YOLO: Frequently Resetting Cyber-Physical Systems for Security

Miguel A. Arroyo, M. Tarek Ibn Ziad, Hidenori Kobayashi,
Junfeng Yang, Simha Sethumadhavan

Columbia University, Computer Science Department, New York, NY, USA

ABSTRACT

A Cyber-Physical System (CPS) is defined by its unique interactions between digital (cyber) computation and physical motion. Their hybrid nature introduces new attack vectors, but also provides an opportunity to design new security defenses. In this paper, we present a new domain-specific security mechanism, YOLO, that leverages physical properties such as inertia to improve security.

YOLO is simple to describe. It goes through two operations: Reset and Diversify, as frequently as possible – typically in the order of a few seconds. Resets mitigate attacks that aim to achieve persistence and enhance the power of diversification techniques. Due to inertia, CPSs can remain safe even under frequent resets.

We introduce an analytical approach to evaluate the feasibility of a YOLO-ized system. Using this analytical model we define the constraints on reset periods in order to maintain the CPS's stability. We evaluate our approach in simulation and on two real systems: an engine control unit (ECU) of a car and a flight controller (FC) of a quadcopter. From our experiments, we determine that resets can be triggered frequently, as fast as every 125ms for the ECU and every second for the FC, without violating safety.

Keywords: cyber-physical systems, defense, diversification, inertia, resilience, reset, security

1. INTRODUCTION

A Cyber-Physical System (CPS) represents the synthesis of computational and physical processes encompassing a wide range of applications in transportation, robotics, manufacturing, and critical infrastructure. Current state of the art defenses for CPSs can be understood as defenses for the physical portion of the CPS, viz., the sensors, and cyber portion of the CPS, viz. the control and communication software. The physical defenses leverage physical properties to detect and mitigate sensor spoofing, sensor jamming, and other similar types of attacks [1, 2]. The cyber defenses are largely ports of techniques originally developed for traditional computing systems. As a result, they may not be as effective in the CPS domain.

In this paper, we propose a new security resiliency mechanism specifically for CPSs involving motion, that leverages their unique cyber and physical properties. A key innovation in our approach is that we take advantage of properties like *inertia*, i.e., the ability of the CPS to stay in motion or at rest, and its ability to tolerate transient imperfections in the physical world, to survive attacks. Our mechanism, You Only Live Once (YOLO), is best understood with an example. Consider an unmanned drone: even if engine power is cut off, it will continue to glide; similarly, even if a few sensor inputs are incorrect, the drone will continue to operate as the system is designed to handle intermittent sensor errors that can occur during normal operation. In YOLO, we take advantage of these features. We intentionally reset the system periodically to clear software state that may have been manipulated by an attacker. During resets, we rely on inertia to continue operating and diversification [3] to prevent the same attacks from breaking the system.

YOLO offers three major security benefits. First, the effects of resets, i.e, wiping memory and bringing software to its initial state, make it difficult for the attacker to persist on the system. Second, the periodicity of

Corresponding authors:
Miguel A. Arroyo: miguel@cs.columbia.edu
Simha Sethumadhavan: simha@cs.columbia.edu

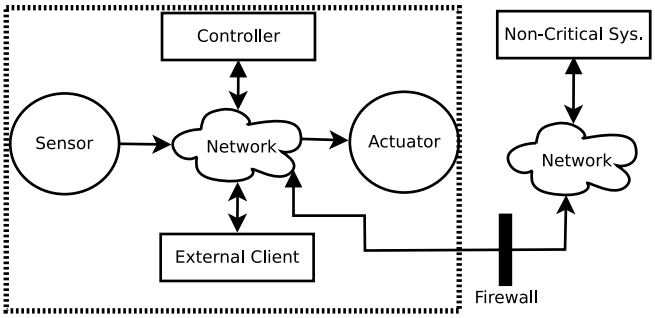


Figure 1: CPS System Model. The portion enclosed in the dotted area corresponds to the safety critical components.

resets enforces a limit as to how quickly an adversary has to carry out an attack. Finally, diversification is used to force an adversary to develop a new attack strategy after every reset.

Realizing YOLO is not without challenges. One key challenge is to determine how to safely apply resets while guaranteeing stability. To evaluate the feasibility of our approach, we define a theoretical model by which designers can evaluate YOLO on their CPS. We validate our methodology by implementing a simulation of a YOLO-ized DC Motor. Our results show that YOLO has a negligible performance effect while limiting the adversary's ability to attack the CPS.

Furthermore, we experimentally evaluate YOLO on two popular stateful CPSs with different inertia, safety requirements, and control complexity. Using an engine controller unit (ECU) and measurements from a real combustion engine, we discuss and measure the performance and safety impacts. Additionally, we also perform measurements on a quadcopter's flight controller (FC) that involves more sophisticated control. We find that YOLO-ized versions of these systems tolerate multiple frequent resets safely.

Closely related to YOLO are works by Abdi et al. [4] and Mertoguno et al. [5]. Mertoguno et al. [5] observe that unique physical properties of CPS can be used to improve their security. They devise a system that takes advantage of inherent redundancy available in CPS along with inertia and to design a Byzantine Fault Tolerant system. Concurrently with our work, Abdi et al. [4] propose a restart based approach to secure CPSs. In contrast to YOLO, their design requires additional hardware components for maintaining the safety of the system during resets. This may make it more difficult to retrofit into legacy systems, increases cost, and requires a larger trusted computing base (TCB). YOLO has no such requirement.

The rest of the paper is organized as follows: In § 2, we present an overview of CPS relevant to our work. We then describe our adversary model in § 3, present YOLO in § 4, provide a discussion on YOLO's security parameters in § 5 and provide a theoretical analysis in § 6. We next provide experimental evaluation on two representative case studies—an ECU and an FC—in § 7. We list our limitations in § 8, we survey related work in § 9, and conclude in § 10.

2. SYSTEM MODEL

In this section, we provide an introduction to cyber-physical systems in terms of their overall architecture and computing resources.

2.1 Architecture

Current CPSs have four main subsystems: a subsystem that interfaces with the physical world that includes the sensors and the actuators, a subsystem that takes inputs from the sensors and generates commands for the actuators—the control subsystem,—a subsystem that takes in external commands, and finally a subsystem the provides non-critical functions. In most CPSs today these subsystems are not isolated from each other creating a quagmire of security problems. However, recognizing these dangers, emerging CPS designs include primitives for isolation (such as separating out the networks in a car), sensor authentication and encryption to name a few.

We set our work in this emerging model and we consider the safety and security of only the software that provides control over the physical system (i.e. the safety critical components). The enclosed portion of Fig. 1 illustrates our model. The rationale for this choice is to focus our attention on aspects that are unique to CPS. A wide variety of systems fall into this model, such as unmanned transport vehicles or industrial control systems.

2.2 Resources

CPS resources can vary wildly. However, a large subset of these systems rely on microcontrollers. They tend to execute from ROM or Flash chips with limited memory (typically hundreds of KBs or a few MBs), have no memory management unit (MMU), only a memory protection unit (MPU); and the performance offered by these microcontroller-based systems is orders of magnitude lower compared to traditional microprocessor-based systems such as servers and smartphones. The primary driving factor for the system design choices is the cost of components, and a need to provide real-time deterministic execution characteristics which preempts use of speculative techniques for achieving high performance. As a result of these stringent resources and requirements, these systems usually run a Real-Time OS (RTOS).

Porting of existing security defenses that were not designed for these low-resource devices can be difficult or sometimes impossible. Similarly, these architectural or platform differences present challenges to mounting certain classes of attacks. For instance, modifying even a small amount of data, say one byte, can require an entire Flash sector to be modified, incurring latencies in the order of hundreds of milliseconds to seconds. So persistent attacks on microcontrollers can take several seconds while the same can be accomplished in hundreds of nanoseconds to microseconds on traditional systems – nearly seven orders of magnitude difference.

3. ADVERSARY MODEL

For this work, we consider an adversary that seeks to exploit the software running on the controller via a network input interface. The assumptions we make are:

- *An attacker has complete knowledge of the system internals.* The physics of the system and the control algorithms used are known to the attacker.
- *The attacker's goal is to sacrifice the integrity of the physical subsystem and bring it under their control.* The adversary's intention is to hijack the targeted system and/or prevent it from achieving its intended mission.
- *An attacker's proximity to the target is ephemeral.* An adversary may not always be within communication proximity; they may be limited by their equipment or other environmental factors.

We do not consider Denial-of-Service (DoS) attacks to be within scope as it goes against our assumption of having the adversary bring the system under their control. While we do not consider attacks on sensors and actuators as our defense targets software, an adversary may use them as entry points to exploit software.

3.1 Attack Surfaces

A CPS has three attack surfaces:

(A) *Sensors & Actuators:* The CPS's interface to the physical world is through sensors and actuators. This is susceptible to physical threats, such as tampering of sensors and/or actuators, sensor spoofing, and jamming.

(B) *Software:* The control software that handles incoming queries, processes sensor data, and computes actuator commands. This is susceptible to more traditional software threats, such as those stemming from memory vulnerabilities, integer overflows, etc.

(C) *Network:* The network that connects the various components to the controller. This is susceptible to threats, such as man-in-the-middle attacks.

Attacks on CPS can be broadly categorized into attacks on the physical or cyber subsystems.

3.1.1 Physical-subsystem Attacks

Physical attacks typically target state estimation and control; in fact, much work has been done on this front. Attacks on state estimation usually manifest themselves in one of two ways: sensor spoofing and sensor jamming.

Most, if not all of these physical attacks require a degree of spatial proximity: the jamming or spoofing device needs to keep up with the motion of the CPS for continued action. Examples of recent work in this area by Davidson et al. [1] describe how to spoof optical flow sensors in quadcopters.

It is important to note that while these attacks are physical in nature, they can affect the cyber component of the system and provide the opportunity for certain cyber attacks to become effective. For example, they may indirectly trigger certain vulnerabilities, such as integer overflows and underflows. Maliciously manipulated sensor values can cause incorrect branches to be executed in the control algorithms, or worse trigger CPU specific vulnerabilities.

3.1.2 Cyber-subsystem Attacks

Due to platform differences, attacks in the cyber portion of a CPS are a subset of those seen on more traditional systems. Memory vulnerabilities, such as code reuse and data corruption, are as much a problem for CPSs as for other systems. For example, Checkoway et al. provide an automotive specific discussion on some of these threats [2].

4. YOLO: You Only Live Once

YOLO is a secure resiliency mechanism tailored for CPSs. It combines two orthogonal, but complementary techniques: reset and diversification. This combination in conjunction with the unique properties of CPSs, plays off each other to provide stronger security than either technique on its own. We discuss the intuition behind YOLO below.

Why Reset? YOLO takes advantage of a simple and universally applicable panacea for a multitude of software issues, resetting. Even among expert users, a reset is one of the preferred solutions for numerous problems in the computing world. The simple intuition behind the effectiveness of this approach is that software is tested most often in its pristine state, as discussed by Oppenheimer et al. [6]. With respect to the overall health of the system, the conditions of a reset provide a predictable and well defined behavior.

From the viewpoint of thwarting an attacker, the restoration of state, whether it be code or data typically helps prevent an adversary's ability to corrupt the system. This provides increased resiliency against a large class of attacks. For example, a simple reset can remove the effects of exploits that live in volatile memory. By frequently performing resets, YOLO limits the effects an adversary might have, as well as, the time needed to complete an attack. In other words, an adversary has a bounded time horizon over which they can affect the system, simply because malicious effects are frequently removed.

Why Diversify? Typically, once a vulnerability is identified, an adversary can continuously carry out an attack as long as the vulnerability remains present and if they are able to keep up with the system. To remedy this, some variability must be introduced into the system; otherwise, the same vulnerability would persist. Diversification introduces randomness to prevent the system from being compromised by the same method continuously. The benefits of such an approach have been shown to be successful in a number of related works [3]. As a consequence of diversification, YOLO is able to lower the adversary's chance of success.

Why Reset & Diversify? Diversification can help protect data that needs to be carried across resets. Some CPSs may require certain data that cannot be re-learned during normal operation. For example, sensor calibration data of a quadcopter can only be obtained while it is not in flight. Therefore, it must be preserved across resets. One mechanism to protect such data is to take advantage of diversification. Diversification can be used to change the location of the data on every reset, making it harder for an adversary to locate it. Another mechanism is encryption. Alternatively, the diversification strategy may encrypt the data with a rotating key that is changed on every reset, making it more difficult for an adversary to manipulate data.

The benefits of diversification alone can in fact be less significant than one would expect [7]. Because these techniques rely on probability, there are cases in which they fail. Resets are used to add resiliency in situations where attacks can succeed.

Why does this work for CPS? CPSs and more specifically *dynamic* CPS — those involving motion — have a number of properties that allow for interesting security techniques to be explored. These properties stem from the design, algorithms, and physics of CPSs. The main properties that YOLO leverage are:

Inertia: CPSs have *inertia* — the resistance of an object to any change in its motion. This also means that these systems take time to react to external stimuli. This characteristic is essential for YOLO as it asserts that these systems can continue operation and exhibit a tolerance to missed events, either by design or due to faults.

Relearning: The fact that CPSs can observe their environment using sensors is fundamental in their design and operation. This implies that the state of the system that caused the actuation can be relearned by the sensors. Roughly speaking, this is the same as storing the state of the system (before actuation) to a data store (e.g. database) outside the system (the environment), and then reading it back to the system (through the sensors). This feature is critical for YOLO as it allows for certain state to be discarded during resets, i.e., transmitted out of the system before a reset, and re-observed once out of reset.

While these properties are unique to CPS, some of their advantages can usually be emulated in traditional computing systems, but at a cost. For example, traditional systems can rely on replication to emulate some of the benefits of inertia. This replication usually involves additional memory or hardware. To account for the lack of observability, traditional systems might require a secure data store or some external entity from which to recover their state. Restoring this state and ensuring consistency may be a more costly than what is typically involved in the CPS domain.

How easily is this realized? The YOLO mechanism does not require complex and expensive additional hardware. A system designer has flexibility in choosing how to trigger resets either via software or a cheap external clock. This makes the task of deploying YOLO on legacy systems much easier.

What YOLO parameters can be tuned? System designers can select appropriate reset & diversification strategies according to their constraints. At a minimum, a reset implementation should clear the running (volatile) memory of the program. Systems with more flexible reset timings should be able to tolerate more complete reset mechanisms, such as restoring code in non-volatile memory. Additionally, depending on the timing constraints of a system, different diversification techniques [3] may be applied.

5. YOLO SECURITY PARAMETERS

As mentioned in § 2, a large fraction of CPSs rely on software running on microcontrollers. Unlike their traditional computing counterparts, cyber-attacks on CPSs encompass not only the effects of software exploits, but additionally the physical reaction to an exploit. Determining the effectiveness of YOLO involves: (a) analyzing timeliness with respect to how long an attack takes and (b) the effects of a reset on an exploit depending on where in memory the exploit resides.

We focus on two main classes of vulnerabilities an adversary is likely to exploit: integer overflow/underflow, and memory safety. Empirical studies have determined these two vulnerabilities to be responsible for nearly 80% of the recent exploits [8].

5.1 Memory

Depending on the adversary's goals, exploits move between different types of memory. In addition to timeliness, the conditions provided by a reset affect exploits differently depending on where in memory they reside.

Volatile: Most exploits typically begin their lives in volatile memory. YOLO is extremely effective against those exploits. There are reasons for an adversary not to move to non-volatile memory, especially if they wish to leave as minimal a trace as possible. An ideal reset restores the entirety of memory. However, this may not be possible for all applications, especially those that need to persist state in volatile memory across reset boundaries. Depending on the complexity of the application the amount of persisted state may also vary in size. The smaller

this state, the more security benefits YOLO can provide. This persisted state can be protected with different encryption schemes depending on its size. In other words, small amounts of persisted state need not pose a significant threat.

Non-volatile: When the goals of an adversary are longer term, they typically must achieve persistence in some form of non-volatile memory. Threats such as *rootkits, spyware,* and *ransomware* all typically aim to make the transition to non-volatile memory. As previously mentioned, an ideal reset should restore the entirety of memory including the non-volatile portion. In some cases, limitations of the platform may make this impractical. For example, the limited lifetime of flash memory imposes restrictions on how often the system can be reprogrammed.

5.2 Timeliness

The timeliness of an attack is based on the time an exploit makes an observation of the system and the time at which an attack is completed. We use the framework developed by Evans et al. [7] to study the effectiveness of YOLO with respect to the time to attack. There are two cases to consider: (i) the time for the *exploit* and (ii) the time for the *attack*. Typically, these terms are used to refer to the same thing, but due to the nature of CPS we make an explicit distinction. The time for the *exploit* corresponds to the time needed by an adversary to achieve full control flow hijack of the software. The time for the *attack* is the time for the exploit in addition to the time for the physical system to react to an adversary's commands—their objective. Therefore, if the reset period for YOLO is less than the time needed for the exploit, the adversary is prevented from ever exploiting the system. Similarly, if the reset period is less than the time for an attack, the adversary is prevented from achieving their goal.

Exploit Time: We first examine the time required to exploit the software. Depending on the level of control a vulnerability affords the adversary, the time required for them to mount an exploit can vary. In the worst case, they may have access to a zero-time operation i.e, an integer vulnerability used to make a security decision which can trigger the malicious behavior instantly. In this situation, YOLO does not prevent the exploit, but will rely on the side-effects of resets to provide resiliency.

To leverage memory safety vulnerabilities on YOLO-ized like systems that use diversification, an adversary must do work proportional to their knowledge of the program layout and the strength of diversification. The strength of diversification affects how difficult it is to trigger the vulnerability and how hard it is to hijack control flow. Assuming defenses that prevent code injection are in place, adversarial strategies can be broadly categorized into: layout blind and layout aware.

Layout blind exploits use information such as crashes to aid in searching the address space of a program for useful gadgets to craft the exploit. Given that timeliness is important to YOLO's security, let us consider the derandomization exploit by Shacham et al. [9] as an example to get a sense of the timescales at which resets should occur at. The system being exploited in their evaluation was a 32-bit $1.8GHz$ Athlon machine. Their fastest recorded time was $29s$ with an average of around 3.6 minutes. Consider the same scenario on a typical microcontroller running at $200MHz$. As a rough estimate, if we simply scale these values by the processor frequencies, this results in a $9x$ slowdown turning a $29s$ exploit into one that takes over four minutes. As long as the YOLO-ized system can reset promptly these types of exploits can be prevented.

Layout aware exploits like Just-In-Time Code Reuse (JIT-ROP) [10] leverage pre-existing knowledge, such as the location of a code pointer, to disclose and harvest the run-time memory of the program to search for useful gadgets to craft the rest of the exploit. These exploits are difficult to practically compare given differences in code size between CPS applications and traditional applications. As proposed, published exploits leveraging memory disclosures similar to JIT-ROP target systems with a JIT engine, such as web browsers. This leaves an open question as to whether these exploits are threats to CPSs where JIT engines are avoided due to timing constraints.

Attack Time: Even if an attacker successfully exploits the system, YOLO relies on the assurances of the physical properties to resist the negative impact of an adversary until the next reset period. Given the types of attacks seen in practice, the adversary's goals usually involve compromising the physical plant. Inertia and other physical properties limit the rate of change of the system. Essentially, this means that attacks take longer as they not only involve the exploit, but the time an adversary requires to influence the system to meet their

goals. For example, Urbina et al. [11] discuss metrics and analyze how these physical properties are used by intrusion detection techniques to place bounds on the rate an adversary disturbs the system. The authors found that if the adversary's rate of attack can be made sufficiently slow, the physical properties will allow the system to resist the negative effects of the attack. In turn, this means the system still meets its original objective.

6. THEORETICAL ANALYSIS

In this section, we introduce an analytical approach to evaluate the feasibility of a YOLO-ized system. We define the constraints on reset periods in order to maintain the CPS's stability.

6.1 Problem Formulation

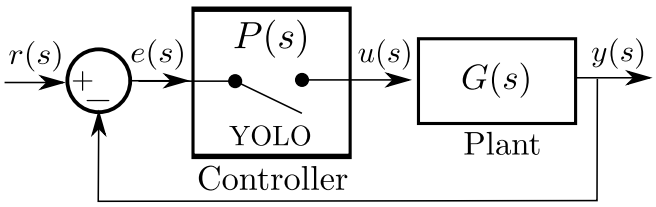


Figure 2: The closed loop model of a CPS. A physical plant with a transfer function $G(s)$, and the YOLO-ized controller, with a transfer function $P(s)$.

The framework shown in Fig. 2 consists of a closed-loop system, which includes a CPS and a YOLO-ized controller. The CPS has a transfer function $G(s)$, while the transfer function of the controller combined with YOLO is represented by $P(s)$. YOLO acts as an ON/OFF switch that resets the controller periodically. To formulate this switching behavior, we refer to the time interval between any two successive resets as T_r. T_r is composed of two main components as shown in (1).

$$T_r = T_u + T_d \tag{1}$$

where T_u is the controller up-time, in which YOLO is inactive, and T_d is the controller down-time, in which the controller is out of service due to the YOLO's effect.

Based on YOLO's state, the closed loop transfer function of Fig. 2 takes two different forms. The first form occurs when YOLO is inactive (i.e. the controller output is directly connected to the plant input, during T_u), as shown in (2). The second form occurs when YOLO is active (i.e, the controller is in the reset mode, during T_d), as shown in (3).

$$F_{inactive}(s) = \frac{y(s)}{r(s)} = \frac{P(s)G(s)}{1 + P(s)G(s)} \tag{2}$$

$$F_{active}(s) = \frac{y(s)}{u(s)} = G(s) \tag{3}$$

where $y(s)$ and $r(s)$ represent the closed loop system output and the input reference, respectively. $u(s)$ is the plant input signal. This system is viewed as a switched system [12], a hybrid dynamical system composed of a family of continuous-time subsystems with rules orchestrating the switching between the subsystems. The state space model of our switched system is given by (4).

$$\begin{cases} \dot{x} = A_i x + B_i r, \\ y = C_i x + D_i r \end{cases} \tag{4}$$

where $x \in \mathbb{R}^n$ is the state, $r \in \mathbb{R}^m$ is the reference input, and $y \in \mathbb{R}^p$ is the system output. The matrices A_i, B_i, C_i, and D_i are the system matrices, where $i \in \{1, 2\}$. By representing (2) and (3) using the general n^{th} order transfer function (5), we define the main matrices using the controllable canonical form.

$$F(s) = \frac{b_0 S^n + b_1 S^{n-1} + \cdots + b_{n-1} S + b_n}{S^n + a_1 S^{n-1} + \cdots + a_{n-1} S + a_n} \tag{5}$$

$$A_i = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix} \qquad B_i = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$$C_i = \begin{bmatrix} (b_n - a_n b_0) & (b_{n-1} - a_{n-1} b_0) & \cdots & (b_1 - a_1 b_0) \end{bmatrix} \qquad D_i = b_0$$

For consistency, we refer to the case, in which YOLO is inactive ($F_{inactive}$) and active (F_{active}), by the index $i = 1$ and $i = 2$, respectively.

6.2 Stability Analysis

To prove that the system is safe under YOLO's reset, the stability of the switched system shown in (4) should be guaranteed. There exist considerable amount of work in control theory literature, which investigate the stability of switched systems [13]. These works mainly depend on Lyapunov functions, a method used to prove the stability of dynamic systems without requiring the actual solution of the system's ordinary differential equations to be available [12].

Among the available stability analyses, we adopt the average "dwell time" approach proposed by Zhai et al. [14] that maintains the stability of a system containing Hurwitz stable (i.e., all eigenvalues lie in the left half complex plane) and unstable subsystems. This configuration maps to the *inactive* and *active* forms of YOLO. The "dwell time" concept is defined as the time between consecutive switchings [13] and represents the minimum boundary that should be respected to guarantee stability. The average "dwell time" allows some consecutive switching to violate the dwell time constraint as long as the total average switching time is no less than a specified constant, τ [13].

Zhai et al. [14] show that if τ is chosen sufficiently large and the total activation time of unstable subsystems (T_d) is relatively small compared with that of Hurwitz stable subsystems (T_u), then exponential stability of a desired degree is guaranteed. The stability conditions are shown in (6) and (7), respectively (see [14] for further details).

$$\tau \geq \frac{a}{\lambda^* - \lambda} \tag{6}$$

$$\frac{T_u}{T_d} \geq \frac{\lambda^+ + \lambda^*}{\lambda^- - \lambda^*} \tag{7}$$

where λ^- and λ^+ are positive scalars representing the minimum and maximum eigenvalues of the stable and unstable subsystems, respectively. For any given $\lambda \in (0, \lambda^-)$, λ^* is chosen arbitrary, where $\lambda^* \in (\lambda, \lambda^-)$. Finally, a is the maximum of a set of scalars a_i, which satisfy the following condition [14].

$$\begin{cases} \|e^{A_i t}\| \leq e^{a_i - \lambda_i t} & i \in \mathbb{S} \\ \|e^{A_i t}\| \leq e^{a_i + \lambda_i t} & i \in \mathbb{U} \end{cases} \tag{8}$$

where $\mathbb{S}$ represents the stable subsystems and $\mathbb{U}$ represents the unstable ones. In our scenario, switching times, T_u and T_d, are fixed. Thus, to maintain stability, $(T_u + T_d)/2$ should be greater than τ. By using the plant and

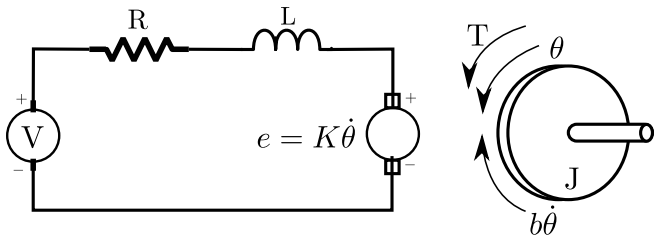


Figure 3: Simple abstraction of a DC motor. The voltage source, v; the rotational speed of the shaft, $\dot{\theta}$; the moment of inertia of the rotor, J; the motor viscous friction constant, b; the electromotive force & motor torque constants K; the electric resistance, R; and the electric inductance, L.

controller parameters, given in (2) and (3), to solve for τ and T_u/T_d, given in (6) and (7), with the help of the above boundary rule (8), a piecewise Lyapunov function ($V(x) = x^T P_i x$) [12] is constructed to ensure the global exponential stability of the system [14]. P_i are positive definite symmetric matrices, $P_i \in \mathbb{R}^n$, which are directly obtainable by solving the linear matrix inequalities (LMIs), given by (9).

$$\begin{cases} (A_i + \lambda_i I)^T P_i + P_i(A_i + \lambda_i I) < 0 & i \in \mathbb{S} \\ (A_i - \lambda_i I)^T P_i + P_i(A_i - \lambda_i I) < 0 & i \in \mathbb{U} \end{cases} \tag{9}$$

where I is the identity matrix. There exist many efficient methods for obtaining P_i numerically, such as the LMI solvers of *Matlab Robust Control Toolbox*. Next, we show that this analysis is easily extended to include any plant and controller.

6.3 Case Study

To evaluate YOLO, we have modeled a closed loop system where the plant, $G(s)$, is a DC motor and the controller, $P(s)$, is a Proportional-Integral-Derivative (PID) controller.

6.3.1 Problem Formulation

Fig. 3 gives a simple abstraction of the DC motor. The open-loop transfer function of the DC motor is given in (10), where the rotational speed, $\dot{\theta}$, is the output and the voltage, v, is the input [15].

$$G(s) = \frac{y(s)}{u(s)} = \frac{K}{(Js + b)(Ls + R) + K^2} \tag{10}$$

We further simplify the model by assuming that $L << R$ and hence obtain a first order system, as shown in (11).

$$G(s) = \frac{K_o}{T_o s + 1} \tag{11}$$

where K_o and T_o are calculated by using (12) and (13), respectively.

$$K_o = \frac{K}{R\,b + K^2} \tag{12}$$

$$T_o = \frac{R\,J}{R\,b + K^2} \tag{13}$$

The PID controller is a simple yet versatile feedback compensator structure [15]. Its transfer function is shown in (14).

$$P(s) = \frac{u(s)}{e(s)} = K_p + \frac{K_i}{s} + K_d s \tag{14}$$

where K_p, K_i, and K_d represent the proportional, the integral, and the derivative gains, respectively.

From (11) and (14), the closed loop transfer functions, $F_{inactive}(s)$ and $F_{active}(s)$, are developed, as shown in (15) and (16).

$$F_{inactive}(s) = \frac{K_o K_d s^2 + K_o K_p s + K_o K_i}{(T_o + K_o K_d)s^2 + (1 + K_o K_p)s + K_o K_i} \tag{15}$$

$$F_{active}(s) = \frac{K_o}{T_o s + 1} \tag{16}$$

By comparing the aforementioned equations with (2) and (3), and using $n = 2$, the values of the matrices A_i, B_i, C_i, and D_i are obtained as shown below. One extra zero-state has been added to $F_{active}(s)$ in order to make it a second order system as $F_{inactive}(s)$.

$$A_1 = \begin{bmatrix} 0 & 1 \\ \frac{-K_o K_i}{T_o + K_o K_d} & \frac{-(1+K_o K_p)}{T_o + K_o K_d} \end{bmatrix} \qquad B_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} \frac{K_o K_i T_o}{(T_o + K_o K_d)^2} & \frac{K_o (K_p T_o - K_d)}{(T_o + K_o K_d)^2} \end{bmatrix} \qquad D_1 = \frac{K_o K_d}{T_o + K_o K_d}$$

$$A_2 = \begin{bmatrix} \frac{-1}{T_o} & 0 \\ 0 & 0 \end{bmatrix} \qquad B_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad C_2 = \begin{bmatrix} \frac{K_o}{T_o} & 0 \end{bmatrix} \qquad D_2 = 0$$

6.3.2 Evaluation

We assume the following typical set of parameters for the DC motor: $J = 0.01$, $b = 0.1$, $K = 0.01$, $R = 1$. By using the following gains for the PID controller; $Kp = 100$, $Ki = 200$, $Kd = 10$, one obtains the following A_i matrices:

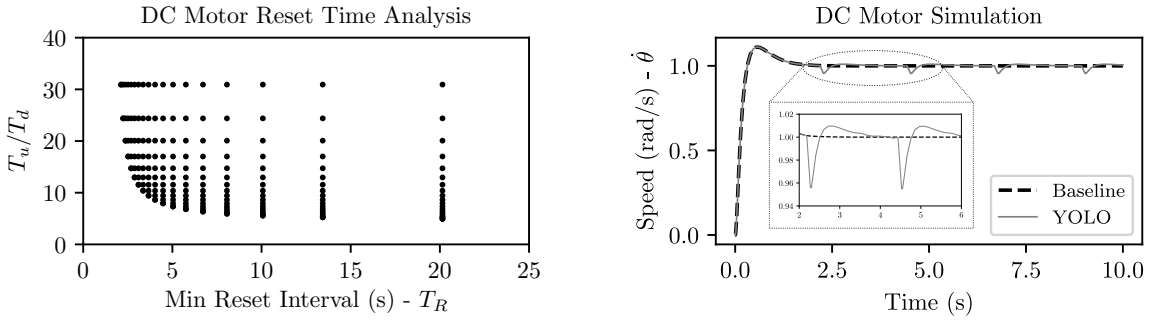
$$A_1 = \begin{bmatrix} 0 & 1.0000 \\ -18.1818 & -10.0009 \end{bmatrix} \qquad A_2 = \begin{bmatrix} -10.01 & 0 \\ 0 & 0 \end{bmatrix}$$

Using basic matrix algebra, one obtains $\lambda(A_1) = \{-7.6125, -2.3884\}$ and $\lambda(A_2) = \{-10.01, 0\}$. From which, $\lambda^- = 2.3884$ and $\lambda^+ = 10.01$ are obtained. We choose $\lambda \in (0, \lambda^-) = 0.1$ and $\lambda^* \in (\lambda, \lambda^-) = 2$. From (8), $a_1 = 0$ and $a_2 = 2.0146$. Hence, a equals 2.0146. Finally, by using (6) and (7), we get $\tau = 1.0603s$ and $T_u/T_d = 30.9202$. This leads to a possible selection of $T_r > 2\tau = 2.15s$ with $T_d = 67ms$.

Additionally, using *Matlab Robust Control Toolbox* to solve (9) would give us the following two positive definite symmetric matrices, which are required to construct Lyapunov stability.

$$P_1 = \begin{bmatrix} 0.4873 & -0.0321 \\ -0.0321 & 0.0479 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 0.0269 & 0 \\ 0 & 0.0551 \end{bmatrix}$$

It is worth noting that choosing different values for the arbitrary scalars λ and λ^* would result in different valid combinations for τ and T_u/T_d. For instance, Fig. 4a summarizes the allowed minimum reset times (T_r) and the corresponding T_u/T_d. Every point on the graph represents one selection of λ and λ^*. This flexibility of λ and λ^* is highly important in order to satisfy any hard constraint proposed by the controller (i.e, the controller might require a certain T_d to operate normally after a reset).



(a) The relation between the minimum reset time T_r and its corresponding T_u/T_d for arbitrary selection of λ and λ^*.

(b) The time response of the DC motor rotational speed for a *Baseline* system and YOLO-ized one.

Figure 4: Simulation results for the DC motor.

Furthermore, we simulate the system shown in Fig. 2 by using *Matlab Simulink* with the aforementioned parameters for the DC motor and controller. Fig. 4b shows the output time response for a *Baseline* system versus a YOLO-ized one ($T_r = 2.15s$) and ($T_d = 67ms$). YOLO introduces approximately a 4% drop in engine speed. This result highlights how YOLO can be realized solely relying on the unique CPS properties.

7. EXPERIMENTAL ANALYSIS

In order to empirically evaluate YOLO, we study two distinct CPSs: an Engine Control Unit (ECU) and a UAV flight controller (FC)*. Each case study provides its own challenges to determining the feasibility of YOLO, because each one is different with respect to the physical component under control and thus the complexity of the controller. The main questions we study are: for which reset periods T_R is the system safe (a) for the ECU and (b) for the FC. Similarly, we also study how the performance of the system is impacted by resets.

7.1 Case Study: Engine Control Unit

7.1.1 Description

An ECU controls the combustion process an engine. For a combustion engine to produce the right amount of power, the ECU must inject fuel into the internal chamber, mix it with air, and finally ignite the air-fuel mixture, all at the right timings.

How it works: There are two rotating parts, the crank and camshaft, inside an engine. The ECU observes their revolutions to determine what the state of the engine is. The number of input signals that must be observed to correctly determine the engine state depends on the shapes of the crank and camshaft. Then, the control algorithm interpolates the time to properly schedule ignition and injection events. Once the ECU has determined in which phase of the combustion cycle the engine is in, it will use other measurements from sensors, such as throttle position, temperature, pressure, air-flow and oxygen to accurately determine the air-fuel mixture to be injected.

Platform: We use the rusEFI open-source ECU and a Honda CBR600RR engine, a very commonly engine used by FSAE racing enthusiasts. The source-code is written in C/C++ running on top of an open-source real-time library operating system called ChibiOS and is designed to run on a STM32F4-Discovery board, a widely popular micro-control unit (MCU). This board contains a 168MHz ARM Cortex-M4 processor with 192KBytes of SRAM and 1MB of non-volatile flash memory. The MCU contains only a Memory Protection Unit (MPU) with eight protection regions, which may be leveraged by diversification techniques.

Reset Mechanism: Realizing YOLO involves selecting an appropriate reset mechanism. For the ECU, we choose to power cycle the MCU, which effectively clears out all hardware state. Simple power cycling, or reboots,

*Videos for some of the experiments can be found at: https://bit.ly/2fxB5QS

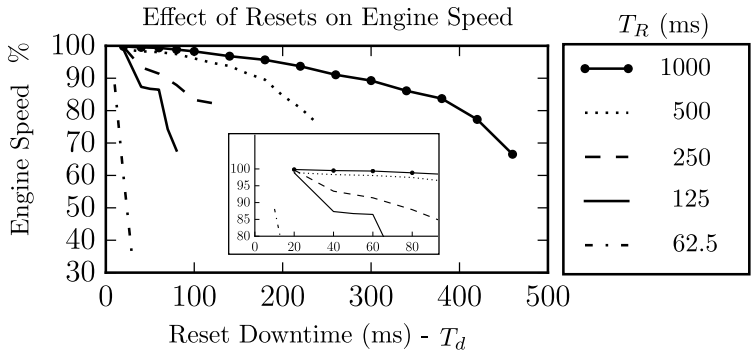


Figure 5: A sweep of the reset interval T_R and reset downtime T_d to study the effects on engine speed. We observe that for certain combinations of T_R and T_d the engine speed approximates 100%.

provide strong security advantages as it can be triggered externally, without any software. Additionally, it protects against attacks which may freeze the configuration of certain hardware peripherals. Power cycling incurs certain costs, specifically the cost of rebooting the chip and the time for the startup routines to reinitialize the controller. However, we found that the cost of rebooting the chip was on the order of microseconds and thus completely inconsequential compared to the latency of the startup routine. The non-interactive version of rusEFI's startup time is approximately $20ms$.

Diversification Strategy: We implement a static variant of Isomeron [16] that provides execution path randomization. Isomeron introduces a hybrid approach that combines code randomization with execution path randomization. Its main objective is to mitigate code-reuse attacks. The high level idea is the following: two copies of the program code are loaded into the same address space and execution is randomly transferred between the two on every function call. The original implementation of Isomeron uses dynamic binary instrumentation techniques which is not feasible on resource constrained devices. We leverage a binary rewriter to implement a static version of Isomeron which makes the technique suitable for our targeted resource constrained devices.

7.1.2 Evaluation

The synchronization time for the ECU is dependent on the number of engine cycles that must be observed. The ECU must observe two engine cycles to determine whether it is synchronized with the engine's rotation. Additionally, it must observe enough engine cycles to compute properties that must be integrated over time (eg. acceleration requires three engine cycles). Assuming an engine speed of 4500RPM (i.e., approx 75Hz), each engine cycle takes $13ms$, where synchronization takes $39ms$. Combining the previous software timing constraints along with the physical safety constraints, we set a range of T_R and T_d to perform our experiments according to (7).

Resets: Fig. 5 shows the change in engine speed—for an unloaded engine[†]—as a percentage for the sweep. Maintaining the engine speed, can be satisfied for a wide range of T_R and T_d where the engine speed is approximately 100%. We note what happens when the engine speed drops significantly. At some point, the ignition & injection steps are unable to generate enough energy to overcome friction, and the engine comes to a stop. We refer to the specific engine speed at which this failure occurs as the stopping threshold. As we more frequently, we observe lower engine speeds without crossing the stopping threshold during operation as stay within the system's safe region. We also note that the actual stall threshold varies non-linearly with T_R and T_d, most likely due to environmental factors and the large variability in the internal combustion process. Under engine load, we expect the stall threshold to vary less due to the added inertia. For our system, we can therefore conclude that there are specific combinations of T_R and T_d for which safety can be satisfied even as the system misses events.

Diversification: For the ECU we studied the effects of our Isomeron implementation. The results showed our version introduced a constant slow down of approximately $2.13x$, primarily due to its use of a hardware random number generator used to implement the Isomeron diversifier that switches between functions. While this slow

[†]No access to a dynamometer was available.

down may seem large, the original application had more than sufficient slack to accommodate it. To put this into perspective, even with the slow down, our ECU was still within typical timing accuracy of commercial systems ($2-3°$ delay). To further test the limits of our system, we swept ignition and injection delays up to $8x$, but saw no observable effects on RPM for our engine.

7.2 Case Study: Flight Controller

7.2.1 Description

The FC is designed to ensure the stability and control of an aircraft. It does so by controlling translation along the x, y, z directions and rotation about the x, y, z axes (i.e, the attitude).

How it works: The flight controller is primarily responsible for ensuring attitude stability while aiding a pilot or performing autonomous flight. It must read all of the sensor data and filter the noise in order to calculate proper output commands to send to its actuators. In particular, we focus on quadrotor helicopters more commonly referred to as quadcopters. Controlling these quadcopters involves operating four independent rotors to the provide six degrees of freedom. Sensors measuring a number of physical properties are then fused together to estimate the position and attitude of the quadcopter. This estimation, similar to the case of ECU, require a certain number of observation samples before an output is produced. This output is then used by other components that determine the best command actions for the system.

Platform: We use the PX4 open-source FC with a DJI F450 quadcopter air-frame, a very common DIY kit favored by enthusiasts. The PX4 FC provides attitude and position control using a series of sensors, such as GPS, optical flow, accelerometer, gyroscope, and barometer. The PX4 controller software includes a variety of flight modes ranging from manual, assisted, to fully autonomous. The source-code is written in C/C++ and supports multiple kinds of OS and hardware targets. Specifically, we use the Pixhawk board based on the same series of MCU as that used in the ECU case study.

Reset Mechanism: Similar to the ECU case study, we first attempted simple reboots. The downtime T_d for PX4 was found to be around $1.5s$. This higher reset time in comparison to the ECU was not unexpected due to the higher complexity of the quadcopter controller. Given the more sensitive physical dynamics of the quadcopter, simple rebooting is not effective, i.e., the quadcopter crashes very often, prompting the need of a more efficient approach. This led us to explore alternate, more performant reset mechanisms. We found that much of the startup time was spent in initializing data structures and setting up the system for operation. So, we create a snapshot of RAM after all the initialization is complete and use it to start the system in approximately $3ms$[‡]. This snapshot can additionally be verified and signed for increased security.

The snapshot is stored in a special region of flash and at the following boot, the saved state is restored. The special flash region is protected, and locked by the MPU. This provides a consistent restoration point for the system's lifetime. Snapshot-ing was implemented as an extension of the NuttX library OS used by the Pixhawk PX4 target. The $3ms$ that the snapshot restoration takes is primarily dominated by the time required to write data from flash to RAM.

When the snapshot is taken, and what data is stored in the snapshot, have implications on the capabilities of the system. Therefore we allow the designer to annotate data regions to be persisted and placed them in a memory location excluded from the snapshot & restoration mechanism. This is accomplished via source code compiler directives and linker modifications.

Depending on the flight mode for the quadcopter the snapshot has different requirements as to what data can be reset and persisted. For the autonomous flight mode for example, coordinates for the quadcopter's flight path could appropriately be made part of the snapshot taking care to use absolute coordinates were possible. However, including the flight path in the snapshot would prevent the quadcopter's path from being modified mid-flight. If this capability is desired, the data would need to be persisted across resets and protected in some way as discussed in § 5. The assisted flight mode has fewer limitations. For the assisted flight mode, which only requires the pilot inputs, a simple snapshot of RAM taken after the sensors have been calibrated is sufficient, as the system can recover the state that it needs by re-observing the environment. For optimal security, the

[‡]Code is available at http://bit.ly/2Wa5JER

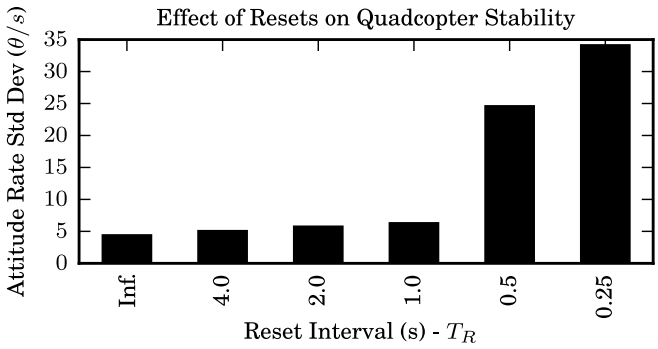


Figure 6: The effects of resets on the quadcopter stability for various reset intervals. These results, quantify our observations from the poll conducted.

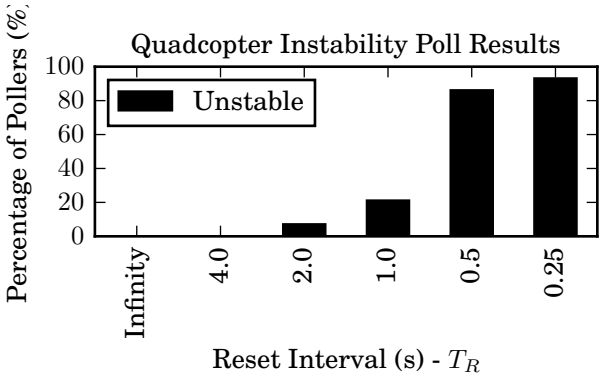


Figure 7: The results of the poll conducted to determine at which reset interval (T_R) the quadcopter will start to become unstable during hover. From this we can determine a minimum T_R for the system.

snapshot could be taken once in a controlled and secure environment, as long as, the system was initialized with the correct parameters.

Diversification Strategy: Here, we chose to implement a simple variation of conventional stack canaries. Canaries, or integers placed in memory just before the stack return pointer, are useful against stack overflows. On each reset, we randomly generate new canaries used by the program.

7.2.2 Evaluation

Resets: To quantify the effects of resets, the standard deviation of the quadcopter's attitude rate over time was used. The results are shown in Fig. 6. The results show little impact on the attitude for $T_R > 1$s and a large spike for smaller values. This indicates that for $T_R \leq 1$s the stability of the system is roughly equivalent to the system without YOLO. At lower T_R periods, we see a large spike in the standard deviations, which correspond to when we observe the system to start oscillating resulting in decreased controllability.

To better gauge the threshold at which a pilot would begin to detect these oscillations, or in other words, the lower limit for T_R, we conducted a survey among a set of 20 students. The survey was conducted using an ABX test methodology where various videos of the quadcopter with YOLO during flight for different T_R were shown. Before conducting the survey, users were shown an example video of a stable and unstable flight. They were then shown videos in a random sequence and asked to determine whether there were any observable oscillations during hover flight. The results are shown in Figure 7 indicating that oscillations become significantly observable somewhere between T_R of 0.5 to 1 second.

Diversification: Similar to the ECU, we sought to observe any difference in the system's behavior introduced by diversification. Our simple strategy in the case of the quadcopter, had a negligible effect. This is not surprising as stack canaries have significantly low overheads. To further stress the effects of delays on our system, we

effectively lowered the control loop rate by $1.5x$ and $2x$ which resulted in approximately a 7.6% and 11.3% change in the attitude standard deviation, respectively. In normal flight, we noticed no visual sign of these effects or any noticeable difference in maneuverability with respect to pilot inputs.

8. LIMITATIONS AND MITIGATIONS

Wear & Tear: YOLO may have miscellaneous effects on CPSs that may not have been observable during our evaluation. These effects can include additional wear and tear of components for example. It is difficult to say whether YOLO may have long term effects. However, after having explored these systems, the physical subsystems are built with ample tolerance margins such that we may never see any effects for the duration of the system's lifetime. For example, the rotors on quadcopters are Brushless DC motors (BLDC). BLDCs rely on electronic commutation as opposed to mechanical commutation and therefore do not suffer from much wear and tear.

Multiple Interacting Components: Our two case studies both look at systems with single computing components. Larger and more complex may have multiple interacting components in which timing and communication challenges arise. In order to realize YOLO on such systems, one possible solution is to think of each component as a microservice and follow a reset strategy similar to that proposed by Candea et al. [17] which involves recursively rebooting services depending on the time slack available to them. Given that the timings of interactions between components is part of the design, this information can be leverage to implement YOLO.

Temporary loss of control: In some CPSs even a temporary loss of control due to resets may be unacceptable. These cases have to be carefully evaluated. If temporary loss of control is completely unacceptable, then multiple replicas are necessary to provide fail safe operation. In this situation, interleaved resets can be used to implement YOLO to enhance security of the ensemble system. The practicality of this approach is validated by its use in the Boeing 787 which recently deployed interleaved resets [18] in response to a software error that triggered a simultaneous reset of all flight control systems (a safety issue, not a security problem).

9. RELATED WORK

CPS security can be broadly categorized along three dimensions: the threats covered by the defense, the nature of the defense, and the suitability of the defense to typical CPS environments. Tbl. 1 presents security techniques relevant to the paper ranging from general security techniques to those specific to CPSs. The references used here, especially for the general-purpose security techniques, are not meant to be exhaustive. These are just some of the most recent and most closely related to YOLO.

Threat Covered: Following the duality of CPSs, we categorize the threats covered into two groups: (a) *Cyber*—These attacks target the cyber subsystem. This group are those seen on traditional computing systems. (b) *Physical*—These attacks target the physical subsystem. More specifically, it is those attacks which target sensors and actuators. The general security techniques only focus on cyber attacks whereas CPS specific techniques primary focus on physical attacks.

Defense Type: We categorize defenses into four groups based on when the methodology that is applied (during construction or deployment; and what properties are provided by the defense under attack): (a) *Secure-by-Construction (SBC)*—These defenses focus on preventing the root cause for vulnerabilities. (b) *Detection*—These defenses focus on detecting system exploitation during deployment. It does not involve response to a detected attack. (c) *Graceful Degradation*—These defenses focus on mitigating the malicious effects of an attack to maintain acceptable levels of performance or to prolong the life of the system. (d) *Impersistence*—These defenses focus on denying the attacker an ability to gain a foothold or getting rid of the foothold the attacker may have gained.

Target: In order to evaluate whether a technique can be widely applied to CPSs, we consider two items: (a) *Microcontroller Compatible*—The technique can be run on a microcontroller with limited processing power and memory such as a Cortex-M based board. (b) *Additional Logic* —This means additional hardware is necessary for implementing the technique.

Table 1: Comparison of related work techniques.

		General Security						CPS Security								
		Control Flow Integrity [19]	Execute Only Memory [20]	Layout Re-Randomization [21, 22]	Control Flow Randomization [16]	Intrusion Detection [23]	Proactive Recovery [24]	Sensor Authentication [25]	Spoofing Detection [26]	Measurement Set Randomization [27]	Impact Mitigation [11]	Resilient State Estimation [28]	BFT++ [5]	Restart-Based Security [4]	YOLO (*this paper*)	
Threat	Cyber	✓	✓	✓	✓	✓	✓						✓	✓	✓	
	Physical							✓	✓	✓	✓					
Defense	SBC	✓											✓	✓	✓	
	Detection		✓	✓	✓	✓	✓	✓	✓				✓			
	Graceful D.							✓		✓	✓	✓	✓	✓	✓	
	Impersist.			✓			✓								✓	
Target	μC Comp.	✓					✓	✓		✓	✓	✓	✓	✓	✓	
	Add. Logic						✓						✓	✓		

While the general security techniques focus on software intrusion detection, the CPS specific techniques focus on detection of attacks on physical interfaces, namely sensors and actuators. YOLO fits into this category by making attacks that exceed the reset interval impossible.

Graceful degradation under physical attacks has been an active research area in CPS security. The main idea here is to continue operating, perhaps suboptimally, even under attack. In most of these papers, the threat that is considered is an availability attack, and thus continued operation is synonymous with security. Recently proposed techniques [11, 27, 28] are approaching a point where non-detectable levels of destruction can only affect the system so slowly that stealthy spoofing attacks will become negligible or can be coped with in practical ways.

Impersistence is one of the crucial aspects in CPS security in that it is almost impossible to re-install the firmware during their operation. Moreover, it is essential for many CPSs that recovery be performed without human intervention. One of the closest techniques to YOLO among general security techniques is proactive recovery [24]. However, their approach assumes redundancy (i.e. additional hardware) in place of inertia.

10. CONCLUSION

It is natural to ask if CPSs are indeed unique, if so, how CPS defenses should be different from general-purpose defenses. This paper provides one answer to this question. We construct a new resiliency mechanism that is only practical on CPSs because of their properties. In contrast to prior work on CPSs, which focus mostly on sensors, our defense leverages both cyber and physical properties for a tailored defense of the cyber portion of the CPS. We show that new, simple to implement, low-resource, and effective defenses are possible if we leverage the unique physical properties of CPS.

We present a new CPS-tailored cyber defense called YOLO that combines reset and diversification. YOLO leverages unique properties of cyber-physical systems such as inertia fo its implementation. In a traditional system, frequent resets will degrade the usability of the system. In CPSs, however, the physical subsystem can continue to move and operate even during resets because of its inertia.

We show that YOLO is an effective practical solution for an engine control unit and a flight controller. From our experiments, we determine that resets can be triggered frequently, as fast as every 125ms for the ECU and every second for the flight controller, without violating safety requirements.

YOLO has the potential to alleviate problems that plague CPSs in industry. First, these systems have a long shelf life typically around 10 years if not more. Therefore, there is a need to provide solutions for the large amount of legacy systems out there. Secondly, there is the issue of the long time-to-market and its implications on computing resources. CPSs usually have a production cycle of 3 to 5+ years meaning that the hardware they use is as old, if not older by the time they are released.

In this work, we have not only demonstrated the feasibility of YOLO, but have highlighted the need to design CPS specific security techniques. Given YOLO's simplicity, we believe it to be a practical solution that will inspire others to consider CPS tailored security. The results of our work show that resets, which may have been previously thought of as unrealistic due to safety can indeed be done. When applicable, YOLO may be especially useful for legacy systems and for emerging unmanned CPSs such as drones.

ACKNOWLEDGMENTS

This work was partially supported by ONR N00014-16-1-2263, ONR N00014-17-1-2788, ONR N00014-15-1-2173, a subcontract from CMU/SEI from ONR, and gifts from Bloomberg and Canon. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US Government or commercial entities. Simha Sethumadhavan has a significant financial interest in Chip Scan Inc.

References

[1] Davidson, D., Wu, H., Jellinek, R., Ristenpart, T., Tech, C., and Singh, V., "Controlling UAVs with sensor input spoofing attacks," in [*10th USENIX Workshop on Offensive Technologies*], (2016).

[2] Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., and Kohno, T., "Comprehensive experimental analyses of automotive attack surfaces.," in [*USENIX Security*], (2011).

[3] Larsen, P., Homescu, A., Brunthaler, S., and Franz, M., "SoK: Automated software diversity," in [*IEEE S&P*], 276–291 (2014).

[4] Abdi, F., Chen, C., Hasan, M., Liu, S., Mohan, S., and Caccamo, M., "Guaranteed physical security with restart-based design for cyber-physical systems," in [*2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*], 10–21 (April 2018).

[5] Mertoguno, J. S., Craven, R. M., Koller, D. P., and Mickelson, M. S., "A physics-based strategy for cyber resilience of cps," in [*Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure*], **DL 11009** (2019).

[6] Oppenheimer, D., Ganapathi, A., and Patterson, D. A., "Why do internet services fail, and what can be done about it?," in [*USENIX symposium on internet technologies and systems*], **67**, Seattle, WA (2003).

[7] Evans, D., Nguyen-Tuong, A., and Knight, J., "Effectiveness of moving target defenses," in [*Moving Target Defense*], 29–48, Springer (2011).

[8] Rains, T., Miller, M., and Weston, D., "Exploitation trends: From potential risk to actual risk," in [*RSA*], (2015).

[9] Shacham, H., Page, M., Pfaff, B., Goh, E.-J., Modadugu, N., and Boneh, D., "On the effectiveness of address-space randomization," in [*ACM CCS*], 298–307 (2004).

[10] Snow, K. Z., Monrose, F., Davi, L., Dmitrienko, A., Liebchen, C., and Sadeghi, A.-R., "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization," in [*IEEE S&P*], 574–588 (2013).

[11] Urbina, D. I., Giraldo, J. A., Cardenas, A. A., Tippenhauer, N. O., Valente, J., Faisal, M., Ruths, J., Candell, R., and Sandberg, H., "Limiting the impact of stealthy attacks on industrial control systems," in [*ACM CCS*], 1092–1105 (2016).

[12] Liberzon, D., [*Switching in Systems and Control*], Birkhauser Basel (2003).

[13] Lin, H. and Antsaklis, P. J., "Stability and stabilizability of switched linear systems: A survey of recent results," *IEEE Transactions on Automatic Control* **54**, 308–322 (Feb 2009).

[14] Zhai, G., Hu, B., Yasuda, K., and Michel, A. N., "Stability analysis of switched systems with stable and unstable subsystems: an average dwell time approach," in [*American Control Conference*], **1**, 200–204 (Sep 2000).

[15] Dorf, R. C. and Bishop, R. H., [*Modern control systems*], Pearson (2011).

[16] Davi, L., Liebchen, C., Sadeghi, A.-R., Snow, K. Z., and Monrose, F., "Isomeron: Code randomization resilient to (just-in-time) return-oriented programming.," in [*NDSS*], (2015).

[17] Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G., and Fox, A., "Microreboot-a technique for cheap recovery.," in [*OSDI*], **4**, 31–44 (2004).

[18] "Alert service bulletin b787-81205-sb270040-00," *Boeing Aircraft Service Bulletin* (2016).

[19] Abadi, M., Budiu, M., Erlingsson, U., and Ligatti, J., "Control-flow integrity," in [*ACM CCS*], 340–353, ACM (2005).

[20] Braden, K., Crane, S., Davi, L., Franz, M., Larsen, P., Liebchen, C., and Sadeghi, A.-R., "Leakage-resilient layout randomization for mobile devices," in [*NDSS*], (2016).

[21] Bigelow, D., Hobson, T., Rudd, R., Streilein, W., and Okhravi, H., "Timely rerandomization for mitigating memory disclosures," in [*ACM CCS*], 268–279 (2015).

[22] Williams-King, D., Gobieski, G., Williams-King, K., Blake, J. P., Yuan, X., Colp, P., Zheng, M., Kemerlis, V. P., Yang, J., and Aiello, W., "Shuffler: Fast and deployable continuous code re-randomization," in [*USENIX OSDI*], (2016).

[23] Roesch, M. et al., "Snort: Lightweight intrusion detection for networks.," in [*LISA*], **99**(1), 229–238 (1999).

[24] Castro, M. and Liskov, B., "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)* **20**(4), 398–461 (2002).

[25] Shoukry, Y., Martin, P., Yona, Y., Diggavi, S., and Srivastava, M., "Pycra: Physical challenge-response authentication for active sensors under spoofing attacks," in [*ACM CCS*], 1004–1015 (2015).

[26] Pasqualetti, F., Dörfler, F., and Bullo, F., "Attack detection and identification in cyber-physical systems," *IEEE Transactions on Automatic Control* **58**(11), 2715–2729 (2013).

[27] Rahman, M. A., Al-Shaer, E., and Bobba, R. B., "Moving target defense for hardening the security of the power system state estimation," in [*ACM Workshop on Moving Target Defense*], 59–68 (2014).

[28] Fawzi, H., Tabuada, P., and Diggavi, S., "Secure estimation and control for cyber-physical systems under adversarial attacks," *IEEE Transactions on Automatic Control* **59**(6), 1454–1467 (2014).