Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Software Rejuvenation Under Persistent Attacks in Constrained Environments

Raffaele Romagnoli * **Paul Griffioen** * **Bruce H. Krogh** **
Bruno Sinopoli ***

* *Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA 15213 (e-mail: {rromagno|pgriffi1}@andrew.cmu.edu)*
** *Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA 15213 (e-mail: krogh@ece.cmu.edu)*
*** *Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO, USA 63130 (e-mail: bsinopoli@wustl.edu)*

Abstract: Software rejuvenation has been proposed to guarantee safety of cyber-physical systems (CPSs) against cyber-attacks. Recent work has demonstrated how this method can be applied to more general control problems such as tracking control. Despite this progress, there are still limitations in applying software rejuvenation to real situations where the presence of persistent attacks and physical environment constraints exist. In this paper we address these issues and propose a secure recovery algorithm that can be deployed not only for recovery against persistent attacks but also in situations where physical environment constraints do not allow the system to tolerate any attack. The effectiveness of the approach is illustrated with a simulation of a quadrotor landing on the ground during recovery from a persistent attack.

Keywords: software rejuvenation, cyber-physical systems, resilience, security

1. INTRODUCTION

Software rejuvenation is a typical procedure used in software engineering to prevent potential software failures within a system (Huang et al., 1995). This kind of approach has recently been proposed to guarantee safety of cyber-physical systems (CPSs) against cyber-attacks (Abdi et al., 2018b; Arroyo et al., 2019). The objective is to periodically reboot the computer system by replacing the possibly corrupted controller with a trusted copy of the control software before the effects of an attack can cause irreversible damage to the physical system. This approach is a prevention mechanism that makes the system resilient to attacks without implementing any attack detection algorithm.

The disadvantage of periodically refreshing the system is that it results in a reduction in the control performance. For example, system resets can only be used if the system is not tracking a trajectory but stations around a specific equilibrium point, such as a rotorcraft UAV hovering at a given position. A solution aimed at reducing this issue has been proposed in Romagnoli et al. (2019a). There are several advantages of this approach with respect to other recent existing methods in the literature (Abdi et al., 2018b,a). The first is that this is an off-line approach based on positively invariant sets and reach set analysis which are used to compute the period between two consecutive software resets such that the worst case attack cannot take the system out of a safety set. The main difference with the Simplex method (Bak et al., 2014) is that when an attack occurs, the information of the state can be compromised. The worst case attack is defined according to certain control constraints and actuator saturation limits. Another difference with respect to other methods is that this approach has been extended to a secure tracking control algorithm (Romagnoli et al., 2019b) which ensures that the system remains within the safety set while also allowing trajectory tracking, or liveness. If the liveness property is satisfied, the system is capable of reaching a desired equilibrium point while periodically rejuvenating.

In this paper, we want to improve the proposed method by making it be able to handle some critical aspects such as the presence of persistent attacks and environmental constraints. In previous work (Romagnoli et al., 2019b), liveness is only guaranteed when no persistent attack is acting on the CPS. However, safety is always guaranteed through a secure control mode that is activated after software refresh if an attack takes the system close to the boundary of the safety set. During secure control the system is driven back towards the current equilibrium point while remaining disconnected from the network, preventing any cyber-attacks from occurring. Despite the fact that the system remains safe during secure control, the lack of communication does not allow the system to update the equilibrium point so that a reference input can be tracked. As a result, it may be the case that the reference trajectory cannot be updated under a persistent attack, causing the liveness property to not be satisfied.

In such cases, a system operator may wish to change the mission and drive the system to a safer place. However, the system still requires communication while being driven to

a safer place, such as when a drone needs to communicate with the ground station. Consequently, this paper introduces a secure recovery algorithm which guarantees that the system can be driven to a safer place while still remaining vulnerable when connected to the network. A similar problem has been solved in Griffioen et al. (2019). Here the secure recovery algorithm reboots the system before the effects of a possible attack can affect or modify the control input. Safety and liveness for the secure recovery algorithm is guaranteed analyzing the dynamics of the system during software refresh which is less conservative than the reach set analysis approach.

In addition, previous work (Romagnoli et al., 2019a,b) has not considered the presence of physical environment constraints. When the system is close to environmental constraints, an attack has the potential to push the system outside of or against those constraints. This paper takes these physical environment constraints into account during both tracking control and during the secure recovery mode.

2. PROBLEM FORMULATION

We model the plant as a continuous time linear time invariant (LTI) system given by the state equation

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{1}$$

where $x \in \mathbb{R}^n$ represents the state, $u \in \mathbb{R}^p$ denotes the control input, and the system is centered about an equilibrium point of 0. We design a stabilizing controller

$$u(t) = -Kx(t), \tag{2}$$

so that the closed-loop dynamics are given by

$$\dot{x}(t) = A_f x(t), \tag{3}$$

where $A_f \triangleq A - BK$ is a Hurwitz matrix. Given a non-zero equilibrium point $x_j \in \mathbb{R}^n$, we can do a coordinate transformation and represent the closed-loop system as

$$\dot{x}(t) = A_f \left(x(t) - x_j\right), \tag{4}$$

which is asymptotically stable and converges to x_j. Since the system is asymptotically stable, we can find a symmetric positive definite matrix $P \succ 0$ such that $A_f^T P + PA_f \prec 0$. Given a scalar $\epsilon > 0$ we can define the following Lyapunov level set

$$\mathcal{E}_j(\epsilon) \triangleq \left\{x | (x - x_j)^T P (x - x_j) \leq \epsilon\right\}, \tag{5}$$

which is positively invariant for (4). Thus for any $x(0) \in \mathcal{E}_j(\epsilon)$, $x(t) \in \mathcal{E}_j(\epsilon)$ for all $t \geq 0$.

For all possible equilibrium points x_j we consider a set of constraints on the state space, which defines the region where the controller can stabilize the system

$$\mathcal{C}(x_j) = \left\{c_i^T (x - x_j) \leq 1,\ i = 1, \ldots, n_c\right\}, \tag{6}$$

where $c_i \in \mathbb{R}^n$ is the normal direction to the hyperplane representing the i^{th} constraint. This is the set of states where the controller can guarantee some fundamental properties of the controlled system. In general those constraints do not match with the physical constraints due to the environment,

$$E = \left\{e_i^T x \leq 1,\ i = 1, \ldots, n_e\right\}, \tag{7}$$

where $e_i \in \mathbb{R}^n$. We note that the origin of the state space is contained in E. Then we have the following two possible situations for each equilibrium point $x_j \in E$:

$$\text{a) } \mathcal{C}(x_j) \subseteq E, \quad \text{or} \quad \text{b) } \mathcal{C}(x_j) \not\subseteq E.$$

Condition a) suggests that around x_j the control system can operate safely without violating physical environment constraints, and condition b) implies that physical environment constraints may be violated. Note that the control constraints do not take the environmental constraints into account, so the state $x(t)$ may be inside $\mathcal{C}(x_j)$ while also lying outside E.

In the presence of attacks we want to guarantee safety of the system using software rejuvenation (Romagnoli et al., 2019a) where the controller is replaced with a trusted copy of it before an attack can take the system out of $\mathcal{C}(x_j)$, and the protection during reboot is guaranteed by disconnecting the system from the network. In this scenario if the system is in condition a), then we can implement the secure tracking control algorithm developed in Romagnoli et al. (2019b). In condition b), software rejuvenation cannot guarantee protection against physical environment constraints because an attack can drive the state of the system anywhere within $\mathcal{C}(x_j)$. The reasons we do not include physical environment constraints in $\mathcal{C}(x_j)$ are that to incorporate the environment constraints: (i) $\mathcal{C}(x_j)$ has to be computed on-line in protected mode, and this can be difficult to implement due to possible limitations of the system architecture with this kind of modality; and (ii) software rejuvenation cannot admit an admissible solution if the state of the system is too close to the physical environment constraints.

To solve this problem, we want to design a safety algorithm that implements software rejuvenation while considering the physical environment constraints and at the same time allowing the system to communicate through the network to receive admissible recovery equilibrium points. This algorithm can also be used to recover the system if it is under persistent attack, which is not addressed in Romagnoli et al. (2019b).

2.1 Attack Model

The CPS described by the controlled system (4) operates normally when connected to the network to provide and receive information depending on the mission to be accomplished. The equilibrium points x_j are transmitted over the network to the control system. When the system is connected to the network it is vulnerable to attacks. We assume that an attacker has knowledge of the system model, can read the state and control inputs when the system is connected to the network, and can arbitrarily modify the control inputs and the controller such that $u(t) \in \mathcal{U} \subset \mathbb{R}^p$, where $\mathcal{U}$ represents the control constraints (see for example Johnson et al. (2017)).

The system is vulnerable only if it is connected to the network. Suppose the system starts from an initial state where it is not corrupted and is not connected to the network. Once it connects to the network, it becomes vulnerable to attacks, but the potential attack only begins to take effect after $T_\Delta > 0$. Consequently, we can consider the system to be working properly for that period of time where the network connection is open. Implementing software rejuvenation and considering the system clean after each software refresh makes the above assumption plausible as introduced in Griffioen et al. (2019) and Lucia et al. (2016). The following assumptions are made on T_Δ:

A1) $T_\Delta > 0$ is long enough to allow the system to receive a new equilibrium point x_j.
A2) During T_Δ the communication is assumed to be secure.

2.2 Software Rejuvenation

Software rejuvenation is a mechanism of protection based on three operating modes of the system (Romagnoli et al., 2019a). *Mission Control* (MC): the system is connected to the network to perform tasks for which it has been designed. In this mode it is also vulnerable to cyber-attacks. *Software Refresh* (SR): the system is disconnected from the network, and the current possibly corrupted controller is replaced by a trusted copy of it. During this mode, the system is running in an open loop, and we assume that the control inputs are held constant at the last control input provided before refreshing the controller. *Secure Control* (SC): after software refresh the system may remain disconnected from the network while the trusted controller is used to recover the system to a specific region before connecting it to the network. The timing diagram that represents the switching between the several modes is shown in Fig. 1. T^j_{MC} is a design parameter representing the length of time the system can tolerate an attack before SR. T_{SR} is the time needed to refresh the controller. While the time periods T_{SR} and T^j_{MC} are fixed a priori, the time in SC mode T_{SC} is not fixed a priori, and the system can only switch back to mission control when the state of the system has been returned to the invariant region associated with mission control. Note that we assume that after SR, the state of the system is immediately available from sensor measurements to check if the system can switch back to mission control or if it needs to be recovered in secure control mode.

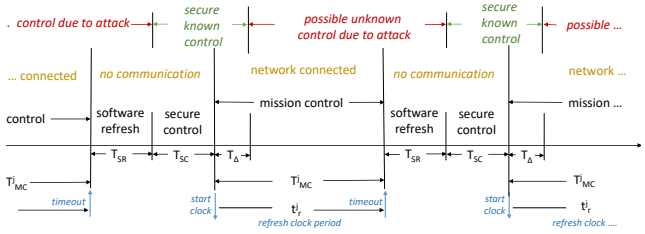

Fig. 1. Software rejuvenation control modes.

Remark 1. Ideally we want T^j_{MC} to be as large as possible because the network is used not only for control purposes but also for communicating other kinds of information required for particular tasks. Keeping the system disconnected from the network can guarantee safety, but it may drastically reduce system performance.

Remark 2. SC mode is a recovery procedure which is implemented by disconnecting the system from the network. This ensures safety during recovery against attacks, but it also prevents any possible interaction if an unexpected situation occurs. Despite the fact that T_{SC} is finite, it is a priori unknown, implying that once the system is in this modality it cannot promptly address unexpected events.

3. SOFTWARE REJUVENATION BASED SECURE TRACKING CONTROL

3.1 Invariant Safety Sets

Given an equilibrium point x_j, we define the safe set as the invariant ellipsoid (5) for system (4) centered at x_j given by

$$\mathcal{E}_j(\epsilon) \text{ s.t. } \mathcal{E}_j(\epsilon) \subseteq \mathcal{C}(x_j), \tag{8}$$

for some $\epsilon \in (0, 1]$. The positive definite matrix P is computed by finding the largest invariant ellipsoid $\mathcal{E}_j(1)$ such that $\mathcal{E}_j(1) \subseteq \mathcal{C}(x_j)$, and this is completed by solving a maximization problem as given in (Romagnoli et al., 2019a) and (Boyd et al., 1994).

Part of the results proposed in this section come from (Romagnoli et al., 2019a,b). However, we reformulate them in this new context by considering E and $T_\Delta > 0$, which is included as part of T^j_{MC} in Fig. 1. Given the time T_{MC} during which we want the system to be connected to the network, we define the time period of unknown control (UC) as

$$T_{UC} \triangleq T_{MC} - T_\Delta + T_{SR}, \tag{9}$$

which represents the time where the control input may be corrupted and unpredictable. We assume that $T_{MC} \geq T_\Delta$.

To ensure that the system can tolerate a worst case attack and still be safely recovered after software refresh, we define the invariant ellipsoid $\mathcal{E}_j(\epsilon')$, where $\epsilon' \in (0, 1]$. Letting $\mathcal{R}^+(x(t_0), T, CP)$ denote an over-approximation of the reachable set for all $t \in [t_0, t_0 + T]$ from initial state $x(t_0)$ under control policy CP, $\mathcal{E}_j(\epsilon')$ represents the largest ellipsoid such that

$$\forall x(t) \in \mathcal{E}_j(\epsilon'),\ \mathcal{R}^+(x(t), T_{UC}, UC) \subseteq \mathcal{E}_j(1). \tag{10}$$

This is associated with finding the largest value of ϵ' such that the reachable set over-approximation $\mathcal{R}^+(x(t), T_{UC}, UC)$ is a subset of $\mathcal{E}_j(1)$. One way to compute ϵ' is presented in Algorithm 1 where ϵ' is iteratively increased until the condition in (10) no longer holds.

Algorithm 1 Ellipsoid Computation Procedure

1: Initialize $\delta \in (0, 1)$
2: Initialize $\epsilon' \in (0, 1]$ s.t. $\mathcal{R}^+(x(t), T_{UC}, UC) \subseteq \mathcal{E}_j(1)\ \forall x(t) \in \mathcal{E}_j(\epsilon')$
3: **while** $\mathcal{R}^+(x(t), T_{UC}, UC) \subseteq \mathcal{E}_j(1)\ \forall x(t) \in \mathcal{E}_j(\epsilon')$
4: $\quad \epsilon' = \epsilon' + \delta$
5: **end while**
6: $\epsilon' = \epsilon' - \delta$

While ϵ' is computed in Algorithm 1 according to the period of uncertain control T_{UC} chosen by the system operator, it may be necessary to decrease T_{UC} in situations where $\mathcal{E}_j(1) \not\subseteq E$. In these cases,the system executes mission control for $T^j_{MC} \triangleq T^j_{UC} + T_\Delta - T_{SR}$ time instead of T_{MC} time. The value of T^j_{MC} that is transmitted over the network with x_j is obtained finding the maximum $T^j_{UC} \in [T_{SR}, T_{UC}]$ such that

$$\mathcal{R}^+(x(t), T^j_{UC}, UC) \subseteq \mathcal{E}_j(1) \cap E\ \forall x(t) \in \mathcal{E}_j(\epsilon'), \tag{11}$$

ensuring that the physical environment constraints are not violated.

Remark 3. The computation of T^j_{UC} using (11) can be implemented online by the operator transmitting the equi-

librium points. However, (11) can also be implemented off-line for a finite number of pre-determined pairs $\{x_j, T_{UC}^j\}$.

Recalling (5), we define the set E' as

$$E' \triangleq \{x_j | \mathcal{E}_j(\epsilon') \subseteq E\}. \quad (12)$$

When $x_j \in E'$ and $x(t) \in \mathcal{E}_j(\epsilon)$, the state will never violate the physical environment constraints E. Here we have let $\epsilon \in [\epsilon', 1]$ so that $\mathcal{E}_j(\epsilon)$ represents the smallest invariant set such that

$$\mathcal{E}_j(\epsilon) \supseteq \mathcal{R}^+(x(t_0), T_{UC}^j, UC) \; \forall x(t_0) \in \mathcal{E}_j(\epsilon'). \quad (13)$$

For the rest of the paper we consider ϵ satisfying condition (13).

We are going to design the secure tracking control algorithm based on software rejuvenation for when $x_j \in E'$. The next section will introduce the recovery mode that is activated when $x_j \notin E'$ or when the system is under persistent attack.

Let the invariant ellipsoid $\mathcal{E}_j(\epsilon'')$ represent the set of states from which mission control can drive the state into $\mathcal{E}_j(\epsilon')$ during T_Δ, where $\epsilon'' \in [\epsilon', 1]$. In other words,

$$\mathcal{E}_j(\epsilon'') = \left\{ x(t) \middle| x(t) = \left(e^{A_f T_\Delta}\right)^{-1} x(t_0),\; x(t_0) \in \mathcal{E}_j(\epsilon') \right\}. \quad (14)$$

This ellipsoid will be used to determine when the system can reconnect to the network and switch from secure control to mission control. There is no fixed time for secure control as it is implemented until $x(t) \in \mathcal{E}_j(\epsilon'')$, but it is possible to prove that T_{SC} is finite.

3.2 Tracking

The notion of safety has to be extended to the case where multiple equilibrium points are provided for reaching a particular position. Assuming that for each x_j the corresponding $\mathcal{E}_j(\epsilon) \subseteq \mathcal{C}(x_j)$, the goal is to ensure safety during the transition between two consecutive equilibrium points. Given $\epsilon_w \in (0, \epsilon']$, we introduce the ellipsoid $\mathcal{E}_j(\epsilon_w)$ which represents the set of states where the equilibrium point can be updated. If $x(t) \in \mathcal{E}_j(\epsilon_w)$ during the first T_Δ period of mission control, then the equilibrium point can be updated to x_{j+1}. Safety is guaranteed if the following condition holds for all the equilibrium points

$$\mathcal{E}_j(\epsilon_w) \subseteq \mathcal{E}_{j+1}(\epsilon'). \quad (15)$$

In the tracking control problem, the software rejuvenation scheme of protection remains the same other than the fact that $\mathcal{E}_j(\epsilon_w)$ is introduced for switching to the next equilibrium point.

Another goal is ensuring that the state of the system does not get stuck around a particular equilibrium point unless it is the final one. This property is named *liveness*. To guarantee this property, $\mathcal{E}_j(\epsilon_w)$ has to be a set of attraction (Brockman and Corless, 1998) for the system subject to software rejuvenation. For more formal details see Romagnoli et al. (2019b). Algorithm 2 presents the software rejuvenation procedure for the tracking control problem.

The main difference with respect to employing software rejuvenation without tracking is that during the first T_Δ period of mission control, the algorithm checks to see if $x(t) \in \mathcal{E}_j(\epsilon_w)$, and if so updates the equilibrium point

Algorithm 2 Software Rejuvenation Algorithm

```
1: while 1
2:     t = 0 (initialize and begin timer)
3:     while t < T_MC^j
4:         Mission Control
5:         if x(t) ∈ E_j(ε_w) and t < T_Δ
6:             Update Equilibrium Point (j = j + 1)
7:         end if
8:     end while
9:     Software Refresh
10:    if x(t) ∉ E_j(ε'')
11:        Close Network Connection
12:    end if
13:    while x(t) ∉ E_j(ε'')
14:        Secure Control
15:    end while
16:    Open Network Connection
17: end while
```

$x_j \to x_{j+1}$. Note that from (15), the transmitted value x_{j+1} can be verified by checking the conditions $x(t) \in \mathcal{E}_{j+1}(\epsilon')$ and $\mathcal{E}_{j+1}(\epsilon') \subseteq E$. If these conditions do not hold, the update is discarded and the system is still safe around the previous equilibrium point. The attacker can drive the system to any arbitrary place if the compromised equilibrium points meet these conditions, but safety is always guaranteed. The issue of the recovery from an attack on the communication channel is still an open problem that will be addressed in future research.

In this scenario if the system is under persistent attack, it can be the case that the equilibrium point is never updated because $x(t)$ never lies in $\mathcal{E}_j(\epsilon_w)$. This can occur if the state is not driven into $\mathcal{E}_j(\epsilon_w)$ during the first T_Δ period of mission control and if the attacker drives the state of the system out of $\mathcal{E}_j(\epsilon'')$ during every instance of mission control. Forcing liveness using secure control may drastically reduce the overall performance of the system since it is isolated. Guaranteeing a minimum level of communication makes it possible to also have remote control of the mission during safety-critical tasks such as recovery.

4. SOFTWARE REJUVENATION BASED SECURE RECOVERY MODE

The previous section considered the tracking control algorithm for equilibrium points $x_j \in E'$. We now consider the situation where $x_j \notin E'$ so that a possible attack can push the system out of the physical environment constraints E. In this case the effect of an attack cannot be tolerated, but the system still needs to be able to communicate to guarantee some fundamental functionalities (e.g. equilibrium points communication).

In order to avoid violating the physical environment constraints from a possible attack and at the same time ensuring the system remains connected to the network, we implement the software rejuvenation scheme using $T_{MC}^j = T_\Delta$. In this case the safe set has to take into account the physical environment constraints. To do so without changing the shape of the invariant set, we define the safety set $\mathcal{E}_j(\epsilon_r)$ according to (5) with $\epsilon_r \in (0, \epsilon']$ and with the previously computed positive definite matrix P which ensures that $\mathcal{E}_j(\epsilon_r) \subseteq \mathcal{C}(x_j)$.

Definition 4. Considering any fixed equilibrium point x_j, the set $\mathcal{E}_j(\epsilon_r)$ is a *recovery safe set* for the system subject

to periodic software refresh with $T_{MC}^j = T_\Delta$ if

$$x(t) \in \mathcal{E}_j(\epsilon_r)\ \forall t \in [t_0, t_0 + T_\Delta + T_{SR}], \tag{16}$$

where t_0 represents the time at the beginning of mission control. The recovery safe set is a positively invariant set.

Using the physical environment constraints and the safety set for an equilibrium point x_j, we define the set of recoverable equilibrium points as

$$E^r \triangleq \{x_j | \mathcal{E}_j(\epsilon_r) \subseteq E\}. \tag{17}$$

During the recovery mode, equilibrium points will only be transmitted over the network that lie inside of E^r. In other words, while it may be the case that $x_j \notin E'$, the system will never be provided with equilibrium points $x_j \notin E' \cup E^r$. During the recovery mode, ϵ_r can be increased or decreased over time to either enlarge or shrink the recovery safe set depending on how close the equilibrium point x_j is to the physical environment constraints E. For different values of ϵ_r, different sets E^r can be computed off-line. We now define safety for the recovery mode.

Definition 5. (Safety). Given a finite sequence of equilibrium points $x_j \in E^r$ with $j = 1, \ldots, s$, the system is safe if

$$x(t) \in \bigcup_{j=1}^{s} \mathcal{E}_j(\epsilon_r)\ \forall t \geq 0. \tag{18}$$

The system is safe if (18) is true because each ellipsoid $\mathcal{E}_j(\epsilon_r)$ lies within the control constraints $\mathcal{C}(x_j)$ and the physical environmental constraints E.

For the recovery mode, we redefine ϵ_w so that $\epsilon_w \in (0, \epsilon_r]$, and liveness is defined as follows.

Definition 6. (Liveness). The system is *live* if there exists a sequence of times $t_1, \ldots, t_s$ where $0 \leq t_1 < \cdots < t_s < \infty$ such that

$$x(t_j) \in \mathcal{E}_j(\epsilon_r),\ j = 1, \ldots, s. \tag{19}$$

The proposed recovery mode is described in Algorithm 3.

Algorithm 3 Software Rejuvenation Recovery Algorithm

```
Open Network Connection
while j < s
    t = 0 (initialize and begin timer)
    while t < T_Δ
        Mission Control
        if x(t) ∈ E_j(ε_w)
            Update Equilibrium Point (j = j + 1)
        end if
    end while
    Software Refresh
end while
```

Theorem 7. The recovery procedure presented in Algorithm 3 guarantees safety and liveness if

i $\mathcal{E}_j(\epsilon_r)$ is a recovery safe set $\forall j = 1, \ldots, s$.
ii $x(t_1) \in \mathcal{E}_1(\epsilon_r)$, where t_1 is the time when the recovery algorithm is initiated.
iii $\mathcal{E}_j(\epsilon_w) \subseteq \mathcal{E}_{j+1}(\epsilon_r)\ \forall j = 1, \ldots, s-1$.
iv For all $x(t_0) \in \mathcal{E}_j(\epsilon_r) \setminus \mathcal{E}_j(\epsilon_w)$, $j = 1, \ldots, s-1$, there exists an $\epsilon'_w \in [\epsilon_w, \epsilon_r)$ such that $x(t_0 + T_\Delta + T_{SR}) \in \mathcal{E}_j(\epsilon'_w)$, where t_0 represents the time immediately following software refresh.

Proof.

(1) Safety: Condition i) implies that $\mathcal{E}_j(\epsilon_r)$ is a positively invariant set, so if $x(\tau) \in \mathcal{E}_j(\epsilon_r)$ for a fixed equilibrium point x_j, then $x(t) \in \mathcal{E}_j(\epsilon_r)\ \forall t \geq \tau$. Condition iii) ensures that $x(t) \in \mathcal{E}_j(\epsilon_r) \cap \mathcal{E}_{j+1}(\epsilon_r)$ when updating the equilibrium point from x_j to x_{j+1}. These two conditions in conjunction imply that if $x(\tau) \in \mathcal{E}_j(\epsilon_r)$ with equilibrium point x_j, then $\forall t \geq \tau$, $x(t) \in \mathcal{E}_i(\epsilon_r)$ with equilibrium point $x_i\ \forall i \geq j$. This result combined with condition ii) implies that $x(t) \in \mathcal{E}_j(\epsilon_r)\ \forall t \geq 0$, $j = 1, \ldots, s$, satisfying (18) and guaranteeing the safety of the system.
(2) Liveness: Because conditions i), ii), and iii) imply that (18) is satisfied and because the equilibrium point is only updated when $x(t) \in \mathcal{E}_j(\epsilon_w)$, liveness is guaranteed if $x(t) \to \mathcal{E}_j(\epsilon_w)$ in a finite amount of time $\forall j = 1, \ldots, s-1$. Condition iv) ensures that this is the case by stating that $\mathcal{E}_j(\epsilon_w)$ is a region of attraction for $x(t)$. It states that after each software refresh, $x(t)$ will reside in an ellipsoid that is smaller than the ellipsoid it resided in after the previous software refresh until $x(t) \in \mathcal{E}_j(\epsilon_w)$.

Let t_i represent the time at the beginning of the i^{th} instance of mission control so that

$$\begin{aligned} x(t_i + T_\Delta) - x_j &= e^{A_f T_\Delta}\left(x(t_i) - x_j\right) \\ &= A_1(T_\Delta)\left(x(t_i) - x_j\right), \end{aligned} \tag{20}$$

$$\begin{aligned} x(t_{i+1} + T_\Delta + t) - x_j &= e^{At}\left(x(t_i + T_\Delta) - x_j\right) \\ &\quad + \int_0^t e^{A(t-\tau)} B d\tau \left(-K\left(x(t_i + T_\Delta) - x_j\right)\right) \\ &= A_2(t)\left(x(t_i + T_\Delta) - x_j\right) \quad \forall t \in [0, T_{SR}], \end{aligned} \tag{21}$$

where $A_1(t) \triangleq e^{A_f t}$ and $A_2(t) \triangleq e^{At} - \int_0^t e^{A(t-\tau)} BK d\tau$. We can now present the following proposition.

Proposition 8. Consider the dynamics of the closed-loop system (4) subject to the recovery mode behavior around a fixed equilibrium point x_j. Conditions i) and iv) from Theorem 7 hold if $\forall t \in [0, T_{SR}]$,

$$A_1^T(T_\Delta) A_2^T(t) P A_2(t) A_1(T_\Delta) - P \prec 0. \tag{22}$$

Proof. Assume that (22) holds $\forall t \in [0, T_{SR}]$. Then

$$\begin{aligned} &(x(t_i) - x_j)^T A_1^T(T_\Delta) A_2^T(t) P A_2(t) A_1(T_\Delta)(x(t_i) - x_j) \\ &-(x(t_i) - x_j)^T P (x(t_i) - x_j) < 0\ \forall t \in [0, T_{SR}]. \end{aligned} \tag{23}$$

Substituting (20) into (23) yields

$$\begin{aligned} &\left(x(t_i + T_\Delta) - x_j\right)^T A_2^T(t) P A_2(t)\left(x(t_i + T_\Delta) - x_j\right) < \\ &< (x(t_i) - x_j)^T P\left(x(t_i) - x_j\right)\ \forall t \in [0, T_{SR}]. \end{aligned} \tag{24}$$

Substituting (21) into (24) yields

$$\begin{aligned} &\left(x(t_i + T_\Delta + t) - x_j\right)^T P\left(x(t_i + T_\Delta + t) - x_j\right) < \\ &< (x(t_i) - x_j)^T P\left(x(t_i) - x_j\right)\ \forall t \in [0, T_{SR}]. \end{aligned} \tag{25}$$

If $x(t_i) \in \mathcal{E}_j(\epsilon_r)$, then (25) implies that $x(t_i + T_\Delta + t) \in \mathcal{E}_j(\epsilon_r)\ \forall t \in [0, T_{SR}]$. Because the system is asymptotically stable in the time period $[t_i, t_i + T_\Delta]$ during mission control, $x(t_i + t) \in \mathcal{E}_j(\epsilon_r)\ \forall t \in [0, T_\Delta]$. Consequently, (16) is satisfied if $x(t_i) \in \mathcal{E}_j(\epsilon_r)$, implying that $\mathcal{E}_j(\epsilon_r)$ is a recovery safe set and that condition i) holds. Furthermore, the strict inequality in (25) implies that if $x(t_i) \in \mathcal{E}_j(\epsilon_r)$, then

$x(t_i + T_\Delta + T_{SR}) \in \mathcal{E}_j(\epsilon'_w)$ for some $\epsilon'_w < \epsilon_r$, satisfying condition iv).

Remark 9. This proposition holds for any initial conditions, and it only depends on T_Δ and T_{SR}. We do not have any degrees of freedom if (22) is not satisfied, but we can use secure control for a fixed amount of time before opening the network connection to drive $x(t_i + T_\Delta)$ closer to x_j. If this solution is adopted, it guarantees that the secure control time interval T_{SC} is known and fixed, in contrast to normal software rejuvenation where we only know that T_{SC} is finite.

5. SIMULATION

We consider the quadrotor system used in (Romagnoli et al., 2019b) to test the proposed algorithm. The dynamics of the quadrotor are nonlinear, but it is a common practice to use a linear model that describes the dynamics around an equilibrium point. When the vehicle is hovering at a certain position where the propellers provide the force needed to counteract the g-force in that particular position, the nonlinear equations can be approximated by a linear model (1) (Beard, 2008). The simulations are carried out using the jMAVSim simulator.

The mission objective is to achieve the position $x_p = 1$ m, $y_p = 1$ m, $z_p = 4$ m starting from $x_p = 1$ m, $y_p = 0$ m, $z_p = 2$ m, where the subscript p is used to indicate the spatial coordinates. We simulate the effects of a persistent attack which repeatedly takes over the control inputs with the goal of crashing the quadrotor into the ground. Then we activate the recovery algorithm for safe landing of the vehicle. There are four control inputs: thrust F, and three torques τ_ϕ, τ_θ, and τ_ψ, where 0 N $\leq F \leq 16$ N and -0.0033 Nm $\leq \tau_{\phi,\theta,\psi} \leq 0.0033$ Nm. The angles ϕ, θ, and ψ describe the attitude of the quadrotor and represent the pitch, roll, and yaw angles respectively. The state space x and physical environment E are defined as

$$x \triangleq \left[\, \dot{x}_p \;\; x_p \;\; \dot{y}_p \;\; y_p \;\; \dot{z}_p \;\; z_p \;\; \dot{\phi} \;\; \phi \;\; \dot{\theta} \;\; \theta \;\; \dot{\psi} \;\; \psi \,\right]^T, \tag{26}$$

$$E = \{x \in \mathbb{R}^n | z_p \geq 0\}, \tag{27}$$

where only the ground is considered as a physical environment constraint. The state space constraints are defined around the equilibrium point $x_j = 0$ and are given by

$$\mathcal{C}(0) \triangleq \left\{ x \in \mathbb{R}^n \left| \begin{array}{c} -1 \leq x_p, y_p \leq -1 \\ -2 \leq z_p, \dot{x}_p, \dot{y}_p \leq -2 \\ -5 \leq \dot{z}_p, \dot{\phi}, \dot{\theta}, \dot{\psi} \leq -5 \\ -\pi/4 \leq \phi, \theta, \psi \leq \pi/4 \end{array} \right. \right\}. \tag{28}$$

For the software rejuvenation based secure tracking control we find the maximum volume ellipsoid contained in (28) as in Romagnoli et al. (2019a). Applying Algorithm 1 allows us to obtain $\epsilon' = 0.01$ for a given $T_{UC} = 0.12$ s, a given software refresh period $T_{SR} = 0.05$ s, and a given $T_\Delta = 0.05$ s. With these values, the condition in Proposition 8 is satisfied, implying that we can define safe recovery sets. Moreover, these values also satisfy the liveness condition needed to properly implement Algorithm 2. For the tracking control mode we consider $\mathcal{E}_j(\epsilon_r)$ with $\epsilon_w = 0.005$. In order to update the equilibrium point during the recovery procedure in Algorithm 3, $\mathcal{E}_j(\epsilon_w) \subseteq \mathcal{E}_{j+1}(\epsilon_r)$. Fig. 2 shows that this is the case for the three different recovery safe sets used by the recovery algorithm in this simulation.

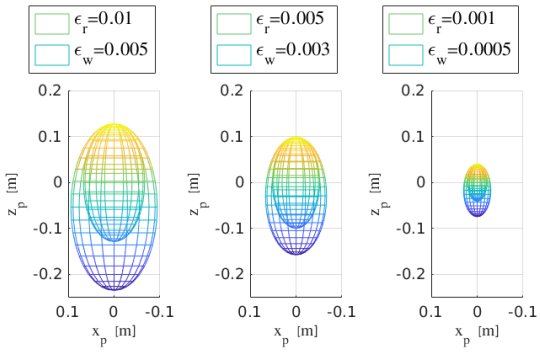

Fig. 2. The three different recovery safe sets (outer sets) $\mathcal{E}_{j+1}(\epsilon_r)$ used for landing. Each inner set $\mathcal{E}_j(\epsilon_w)$ is contained within the recovery safe set of the next equilibrium point.

The goal of the safe recovery algorithm is to guarantee a safe landing for the quadrotor, and therefore we want to take it as close as possible to the ground. To do so, we use three different safe recovery sets. One set is used where $\epsilon_r = 0.01$ and $\epsilon_w = 0.005$, the same as that used for tracking control, and this set is used for all the equilibrium points x_j where $z_p > 0.2$ m. If x_j is such that 0.11 m $< z_p \leq 0.2$ m, we shrink the recovery safe set by using $\epsilon_r = 0.005$ and $\epsilon_w = 0.003$. If x_j is such that $z_p \leq 0.11$ m, we again shrink the recovery safe set by using $\epsilon_r = 0.001$ and $\epsilon_w = 0.0005$. In this way the quadrotor can safely descend to 0.06 m above the ground where it is able to land using only the secure control mode.

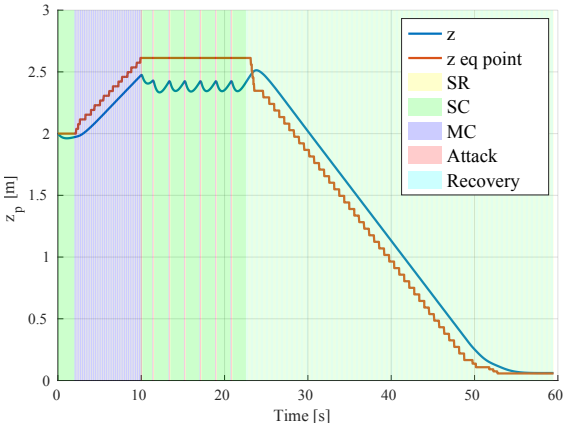

Fig. 3. Behavior of the system state and associated equilibrium points projected on z_p, and the timeline of the control modes.

As Fig. 3 depicts, the system is running in SC mode before the simulation begins, and the initial state of the system belongs to E'. After that the mission starts and the equilibrium points are updated since the liveness of the system is preserved subject to software rejuvenation. At $t = 10$ s a persistent attack is initiated, but safety is guaranteed since Algorithm 2 activates SC after SR. However, the equilibrium point cannot be updated under this persistent attack. Because the equilibrium point is unable to be updated, the secure recovery algorithm is activated at $t \approx 22$ s, and the quadrotor begins its descent until it reaches 0.06 m above the ground. The behavior of the Lyapunov function is reported in Fig. 4, where the spikes are due to updating the equilibrium points.

During the persistent attack, the state of the system is continuously taken out of $\mathcal{E}_j(\epsilon'')$ despite the fact that SC recovers it after each software refresh. When the system is under attack, the state does not leave $\mathcal{E}_j(\epsilon)$. During the recovery mode, the initial recovery safe set is $\mathcal{E}_j(0.01)$ for $22\text{ s} \leq t \leq 47\text{ s}$, $\mathcal{E}_j(0.005)$ for $47\text{ s} < t \leq 52\text{ s}$, and $\mathcal{E}_j(0.001)$ for $t > 52$ s.

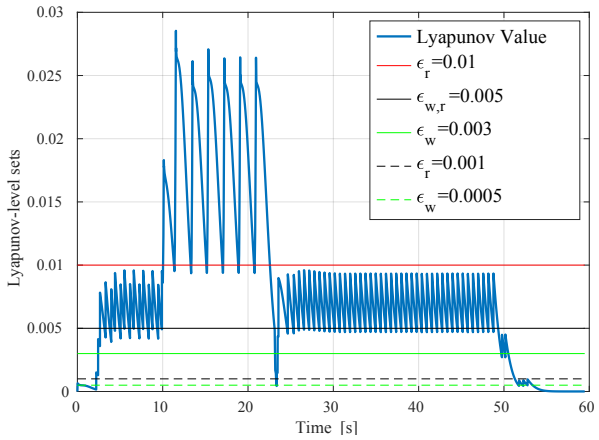

Fig. 4. Behavior of the Lyapunov function and representation of the different level sets for different values of ϵ_r and ϵ_w.

6. CONCLUSION

In this paper we have addressed the issues that can arise when applying software rejuvenation in the presence of persistent attacks and environmental constraints. By rebooting the system before the effects of a possible attack affect the control input, we have designed a secure recovery algorithm that guarantees safety and liveness of the system when it is vulnerable. This algorithm is also used for situations where the system is in the proximity of physical environment constraints so that the effects of any attack cannot be tolerated. The effectiveness of this approach has been illustrated with the simulation of a quadrotor under persistent attack where the secure recovery algorithm allows the quadrotor to land at a specific safe point. Future developments will be devoted to studying the case of attacks on the communication channel and on-board equilibrium point generation.

ACKNOWLEDGEMENTS

Copyright 2019 Carnegie Mellon University and Bruno Sinopoli. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. DM19-1178

REFERENCES

Abdi, F., Chen, C.Y., Hasan, M., Liu, S., Mohan, S., and Caccamo, M. (2018a). Guaranteed physical security with restart-based design for cyber-physical systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, 10–21. IEEE Press.

Abdi, F., Chen, C.Y., Hasan, M., Liu, S., Mohan, S., and Caccamo, M. (2018b). Preserving physical safety under cyber attacks. *IEEE Internet of Things Journal*.

Arroyo, M.A., Ziad, M.T.I., Kobayashi, H., Yang, J., and Sethumadhavan, S. (2019). Yolo: frequently resetting cyber-physical systems for security. In *Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2019*, volume 11009, 110090P. International Society for Optics and Photonics.

Bak, S., Johnson, T.T., Caccamo, M., and Sha, L. (2014). Real-time reachability for verified simplex design. In *2014 IEEE Real-Time Systems Symposium*, 138–148. IEEE.

Beard, R. (2008). Quadrotor dynamics and control, rev 0.1. Technical Report 2008-05-05, Brigham Young University.

Boyd, S., Ghaoui, L.E., Feron, E., and Balakrishnan, V. (1994). *Linear Matrix Inequalities in System and Control Theory*, volume 15. SIAM.

Brockman, M.L. and Corless, M. (1998). Quadratic boundedness of nominally linear systems. *International Journal of Control*, 71(6), 1105–1117.

Griffioen, P., Romagnoli, R., Krogh, B.H., and Sinopoli, B. (2019). Secure networked control via software rejuvenation. In *58th IEEE Conference on Decision and Control (CDC)*.

Huang, Y., Kintala, C., Kolettis, N., and Fulton, N. (1995). Software rejuvenation: Analysis, module and applications. In *Proceedings of 25th International Symposium on Fault Tolerant Computing*, 381–390. IEEE Computer Society, Pasadena, CA.

Johnson, B., Caban, D., Krotofil, M., Scali, D., Brubaker, N., and Glyer, C. (2017). Attackers deploy new ics attack framework "triton" and cause operational disruption to critical infrastructure.

Lucia, W., Sinopoli, B., and Franze, G. (2016). A set-theoretic approach for secure and resilient control of cyber-physical systems subject to false data injection attacks. In *2016 Science of Security for Cyber-Physical Systems Workshop (SOSCYPS)*, 1–5. IEEE.

Romagnoli, R., Krogh, B.H., and Sinopoli, B. (2019a). Design of software rejuvenation for cps security using invariant sets. In *2019 American Control Conference (ACC)*, 3740–3745.

Romagnoli, R., Krogh, B.H., and Sinopoli, B. (2019b). Safety and liveness of software rejuvenation for secure tracking control. In *2019 18th European Control Conference (ECC)*, 2215–2220.