Targeted Attacks on Teleoperated Surgical Robots: Dynamic Model-based Detection and Mitigation

Homa Alemzadeh, Daniel Chen, Xiao Li*, Thenkurussi Kesavadas*, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
{alemzad1, dchen8, kalbarcz, rkiyer}@illinois.edu
*Health Care Engineering Systems Center, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
{xiaoli16, kesh}@illinois.edu

***Abstract*—This paper demonstrates targeted cyber-physical attacks on teleoperated surgical robots. These attacks exploit vulnerabilities in the robot's control system to infer a critical time during surgery to drive injection of malicious control commands to the robot. We show that these attacks can evade the safety checks of the robot, lead to catastrophic consequences in the physical system (e.g., sudden jumps of robotic arms or system's transition to an unwanted halt state), and cause patient injury, robot damage, or system unavailability in the middle of a surgery. We present a model-based analysis framework that can estimate the consequences of control commands through real-time computation of robot's dynamics. Our experiments on the RAVEN II robot demonstrate that this framework can detect and mitigate the malicious commands before they manifest in the physical system with an average accuracy of 90%.**

Keywords—Targeted Attacks, Malware, Telerobotics, Robotic Surgery, RAVEN II robot, Cyber-physical systems

I. Introduction

Robotic surgical systems are among the most complex medical cyber-physical systems. They enable performing minimally invasive procedures with better visualization and increased precision using 3D magnified views of the surgical field and tele-manipulated arms and instruments that mimic human hand movements. During 2007-2013, over 1.74 million robotic procedures were performed in the U.S. across various surgical specialties, including gynecology, urology, general surgery, cardiothoracic, and head and neck surgery [1]. The next generation of surgical systems are envisioned to be teleoperated robots that can operate in remote and extreme environments such as disaster-stricken areas, battlefields, and outer space [2].

Past studies have emphasized the importance of security attacks that compromise the communication channels in medical devices such as implantable cardiac defibrillators [3], wearable insulin pumps [4], and teleoperated surgical robots [5]-[8]. For example, studies [7] and [8] demonstrated denial of service (DOS) and man-in-the-middle (MITM) attacks on the network communication between the teleoperation console and the control system of a surgical robot. To the best of our knowledge, no previous work has discussed the possibility of directly compromising the control systems of surgical robots. It is usually assumed that getting access to the robot control system is unlikely.

In this paper, we demonstrate cyber-physical attacks on the *control system* of surgical robots in the event when the attacker is able to install a malware to strategically inject faults into the control system at critical junctures during surgery. In order to install the malware, we assume that the attacker has access to the system as an insider or through remote code execution. The malware modifies the control commands while preserving their legitimate format, making this type of attacks difficult to detect without understanding the dynamics of the robot's manipulators.

To detect and mitigate such attacks, we have developed a model-based analysis framework based on the dynamics of the surgical robot and use it to preemptively determine if a command is malicious before the actual execution of the command can progress in the physical robot. We validated the detection experimentally using two real attack scenarios involving injection of unintended user inputs and unintended motor torque commands.

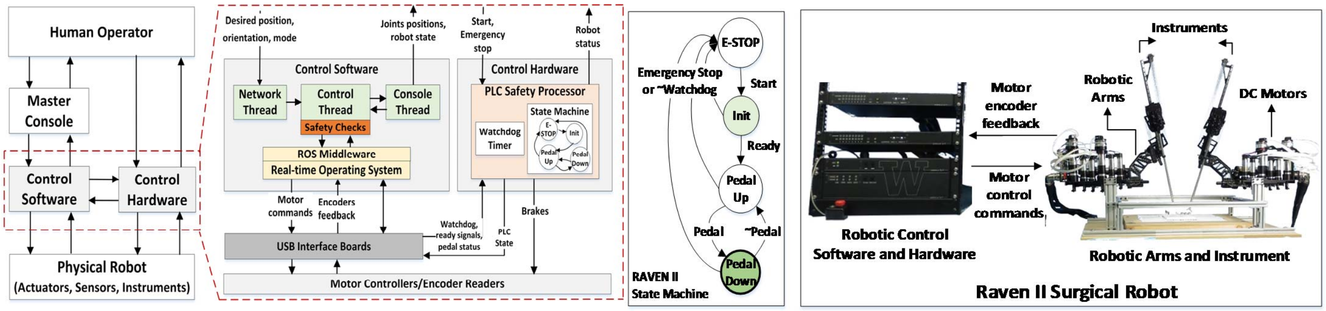
The attacks are deployed via a self-triggered malware with embedded: (i) *logging* mechanisms for collecting and analyzing measurements from the surgical robot in order to identify the critical states and (ii) *fault-injection* mechanisms for inserting malicious commands into the robot control system. The deployment of the malware presumes that the attacker has penetrated the hospital network by exploiting vulnerabilities in the underlying hospital network and has obtained access to the robot control system by exploiting a zero-day remote code execution vulnerability (similar to the ones listed in Table III). This is a credible threat as recent reports indicate the existence of many vulnerabilities in hospital networks [9], in commonly used hospital medical devices [10], and in the software firewall of surgical robots [11], that allow attackers to gain access to critical medical devices.

The cyber-physical attack scenarios presented in this paper have the following important characteristics that complicate their detection and diagnosis:

1) Attacks exploit the TOCTOU (time of check-time of use) vulnerability between the safety checks on the commands and the actual execution of the commands.
2) Attacks are initiated in the cyber domain by modifying the control commands while preserving their legitimate format and syntax, i.e., no change to the control flow (in terms of the sequence of the functional blocks invoked) and no

978-1-4673-8891-7/16 $31.00 © 2016 IEEE
DOI 10.1109/DSN.2016.43

IEEE computer society



(a) (b) (c) (d)

Figure 1. (a) Typical control structure in surgical robots, (b) Software and hardware control loops in the RAVEN II robot, (c) Operational state machine of the RAVEN II robot, d) RAVEN II surgical platform [12].

change to the performance of the target program (preserving the real-time constrains of the robot control software).

3) Attacks directly result in catastrophic consequences in the physical domain (e.g., abrupt jump of the robotic arms), causing damage to the robot or harm to the patient. They are hard to distinguish from incidents caused by system or human induced accidental failures and therein lies the importance of these attacks–answering the question why attacker does not simply kill the robot. If deployed on wide scale, such attacks could cause major disruption and damage to surgical facilities and cause financial or legal impacts.

We illustrate the attacks by implementing a prototype of the malware targeting the RAVEN II robot, an open-source platform for research in teleoperated robotic surgery [12]. We use the RAVEN robot as our experimental platform for several reasons: (i) it contains the typical control and safety mechanisms used in the state-of-the-art robotic surgical systems, (ii) it is a platform indicative of the next-generation teleoperated surgical robots with both remote operability and networking features, and (iii) it is accessible for demonstrating security attacks and studying their impact without the need to interrupt real surgical procedures or risk of harming patients.

Our experiments on the RAVEN II robot demonstrate that: a) injecting malicious commands to motor controllers can lead to abrupt jumps of a few millimeters in the robot manipulators within only a few milliseconds and b) our dynamic model-based analysis framework can detect malicious commands and mitigate their impact before they manifest in the physical system, with an average accuracy of 90%.

II. Background

A. Robotic Surgical Systems

Surgical robots are designed as human-operated robotically-controlled systems, consisting of a *teleoperation console*, a *robot control system*, and a *patient-side cart* (which hosts the robotic arms, holding the surgical endoscope and instruments). The most critical component of the robot control is the electronic control system, which is responsible for the following:

- *Receiving the surgeon's commands* issued using master manipulators and foot pedals on the teleoperation console.
- *Translating the surgeon's commands* into the corresponding surgical robot movements.
- *Providing video feedback* of the surgical field (inside patient's body) to the surgeon through 3D vision on the teleoperation console.
- *Performing safety checks* on to ensure the safe operation of the surgical robot.

Figure 1(a) shows the typical control system structure of a surgical robot based on our review of publicly available documents on commercial and open-source robotic surgical platforms including da Vinci Surgical System [13][14], the da Vinci Research Kit [15], and the RAVEN II robot [12][16]. In this paper we use the RAVEN II robot as an experimental platform for implementing the attack scenarios and characterizing the robot's resiliency to those attacks. We treat the RAVEN robot as a grey box system. (i.e., we do not have any access to the robot's source code.)

B. RAVEN II Robotic Surgical Platform

Figure 1(d) depicts the configuration of the RAVEN II system. The desired position and orientation of robotic arms, foot pedal status, and robot control mode are sent from the teleoperation or master console (not shown in the figure) to the robotic control software over the network using the Interoperable Teleoperation Protocol (ITP), a protocol based on the UDP packet protocol. The control software receives the user packets, translates them into motor commands, and sends them to the control hardware, which enables the movement of robotic arms and surgical instruments. The robot consists of two cable-driven surgical manipulators attached with tool interfaces and the instruments. Each surgical manipulator is operated by DC motors and has seven degrees of freedom [12].

As shown in Figure 1(b), the control software runs as a node (process) on the Robotic Operating System (ROS) middleware [17] on top of a real-time (RT-Preempt) Linux kernel. It communicates with the motor controllers and a Programmable Logic Controller (PLC) through two custom 8-channel USB interface boards. The interface boards include commodity programmable devices, digital to analog converters, and encoder readers. The motor controllers send movement commands (torque values calculated based on the desired joint positions) to the DC motors and read back the encoder values from the motors (to estimate the current joint positions). The PLC controls the fail-safe brakes on the robotic joints and monitors the system state by communicating with the robotic software.

As shown in Figure 1(c), the RAVEN control system goes through an initialization phase before getting ready for the operation. During the initialization phase, the mechanical and electronic components of the system are tested to detect any faults or problems. After successful initialization, the robot enters the “Pedal Up” state, in which the robot is ready for teleoperation but the brakes are engaged. When the foot pedal is pressed by the human operator, the robot moves to the “Pedal Down” state. In this state the brakes are released, allowing the teleoperation console to control the robot [12][16].

Figure 2 shows the kinematic chain of the RAVEN control software. The operator commands are sent to the control software as incremental motions (desired end-effector positions (*pos_d*) and orientations (*ori_d*)). The current end-effector’s configurations (*pos* and *ori*) are calculated based on motor encoder feedback using forward kinematics function. The inverse kinematics calculates the joint (*jpos_d*) and motor (*mpos_d*) positions that are required to obtain the desired end-effector configurations and positions. Finally, the amount of torque needed for each motor to reach its new position is obtained from a Proportional-Interal-Derivitive (PID) controller. The motor torques are then transferred in the form of DAC commands (*DAC_value*) to the motor controllers on the USB boards, to be executed on the motors [18].

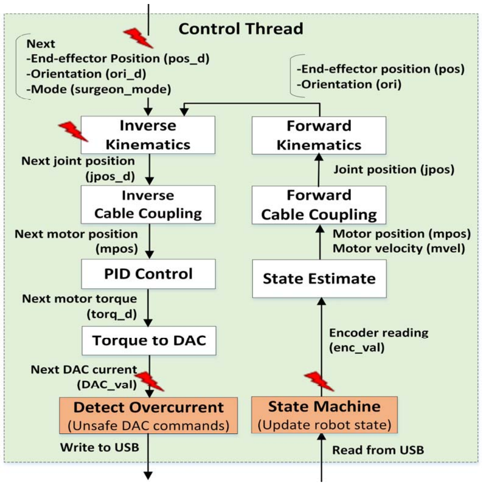


Figure 2. The kinematics chain in the RAVEN II control software

The RAVEN II robot has the following safety mechanisms [12]:

- A physical start button should be pressed to take the robot out of the emergency stop (“E-STOP”) state. At any time pressing the emergency stop button will immediately stop the robot by putting the PLC and control software into the “E-STOP” state (see Figure 1(c)).
- Whenever the human operator lifts the foot from the pedal, the system enters the “Pedal Up” state and engages the fail-safe power-off brakes on the motors and disengages the master console from manipulating the surgical arms.
- The control software performs safety checks on the motor controller commands before they are sent to the USB I/O boards. These safety checks compare the electrical current commands sent to the digital to analog converters (DACs) with a set of pre-defined thresholds to ensure the motors and arm joints do not move beyond their safety limits.
- The control software sends a periodic (I’m alive) square-wave watchdog signal to the PLC through the USB boards. Upon detecting any unsafe motor commands, the control software stops sending the watchdog signal. The PLC safety processor monitors the watchdog signal and in absence of the watchdog signal puts the system in the Emergency-Stop (“E-STOP”) state.

III. Cyber-Physical Attacks on The RAVEN II Robot

Previous studies on fault-injection based safety assessment of RAVEN II system have shown several vulnerabilities in the safety mechanisms of the robot [19][20]. In this paper, we show that malicious parties can exploit such vulnerabilities to perform cyber-physical attacks that are difficult to be detected without modeling the robot’s dynamics.

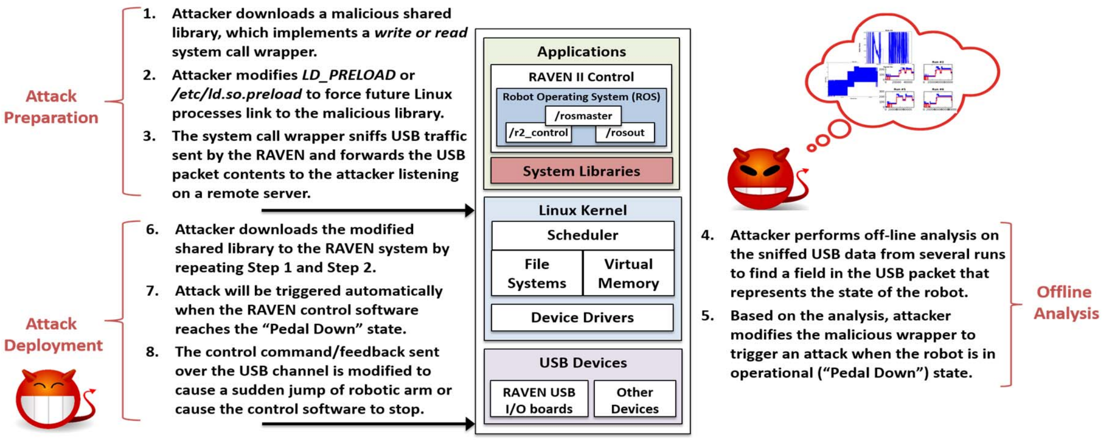
Table I. Variants of Attacks On Robot Control Structure

Target Layer	Target System Library	Malicious Action	Observed Impact
Master Console and Control Software	Socket comm. (bind, received_from)	Change -port number -packet content	Hijack trajectory Unwanted state (E-STOP)
Control Software	Math (sin, cos)	Add drift to -output -input	Unwanted state (IK-fail)
Control Software and Hardware	Interface (read, write)	Change -robot state in PLC	Homing Failure
Software and Physical Robot		Change -motor commands -encoder feedback	Abrupt Jump/ Unwanted state (E-STOP)

The attacks exploit the dynamic loading feature for system libraries in the underlying Linux OS and vulnerabilities in RAVEN II software-hardware interface to inject malicious actions at different layers of the robot control structure (shown in Figure 1(a)). The attacks can cause a variety of adverse impacts on the robot functionality, the patient, and these impacts are potentially difficult to distinguish from unexpected failures. Table I summarizes variants of those attacks, categorized by the target layer in the control structure (see the red marks in Figure 2), the target system library, the type of malicious action, and their observed impact on the system (as reported in [20]). We specifically focuse on two attack scenarios that cannot be detected and mitigated by the existing safety mechanisms in the RAVEN II robot:

A. **Injection of unintended user inputs** after they are received by the control software. These attacks either cause hijacking the control of the robot by performing an action that was not initiated by the operator or lead to unintended jumps and unwanted halt states.

B. **Injection of unintended motor torque commands** after they have passed the safety checks and before transmission to the USB interface boards and motor controllers. These attacks can lead to unintended moves and abrupt jumps of the robot or unwanted halt states.



1. Attacker downloads a malicious shared library, which implements a *write or read* system call wrapper.
2. Attacker modifies *LD_PRELOAD* or */etc/ld.so.preload* to force future Linux processes link to the malicious library.
3. The system call wrapper sniffs USB traffic sent by the RAVEN and forwards the USB packet contents to the attacker listening on a remote server.
4. Attacker performs off-line analysis on the sniffed USB data from several runs to find a field in the USB packet that represents the state of the robot.
5. Based on the analysis, attacker modifies the malicious wrapper to trigger an attack when the robot is in operational ("Pedal Down") state.
6. Attacker downloads the modified shared library to the RAVEN system by repeating Step 1 and Step 2.
7. Attack will be triggered automatically when the RAVEN control software reaches the "Pedal Down" state.
8. The control command/feedback sent over the USB channel is modified to cause a sudden jump of robotic arm or cause the control software to stop.

Figure 3. Attack scenario B (injection of unintened motor torque commands) in RAVEN II surgical robot

We exemplify the attacks by deploying attack scenario B (described above) on the RAVEN II robot. We used a desktop computer running RAVEN II software on top of ROS Indigo and Linux Ubuntu 14.04 LTS with SMP Preempt Real-time kernel. The machine contained an Intel Core i5 CPU@2.90 GHz and 8GB of RAM. The malicious code was implemented using bash, Python scripts, and ROS commands and was executed in the user space (no root privilege was needed to execute the malware).

A. Attack Model

We focus on the steps taken *after* the attacker has obtained remote access to a robot control system on a hospital's network. The attacker can gain such access by exploiting weaknesses such as vulnerable services, unpatched medical devices, stolen credentials, or insider attacks to penetrate the hospital network. Once in the hospital network, the attacker can move laterally across devices within the hospital, steal additional credentials and discover vulnerabilities until the target robot control system is located and penetrated. The attacks discovered by TrapX Security, Inc. [10], the Stuxnet attack [21], and the discovered vulnerability in the firewall of a commercial robot [11] serve as examples of how these penetration attacks could be performed. Table V shows the common entry points exploited in recent attacks detected on hospital networks. The purpose is to assert that access to the robot control system in present day environments is not only feasible but quite probable.

After getting access to the robot, the intention of the attacker is to remain on the target system without being detected for as long as possible in order to (i) collect data from the system, (ii) analyze the collected data to create an operational profile of the robot and determine the best time for activating the attack, and (iii) trigger the attack at the desired critical time.

We assume the attacker does not have access to the source code or internal design of the robot. The attacker gathers information about the system configuration and potential vulnerabilities of the robot through publicly available documents (e.g., previous publications on vulnerabilities of RAVEN II robot [8][19][20]) or through a vulnerability discovery process consisting of targeted probing and fuzzing.

There are specifically two pieces of information that the attacker must have about the robot in order to perform a successful attack: (i) the state machine representing robot operations and (ii) a side channel that can be used to extract the current state of the robot in order to determine the best time to trigger an attack. The attacker also needs at least a user privilege to download and run the malicious code on the system.

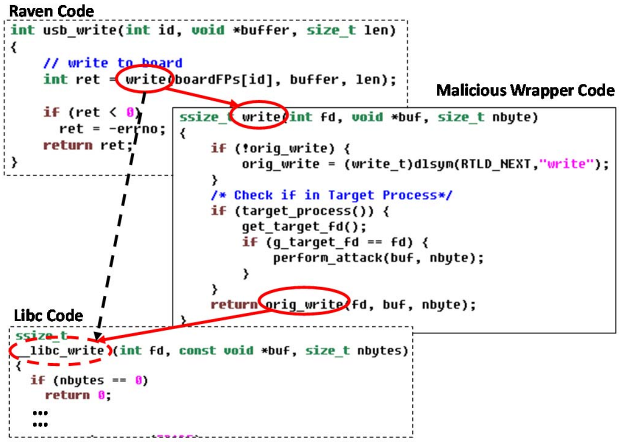
B. Attack Description

In the attack scenario illustrated in Figure 3, an attacker (who penetrated RAVEN control system) first eavesdrops (intercepts) on the USB communication between the RAVEN control software and the USB I/O boards. The intercepted packets are analyzed offline to extract the state information of the surgical robot, i.e., determine the state of the robot according to the operational state machine depicted in Figure 1(c). The extracted data are then used to build a malware for triggering (injecting) an attack at a critical time during the robot's operation, i.e., when the robot is operating in the "Pedal Down" state.

Figure 3 describes the steps to execute the attack on a RAVEN II robot: These steps are grouped into three phases: *Attack Preparation Phase*, *Analysis Phase*, and *Deployment Phase*. The Attack Preparation Phase and the Analysis Phase need to be performed only once to obtain the information necessary to design and implement the final malware capable of triggering an attack when the robot is most vulnerable. The details of each phase are described next.

1) Attack-Preparation Phase: The goal of the Attack-Preparation phase is to eavesdrop on the communication between the RAVEN control software and the USB I/O boards and send that information to the attacker for offline analysis. This is achieved by (i) downloading and installing a malicious shared library on the RAVEN control system, (ii) forcing processes on the system to link to the malicious shared library, and (iii) logging the RAVEN USB communication and forwarding it to the attacker on a remote server using UDP packets.

In a Linux system most programs do not communicate directly with the kernel. Instead, the program invokes a function in a runtime library (e.g., *libc*), which performs the necessary preparation of the arguments and then triggers the corresponding system call (see Figure 4 for an example of calling the *write* system call in the RAVEN control software). When a program



```
int usb_write(int id, void *buffer, size_t len)
{
    // write to board
    int ret = write(boardFPs[id], buffer, len);

    if (ret < 0)
      ret = -errno;
    return ret;
}
```

```
ssize_t write(int fd, void *buf, size_t nbyte)
{
    if (!orig_write) {
        orig_write = (write_t)dlsym(RTLD_NEXT,"write");
    }
    /* Check if in Target Process*/
    if (target_process()) {
        get_target_fd();
        if (g_target_fd == fd) {
            perform_attack(buf, nbyte);
        }
    }
    return orig_write(fd, buf, nbyte);
}
```

```
ssize_t
__libc_write(int fd, const void *buf, size_t nbytes)
{
  if (nbytes == 0)
    return 0;
  ...
  ...
```

Figure 4. The malicious write system call loaded as a wrapper around the original *write* system call on the system. The dashed line shows the original program flow. The solid lines show the program flow after *LD_PRELOAD* is set to point to the malicious wrapper.

starts, the runtime linker searches the default path to find the runtime library to be linked. If an environment variable *LD_PRELOAD* or the directory */etc/ld.so.preload* is defined in the system, then the linking process is forced to first search, load, and link to the library object in the path pointed by the *LD_PRELOAD* or */etc/ld.so.preload* [22]. If the alternative library object has a function with the same name as function defined in the original runtime library (e.g., *read* or *write*), the alternative library's function will be called. This allows the alternative library to "wrap" the runtime library function, intercepting system calls. The alternative library function can call the original system call, not call it, or do some malicious task before calling it. This approach has been used by several rootkits to hide their operations [23].

In implementing the attack scenario B, we exploited the Linux dynamic linking feature to install malicious system call wrappers for the *write* system call in order to eavesdrop on the commands sent to the robot motor controllers and the safety PLC through USB. An attacker with the user privilege, can add the *LD_PRELOAD* environment variable to user's startup profile (e.g., *.bashrc*), so all future terminals started by this user will have the *LD_PRELOAD* environment variable set to point to the malicious shared library. The attacker with root privilege can add the path to the malicious shared library to */etc/ld.so.preload*, so that new processes started by any user on the system link to the malicious shared library. This means that when any future process makes a *write* system call, the system call wrapper in the malicious shared library will be called (see the malicious wrapper code in Figure 4).

2) Offline Analysis Phase: The goal of the Analysis Phase is to discover state information of the surgical robot from the logged USB communication. From the publicly available documents on the RAVEN II robot (e.g., [12][16]), the attacker can infer that the state information (the robot can be in one of four states depicted by the operational state machine; see Figure 1(c)) must be transmitted between the RAVEN control software and the USB I/O boards. The attacker performs an offline analysis on the USB packets (step 4 in Figure 3) collected from several robot runs—from initialization to the end of a teleoperation session—to identify fields in the USB packets that carry robot's state information.

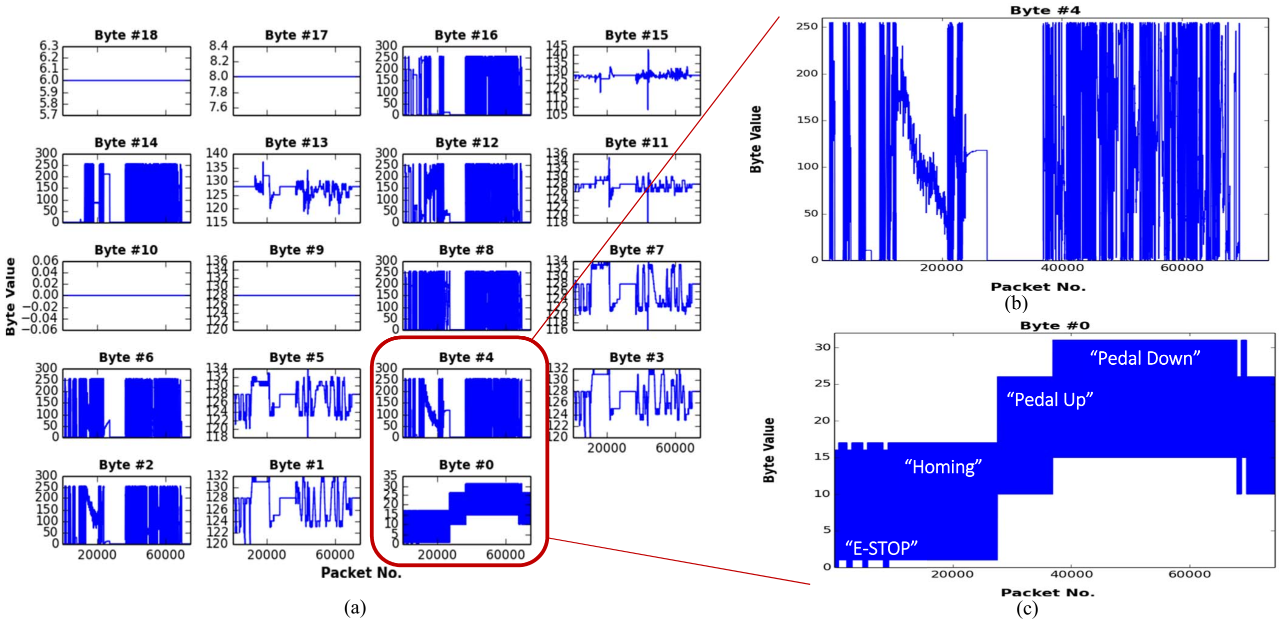
Since the attacker does not know the format of the USB packets, a simple approach to analyze them is to look at the values of the packets byte by byte over time to see whether there are patterns indicating a specific byte that may contain the state information. Figure 5(a) illustrates sample USB packets (values of the *buf* parameter for the *write* system call) collected in one run of the robot. Each subplot shows the value of each of the 18 bytes over the course of a run. During this run, the RAVEN robot was teleoperated using the manipulators on a remote console.

By analyzing multiple runs, attacker can discover that *Byte 0* switches among 8 different values in a surgical run whereas other bytes either stay constant or switch between many values. For example, Figure 5(b) and Figure 5(c) show the enlarged plot of *Byte 4* and *Byte 0*, respectively. A more detailed look at the values of *Byte 0* reveals that the fifth bit toggles periodically between 0 and 1 (e.g., 0X0F toggles to 0X1F). If we take that bit out, then *Byte 0* only switches among 4 values. Figure 6 shows the patterns of *Byte 0* over nine different runs of the robot. Our further investigation into the RAVEN II specifications revealed that the fifth bit of *Byte 0* might be the watchdog signal, a square-wave signal toggling periodically between 0 and 1 to communicate the healthy status of the robot control software to the PLC safety processor [16].

Now, the attacker can combine this information with the knowledge that the RAVEN robot state machine navigates through 4 distinct states during a teleoperation. It begins from a stopped state ("E-Stop"), then upon hitting the start button, it performs an initialization process ("Init"), then moves to a standby state ("Pedal Up"), and during the surgical procedure, moves between the standby state ("Pedal Up"), and the operational state ("Pedal Down") (see Figure 1(b)). Putting these two pieces of information together, the attacker can conclude, based on several runs of collected data, that *Byte 0* most likely represents the state of the surgical robot and the values 31 (0x1F) or 15 (0x0F) in *Byte 0* indicate that the robot is engaged and in operation (in the "Pedal Down" state). The red dashed lines in each subplot of Figure 6, highlight steps corresponding to the different operational states of the robot that can be inferred from this data. Similar analysis can be done on the data collected from the *read* system calls to eavesdrop on the feedback received from motor encoders (not shown here due to space limitations).

3) Attack Deployment Phase: The goal of the Deployment Phase is to install a malicious code that triggers an attack on the RAVEN II robot when it is engaged in the middle of a surgical operation. Based on the offline analysis, the attacker can use *Byte 0* as a trigger to determine when to activate an attack on the robot. There can be other triggers in addition to *Byte 0*, but *Byte 0* can indicate when the surgical robot is in the operational ("Pedal Down") state. Attacking the robot in other states may not have the desired malicious effect, e.g., in the "E-STOP" or "Pedal Up" states, the robot is not engaged and the motor brakes are applied, so no commands sent to the motors will be executed.

The attacker modifies the *write* system call wrapper in the malicious shared library to perform an attack when *Byte 0* (in the USB packets) indicates the "Pedal Down" state in the robot's operation. The attack consists of modifying the values



(a)

(b)

(c)

Figure 5. The contents of packets transferred in one run of the RAVEN II robot from the robot to one of the USB boards (by calling write systems call). (a) Each subplot corresponds to a byte in the USB packets. (b) *Byte #4* switches between many different values. (c) *Byte #0* switches between 8 different values and if the fifth bit is taken out, it switches between 4 values corresponding to the four distinct states of the robot.

of other bytes in the USB packets, that represent the control commands sent to the USB I/O boards by the control software to drive the motors on the robotic arms.

Previous assessment of the RAVEN control software by fuzzing the USB packets transferred between the robotic software and USB I/O boards revealed that the motor commands issued by the control software are checked before being sent to the custom USB boards (to make sure they do not exceed safety limits and the desired joint positions are not outside of the robot workspace). However, the integrity of the packets is not checked after the USB boards receive them. Since the USB I/O boards do not verify the integrity of the received USB data, a corrupted or incorrect motor command can pass to the motors causing the robot arm to move to an undesired location and potentially damage the system or harm the patient.

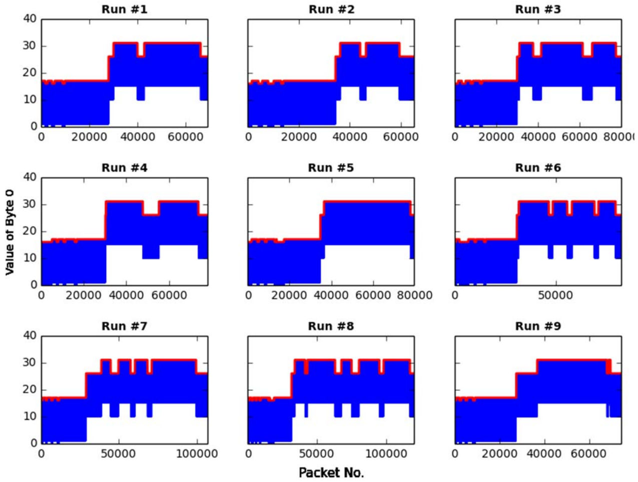


Figure 6. The values of *Byte 0* in the packets transferred from the robot to one of the USB boards in a sample of nine different runs. The robot state (highlighted in red) can be inferred from the changes in the value of *Byte0*.

Figure 4 shows the modified version of the wrapper. The attacker can deploy the modified shared library to any RAVEN machine using steps 1 and 2 in the Attack Preparation Phase (see Figure 3). Now, with every invocation of the *write* system call made by the RAVEN control software, instead of logging the USB communication, the malicious wrapper checks *Byte 0* of the *buf* parameter and automatically triggers an attack if *Byte 0* indicates that the robot is in the "Pedal Down" state.

C. *Attack Evaluation*

To asses the impact of the attacks on the progress of the surgery and the health of patient, we simulated the attacks on the *write* system calls in a surgical simulator for RAVEN II robot as well as on a real RAVEN II robot. By implementing the attacks on the simulator, we were able to verify the impact of the attacks before testing them on the actual robot, which prevents causing damage to the robotic arms and instruments.

1) Impact on the Physical System: The corruption of packets sent by the control software to the USB I/O boards was achieved using malicious wrapper around the *write* system call to inject a random value (e.g., between 0 and 100) to one of the bytes (other than *Byte 0*). This corruption caused abrupt jumps of the robotic arms, leading both the RAVEN II software and hardware to go into the "E-STOP" state. In a few cases, the abrupt jump of robotic arms, caused the breaking of the cables on the robot. The visualization of this scenario in the simulator and on the actual robot is available in [19] and [24].

This disturbance of the robot operation may lead to an interruption in the surgery, damage to the robotic instruments due to collision, or harm to the patient in the form of tearing or perforation of tissues if the instruments were inside the body. If the malicious wrapper is loaded by setting the *LD_PRELOAD* in the *bashrc* file of the target user, the malware will be reloaded to the system on each run of the robot even after restarting the system. Consequently, the "E-STOP" condition would happen on every invocation of the system call and practically make the robot unavailable to the surgical team.

As discussed in [25], in several safety incidents reported to the U.S. Food and Drug Administration (FDA), unexpected movement of robotic instruments due to accidental mechanical or electrical malfunctions or unintentional human errors (not malicious attacks) led to tearing or perforation of patient tissues, bleeding, and minor or severe injuries. Our results show that similar adverse incidents can be caused by malicious tampering with the robotic system and potentially harm patients or interrupt the surgical procedure without being identified as malicious activity.

2) Impact on the Cyber Domain: We also measured the performance overhead of the malicious system call wrappers on the normal operation of the robot and other processes running in the system. Table II shows the performance overhead of the malicious wrappers, measured by the execution time of the *write* system call wrapper in the RAVEN control process. We collected measurements before and after installing the malicious library wrapper in 50,000 runs of the system call.

TABLE II. PERFORMANCE OVERHEAD OF MALICIOUS SYSTEM CALL

	Time (µs)	Min	Max	Mean	Std.
RAVEN Process	**Baseline System Call**	0.9	12.7	1.3	0.2
	With Malicious Wrapper **Logging**	7.9	38.1	20.0	7.5
	Injection	1.5	6.7	3.6	1.1

The average execution time of the baseline *write* system call in the RAVEN process was around 1.3 microseconds. The malicious wrapper for logging the USB packets sent by the control software (including checking the process name and the file descriptor and sending the UDP packets to the remote attacker) on average added 18.7 microseconds to the execution time of the *write* system call in the RAVEN process. The malicious wrapper that injected the malicious bytes to the USB packets (including checking for the process name and file descriptor, checking the packet contents to determine if the desired robot state is reached, and overwriting the malicious value) added about 2.3 microseconds to the baseline *write* system call execution time. These overheads are within the timing constraints (1 millisecond) of the real-time process running the robot control software. So the malicious wrapper does not have any adverse impact on the performance of robot control and its effect would not be noticed by the human operators or users of the system.

D. Why this Attack is not Easy to Detect?

In the presented attack scenarios two important vulnerabilities allow the attacker to identify the critical time during robot operation and inject the malicious commands: (i) Linux dynamic loading feature for shared libraries and (ii) leaking of robot state information from the packets transferred between the robot control software and the USB I/O boards.

Malicious shared library attacks or dll hijacking attacks have been around. However, the security community has not paid much attention to this type of intrusions, because to be successful, such attacks require access to the file system on the target machine or a remote shell access. Several recent reports on attacks to safety-critical cyber-physical systems show the existence of many vulnerabilities that allow remote malicious access. Table V shows the entry points and vulnerabilities exploited by the recent real attacks on the hospital networks and commonly used medical devices. Table III shows examples of recent zero-day vuluneralbities in different operating systems allowing remote code execution, which could be used to download and setup the right scenarios for malicious shared library attacks.

The malicious shared library attacks presented here cannot be easily detected in the cyber-domain by the existing malware detection techniques, because:

1. Malicious actions are confined to the robot control software:
   a) no separate processes are created to run the malware.
   b) no system-wide malicious activities are performed
   c) the performance of target application is not affected
2. No changes are made to the control flow of the target process. The functions in the shared library are invoked by the process following its normal execution flow.
3. No anomaly in the syntax of robot control commands are introduced.

Furthermore, the surgical robot puts rather stringent real-time constrains on the system operation (e.g., in RAVEN II the operational cycle is 1 millisecond). The robot control loop plus any real-time detection and mitigation actions must complete within 1 millisecond to avoid potential deviation in system dynamics, leading to robot damage or patient harm. Traditional malware detection techniques (e.g., signature- or anomaly-based and control flow checking), encryption mechanisms (e.g., "bump-in-the-wire" (BITW) solutions [31][32]), and remote software attestation [33][34] may introduce significant overhead in the system operation and still not eliminate the possibility of TOCTOU exploits. In order to address this challenge, we develop dynamic model-based detection and mitigation mechanisms as discussed next.

TABLE III. RECENT ZERO-DAY VULNERABILITIES ALLOWING REMOTE CODE EXECUTION OR PRIVILEGE ESCALATION

Date [Ref]	CVE	Vulnerability	Affected Systems	Impact
Jul. 2015 [26]	CVE-2015-5123	Flash Player	Linux, Windows, OS X	Gain administrator shell on target machine
Jan. 2015 [27]	CVE-2015-0235 (GHOST)	Glibc	Linux	Remote code execution
Oct. 2014 [28]	CVE-2014-4113	Privilege Escalation	Windows	Escalate to SYSTEM Privilege
Sep. 2014 [29]	CVE-2014-6271 (Shellshock)	Bash shell	Linux, Unix, OS X	Remote code execution
Aug. 2015 [30]	CVE-2015-5783	OS X 10.10	Mac	Gain root access

IV. Dynamic Model-based Detection and Mitigation

In this section we describe the dynamic model-based analysis framework that we developed for (i) assessing the impact of attacks on the robot physical system and (ii) preemptive detection of the attacks and mitigating their impact before they manifest in the physical domain (see Figure 7). We validated the detection experimentally using two real attacks involving injection of unintended user inputs (scenario A) and unintended control motor torque commands (scenario B).

The dynamic model allows us to determine the subsequent state of robot end-effectors and the motors incrementally based on the information on the current state and the real time input received from the RAVEN control software. The methods for modeling the serial chain robot manipulators and RAVEN II robot dynamics are well understood in the literature and we briefly outline them for completeness. What is important here is to ensure that the output of the dynamic model closely follows the actual robot movements in real-time so that the detection is performed accurately.

To preemptively detect and mitigate the impact of attacks, the detection mechanisms need to dynamically estimate the consequence of executing a command on the physical system to ensure the final end-effector movements are within specified safety limits and within the workspace of the robot. There are two main challenges for implementing such monitoring mechanisms at lower layers of the control structure (e.g., at the interface device or the motor controllers):

1) The detector needs to estimate:
   a. Next motor (*mpos*) and joints positions (*jpos*) that will be achieved upon executing a given DAC command.
   b. End-effector positions (*pos*) and orientations (*ori*) that result from those commands in the next control loop.

If the estimated next joint position and end-effector position and ortientation values are beyond a safety limit (defined by a threshold value) from their current values, the DAC command should not be delivered to the motors and the robot should move to a emergency E-STOP state (see Figure 7(b)). Finding a solution to the above estimation problems requires modeling the dynamics of physical robot (motors and joint dynamics) for estimating the next motor and joint positions.

2) The robotic control systems often face tight real-time constraints. For example, the RAVEN II control loop has a real-time requirement of receiving and processing each packet from the USB boards and sending the next control command to the motor controllers every 1 millisecond.

Thus, any preemptive detection mechanism implemented at the software or software-physical interface layers should perform the dynamic state estimations within the real-time constraints imposed by the robot control design.

A. Framework Overview

Figure 7(a) shows the dynamic model-based simulation framework that we developed to assess the impact of the attacks on the physical system and validating the detection and mitigation mechanisms. The framework consists of:

- **A master console emulator** that mimics the teleoperation console functionality by generating user input packets based on previously collected trajectories of surgical movements made by a human operator and sends them to the RAVEN control software.

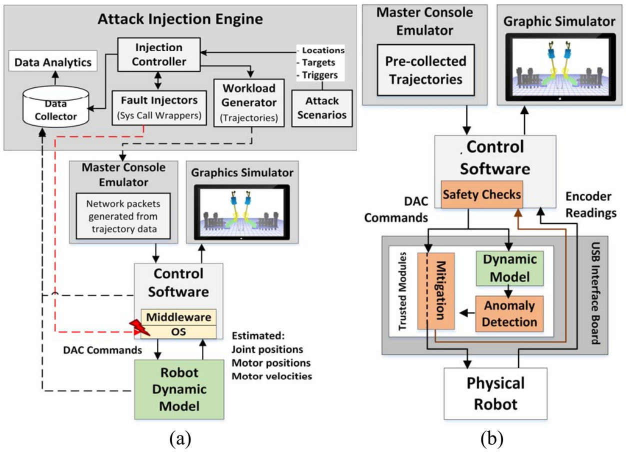


Figure 7. (a) Simulation framework for assessment of the impact of attacks. (b) Dynamic-model based detection and mitigation mechanisms.

- **A graphic simulator** that animates the robot movements in real time by listening to the ROS topic generating the robot state and mapping robotic arms and instruments movements to CAD models of robot mechanical components in a 3D virtual environment.
- **A dynamic model** of the RAVEN II physical system, which integrates the motor dynamics and robot manipulator dynamics together to model the physical system behavior in real time.
- **An attack injection engine** which can create attack scenarios targeting different layers of robot control structure by injecting faults into the robot control software modules.

1) Dynamic Model: We simulated the functionality of RAVEN II surgical robot by developing a software module that mimics the dynamical behavior of the robotic actuators. This is done by modeling the MAXON RE40 and RE30 DC motors used by the robot [12] as well as the robot manipulators (joints).

As shown in Figure 7, this model is integrated with the RAVEN control software and can run with or without the physical robot. At each cycle of software control loop (shown in Figure 2) the model receives the same control commands (DAC values) sent to the physical robot (calculated based on the desired joint and motor positions for the next time step) and estimates the next motor and joint positions.

The challenge in developing the model is to be able to perform estimations within the time constrains of the robot's single iteration through the control loop. To reduce computational cost while maintaining the model accuracy as well as the system real time guarantees, we model the robot manipulator dynamics using the first three (out of seven) degrees of freedom only (two rotational joints plus one translational joint). This is reasonable because the first three joints are positioning joints which contribute most to the instruments' end-effectors' positions, while the other four degrees of freedom are instrument joints, mainly affecting the orientation of the end-effectors. The model estimates the next

Integration Method (Step Size:1 ms)	Avg. Time/ Step (ms)	Joint 1 Avg. Error (% deg.)		Joint 2 Avg. Error (% deg.)		Joint 3 Avg. Error (%)	
		mpos	jpos	mpos	jpos	mpos (deg)	jpos (mm)
4-th Order Runge Kutta	0.032	115.0 (2.4)	0.9 (2.4)	178.1 (1.5)	1.8 (2.0)	181.9 (0.3)	1.4 (0.4)
Euler	0.011	136.6 (2.4)	1.0 (2.4)	132.8 (1.4)	1.4 (1.9)	180.6 (0.3)	1.3 (0.3)

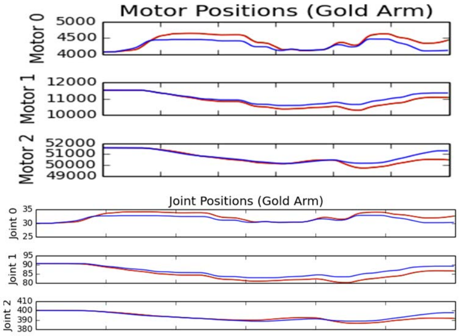


Figure 8. Validation of dynamic model: Trajectories generated by the dynamic model and the robot and average estimation error

states of the first three motors and the corresponding joint states, including shoulder joint (rotational), elbow joint (rotational), tool insertion/retraction (translational) on one arm.

Two sets of second-order ordinary differential equations were used to describe the dynamic model of the robot, including link (joint) and motor dynamics, similar to [35]. The robot mechanical properties, such as link mass, inertia, and center of mass location were obtained from the CAD models of the joints. The coefficients of these models were obtained via manual tuning based on [35], so that the dynamic model trajectory and the real robot trajectory are close.

The 4th order Runge-Kutta and explicit Euler methods were used for calculating the solutions for these equations using the numerical integration solver (odeint) package in C++. We validated this dynamic model by comparing the operational trajectory of the RAVEN II robot with the corresponding trace generated by the dynamic model. Specifically, we measured the performance of the dynamic model in terms of the average estimation error and the required time for performing the estimation at each robot control cycle. Figure 8 shows the average run time and average motor and joint position errors for the 4-th order Runge Kutta and Euler solvers, by calculating the average of mean absolute errors estimated for each trajectory, over 10 different runs of model and robot together. As shown in the table, for the specific trajectories experimented here, the Euler technique with a step size of 1 millisecond provides us with the best trade-off between execution time and average trajectory error. The average execution time of 0.011 milliseconds is within the timing constraint of 1 millisecond of the RAVEN control loop, which enables running of the model in parallel with the robot control software.

Figure 8 also shows trajectories of the first three joints and motors when running the model (blue plot) in parallel with the physical system (red plot) and both receiving the same control input, calculated based on the encoder feedback from the real robot. As we see in the figure the model closely follows the trajectory of the actual robot.

2) Attack Injection Engine: The core of the attack injection engine is a software implemented fault-injection tool that can be programmed to install wrappers around different system calls in the control software to create the attack scenarios shown in Table I. The attack injector can inject malicious inputs/commands with different values and activation periods to the control software at different times during a running trajectory (e.g., a surgical operation).

B. Assessing the Impact of Attacks

We used the dynamic model-based simulation framework in Figure 7(a) for assessing the impact of attack scenarios A and B by injecting a variety of unintended user inputs (malicious desired end-effector positions) and malicious motor torque commands to RAVEN control software. The simulation framework enabled us to assess the resiliency of the robot by performing thousands of injections without causing damages to the real robot. Representative fault injection experiments were repeated on the actual robot to validate the consistency between the robot and model behavior. We made the following observations by simulating these attack scenarios:

1. Malicious torque commands that inject small errors to the DAC values do not have any impact on the robot state, unless they are activated for periods of larger than 64 milliseconds. If injected for shorter periods (e.g., 2-4 milliseconds), they can cause abrupt jumps in the motor velocities but the impact do not propagate to the next control loop and do not impact motor, joint, and end-effector positions, unless larger values are injected for longer periods. This is due to the fact that the PID controller inside the control corrects the errors in motor velocity and motor positions at each cycle of the control loop. Therefore, to corrupt the physical state of the robot, the attacker needs to keep injecting malicious values to the commands over a long enough period of time.
2. The existing safety checks in RAVEN cannot detect the abrupt jumps resulted from malicious torque commands (injected after the software safety checks are done) until the physical system state is corrupted to a point where the PID control cannot fix the errors anymore. This is because:
   (i) these safety mechanisms only check the DAC commands calculated in software being sent to the robot by comparing it to a fixed threshold. They do not take into account the semantics of the control commands and their consequences in the physical system, i.e., impact of a DAC command on the state of the robot, motor positions and velocities, joint positions, and end-effector positions.

   (ii) the safety check are done at the latest computation step in the control software before sending the commands to physical system. Therefore, there is a TOCTOU gap, from the time the commands are checked to the time they are executed on the physical system, allowing attackers to target the system.

C. Anomaly Detection and Attack Mitigation

In order to preemptively detect the adverse impact of the attacks on the physical robot, we integrate the dynamic model-based analysis framework with the robot control system, to estimate the consequences of control commands before they are sent to the motor controllers and are executed on the physical robot (Figure 7(b)). Our goal is to detect if a given command will cause an unsafe jump of more than 1 millimeter on the robot end-effector position within a short period of 1-2 milliseconds (based on feedback from expert surgeons).

We design an anomaly detection mechanism that intercepts the DAC commands sent by the RAVEN control software and estimates the values for the next motor velocities and positions and joint positions using the robot dynamic model in real time. The detector raises an alert whenever the estimated instant velocity and acceleration on the first three motors and joints (the difference between the estimated values for the next step and current values) are beyond a pre-defined safety threshold (defined as 1 millimeter jump on end-effectors). The thresholds used for detecting anomalies are learned through measuring the maximum instant velocities of each of the variables over 600 fault-free runs of the model with two different trajectories containing sufficient variability in the movement. To eliminate the sensitivity of sample statistics to outliers and possible noise in measurements, we chose values between the 99.8–99.9th percentiles of instant velocity as the threshold for each variable. In order to reduce false alarms due to model inaccuracies and natural noise in the trajectory, the detector fuses the alarms based on the motor acceleration, motor velocity, and joint velocity and raises an alert only when all three variables indicate an abnormality.

Table IV shows the performance of dynamic-model based anomaly detection mechanism compared to the existing detection and emergency stop (E-STOP) mechanisms in the RAVEN II robot in terms of detection accuracy (ACC), true positive rate (TPR), false positive rate (FPR), and F1-score (which is a unified measure of precision and recall). The results were achieved from 1,925 experiments simulating the attack scenario A and 1,361 simulation runs of the attack scenario B. Figure 9 shows the impact of attack activation period and injected error values in scenario B on the probability of adverse impact on the robot physical system (abrupt jumps of end-effector positions) and probability of attack detection and mitigation by the dynamic-model based detection and the robot safety mechanisms. Each attack scenario with specific distance error and activation period was repeated for at least 20 times to achieve confidence in the probability estimates. The conditional probability of attacks given each injected error value *v* and activation period *d* was estimated by calculating marginal conditional probabilities from the measured data.

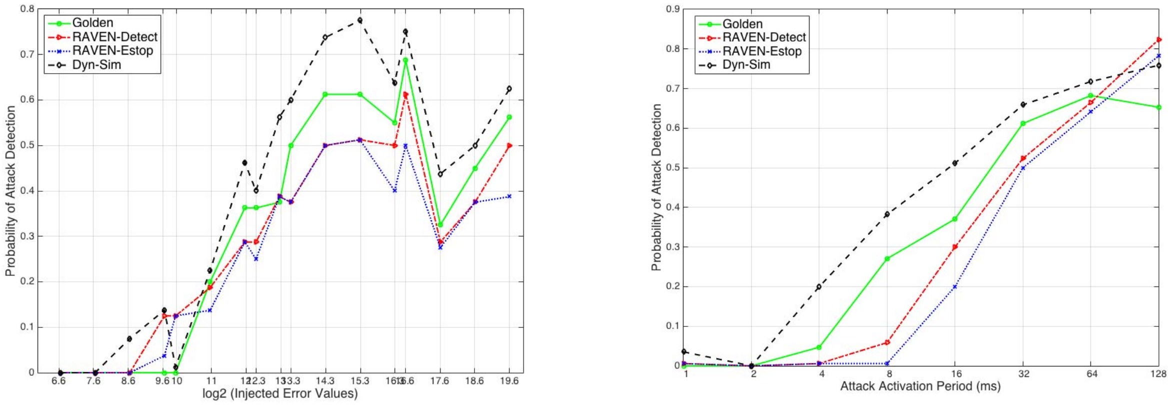
Table IV. Dynamic-model based detection performance evaluation, compared to RAVEN detector

Attack Scenario	Technique	ACC	TPR	FPR	F1
A (User inputs)	Dynamic Model	88.0	89.8	12.4	74.8
	RAVEN	84.6	53.3	7.7	57.8
B (Torque commands)	Dynamic Model	92.0	99.8	11.8	89.1
	RAVEN	90.7	81.0	4.6	85.1

Figure 9 shows that by injecting larger error values and increasing the activation period the probability of adverse impact on the physical system increases. Our dynamic-model based anomaly detection has higher probability of *preemptively* detecting the attacks before their impact manifests in the physical system than the RAVEN safety checks that only detect the impact after it has *already happened*. As shown in Table IV, the dynamic-model based detector could detect the simulated attacks scenarios with an averaged accuracy of 90% and average F1-score of 82%. For attack scenarios A and B, there were respectively 152 and 84 cases where the dynamic–model detected an abrupt jump on end-effectors while the RAVEN checks did not detect them. There were a total of 13 true cases (in scenario A) that our detector missed but RAVEN detected.

The probability of RAVEN safety mechanisms in detecting and mitigating the adverse impact is always lower than the probability of adverse impact, i.e., the RAVEN safety checks cannot detect all the adverse scenarios. Thus, the attacker has a chance of causing an adverse impact on the physical system by carefully engineering injections with values that will not be detected by the robot for even short periods of 2-16 milliseconds (Figure 9(b)). But this chance is reduced when injecting larger error values for longer periods (of more than 64 milliseconds).

Upon detection of potential adverse impact on the physical system, the impact of attacks can be mitigated by either correcting the malicious control command by forcing the robot to stay in a previously safe state or stopping the commands from execution and put the control software into a safe state (E-STOP). The ideal location for insertion of detection and mitigation mechanisms are at lower layers of control structure



(a)

(b)

Figure 9. Attack detection probability vs. injected error values and attack activation period

and just before the commands are going to be executed on the physical robot. This will decrease the probability of TOCTTOU exploits by requiring attackers to compromise the hardware controllers which are harder to access compared to control software. In the RAVEN II robot, the last computational component before the motor controllers is the microcontroller inside the USB interface board. The implementation of the methods for calculating a numerical solution for the ODEs of the dynamic model might incur high computational costs in simple hardware controllers (e.g., an 8-bit AVR microcontroller with 128KB flash memory in RAVEN). One possible solution is to implement the parallel version of these estimation techniques on a custom trusted hardware module and run them concurrently with the robot control system.

V. Related Work

A. Security of Teleoperated Surgical Robots

Previous work on security of telerobotic surgical systems mainly focused on network and communication-based attacks.

Bonaci et al. performed an experimental analysis of different cyber-security attacks on the communication between the surgeon's console and the robot on a RAVEN II platform [8]. They evaluated the threats posed by attacks that modify or manipulate the intent of the surgeon or hijack control of the robot. They showed that causing the user input packets to be delayed or get lost in transit to the robot might lead to jerky motions of the robotic arms or difficulty in performing tasks by human operators. However, the modification of packet contents led the safety software to detect the over-current commands sent to the robot, stop the robot's electrical and mechanical components, and prevent harm to the patient.

Tozal et al. used an information coding approach to design a Secure and Statistically Reliable UDP (SSR-UDP) protocol that ensures confidentiality and reliability of telesurgical communications in wireless environments [5]. Lee et al. proposed Secure ITP, a security enhancement to the Interoperable Telesurgury Protocol (ITP), introducing Transport Layer Security (TLS) and Datagram TLS (DTLS) protocols for authenticating the teleoperation console and slave robot as well as the surgeon and patient [6].

Most of the previous studies assumed that compromising a surgeon's control console or the robot control system is less likely because physical access to the system is prohibited through strict monitoring [8]. Only Coble et al. studied the possibility of compromising the robot software in unattended environments, such as the battlefield. They proposed the remote verification of system software and configuration files before execution, using remote software attestation [34].

B. Attacks on the Hospital Networks

In this work, we assume that attackers exploit one of the existing vulnerabilities in the hospital networks as described in the previous work to get access to the telerobotic surgical systems, without being detected by regular security monitoring mechanisms, such as intrusion detection systems or remote software attestation techniques. Table V presents a summary of the recent reports on real attacks to hospital networks.

For example, TrapX Security Inc. recently discovered three targeted attacks on a hospital's network that passed through the protection of antivirus software, intrusion detection systems, and firewalls. In one case, the vulnerabilities in a blood gas analyzer was exploited to establish a backdoor to the whole hospital network, allowing the attackers to install a malware on the system and steal patient data records from the hospital. In another case, the attackers gained unauthorized access to a clinic workstation, by stealing credentials of an employee visiting a malicious website and installing a malware in that machine [10]. In another recent study on a wide range of medical devices in several hospitals, researchers from Essentia Health discovered that the internal firewalls used for protecting surgical robots from external connections might crash upon running a vulnerability scanner against them and enable unauthorized access to the robot [11]. In addition, there have been several recalls and adverse events reported to the FDA on random attacks on hospital networks in which malware or viruses infected medical devices such as imaging systems, causing interruptions in patient therapy [39][40].

VI. Conclusion

In this paper, we described the anatomy of targeted attacks against the control systems of teleoperated surgical robots. We demonstrated these attacks on the RAVEN II surgical robot and experimentally evaluated their impact on the operation of the robot control system and patients. Our results showed that the attacks can cause either sudden jumps of the robotic arms or

Table V. Potential Entry Points to Get onto a Hospital Network and Examples of Real Attacks Compromising Them

Attack Entry Points	Description	Examples of Real Attacks and Detected Vulnerabilities	Ref. (Year)
Third party networks	Hospital networks are often connected to third party laboratories, pharmacies, and vendor networks that, if compromised, can let data breaches or penetrations into the hospital networks as well.	Two medical centers and more than 3.9 million individuals were affected by a data breach through a third party portal/personal health record platform.	[36] (2015)
Computers used by physicians, nurses, or technicians	The computers used by physicians, nurses, and vendor support technicians for remote access to the hospital network, could be compromised through credential stealing, virus, and malware.	Email phishing attack compromised personal information of 3,300 patients.	[37] (2015)
Vulnerable office devices	Office devices such as network attached desktops, printers, faxes, scanners, and security cameras with default or weak passwords or vulnerable firmware could be an easy entry point.	Default username/passwords for the multi-function printers and security cameras could be used for access to other devices on the hospital.	[38] (2014)
Vulnerable or miss-configured firewalls, access points, gateways	Incorrect configurations in the Wifi access points or gateway machines could expose vulnerabilities or leak information, such as device ID or hospital network layout, to the public.	Incorrect configuration of a gateway computer leaked critical information that made it possible for attackers to locate vulnerable devices within the hospital's network.	[9] [11] (2014)
Vulnerable medical devices	Medical devices on hospital network may have default/weak passwords or unpatched software/firmware, which can be compromised.	Three real-world attacks were detected, where a blood gas analyzer, a PAC system, and an X-ray machine were hijacked to open backdoors in hospital networks.	[10] (2015)

unavailability of the system due to an unwanted transition to a halt state in the middle of surgery. We presented defense mechanisms that combine understanding of the semantics of both software and physical components to predict the adverse consequences of attacks within the real-time constraints of the control system. The mitigation and assessment methods presented here can be applied to safety and security validation of a wider range of safety-critical cyber-physical systems.

VII. Acknowledgements

This work was partially supported by the National Science Foundation under Award Numbers CNS 13-14891 and CNS 15-45069, and the National Security Agency under Grant Number H98230-14-C-0141.

References

[1] Intuitive Surgical Inc., “Annual Report 2013” Available: http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9MjIzOTk3fENoaWxkSUQ9LTF8VHlwZT0z&t=1.

[2] J. Rosen and B. Hannaford, “Doc at a distance,” *IEEE Spectrum,* vol. 43, no. 10, pp. 34-39, 2006.

[3] D. Halperin, et al., “Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses,” *IEEE Symposium on Security and Privacy*, 2008, pp. 129-142.

[4] C. Li, et al., “Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system,” *IEEE Conf. on. e-Health Networking Applications and Services (Healthcom)*, 2011, pp. 150-156.

[5] M. Tozal et al., “On secure and resilient telesurgery communications over unreliable networks,” *IEEE Conf. on Computer Communications Workshops,* 2011, pp. 714-719.

[6] G. S. Lee and B. Thuraisingham, “Cyberphysical systems security applied to telesurgical robotics,” *Computer Standards & Interfaces,* vol. 34, no. 1, pp. 225-229, 2012.

[7] T. Bonaci et al., “Experimental analysis of denial-of-service attacks on teleoperated robotic systems,” *the ACM/IEEE Sixth International Conf. on Cyber-Physical Systems*, 2015, pp. 11-20.

[8] T. Bonaci et al., “To make a robot secure: An experimental analysis of cyber security threats against teleoperated surgical robots,” *arXiv:1504.04339*, 2015.

[9] K. Zetter, “Hospital Networks Are Leaking Data, Leaving Critical Devices Vulnerable”, Wired Magazine, 2014 [Online]. Available: http://www.wired.com/2014/06/hospital-networks-leaking-data/.

[10] TrapX Security, Inc., “Anatomy of an Attack – MEDJACK,” 2015 [Online]. Available: http://deceive.trapx.com/AOAMEDJACK_210_Landing_Page.html.

[11] K. Zetter, “It’s Insanely Easy to Hack Hospital Equipment,” Wired Magazine, 2014 [Online]. Available: http://www.wired.com/2014/04/hospital-equipment-vulnerable/.

[12] B. Hannaford, et al., “Raven-II: An open platform for surgical robotics research,” *IEEE Trans. Biomed. Eng.*, vol. 60, no. 4, pp. 954-959, 2013.

[13] Intuitive Surgical Inc., “The da Vinci® Surgical System” Available: http://www.intuitivesurgical.com/products/davinci_surgical_system.

[14] G. Guthart and J. K. Salisbury Jr, "The Intuitive Telesurgery System: Overview and Application," *IEEE ICRA*, 2000.

[15] P. Kazanzidesf et al., “An open-source research kit for the da Vinci Surgical System,” in *IEEE ICRA*, 2014, pp. 6434-6439.

[16] M. J. Lum, et al., “The RAVEN: Design and validation of a telesurgery system,” *The International Journal of Robotics Research,* vol. 28, no. 9, pp. 1183-1197, 2009.

[17] M. Quigley et al., “ROS: An open-source Robot Operating System,” *ICRA Workshop on Open Source Software,* vol. 3, no. 3.2, p. 5, 2009.

[18] Biorobotics Lab, University of Washington, “RAVEN II Source Code,” [Online]. Available: http://astro.ee.washington.edu/raven2docs/.

[19] H. Alemzadeh et al., “A software framework for simulation of safety hazards in robotic surgical systems,” *SIGBED Review,* vol. 12, no. 4, 2015*, Special Issue on Medical Cyber Physical Systems Workshop*.

[20] H. Alemzadeh, et al., “Systems-theoretic safety assessment of robotic telesurgical systems,” *Computer Safety, Reliability, and Security*, LNCS, vol. 9337, pp. 213-227, Springer, 2015.

[21] N. Falliere, L. O. Murchu, E. Chien, “W32. stuxnet dossier,” Symantec Corp, White Paper 2011.

[22] M. Kerrisk, “LD.SO(8) - Linux Programmer's Manual”, May 2015 [Online]. Available: http://man7.org/linux/man-pages/man8/ld.so.8.html.

[23] Blackhat Academy, InfoSec Institute, “Jynx2 Sneak Peek & Analysis,” March 2012 [Online]. Available: http://resources.infosecinstitute.com/jynx2-sneak-peek-analysis/.

[24] Visualization of attack scenario B on the RAVEN II robot. Available: https://goo.gl/uzQ2kl.

[25] H. Alemzadeh, et al., “Adverse events in robotic surgery: A retrospective study of 14 years of FDA data,” to appear in *PLOS ONE Journal,* 2016 [Online]. Available: http://arxiv.org/abs/1507.03518.

[26] “CVE-2015-5123,” [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5123.

[27] “CVE-2015-0235,” [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-0235.

[28] “CVE-2014-4113,” [Online]. Available: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4113.

[29] “CVE-2014-6271,” [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271.

[30] “CVE-2015-5783,” [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5783.

[31] Schweitzer Engineering Laboratories Inc., “SEL-3021-1 Serial Encrypting Transceiver Data Sheet” [Online]. Available: https://www.selinc.com/WorkArea/DownloadAsset.aspx?id=2854.

[32] P. P. Tsang and S. W Smith, “YASIR: A low-latency, high-integrity security retrofit for legacy SCADA systems,” *The IFIP TC 23rd International Information Security Conf.*, 2008, pp. 445-459.

[33] C. Castelluccia, et al., “On the difficulty of software-based attestation of embedded devices,” *the 16th ACM Conf. on Computer and communications security*, pp. 400-409, 2009.

[34] K. Coble, et al., “Secure software attestation for military telesurgical robot systems,” *Military Communications Conf.,* pp. 965-970, 2010.

[35] M. Haghighipanah, et al., “Improving Position Precision of a Servo-Controlled Elastic Cable Driven Surgical Robot Using Unscent Kalman Filter,” *IROS*, 2015.

[36] D. Sears, “Healthcare Breach Shines Spotlight on Third Party Security Risks,” SecurityScorecard Insights & News, 2015 [Online]. Available: http://blog.securityscorecard.com/2015/06/17/healthcare-breach-shines-spotlight-on-third-party-security-risks/.

[37] E. Snell, “Phishing Attack Affects 3,300 Partners HealthCare Patients,” 2015 [Online]. Available: http://healthitsecurity.com/news/phishing-attack-affects-3300-partners-healthcare-patients.

[38] B. Filkins, “SANS-Norse Health Care Cyberthreat Report,” 2014 [Online]. Available: https://www.sans.org/reading-room/whitepapers/analyst/health-care-cyberthreat-report-widespread-compromises-detected-compliance-nightmare-horizon-34735.

[39] U.S. Food and Drug Administration, “Class 2 Device Recall iLab Ultrasound Imaging System, models 120INS and 240INS,” Available: http://www.accessdata.fda.gov/SCRIPTs/cdrh/cfdocs/cfres/res.cfm?id=73287

[40] U.S. Food and Drug Administration, “MAUDE Adverse Event Report: PHILIPS MEDICAL SYSTEMS XCELERA LLZ,” Available: http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfMAUDE/Detail.CFM?MDRFOI__ID=1568388.