Chapter 11

SECURITY ANALYSIS OF SOFTWARE UPDATES FOR INDUSTRIAL ROBOTS

Chun-Fai Chan, Kam-Pui Chow and Tim Tang

Abstract Robots are widely deployed in industrial manufacturing environments. Cyber compromises of industrial robots pose threats to products and services, to the robots as well as to human workers. Previous security studies of robots have focused on network service vulnerabilities and privileged execution. However, research has not examined robot software updates and their security features. This chapter investigates the security features of software updates for a Universal Robots UR3 cobot, one of the most commonly-used collaborative industrial robots.

Keywords: Industrial robots, Universal Robots UR3 cobot, software updates

1. Introduction

Robots are widely deployed in industrial manufacturing environments. Cyber compromises of industrial robots pose threats to products and services, to the robots as well as to human workers. Security studies of robots have focused on network service vulnerabilities and privileged execution. However, research has not examined robot software updates and their security features.

This chapter investigates the security features of software updates for a Universal Robots UR3 cobot, one of the most commonly-used collaborative industrial robots. The security analysis reveals four hitherto unknown vulnerabilities in the software update process that can lead to total compromise of the cobot. Several recommendations for the cobot manufacturer and cobot operators are presented to reduce or mitigate the risks.

Published by Springer Nature Switzerland AG 2022
J. Staggs and S. Shenoi (Eds.): Critical Infrastructure Protection XV, IFIP AICT 636, pp. 229–245, 2022.
https://doi.org/10.1007/978-3-030-93511-5_11

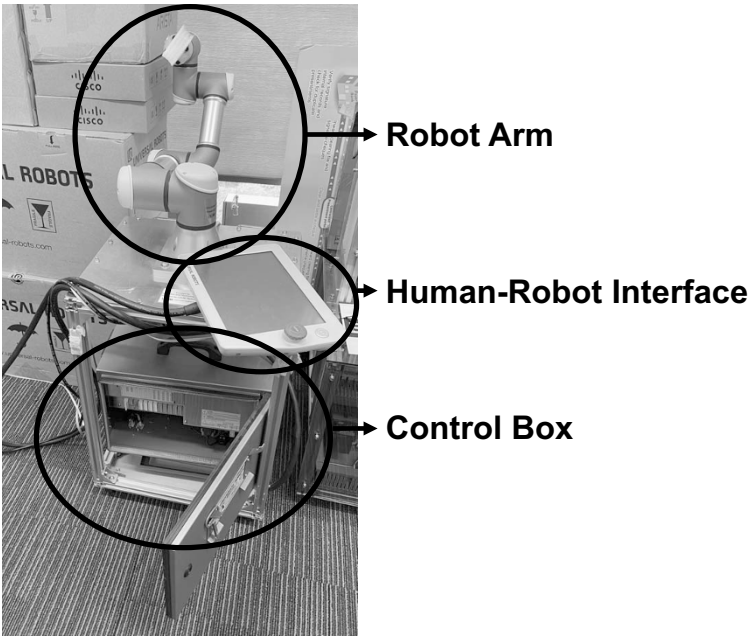

Figure 1. Principal UR3 cobot components.

2. Collaborative Robot

Collaborative robots, called cobots, are designed to operate in a shared space with human workers [2]. Universal Robots, based in Odense, Denmark, is a leading manufacturer of industrial cobots. The company sold its first cobot in 2008 [14] and continues to dominate the cobot sector. The industrial research firm, Interact Analysis [11], reports that Universal Robots had a global market share of almost 50% in 2017. In 2020, Universal Robots sold its 50,000$^{\text{th}}$ cobot [3].

Figure 1 shows a UR3 cobot manufactured by Universal Robots. Its principal components are a robot arm, human-robot interface and control box:

- **Robot Arm:** The robot arm, made of extruded aluminum, comprises tubes and joints that can be controlled on three or more axes, and moved flexibly according to pre-defined instructions [13]. The wrist joints in the arm can rotate 360 degrees and the end joints have infinite degrees of freedom.
- **Human-Robot Interface:** The human-robot interface (HRI) is a touchscreen device similar to a tablet with a wired connection

Figure 2. Control box lock and key.

to a cobot's control box. The human-robot interface houses a Polyscope graphical user interface, which enables a human operator to execute programs and monitor the status of the cobot.

- **Control Box:** The control box of a cobot is enclosed in a chassis. It contains the physical input/output ports and electronic components that connect to the robot arm, human-robot interface and other peripherals. The computer and communications systems for the robot arm and human-robot interface are located in the control box. An operator powers on the control box before booting and controlling the robot arm. Universal Robots provides software resources that support human-robot interactions, cobot programming and cobot movement visualization. These resources can be downloaded from the Universal Robots website [14].

A UR3 cobot incorporates pre-installed safety and security features to protect the cobot and human operators:

- **Physical Security:** A physical lock on the control box prevents tampering with the internal components of the cobot (Figure 2). However, the control box key has only two simple teeth, enabling the lock to be opened with a suitable screwdriver.

- **Software-Defined Safety Settings:** A human operator can specify safety planes that a cobot cannot breach. These safety settings

Figure 3. Safety settings and safety settings modification checksum.

cannot be overwritten by other programmed actions, which reduces the potential harm caused by the cobot.

- **Safety Settings Modification Checksum:** The safety settings are protected by a checksum that is computed automatically. The checksum, which is circled in Figure 3, enables a human operator to verify that the settings have not been changed. When an authorized user modifies the safety settings, the checksum is changed accordingly.

- **Human-Robot Interface Password Protection:** In addition to the Linux shell password, the Polyscope software has two additional password protection mechanisms. One is the system password, which a cobot operator is required to input before performing restricted operations such as changing the system settings and programming the robot arm. The other is the safety password, which protects the safety settings that restrict the locations and movements of the robot arm from being modified. As shown in Figure 4, both the passwords cannot be changed without entering the current passwords.

- **Encrypted Software Update:** The cobot software update file is available at the Universal Robots website. The software update

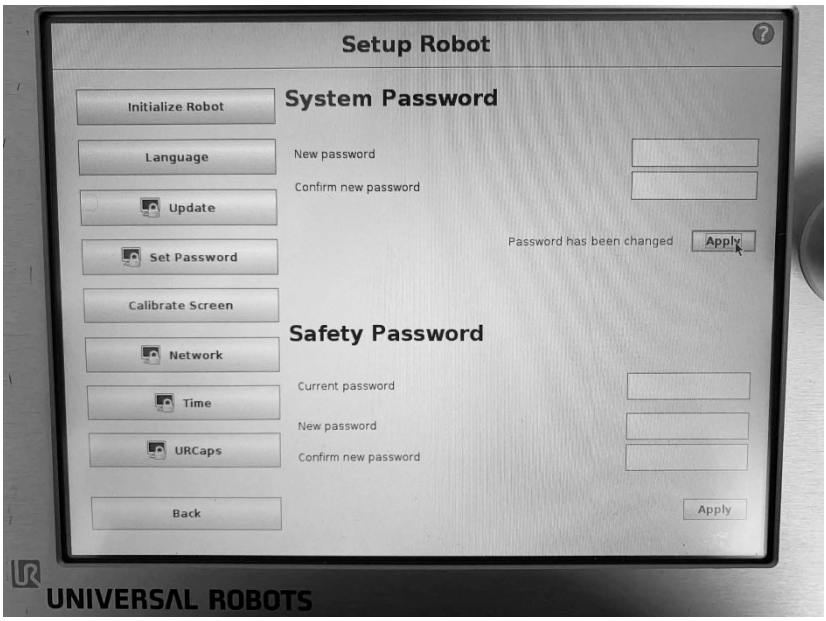

Figure 4. System and safety passwords.

file, which is encrypted using 3TDES, contains the cobot joint firmware update, Polyscope update and update scripts.

3. Previous Work

A growing body of research has focused on the security aspects of industrial robot firmware. Quarta et al. [9] show how attackers can gather information about robot firmware and develop attacks. Information about robots can be obtained without having to purchase them. Many vendors post official support materials on their websites and even allow interested parties to download them free of charge. The most common materials are manuals that provide general information about robots. In the case of advanced robots, additional materials such as software for controlling and simulating robots, as well as drivers and firmware are available. For example, the RobotStudio suite for ABB robots includes a portion of the firmware in the RobotWare distribution [1]. The suite also contains a simulator in the form of a shared library with the entire codebase, which can be analyzed for vulnerabilities. Additionally, platform-specific technical information is available without purchasing a robot.

Researchers have discussed firmware components and how firmware attacks can be performed. Eclypsium [3] describes key firmware components, including system firmware and boot firmware (BIOS, UEFI, EFI and MBR), along with the system management mode and baseboard management controllers that are common components of digital devices.

Rieck [10] discusses the reverse engineering of fitness tracker firmware. The firmware protection mechanism in the fitness tracker analyzed by Rieck is similar to that used by Universal Robotics for software update protection. The fitness tracker firmware comprises two parts, a header and content. The header part has a `table_checksum` field whose value is computed automatically from the content part to verify that no errors occurred during firmware transmission. However, no mechanism is in place to maintain the integrity of the header. Thus, an attacker with access to the firmware can modify the firmware, generate the new checksum and store it in the firmware header. The manipulated firmware can be installed successfully because the checksum and `table_checksum` match.

Shim et al. [12] leveraged Rieck's research to access the firmware of a mobile application that controls a wearable fitness tracker. The firmware was disassembled using the IDA Pro reverse engineering tool. The firmware code was shown to be modifiable by changing the bitmaps of certain characters. The modified firmware was subsequently inserted in the fitness tracker without any warnings or errors.

Apart from research on firmware vulnerabilities, a Robot Vulnerability Database has been created for the public to submit vulnerabilities discovered in industrial robots, which are subsequently verified by experts [7]. A total of 92 vulnerabilities relating to the Universal Robotics UR3 cobot are recorded in the database. Eighty-one vulnerabilities relate to unpatched libraries or binaries in the Linux distribution and eleven relate to the software and firmware. Five of the eleven software/firmware vulnerabilities relate to unauthenticated communications, two to buffer overflows and four to unbounded local privilege execution. None of the reported UR3 cobot vulnerabilities relate to software/firmware integrity.

4. Experiments and Results

A cobot simulation environment was created by downloading and installing Universal Robots offline simulators as virtual machines [15]. The virtual machines were executed in VMWare under a Windows 10 Pro operating system. Experiments were performed with Universal Robots virtual machine CB version 3.14.3 on an i686 architecture running an

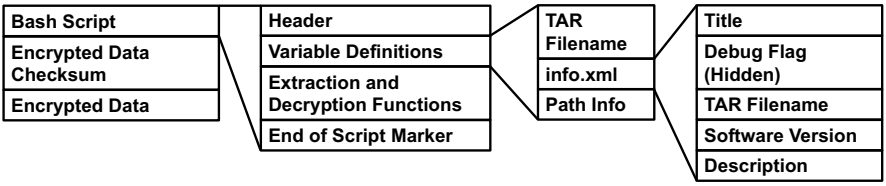

Figure 5. URUP file structure.

Ubuntu 14.04.3 Linux distribution. In addition, experiments were conducted with Universal Robots CB update file versions 3.5.4, 3.6.1, 3.12.1 and 3.14.3. The simulated results were then verified using a Universal Robots UR3 cobot executing CB versions 3.5.1 and 3.5.4 on an i686 architecture with a Debian 7 Linux distribution.

4.1 Software Update File

The software update file (URUP) was examined with the Linux `file` tool to determine if it matched a known file signature. The tool suggested that the update file was a bash script. However, inspection of the update file using a `vim` editor showed that it had bash script text and binary data. Static analysis of the bash script and reverse engineering of the Polyscope program (discussed below) revealed that the URUP update file has the structure shown Figure 5.

A URUP file has three parts. The first part of the update file is a bash script. The first line of the script is a typical bash script header that indicates the bash interpreter file location. After the header line, multiple variables are defined, which include the output TAR filename and the `info.xml` file and extraction path locations. The `info.xml` file content was extracted and read using the Polyscope software update routine (described later).

The `info.xml` file contains its title and description, which are displayed on the human-robot interface screen. The XML file also contains the software version of the URUP update file (which is checked against the existing installed version) and the output TAR filename (same as the TAR filename in the bash script variable definition). Additionally, the XML file may contain flags such as debug and remote server flags that could be used by the update routine, but they were not seen in the URUP update file. Following the variable definitions is a function for extracting `info.xml` from the URUP file and a function for extracting and decrypting the encrypted binary data.

The logic of the function for extracting and decrypting the encrypted binary data is as follows:

- Locate the end of script marker.
- Pipe the remaining file content to a decryption program provided by OpenSSL.
- Decrypt the binary encrypted data using the 3TDES algorithm and symmetric key stored in an environment variable.
- Save the decrypted stream to a file.
- Locate the encrypted data checksum in the update file.
- Compute the MD5 hash of the decrypted file.
- Compare the MD5 checksums.
- Move the decrypted TAR file to a location specified in the input argument.

The second part of the update file is a single line that stores the MD5 checksum of the data contained in the third part of the file.

The third and last part of the update file is a chunk of binary data. According to its binary data header, it corresponds to salted encrypted data. Analysis of the bash script decryption function revealed that the data is encrypted using 3TDES in the CBC mode with a 24 byte key and salt.

4.2 Symmetric Key

Armed with knowledge about the structure and execution flow of the URUP update file, attempts were made to extract the encrypted data in the last part of the update file using the same decryption parameters in the bash script. However, the critical symmetric key was missing. Analysis of the bash script indicated that the symmetric key should be stored in an environment variable. However, examination of the bash shell consoles in the physical and simulated environments did not reveal the presence of the environment variable.

Since the Universal Robots documentation did not have any related information, the only option was to reverse engineer the Polyscope software that provides a graphical user interface to trigger the software update process (Figure 6).

The Universal Robots Polyscope software was developed in Java. The JD-GUI tool [4] was used to decompile the class files in the JAR files.

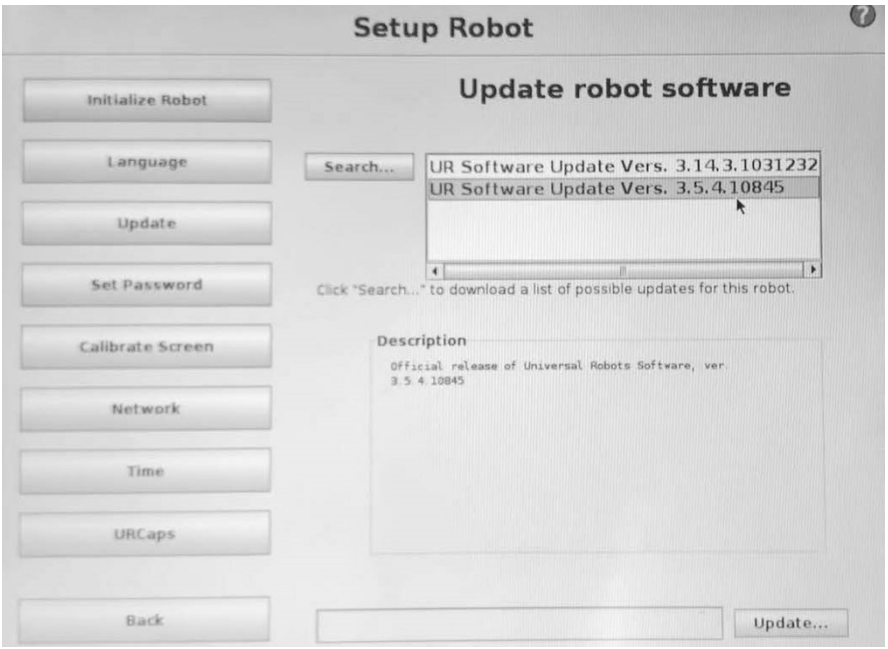

Figure 6. Updating software via the Polyscope graphical user interface.

The decompiled source code was searched for a subroutine that triggers the loading of the URUP update file – this revealed the location where the environment variable is defined. It turned out that the variable is defined just before the bash script is invoked, which is why it could not be located in the bash shell console environment. Surprisingly, the analysis revealed that the symmetric key is saved as a hardcoded string in the Java program (Figure 7).

Multiple versions of the Universal Robots software (version 3.5.4 to the latest version 3.14.3) were decompiled and analyzed. The analyses revealed that all the versions have the same symmetric key saved as a hardcoded string in their Java programs.

4.3 Software Update Process Flow

The decompiled Java source code and other custom system script files were reverse engineered to determine the details of the software update process. The general software update process flow was determined to be as follows:

```
BashScriptRunner bs = new BashScriptRunner("bash " + this.path +
                      " --unpack " + tmpDir.getCanonicalPath());
bs.addEnvVar("RUR", "XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
bs.execute();
```

Figure 7. Hardcoded symmetric key (masked) in the decompiled Polyscope JAR file.

- After a USB drive is plugged into the human-robot interface, an automount service mounts the USB drive to a folder with the prefix (`usbdisk*`) under `/programs`.
- When the Search button is pressed on the Polyscope software update screen, the Java program searches all the URUP files in the mounted folder.
- For each URUP file, the bash script is executed to extract the `info.xml` file content to a different temporary folder.
- The related information of each `info.xml` file is displayed on the human-robot interface screen.
- When a file is selected and the Update button is clicked, the Java program invokes the bash script of the file to extract, decrypt and move the binary data to a temporary folder.
- Major and minor software update versions are checked against the current version and only the compatible version is permitted to be updated.
- The untarred files are saved to `/root/update` and a system reboot is performed.
- After the reboot, the post upgrade script `post.sh` is invoked to copy the files to the appropriate locations.

5. Software Update Process Vulnerabilities

Insights gained from the URUP update file structure and software update process flow supported the vulnerability discovery efforts. This section describes four new vulnerabilities of the UR3 cobot software update mechanism and their exploitation.

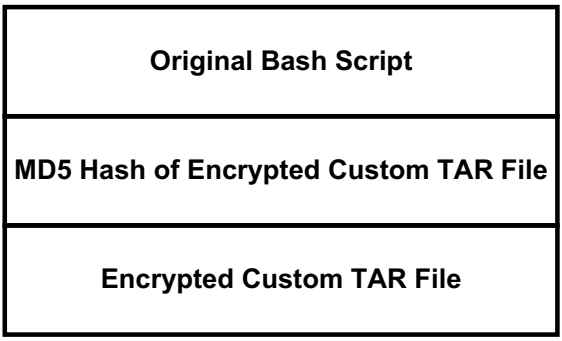

Figure 8. Rogue software update file structure.

5.1 Malicious Software Update File Creation

Because the symmetric key is known and shared by multiple versions of update files, malicious update files with encrypted content can be created to masquerade as legitimate update files.

Two methods were attempted:

- **Method 1:** The first method created an encrypted custom TAR file using the symmetric key that was discovered. An MD5 checksum was computed for the TAR file. The original bash script, new MD5 checksum and encrypted TAR file were concatenated to produce a rogue update file with the structure shown in Figure 8.

- **Method 2:** The second method rewrote the bash script in the first part of the update file. This bypassed the decryption process without needing the symmetric key. At a triggering point in the update process, the content was dumped to a temporary folder and the update process continued its execution.

Method 1 is stealthier than Method 2. Method 2 is easier to perform than Method 1. Also, it enables other actions to be executed that lead to another vulnerability discussed below.

Additionally, an update file with an arbitrary version number can be created by modifying the variables in the bash script as shown in Figure 9. This could mislead a cobot operator to believing that the rogue file is actually a more recent update file. Since the UR3 cobot does not permit software to be downgraded, once a new rogue version is installed, the cobot cannot be rolled back to the previous official version.

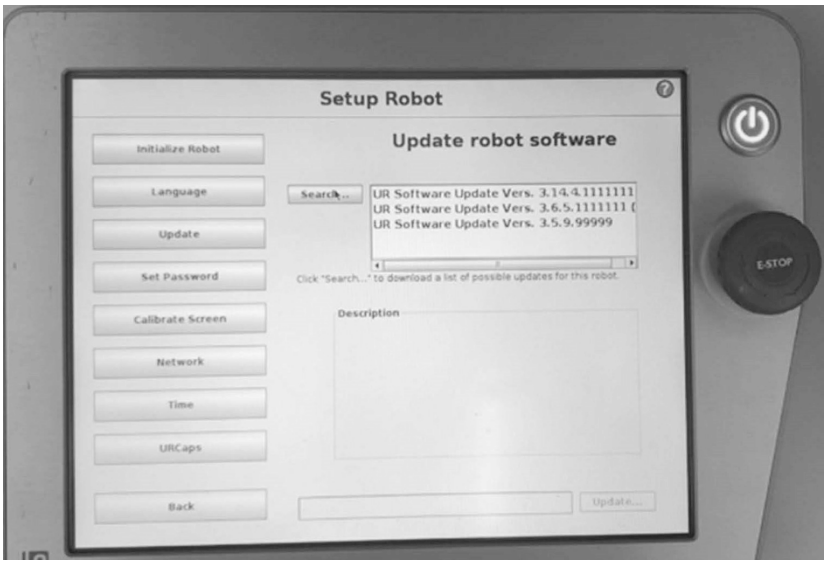

Figure 9. Arbitrary update file version.

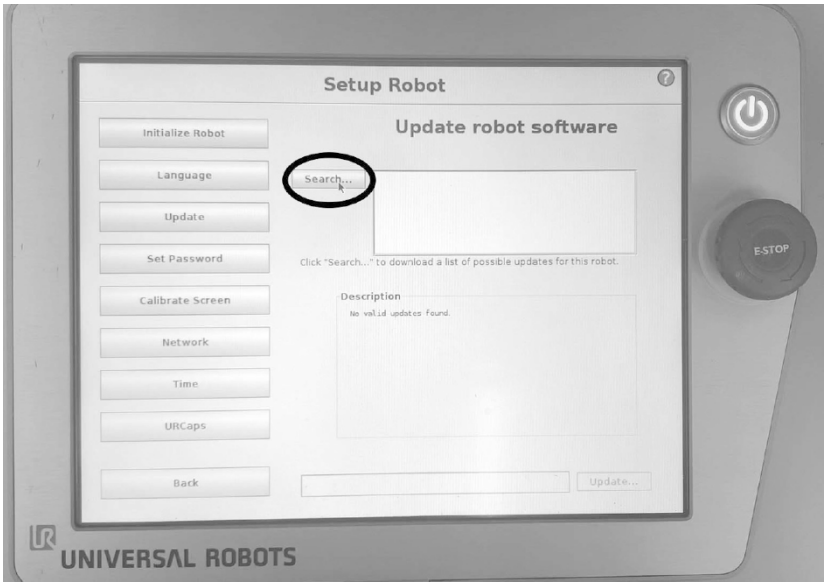

Figure 10. Search function in the software update user interface.

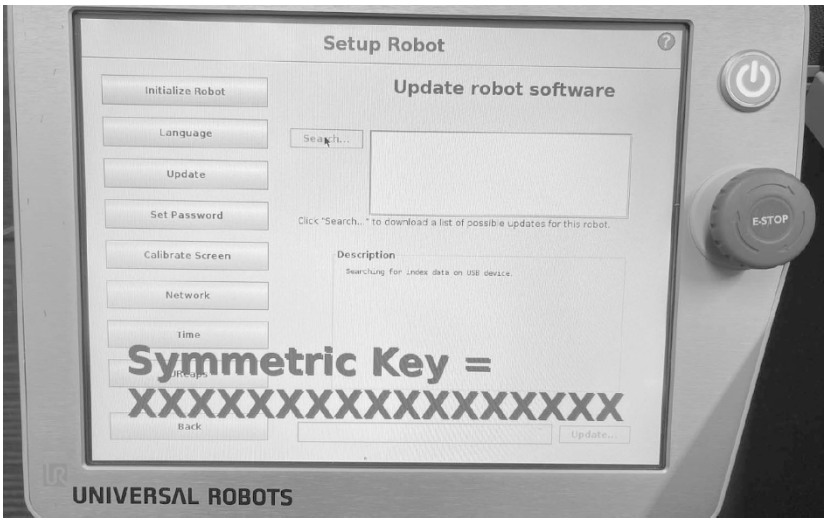

Figure 11. Arbitrary script execution after pressing the Search button.

5.2 Arbitrary Script Execution

The software update process flow analysis revealed that the Search function attempts to locate all the URUP files at the USB mount point (Figure 10). Following this, it tries to extract the `info.xml` files from the URUP files.

However, the `info.xml` extraction function is also in the bash script in the update file. Rewriting the extraction function in the bash script enables the execution of an arbitrary script when a user presses the Search button. Figure 11 shows an example of arbitrary script execution. In the example, the extraction function was modified to dump the symmetric key on the screen.

5.3 Password Integrity

Forensic analysis of the cobot system revealed that two files are modified when the system password and safety password are changed. Both are hidden files with filenames starting with "." and located in the root directory. Each file has only one line corresponding to the encrypted password string.

Experiments were conducted to change content by replacing the encrypted string with another encrypted string for which the plaintext password was known. It was discovered that, after the user interface is

reloaded, the system accepts the new password and permits changes to all the system and safety settings without requiring the previous passwords to be entered. Even worse, when the password files were removed, all the restrictions imposed by password protection vanished and any and all settings could be changed. Leveraging this vulnerability along with the arbitrary script execution vulnerability described above enabled the creation and execution of a rogue update file that triggered the elimination of all the password restrictions on the cobot.

5.4 Arbitrary File Creation

The Polyscope software has an Expert Mode that provides advanced features for users. The features include creating folders and editing system files. A correct password has to be entered to access the Expert Mode, but the password is available because it is hardcoded in the Java program and is also listed in the user manual.

The vulnerability was exploited to create a fake USB mount point directory and implant a rogue URUP update file in the directory. The arbitrary script execution vulnerability described above was exploited to write a script to execute arbitrary commands. Additionally, the password integrity vulnerability described above was leveraged to remove all the password-imposed restrictions.

These exploits enable any user with physical access to the human-robot interface screen to execute any command and change the system and safety settings of the robot arm even when the USB and network ports are blocked. The impacts are serious – potentially causing harm to the cobot and its products and services as well as its human operators.

6. Discussion

In theory, a carefully-crafted rogue update file would leave minimal, if any, traces in the cobot filesystem by removing and overwriting all its intermediate files after execution. The digital forensic method proposed by Gong et al. [5] may not be able to discover the execution of the rogue script. This is because script execution does not require system login and, therefore, no records exist in `bash_history` and the system login record. Additionally, a log would not be written in `log_history.txt` because it is not a Polyscope software operation.

Based on the severity of the vulnerabilities discovered in the software update process, the following actions are recommended:

- Authenticity checks should be enforced at multiple levels. First, a software update file should not use a hardcoded symmetric key to encrypt content; the cobot manufacturer should consider using

public key cryptography to validate software update signatures before installing the software update. Second, all human-robot interface users should be authenticated before gaining access or performing any operations, especially invoking functions that can modify the filesystem.

- The logic flow of the software update process should be reviewed and modified with security in mind. For example, `info.xml` file extraction and software update decryption should be implemented by functions within Polyscope. Additionally, software updates should not be trusted until their authenticity is verified.
- The operating system and network services should be hardened to minimize the attack surface.
- Finally, physical access to the cobot should be restricted, including blocking or locking USB ports and the human-robot interface until users are authenticated and only if access is needed.

7. Conclusions

This chapter presents a security analysis of the software update protection mechanisms of the Universal Robots UR3 cobot, one of the most popular industrial cobots. The security analysis has revealed four hitherto unknown vulnerabilities in the software update process that can lead to total compromise of the cobot. Several recommendations for the cobot manufacturer and cobot operators are presented to reduce or mitigate the risks.

Future research will to employ digital forensic and reverse engineering techniques to identify new vulnerabilities that could be exploited to cause harm to the cobot and its products and services as well as its human operators.

Acknowledgement

The authors wish to thank the Logistics and Supply Chain MultiTech R&D Centre for providing Universal Robots equipment and for sharing its previous research results, both of which have contributed to the research described in this chapter.

References

[1] ABB, Download RobotStudio with RobotWare and PowerPacs, Zurich, Switzerland (new.abb.com/products/robotics/robotstudio/downloads), 2021.

[2] L. Apa, Exploiting Industrial Collaborative Robots, IOActive, Seattle, Washington (www.ioactive.com/exploiting-industrial-collaborative-robots), August 22, 2017.

[3] Eclypsium, Anatomy of a Firmware Attack, Portland, Oregon (eclypsium.com/wp-content/uploads/2020/09/Anatomy-of-a-Firmware-Attack-2020.pdf), 2020.

[4] emmanuel, A Standalone Java Decompiler (JD-GUI 1.6.6), GitHub (github.com/java-decompiler/jd-gui), 2019.

[5] Y. Gong, K. Chow, Y. Mai, J. Zhang and C. Chan, Forensic investigation of a hacked industrial robot, in *Critical Infrastructure Protection XIV*, J. Staggs and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 221–241, 2020.

[6] International Federation of Robotics, Record 2.7 million robots work in factories around the globe, Frankfurt, Germany (ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe), September 24, 2020.

[7] V. Mayoral Vilches, L. Usategui San Juan, B. Dieber, U. Ayucar Carbajo and E. Gil-Uriarte, Introducing the Robot Vulnerability Database (RVD), arXiv: 1912.11299 (arxiv.org/abs/1912.11299), 2020.

[8] I. Priyadarshini, Detecting and mitigating robotic cyber security risks, in *Cyber Security Risks in Robotics*, R. Kumar, P. Pattnaik and P. Pandey (Eds.), IGI Global, Hershey, Pennsylvania, pp. 333–348, 2017.

[9] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. Zanchettin and S. Zanero, An experimental security analysis of an industrial robot controller, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 268–286, 2017.

[10] J. Rieck, Attacks on Fitness Trackers Revisited: A Case Study on Unfit Firmware Security, arXiv: 1604.03313 (arxiv.org/abs/1604.03313), 2016.

[11] A. Sharma, Universal Robots continues to dominate cobot market but faces many challengers, Interact Analysis, Irthlingborough, United Kingdom (www.interactanalysis.com/universal-robots), November 6, 2018.

[12] J. Shim, K Lim, J. Jeong, S. Cho, M. Park and S. Han, A case study on vulnerability analysis and firmware modification attack on a wearable fitness tracker, *IT Convergence Practice*, vol. 5(4), pp. 25–33, 2017.

[13] Universal Robots, User Manual: UR3/CB3, Version 3.5.5, Odense, Denmark (s3-eu-west-1.amazonaws.com/ur-support-site/32340/UR3_User_Manual_en_Global-3.5.5.pdf), 2018.

[14] Universal Robots, About Universal Robots, Odense, Denmark (www.universal-robots.com/about-universal-robots), 2021.

[15] Universal Robots, Legacy Download Center, Odense, Denmark (www.universal-robots.com/articles/ur/documentation/legacy-download-center), 2021.