:	mul	r6, r6, r4	
	0xb6193fa0:	str	rб, [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	DISASSEMB	LY
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



- Where to patch?
 - Vulnerability Localization with DDG*

	BLOCK	DISASSEME	3LY					BLOCK [ISASSEM	3LY	
	0xb6193f64:	sub	sp, sp, #0x10					0xb61948b0:	push	{sl, lr}	
	0xb6193f68:	ldr	r4, [pc, #0x124]					0xb61948b4:	mov	sl, sp	
	0xb6193f6c:	add	r6, sl, r4					0xb61948b8:	push	{r0. r6}	
	0xb6193f70:	str	r6, [sp]					0xb61948bc:	ldr	r6, [sl, #0x10]	
	0xb6193f74:	ldr	r6, [sl, #0x10]				V	0xb61948c0:	cmp	r6, #0	
	0xb6193f78:	mov	r4, #0				,	0xb61948c4:	bne	#0xb61948cc	
oL	0xb6193f7c:	sub	r4, r4, r6		M						Ru
àti	0xb6193f80:	ldr	r6, [sl, #8]								In.
S	0xb6193f84:	lsl	r7, r4, #3	1							
JI	0xb6193f88:	add	r5, r6, r7								ne
dd	0xb6193f8c:	str	r5, [sp, #4]								
\triangleleft	0xb6193f90:	ldr	r6, [sl, #0x14]					BLOCK	DISASSE	MBLY	
\bigcirc	0xb6193f94:	add	r6, r6, #1					0x80573c4:	sub	r2, r2, #4	SJO
	0xb6193f98:	mov	r4, #8					0x80573c8:	ldr	r5, [ip, #-4]	ary
	0xb6193f9c:	mul	r6, r6, r4					0x80573cc:	cmp	r2, #3	
	0xb6193fa0:	str	r6, [sp, #8]					0x80573d0:	mov	r1, ip	
	0xb6193fa4:	ldr	fp, [pc, #0xe4]					0x80573d4:	mov	r3, r4	
	0xb6193fa8:	ldr	r6, [fp]					0x80573d8:	str	r5, [r4, #-4]	
	0xb6193fac:	andvs	r0, r0, r0					0x80573dc :	bls	#0x8057400	
	0xb6193fb0:	mov	lr, pc		20	h					
	0xb6193fb4:	mov	pc, r6		25	7				#BHASI	



- Where to patch?
 - Vulnerability Localization with DDG*

	BLOCK	DISASSEMB	SLY				BLOCK D	ISASSEME	3LY		
	0xb6193f64:	sub	sp, sp, #0x10			Øxb	61948b0:	push	{sl, lr}		
	0xb6193f68:	ldr	r4, [pc, #0x124]			Øxb	61948b4:	mov	sl, sp		
	0xb6193f6c:	add	r6, sl, r4			Øxb	61948b8:	push	{r0. r6}		
	0xb6193f70:	str	r6, [sp]			Øxb	61948bc:	ldr	r6, [sl,	#0x10]	
	0xb6193f74:	ldr	r6, [sl, #0x10]			Øxb	61948c0:	cmp	r6, #0	2	
	0xb6193f78:	mov	r4, #0			Øxb	61948c4:	bne	#0xb61948	cc	
JO	0xb6193f7c:	sub	r4, r4, r6	Л							25
Iti	0xb6193f80:	ldr	r6, [sl, #8]								UI
C C C	0xb6193f84:	lsl	r7, r4, #3								tir
	0xb6193f88:	add	r5, r6, r7		A						n€
bd	0xb6193f8c:	str	r5, [sp, #4]			_					
\forall	0xb6193f90:	ldr	r6, [sl, #0x14]			-	BLOCK	DISASSE	MBLY		lic
\odot	0xb6193f94:	add	r6, r6, #1			0:	x80573c4:	sub	r2, r2, #	‡4	ora
Ш	0xb6193f98:	mov	r4, #8		X	0	x80573c8:	ldr	r5, [ip,	#-4]	ne
	0xb6193f9c:	mul	r6, r6, r4			0:	k80573cc:	cmp	r2, #3		\leq
	0xb6193fa0:	str	r6, [sp, #8]			0	k80573d0:	mov	r1, ip		
	0xb6193fa4:	ldr	fp, [pc, #0xe4]			0:	x80573d4:	mov	r3, r4		
	0xb6193fa8:	ldr	r6, [fp]			0:	x80573d8:	str	r5, [r4,	#-4]	
	0xb6193fac:	andvs	r0, r0, r0			0:	k80573dc :	bls	#0x805740	00	
	0xb6193fb0:	mov	lr, pc	20							
	0xb6193fb4:	mov	pc, r6	30					**		



- How to patch?
 - First, look at branching

BLOCK	DISASSEMB	LY	
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]	\square
0xb6193f84:	lsl	r7, r4, #3	1.5
0xb6193f88:	add	r5, r6, r7	p
0xb6193f8c:	str	r5, [sp, #4]	q
0xb6193f90:	ldr	r6, [sl, #0x14]	lic
0xb6193f94:	add	r6, r6, #1	at
0xb6193f98:	mov	r4, #8	io
0xb6193f9c:	mul	r6, r6, r4	n
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	
Addr	ess into the	Jump Table	



- How to patch?
 - First, look at branching

٢	Function Address 1	
um	Function Address 2	2
рТ	Function Address 3	
ab	Function Address 4	
e	00000000 [Empty]	

BLOCK	DISASSEMBL	_Y	
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]	\mathbf{O}
0xb6193f84:	ls1	r7, r4, #3	1
0xb6193f88:	add	r5, r6, r7	q
0xb6193f8c:	str	r5, [sp, #4]	đ
0xb6193f90:	ldr	r6, [sl, #0x14]	ic
0xb6193f94:	add	r6, r6, #1	at
0xb6193f98:	mov	r4, #8	io
0xb6193f9c:	mul	r6, r6, r4	n
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	
Addr	ess into the .	Jump Table	



- How to patch?
 - First, look at branching



BLOCK DISASSEMBLY -----

sp, sp, #0x10

r6, sl, r4

r6, [sp]

r4, [pc, #0x124]

r6, [sl, #0x10]

sub

ldr

add

str

ldr

0xb6193f64:

0xb6193f68:

0xb6193f6c:

0xb6193f70:

0xb6193f74:



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

BLOCK	DISASSEMB	BLY
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] O
0xb6193f90:	ldr	r6, [sl, #0x14] 🖸
0xb6193f94:	add	r6, r6, #1 🏻 🏨
0xb6193f98:	mov	r4, #8 Ō
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
Addr	ess into the	Jump Table



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

Jump Table

BLOCK 0xb6193f64: 0xb6193f68: 0xb6193f6c: 0xb6193f70: 0xb6193f74: 0xb6193f78: 0xb6193f78: 0xb6193f80: 0xb6193f88: 0xb6193f88: 0xb6193f88: 0xb6193f90: 0xb6193f94: 0xb6193f98: 0xb6193f98: 0xb6193f98: 0xb6193f98:	DISASSEMB sub ldr add str ldr mov sub ldr lsl add str ldr add mov mul str ldr ldr jdr	LY sp, sp, #0x10 r4, [pc, #0x124] r6, sl, r4 r6, [sp] r6, [sl, #0x10] r4, #0 r4, r4, r6 r6, [sl, #8] r7, r4, #3 r5, r6, r7 r5, [sp, #4] r6, [sl, #0x14] r6, r6, #1 r4, #8 r6, r6, r4 r6, [sp, #8] fp, [pc, #0xe4] r6, [fp_ OFESET]	IEC Application
0xb6193f9c: 0xb6193fa0: 0xb6193fa4: 0xb6193fa8: 0xb6193fac: 0xb6193fb0: 0xb6193fb0:	mul str ldr ldr andvs mov mov	r6, r6, r4 r6, [sp, #8] fp, [pc, #0xe4] r6, [fp ,OFFSET] r0, r0, r0 lr, pc pc, r6	5
Addr	ess into the	Jump Table	

2

Function Address 1

Function Address 2

Function Address 3

Function Address 4

Patch Address



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

Jump Table

		0xb6193f64:	sub	sp. sp. #0x10
		0xb6193f68:	ldr	r4. [pc. #0x124]
id fix		0xb6193f6c:	add	r6, s1, r4
		0xb6193f70:	str	r6, [sp]
branch to the	9	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
Patch	←	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
Function Address 1		0xb6193fb0:	mov	lr, pc
Function Address T		0xb6193fb4:	mov	рс, тб
Function Address 2		Addre	ss into the	Jump Table
Function Address 3				
Function Address 4	2			
Patch Address				

BLOCK DISASSEMBLY -----



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

a			0xb6193f6c:	add	r6, sl, r4	
		-	0xb6193f70:	str	r6, [sp]	
a	branch to the	e	0xb6193f74:	ldr	r6, [sl, #0x10]	
			0xb6193f78:	mov	r4, #0	
			0xb6193f7c:	sub	r4, r4, r6	
			0xb6193f80:	ldr	r6, [sl, #8]	
			0xb6193f84:	lsl	r7, r4, #3	\mathbf{b}
		-	0xb6193f88:	add	r5, r6, r7	p
	Patch	← −−−	0xb6193f8c:	str	r5, [sp, #4]	p
			0xb6193f90:	ldr	r6, [sl, #0x14]	ic
		3	0xb6193f94:	add	r6, r6, #1	at
			0xb6193f98:	mov	r4, #8	0
			0xb6193f9c:	mul	r6, r6, r4	\supset
			0xb6193fa0:	str	r6, [sp, #8]	
		_	0xb6193fa4:	ldr	fp, [pc, #0xe4]	▲]
			0xb6193fa8:	ldr	r6, [fp ,OFFSET]	
			0xb6193fac:	andvs	r0, r0, r0	
~	Function Address 1		0xb6193fb0:	mov	lr, pc	
μ			0xb6193†b4:	mov	pc, r6	
m	Function Address 2		Address	s into the	Jump Table	
o T	Function Address 3					
ab	Function Address 4	2				
Ð	Patch Address					

BLOCK DISASSEMBLY -----

sp, sp, #0x10

r4, [pc, #0x124]

sub

ldr

0xb6193f64:

0xb6193f68:



Patch

Function Address 1

Function Address 2

Function Address 3

Function Address 4

Patch Address

• How to patch?

{sl, lr}

{r0, r6}

#0xb61948cc

r6, [sl, #0x10]

sl, sp

r6, #0

BLOCK DISASSEMBLY -----

push

mov

push

ldr

cmp

bne

0xb61948b0:

0xb61948b4:

0xb61948b8:

0xb61948bc:

0xb61948c0:

0xb61948c4:

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

6

Jump Table

BLOCK	DISASSEMB	LY				
0xb6193f64:	sub	sp, sp, #0x10				
0xb6193f68:	ldr	r4, [pc, #0x124]				
0xb6193f6c:	add	r6, s1, 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] O				
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	рс, тб				
Address into the Jump Table						
, (aa		o anno - aoro				

2

5

3



- How?
 - Missing bound check
 - Overwrite small buffer
 - Overwrites critical runtime return addresses
- Impact
 - Messes with the control flow
 - DoS
 - Requires runtime reboot





- How?
 - Missing bound check
 - Reads from the shared stack
 and code
- Impact
 - Can leak secrets
 - Does not crash the runtime





Experiment Setup





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

				0xb6193f6c:	add	r6, sl, r4	
				0xb6193f70:	str	r6, [sp]	
				0xb6193f74:	ldr	r6, [sl, #0x10]	
				0xb6193f78:	mov	r4, #0	
				0xb6193f7c:	sub	r4, r4, r6	
٦	Function Address 1			0xb6193f80:	ldr	r6, [sl, #8]	\square
ur	Function Address 2			0xb6193f84:	lsl	r7, r4, #3	1
_ du				0xb6193f88:	add	r5, r6, r7	q
	Payload Address			0xb6193f8c:	str	r5, [sp, #4]	q
a'	Eurotion Addross 4			0xb6193f90:	ldr	r6, [sl, #0x14]	ic
Ы	Function Address 4			0xb6193f94:	add	r6, r6, #1	at
Ð	Function Address 5			0xb6193f98:	mov	r4, #8	io
		2		0xb6193f9c:	mul	r6, r6, r4	n
			<u>.</u>	0xb6193fa0:	str	r6, [sp, #8]	
				0xb6193fa4:	ldr	fp, [pc, #0xe4]	
		L		0xb6193fa8:	ldr	r6, [fp]	
				0xb6193fac:	andvs	r0, r0, r0	
				0xb6193fb0:	mov	lr, pc	
				0xb6193fb4:	mov	pc, r6	

BLOCK DISASSEMBLY -----

sp, sp, #0x10

рс,

sub

ldr

0xb6193f64:

0xh6193f68



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

					Øxb	6193f68:	ldr	r4,	[pc, #0x124]	
					Øxb	6193f6c:	add	rб,	sl, r4	
					Øxb	6193f70:	str	rб,	[sp]	
					Øxb	6193f74:	ldr	rб,	[sl, #0x10]	
					Øxb	6193f78:	mov	r4,	#0	
					Øxb	6193f7c:	sub	r4,	r4, r6	
	Functior	n Address 1			Øxb	6193f80:	ldr	rб,	[sl, #8]	\mathbf{O}
UN	Function	n Address 2			Øxb	6193f84:	lsl	r7,	r4, #3	Ň
qu			←		Øxb	6193f88:	add	r5,	r6, r7	q
	Payloa	d Address			Øxb	6193f8c:	str	r5,	[sp, #4]	р
a	Function	n Address <i>4</i>			Øxb	6193f90:	ldr	rб,	[sl, #0x14]	ic
	T drietion				Øxb	6193f94:	add	rб,	r6, #1	ati
	Function	n Address 5			Øxb	6193f98:	mov	r4,	#8	0
			(2)		Øxb	6193f9c:	mul	rб,	r6, r4	
					Øxb	6193fa0:	str	rб,	[sp, #8]	
					Øxb)6193fa4:	ldr	fp,	[pc, #0xe4]	
Ma	alicious	Pavload ┥		<u> </u>	0xb	06193fa8:	ldr	r6,	[fp]	
					Øxb	6193fac:	andvs	_ r@	, r0, r0	
				4	Øxb	6193fb0:	mov	lr,	рс	
BLOC	K DISASSEM	BI Y			Øxb	6193†b4:	mov	pc,	r6	
xb61948b0	: push	{s],]r}	< -							
xb61948b4	: mov	sl. sp		2						
xb61948b8	: push	{r0, r6}								
xb61948bc	: ldr	r6, [sl, #0>	10]							
xb61948c0	: cmp	r6, #0								
xb61948c4	: bne	#0xb61948cc								

BLOCK DISASSEMBLY -----

sp, sp, #0x10

sub

0xb6193f64:



Experiment Setup





- 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



- 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.