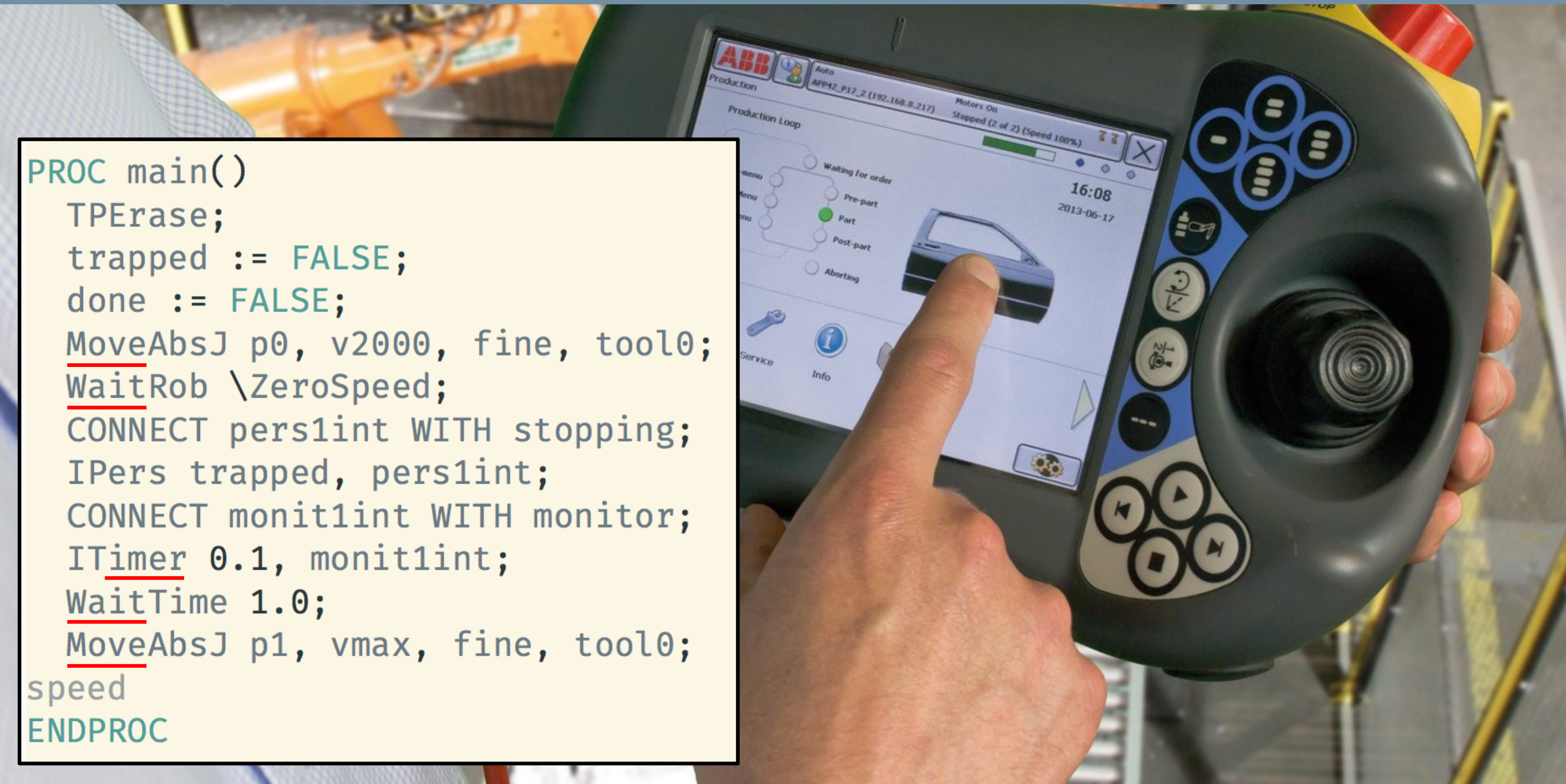
Detecting Insecure Code Patterns in Industrial Robotics Programs

Marcello Pogliani[1], Federico Maggi[2], Marco Balduzzi[2], Davide Quarta[3], Stefano Zanero[1]

[1]Politecnico di Milano – [2]Trend Micro Research – [3]EURECOM

ACM ASIA CCS, 5-9 October 2020

Programming Industrial Robots




```
PROC main()
  TPErase;
  trapped := FALSE;
  done := FALSE;
  MoveAbsJ p0, v2000, fine, tool0;
  WaitRob \ZeroSpeed;
  CONNECT pers1int WITH stopping;
  IPers trapped, pers1int;
  CONNECT monit1int WITH monitor;
  ITimer 0.1, monit1int;
  WaitTime 1.0;
  MoveAbsJ p1, vmax, fine, tool0;
speed
ENDPROC
```

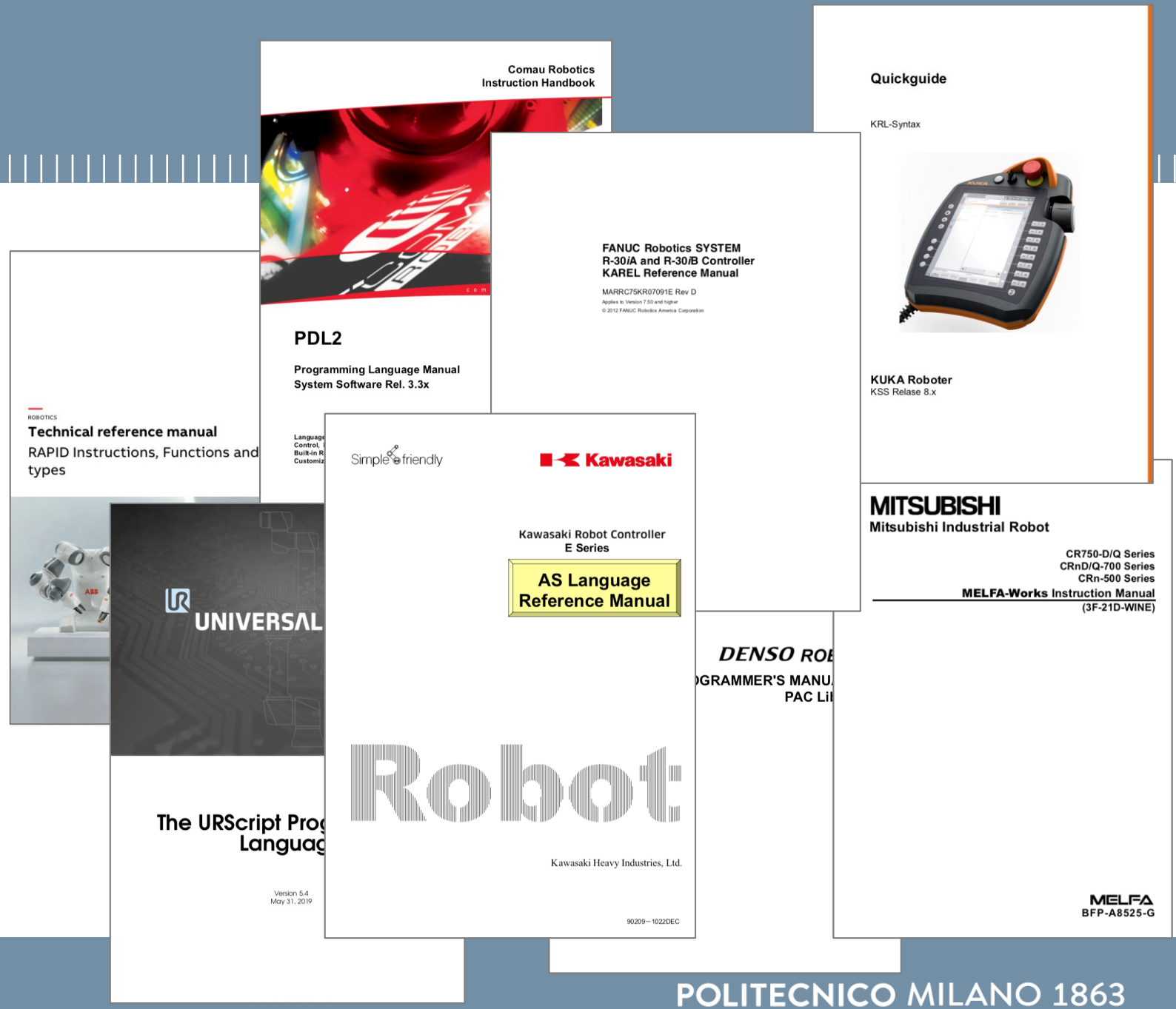
```
MODULE VulnWebServer
    PROC main()
        SocketCreate server;
        SocketBind server, '0.0.0.0', 1234;
        SocketListen server;
        SocketAccept server, sock;
        WHILE true DO
            SocketReceive sock, \RawData:=data;
            fileName := ParseCommand(data);
            Open fileName, res;
            ReadAndSendFile(\file:=res, \socket:=sock);
        ENDWHILE
    ENDPROC
ENDMODULE
```

Network (TCP Sockets)

File System

Programming Industrial Robots
Proprietary Languages

Language	Vendor
RAPID	ABB
KRL	KUKA
MELFA BASIC	Mitsubishi
AS	Kawasaki
PDL2	COMAU
PacScript	DENSO
URScript	Universal-Robot
KAREL	FANUC



Programming Industrial Robots: Features
Sensitive Features

Vendor	File System	Directory Listing
ABB	✓	✓
KUKA	✓	
Mitsubishi	✓	
Kawasaki		
COMAU	✓	Indirect
DENSO		
Universal-Robot		
FANUC	✓	✓

Programming Industrial Robots: Features
Sensitive Features

Vendor	File System	Directory Listing	Load Module From File	Call By Name
ABB	✓	✓	✓	✓
KUKA	✓			
Mitsubishi	✓			
Kawasaki				
COMAU	✓	Indirect	✓	✓
DENSO			✓	✓
Universal-Robot				
FANUC	✓	✓	✓	✓

Programming Industrial Robots: Features
Sensitive Features

Vendor	File System	Directory Listing	Load Module From File	Call By Name	Communication
ABB	✓	✓	✓	✓	✓
KUKA	✓				✓
Mitsubishi	✓				✓
Kawasaki					✓
COMAU	✓	Indirect	✓	✓	✓
DENSO			✓	✓	✓
Universal-Robot					✓
FANUC	✓	✓	✓	✓	✓

- The existence of such primitives is not (by itself) a security issue
- However…
  - External I/O → untrusted input
  - Sensitive resources (e.g., movement, IP)

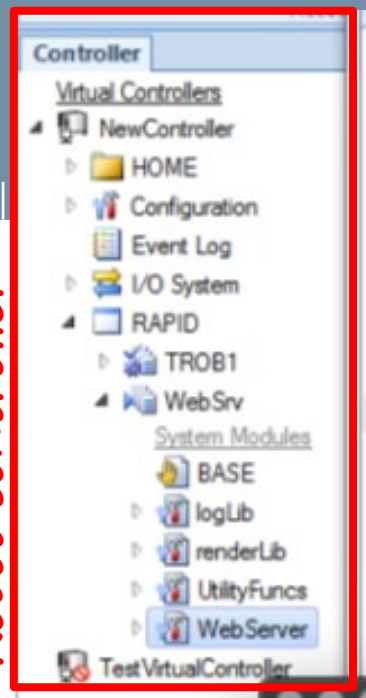
Can vulnerabilities exist in IRPL programs?

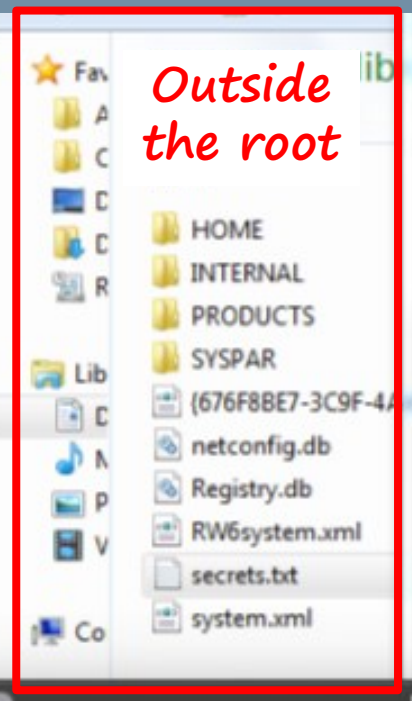
Industrial Robot Programming Languages
Taint-style Vulnerabilities

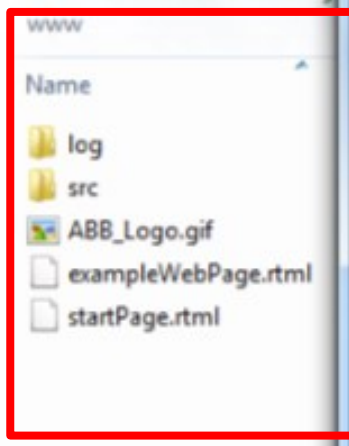
```
MODULE VulnWebServer
    PROC main()
        SocketCreate server;
        SocketBind server, '0.0.0.0', 1234;
        SocketListen server;
        SocketAccept server, sock;
        WHILE true DO
source      SocketReceive sock, \RawData:=data;
            fileName := ParseCommand(data);
sink        Open fileName, res;
            ReadAndSendFile(\file:=res, \socket:=sock);
        ENDWHILE
    ENDPROC
ENDMODULE
```

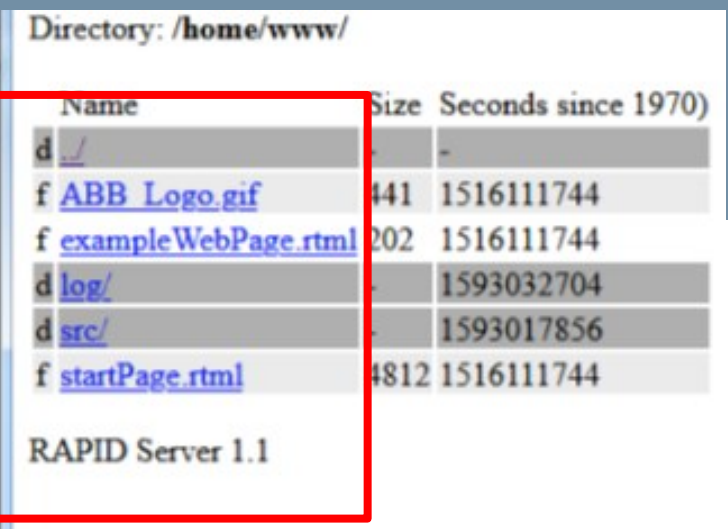
Directory Traversal
Arbitrary File Access

Robot controller



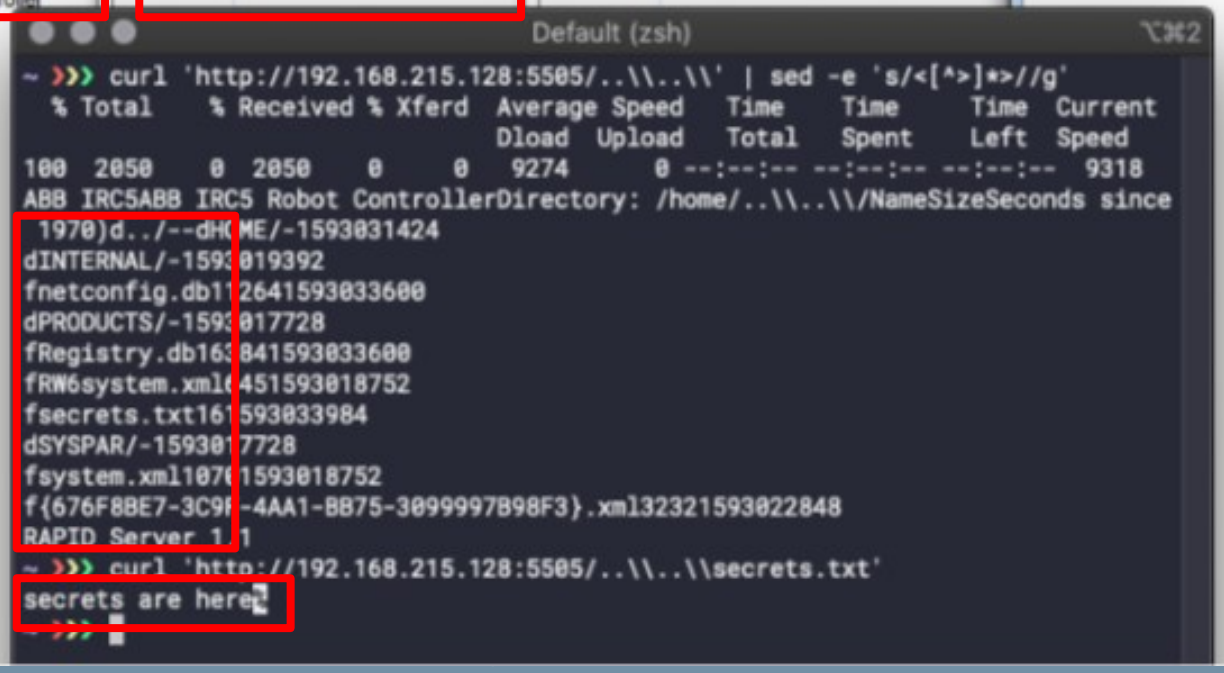




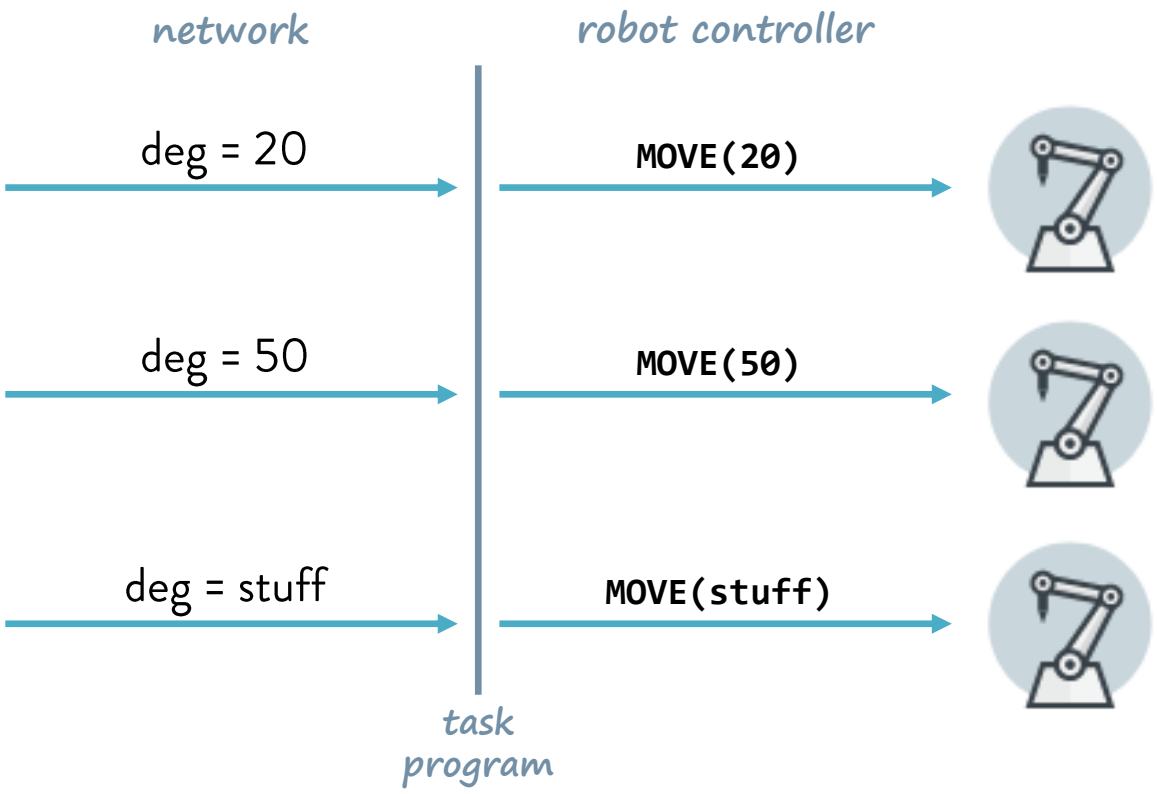


Web server root

Secrets stolen



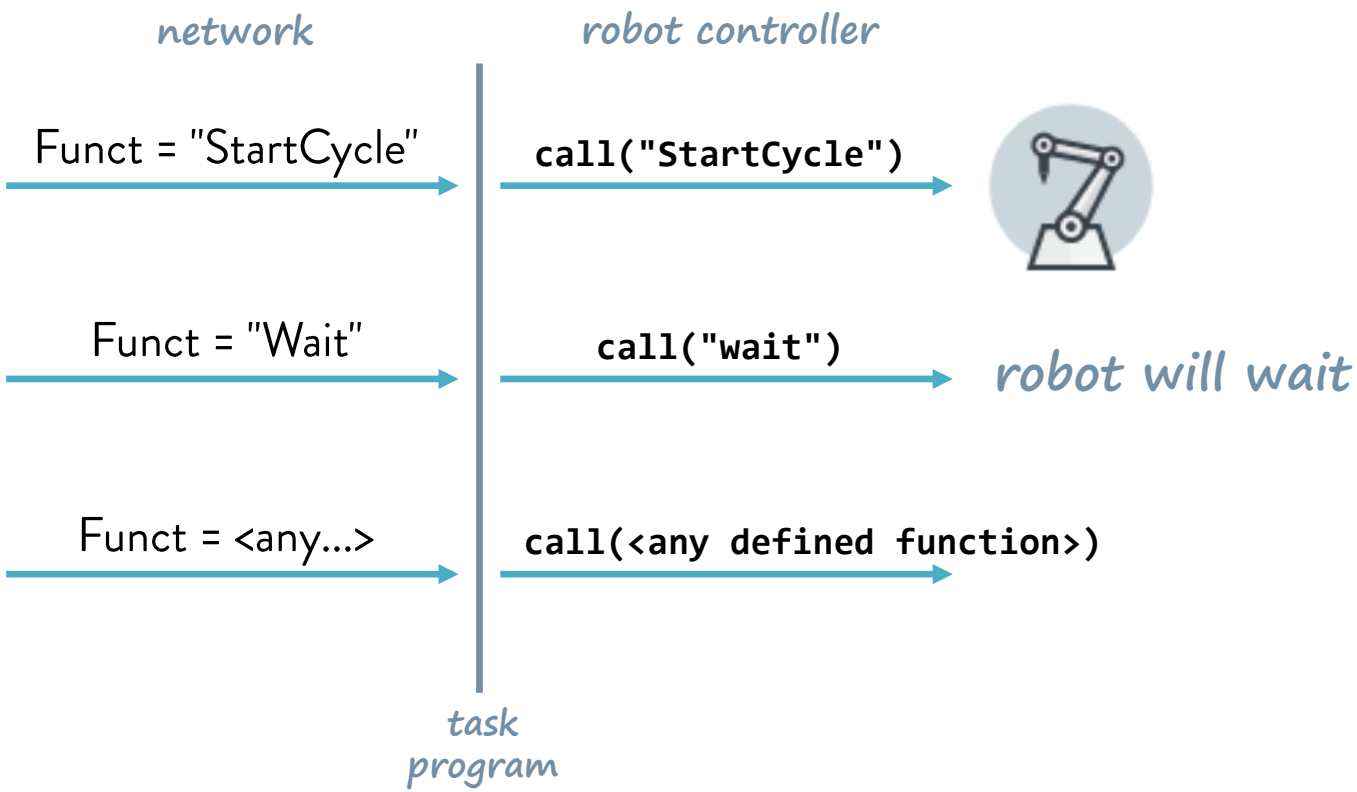
Other Vulnerabilities: Unrestricted Movement



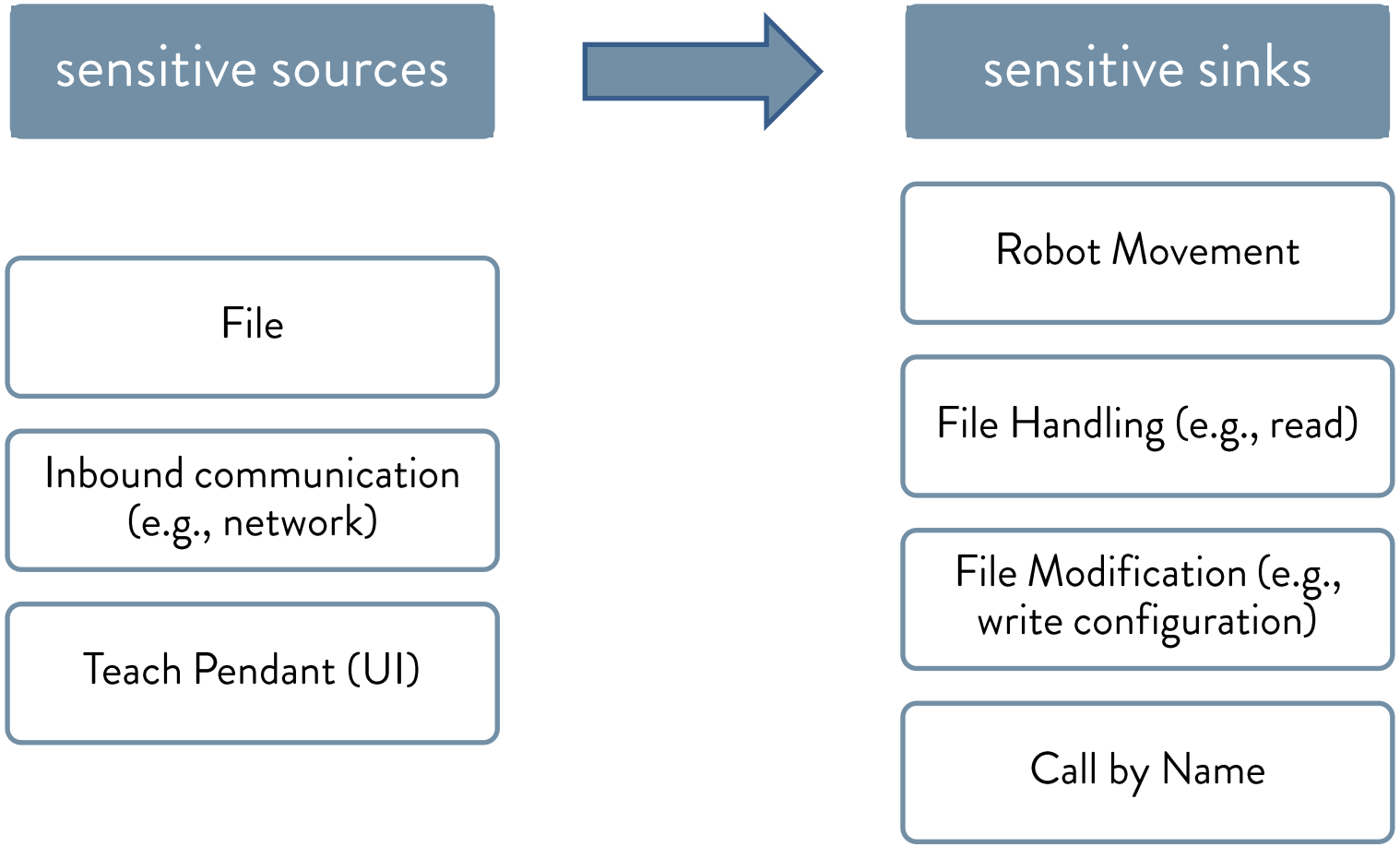
Other Vulnerabilities: Unrestricted Movement

```
DEF external_movement()
    DECL axis pos_cmd

    eki_init("ExiHwInterface")
    eki_open("EkiHwInterface")

    LOOP
        eki_getreal("EkiHwInterface", "RobotCommand/Pos/#A1", pos_cmd.a1)
        eki_getreal("EkiHwInterface", "RobotCommand/Pos/#A2", pos_cmd.a2)
        eki_getreal("EkiHwInterface", "RobotCommand/Pos/#A3", pos_cmd.a3)
        eki_getreal("EkiHwInterface", "RobotCommand/Pos/#A4", pos_cmd.a4)
        eki_getreal("EkiHwInterface", "RobotCommand/Pos/#A5", pos_cmd.a5)
        eki_getreal("EkiHwInterface", "RobotCommand/Pos/#A6", pos_cmd.a6)

        PTP joint_pos_cmd
    ENDLOOP
END
```

Other Vulnerabilities: Arbitrary Function Execution



Recap: Sources and Sinks



A Word About Malicious Patterns

```
MODULE Dropper
    PROC main_loop()

        ! ... variable declaration
        ! ... socket creation and initialization

        WHILE TRUE DO
            SocketReceive clientsock, \Str:=data;
            name := ParseName(data)
            Open diskhome + "/" + name + ".mod", f;
            WHILE data DO
                SocketReceive clientsock, \Str:=rec;
                Write f, rec;
            ENDWHILE
            Load \Dynamic, diskhome \File:=name + ".mod";
            %name + ":main"%; ! call function by name
        ENDWHILE
    ENDPROC
ENDMODULE
```

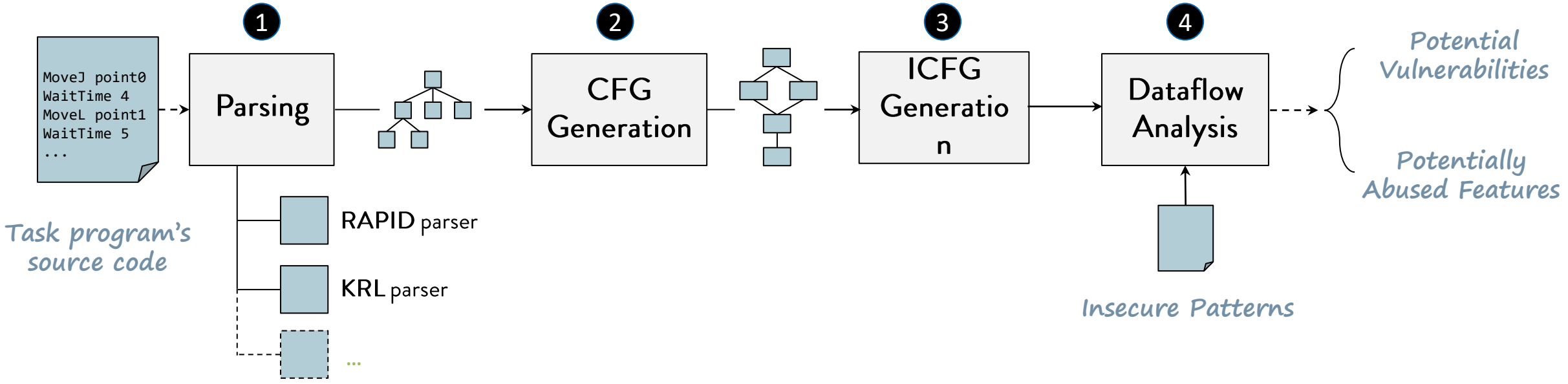
1. Read data from the network
2. Write data to file

3. Load that file as code

Table 4: Examples of malicious patterns.

Case	Feature	Source	Sink
Information stealer	Exfiltration	File	→ Outbound Network
	Exfiltration	Config	→ Outbound Network
	Harvesting	Dir. list	→ File
Dropper	Download	Communication	→ File (code)
	Execute	File (code)	→ Call by name

A Static Source Code Analyzer for Industrial Robots Programming Languages
Overall Architecture



A Static Source Code Analyzer for Industrial Robots Programming Languages

Output


```
"result": [
    {
        "sources": [
            {
                "src_var": "joint_pos_cmd",
                "source": "eki_getreal",
                "source_fn": "eki_hw_iface_get",
                "source_line_no": 178,
                "source_filename": "kuka_eki_hw_interface.src"
            },
            {
                "src_var": "joint_pos_cmd",
                "source": "eki_getreal",
                "source_fn": "eki_hw_iface_get",
                "source_line_no": 175,
                "source_filename": "kuka_eki_hw_interface.src"
            },
            {
                "src_var": "joint_pos_cmd",
                "source": "eki_getreal",
                "source_fn": "eki_hw_iface_get",
                "source_line_no": 176,
                "source_filename": "kuka_eki_hw_interface.src"
            },
            {
                "src_var": "joint_pos_cmd",
                "source": "eki_getreal",
                "source_fn": "eki_hw_iface_get",
                "source_line_no": 179,
                "source_filename": "kuka_eki_hw_interface.src"
            },
            {
                "src_var": "joint_pos_cmd",
                "source": "eki_getreal",
                "source_fn": "eki_hw_iface_get",
                "source_line_no": 180,
                "source_filename": "kuka_eki_hw_interface.src"
            },
            {
                "src_var": "joint_pos_cmd",
                "source": "eki_getreal",
                "source_fn": "eki_hw_iface_get",
                "source_line_no": 177,
                "source_filename": "kuka_eki_hw_interface.src"
            }
        ],
        "sink_var": "joint_pos_tgt",
        "sink": "ptp",
        "sink_fn": "kuka_eki_hw_interface",
        "sink_line_no": 73,
        "sink_filename": "kuka_eki_hw_interface.src"
    }
],
```


```
69      ; Loop forever
70      $advance = 5
71      loop
72        elements_read = eki_hw_iface_get(joint_pos_tgt)  ; Get new command from buffer if present
73        ptp joint_pos_tgt                                ; PTP to most recent commanded position
74      endloop
75
76      ; Note: EKI channels delete on reset or deselect of this program
77      ;       See <ENVIRONMENT>Program</ENVIRONMENT> EKI config element
78    end
```

```
159   ; eki_hw_iface_get
160   ; Tries to read most recent elemnt from buffer. q left unchanged if empty.
161   ; Returns number of elements read.
162   deffct int eki_hw_iface_get(joint_pos_cmd :out)
163      decl eki_status eki_ret
164      decl axis joint_pos_cmd
165
166      if not $flag[1] then
167        return 0
168      endif
169
170      eki_ret = eki_checkbuffer("EkiHwInterface", "RobotCommand/Pos/@A1")
171      if eki_ret.buff <= 0 then
172        return 0
173      endif
174
175      eki_ret = eki_getreal("EkiHwInterface", "RobotCommand/Pos/@A1", joint_pos_cmd.a1)
176      eki_ret = eki_getreal("EkiHwInterface", "RobotCommand/Pos/@A2", joint_pos_cmd.a2)
177      eki_ret = eki_getreal("EkiHwInterface", "RobotCommand/Pos/@A3", joint_pos_cmd.a3)
178      eki_ret = eki_getreal("EkiHwInterface", "RobotCommand/Pos/@A4", joint_pos_cmd.a4)
179      eki_ret = eki_getreal("EkiHwInterface", "RobotCommand/Pos/@A5", joint_pos_cmd.a5)
180      eki_ret = eki_getreal("EkiHwInterface", "RobotCommand/Pos/@A6", joint_pos_cmd.a6)
181
182      return 1
183   endfct
```

Vulnerabilities in Publicly Available Programs

- Dataset of ABB RAPID and KUKA KRL programs
- Open source programs
  - Mostly educational or training or example code
  - Dataset problem ☹

- We found vulnerabilities in:

  10/22 (/14 with at least a source) RAPID

  6/49 (/8 with at least a source) KRL

Vulnerabilities in Publicly Available Programs
Detection Results

Vulnerability	Projects	Files	Root Cause
Network → Remote Function Exec	2	2	Dynamic code loading
Network → File Access	1	4	Unfiltered open file
Network → Arbitrary Movement	13	34	Unrestricted Move Joint or Move to point
Detection Errors	*2*	*12*	*Interrupts*

RobotApps Web Server

Vulnerabilities in Publicly Available Programs
Motion Servers as Cross-Platform Adapters

ros-industrial / kuka_experimental

Watch 30 | Star 96 | Fork 107

Code | Issues 25 | Pull requests 16 | Actions | Security | Insights

Experimental packages for KUKA manipulators within ROS-Industrial (http://wiki.ros.org/kuka_experimental)

kuka | ros-industrial | urdf | rsi | ros-control

114 commits | 2 branches | 0 packages | 0 releases | 13 contributors | Apache-2.0

Branch: indigo-devel | New pull request | Find file | Clone or download

gavanderhoorn readme: load badge from Kinetic devel job. ✓ Latest commit 984e1f2 on Oct 14, 2019

kuka_eki_hw_interface — eki_hw_interface: add cmd buffer length limit to avoid overfeeding co... — 17 months ago

Defense and Remediation Approaches

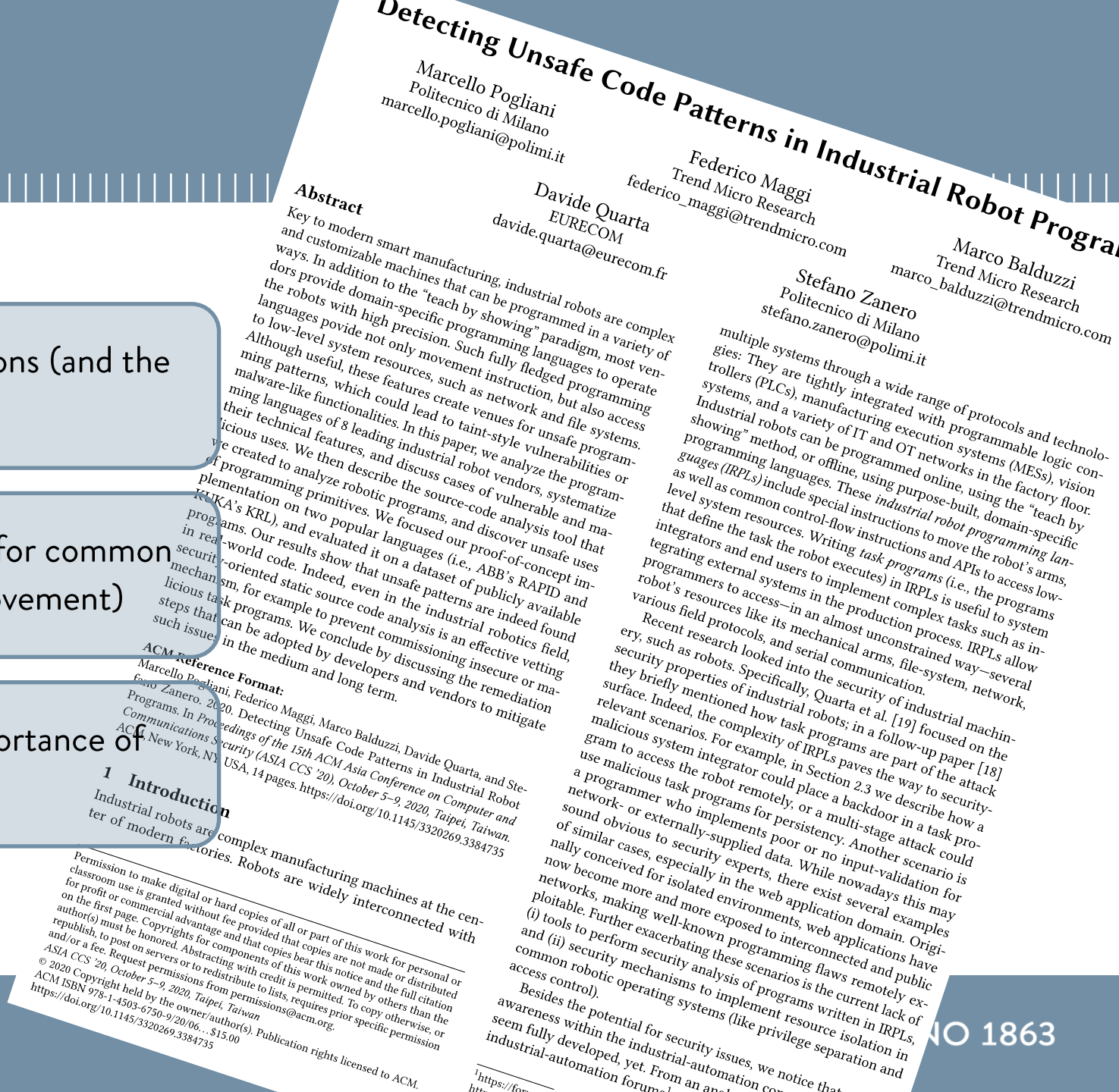
- **Secure communication**: hard to implement without language support
- **Input validation**: hard to fix – what to do when invalid input comes in?
- **Privilege separation**: requires changes at the OS/runtime level
- **Code signing**: probably 5-10 years to see wide adoption

Conclusions

Vulnerabilities are a recurring case in IRPL applications (and the Industry 4.0 paradigm exacerbates this)

Need of rehauling languages, including secure APIs for common vulnerability-prone tasks (e.g., socket-controlled movement)

"appification" of robotics and manufacturing → importance of analysis and vetting tools for 3rd party code



Detecting Unsafe Code Patterns in Industrial Robot Progra

Marcello Pogliani
Politecnico di Milano
marcello.pogliani@polimi.it

Federico Maggi
Trend Micro Research
federico_maggi@trendmicro.com

Marco Balduzzi
Trend Micro Research
marco_balduzzi@trendmicro.com

Davide Quarta
EURECOM
davide.quarta@eurecom.fr

Stefano Zanero
Politecnico di Milano
stefano.zanero@polimi.it

Abstract

Key to modern smart manufacturing, industrial robots are complex and customizable machines that can be programmed in a variety of ways. In addition to the "teach by showing" paradigm, most vendors provide domain-specific programming languages to operate the robots with high precision. Such fully fledged programming languages povide not only movement instruction, but also access to low-level system resources, such as network and file systems. Although useful, these features create venues for unsafe programming patterns, which could lead to taint-style vulnerabilities or malware-like functionalities. In this paper, we analyze the programming languages of 8 leading industrial robot vendors, systematize their technical features, and discuss cases of vulnerable and malicious uses. We then describe the source-code analysis tool that we created to analyze robotic programs, and discover unsafe uses of programming primitives. We focused our proof-of-concept implementation on two popular languages (i.e., ABB's RAPID and KUKA's KRL), and evaluated it on a dataset of publicly available programs. Our results show that unsafe patterns are indeed found in real-world code. Indeed, even in the industrial robotics field, security-oriented static source code analysis is an effective vetting mechanism, for example to prevent commissioning insecure or malicious task programs. We conclude by discussing the remediation steps that can be adopted by developers and vendors to mitigate such issues in the medium and long term.

ACM Reference Format:
Marcello Pogliani, Federico Maggi, Marco Balduzzi, Davide Quarta, and Stefano Zanero. 2020. Detecting Unsafe Code Patterns in Industrial Robot Programs. In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20), October 5–9, 2020, Taipei, Taiwan. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3320269.3384735

1 Introduction

Industrial robots are complex manufacturing machines at the center of modern factories. Robots are widely interconnected with multiple systems through a wide range of protocols and technologies: They are tightly integrated with programmable logic controllers (PLCs), manufacturing execution systems (MESs), vision systems, and a variety of IT and OT networks in the factory floor. Industrial robots can be programmed online, using the "teach by showing" method, or offline, using purpose-built, domain-specific programming languages. These *industrial robot programming languages (IRPLs)* include special instructions to move the robot's arms, as well as common control-flow instructions and APIs to access low-level system resources. Writing *task programs* (i.e., the programs that define the task the robot executes) in IRPLs is useful to system integrators and end users to implement complex tasks such as integrating external systems in the production process. IRPLs allow programmers to access—in an almost unconstrained way—several robot's resources like its mechanical arms, file-system, network, various field protocols, and serial communication.

Recent research looked into the security of industrial machinery, such as robots. Specifically, Quarta et al. [19] focused on the security properties of industrial robots; in a follow-up paper [18] they briefly mentioned how task programs are part of the attack surface. Indeed, the complexity of IRPLs paves the way to security-relevant scenarios. For example, in Section 2.3 we describe how a malicious system integrator could place a backdoor in a task program to access the robot remotely, or a multi-stage attack could use malicious task programs for persistency. Another scenario is a programmer who implements poor or no input-validation for network- or externally-supplied data. While nowadays this may sound obvious to security experts, there exist several examples of similar cases, especially in the web application domain. Originally conceived for isolated environments, web applications have now become more and more exposed to interconnected and public networks, making well-known programming flaws remotely exploitable. Further exacerbating these scenarios is the current lack of (i) tools to perform security analysis of programs written in IRPLs, and (ii) security mechanisms to implement resource isolation and common robotic operating systems (like privilege separation and access control).

Besides the potential for security issues, we notice that awareness within the industrial-automation community does not seem fully developed, yet. From an academic and industrial-automation forum[1]

[1] https://for...

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASIA CCS '20, October 5–9, 2020, Taipei, Taiwan
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6750-9/20/06…$15.00
https://doi.org/10.1145/3320269.3384735

Detecting Insecure Code Patterns in Industrial Robotic Programs

Questions? Feedback?

marcello.pogliani@polimi.it *info@robosec.org*

find these slides at https://robosec.org

TREND MICRO | research

POLITECNICO MILANO 1863 NECST laboratory