NYU TANDON SCHOOL OF ENGINEERING

MoMA LAB

جامعة نيويورك أبوظبي
NYU ABU DHABI

ICSFuzz: Manipulating I/Os and Repurposing Binary Code to Enable Instrumented Fuzzing in ICS Control Applications

Dimitrios Tychalas
NYU Tandon School of Engineering

Hadjer Benkraouda
New York University Abu Dhabi

Michail Maniatakos
New York University Abu Dhabi

USENIX Security 2021

8/12/21

Introduction

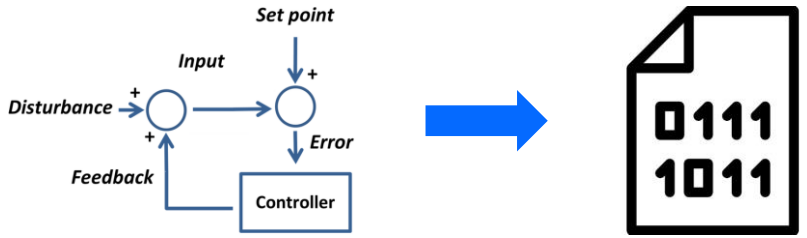
ICS Landscape

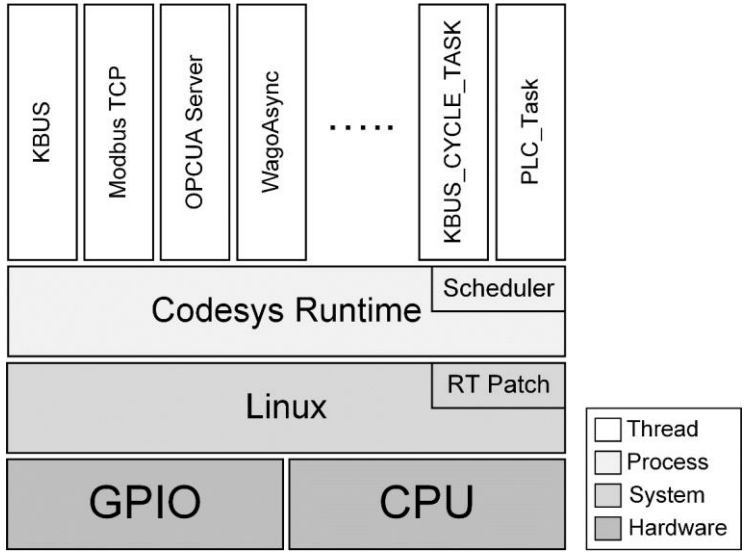
- Industrial Control Systems (ICS) evolving
  - Industry 4.0, IIoT
- IT/OT convergence
- Most research focused on network/system/operational security
  - What about have software bugs?

Introduction

Control Application Intro

- Control logic in PLCs compiled into a software application
  - Assembly instructions
  - Unique format
- Application written in dedicated languages
  - Third-party libraries
- Application hosted in a runtime process
  - Codesys platform a.k.a. a soft PLC
- Runtime is hosted in a Linux-based OS
  - Single binary, multi-threaded





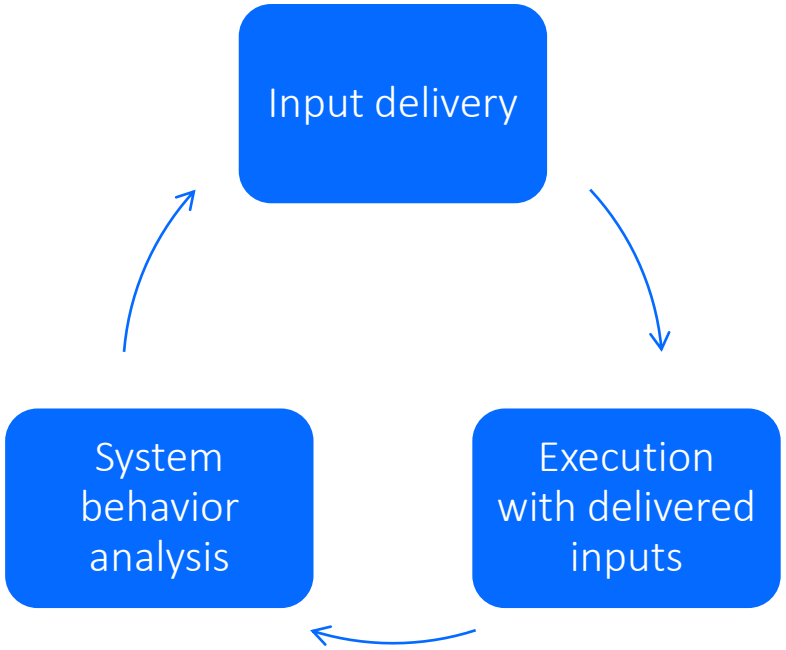
Research Questions/Threat Model

- Can control logic applications have software bugs?
  - Fuzzing!
- Can we fuzz a control applications?
  - Format is not readily accepted by typical fuzzers
- Can fuzzing uncover exploitable bugs?
  - Reverse shell on a PLC sounds exciting!

Fuzzing control applications

Fuzzing basics

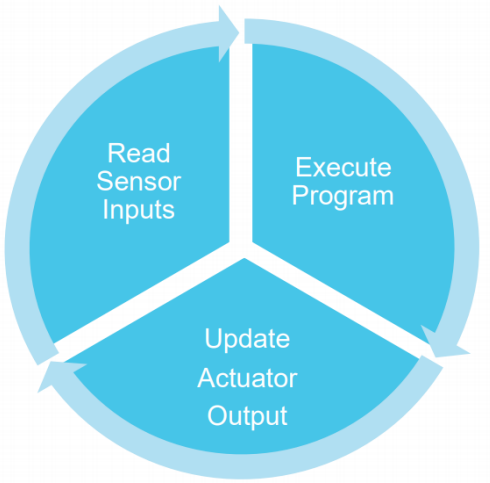
- What is fuzzing?
  - Input control
  - Execution control
  - Execution feedback

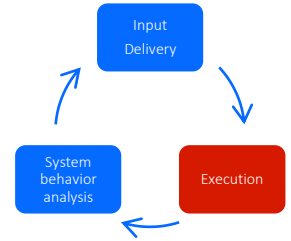


Fuzzing control applications

Execution control

- Execution control
  - Scan cycle task
  - Automated cyclic execution!

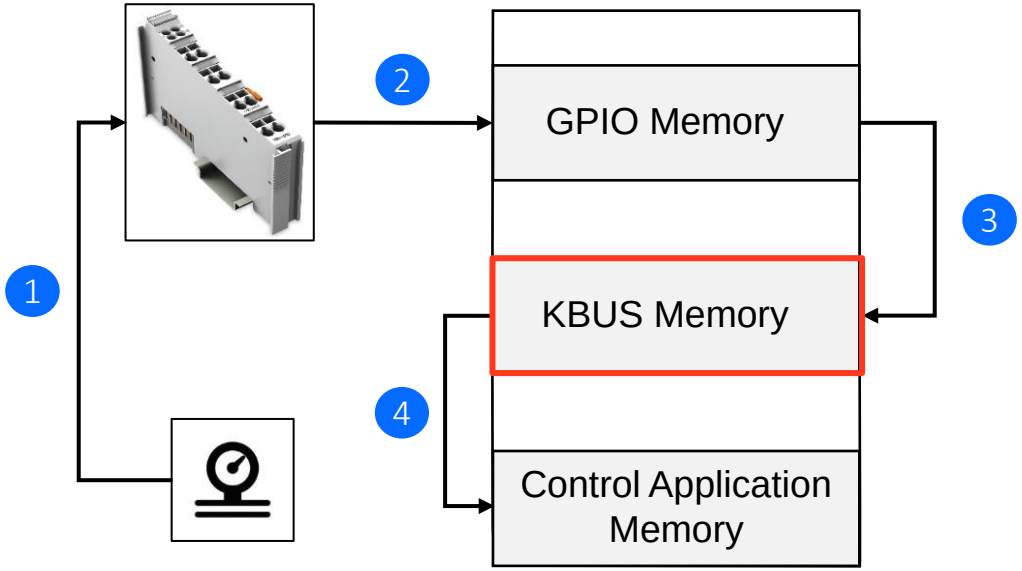


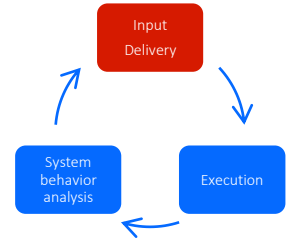


Fuzzing control applications

Input Control

- How to control input?
  - Trace input delivery throughout the stack
  - Sensor -> PLC I/O -> GPIO -> KBUS -> PLC binary
  - Isolate most controllable input delivery stage
  - Force new values through system structures
- How to synchronize input delivery?
  - KBUS_CYCLE_TASK
  - Custom system calls (ioctl's)

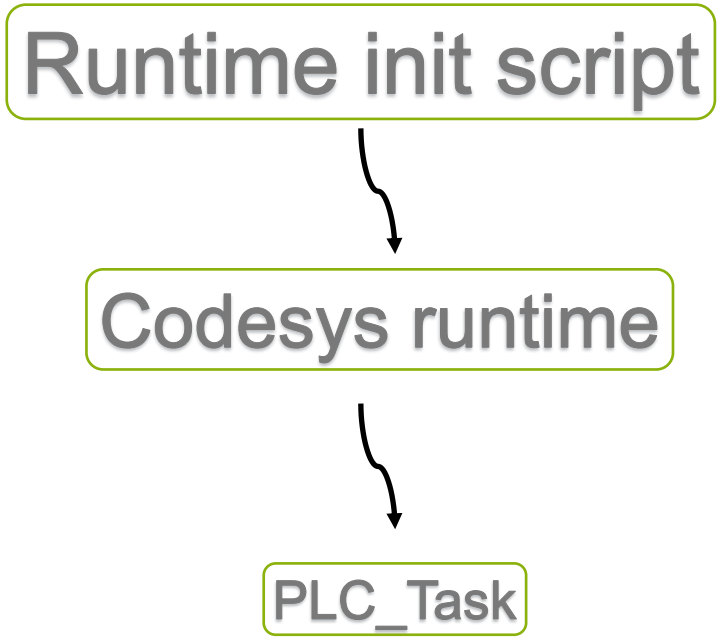




Fuzzing control applications

Execution feedback

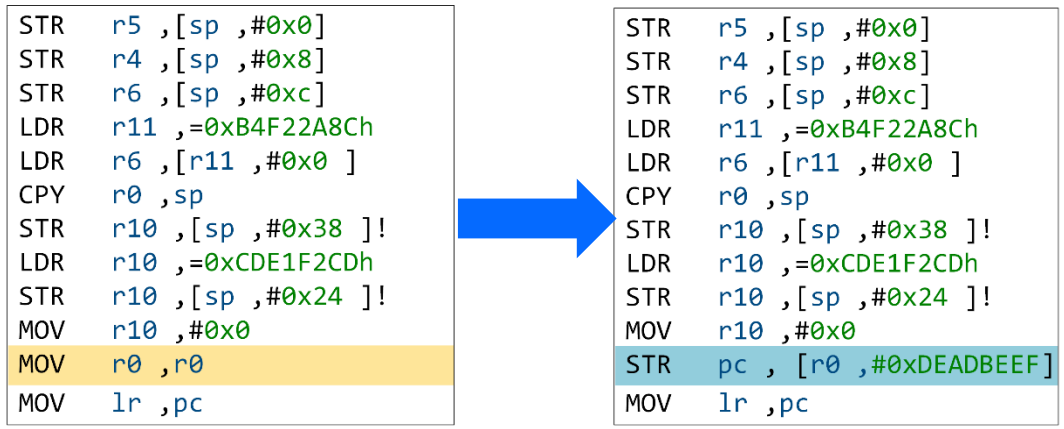
- Execution feedback
  - Trace the PID family
  - Find a malleable predecessor
  - Trace control application exit through predecessor



Fuzzing control applications

Instrumentation

- Can we get feedback on the execution of the control application?
  - Instrumentation!
- What to do?
  - No source code
  - Proprietary compiler
- Look for opportunities!
  - NOPs
  - Replace NOPs with controllable code
    - i.e. store the current program counter (PC)
  - Approximate coverage with the PC information

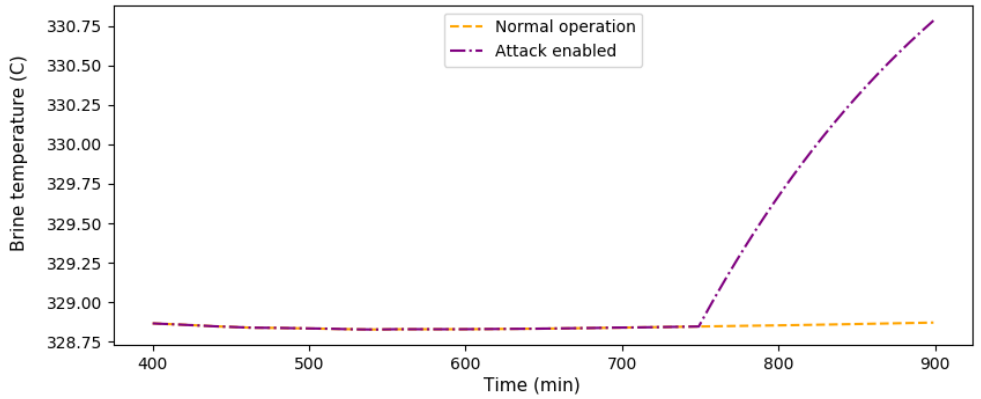


Fuzzing control applications

Experiments

- Fuzz binaries found in the wild
  - Problem: PLC binaries available online are too simple
  - Try something beefier

- Fuzzing an open-source desalination process
  - Bingo!
  - Binary crashed
  - DoS possible



Fuzzing control applications

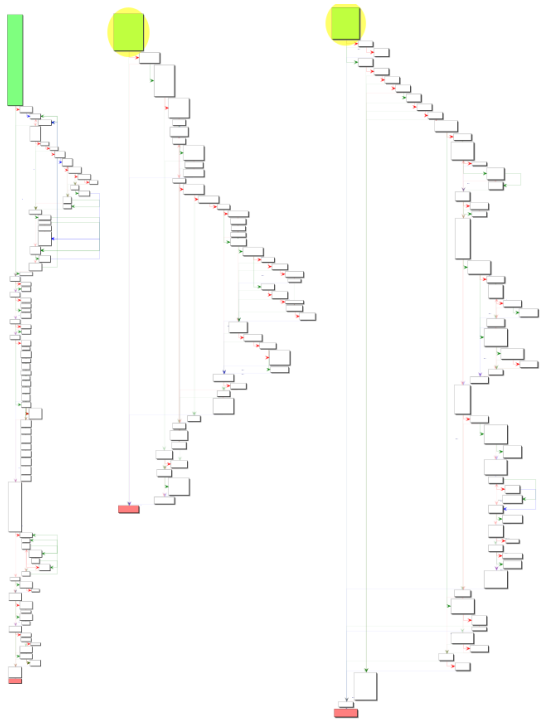
Experiments (contd.)

- Dig a little bit deeper
  - Can we make more complex PLC binaries
    - (yes we can!)
- Introduce interesting stuff
  - Complex structures (arrays, strings, etc.)
  - Notorious functions (`memcpy`, `memmove` etc.)
  - Fuzz away
- Crashes, lots of crashes
  - Buffer overflows
  - (Way) Out of bounds read/writes
- Possibility for complex exploits
  - Reverse shell through TCP
  - Rootkit insertion

Control Application	Execution Speed (inputs/sec)	First Crash (time mm:ss)	First crash (inputs)	Crashes (1hr)
bf_mcpy_1	70.88	3:54	15270	32
bf_mcpy_6	64.2	3:08	12172	21
bf_mcpy_8	66.06	4:39	18216	17
bf_mcpy_12	62.11	7:06	26645	9
bf_mset_1	64.56	3:28	13441	21
bf_mset_3	62.68	2:54	10906	24
bf_mset_5	68.8	4:14	17554	16
bf_mset_9	69.76	10:23	43530	7
bf_mmove_1	64.63	2:56	11245	28
bf_mmove_4	63.1	2:39	10070	24
bf_mmove_7	66.31	3:49	15317	15
bf_mmove_12	64.53	13:03	50643	6
oob_1_arr_1	71.86	0:55	3880	39
oob_1_arr_6	77.03	1:43	8085	28
oob_1_arr_9	69.78	1:45	7326	27
oob_1_arr_13	75.2	3:27	27241	19
oob_2_arr_1	73.53	1:57	8558	35
oob_2_arr_5	71.1	2:45	22759	27
oob_2_arr_8	69.8	3:08	13366	22
oob_2_arr_13	70.95	3:12	13401	19
divby0_1	73.68	N/A	N/A	0

Fuzzing the runtime

- What about the runtime itself?
  - Control applications can write way outside its memory space
    - Fuzzing crashed the whole runtime
- Must assess the runtime itself
  - Complex
  - Super large
  - Built-in anti-debugging

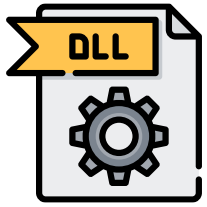


Functions - 4992 items

Label	Loca...	Function Sig...	Function Size
_init	000337ec	int _in...	12
SysFileCopy_	0003380c	thunk u...	28
SysSemProcess...	000338...	thunk u...	12
SysSemEnter	000338...	thunk u...	12
SysSemProcess...	000338...	thunk u...	12
SysEventOSHo...	0003383c	thunk u...	12
sync	000338...	thunk v...	12
SysMemOSHoo...	000338...	thunk u...	12
cgroup_attach...	000338...	thunk u...	12
SysSockGetHos...	0003386c	thunk u...	12
SysExceptMapE...	000338...	thunk u...	12
SysTimeRtcCon...	000338...	thunk u...	12
expf	000338...	thunk f...	12
qsort	0003389c	thunk v...	12
sigemptyset	000338...	thunk i...	12
daemon	000338...	thunk i...	12
strerror	000338c0	thunk c...	12
SysTimeRtcCon...	000338cc	thunk u...	12
tanf	000338...	thunk f...	12
__ctype_toupp...	000338...	thunk _...	12
SysComWrite	000338f0	thunk u...	12
SysSemCreate	000338fc	thunk u...	12
__sigsetjmp	000339...	thunk u...	12
SysSemTry	000339...	thunk u	12

Fuzzing the runtime

- Divide and conquer
  - Target dynamically loaded libraries
  - Reverse-engineer context
  - Instantiate in C harness
  - Fuzz the produced ELF!
    - Fuzzers galore




```
kbus_ksock_write_data(socket, data, length);
```

```
kbus_ksock_t ksock = fopen("/dev/kbus0", "r");
```

```
#include LIBDBUSCOMMON_H
int main(int argc, char **argv){
    kbus_ksock_t ksock = fopen("/dev/kbus0", "r");
    kbus_ksock_write_data(ksock, &argv, 32);
}
```

Fuzzing the runtime

Experiments

- Limited function eligibility
  - ~250 out of more than 5000
- Focus on "external facing" functions
  - Network protocols
  - Parsers
- Found crashes
  - Recognized a couple of known CVEs

Function	Description	Crashes (1hr)
KbusRegisterRequestWrite	Kbus write function	2
KbusRegisterRequestRead	Kbus read function	1
kbus_ksock_write_data	Kbus write function	4
kbus_ksock_read_data	read function	7
XMLParse	XML Parsing Function	8
SysSockRecv[5]	TCP receive data	6
CMAddCoomponentKbus	Kbus instantiation	4
pthread_create	Creates runtime thread	8
pthread_rwlock_unlock	Updates thread privileges	2
pthread_join	Joins PLC task threads	1
pthread_setschedparam	Sets scheduler thread policy	1
GetLoginName	Receives input login name	7
SysLibStrcpy	String copy custom function	2
SysLibStrcmp	String compare custom function	5
SysComWrite	System communication output	7
SysComRead	System communication input	2
GetHookName	Get name of hooked function	6
CopyRtsMetrics	Copies PLC data	8
getspnam	Returns info from shadow file	5

Conclusion

- PLC binaries are inherently robust
  - High-level nature of the PLC programming languages
  - Very well-defined problems they are addressing
- However, the control binaries get more complex
  - Mimicking typical software evolution
  - Exploitable vulnerabilities can be introduced
- PLC runtimes suffer like high-level software
  - E.g. typical C/C++ developed software
  - This can compromise the whole industrial control system computation stack
- Fuzzing is great!
  - Even for Industrial Control Systems
  - Despite the presence of heavy I/O and scan cycles

MoMA LAB | nyuad.nyu.edu/momalab

Thank you

Dimitris Tychalas
dimitris.tychalas@nyu.edu

Hadjer Benkraouda
hb992@nyu.edu

Mihalis Maniatakos
mihalis.maniatakos@nyu.edu

NYU TANDON SCHOOL OF ENGINEERING

جامعة نيويورك أبوظبي
NYU ABU DHABI