BayesMem: An End-to-End Bayesian Memory-Deduplication based Rowhammer Attack on Industrial Control Systems

ABSTRACT

Industrial Control Systems (ICSs) have distributed and interconnected components in Industry 4.0. This decentralized nature opens the "Pandora's Box" of unknown threats that could come from very unconventional ways. Over the last decades, several papers have been published to show attacks by exploiting bugs in PLCs, sensing nodes, and communication protocols in different layers of ICSs. However, in this paper, we show a new attack primitive-*BayesMem* using the *Bayesian estimation* based memory deduplication and the Rowhammer attack in cloud settings that works without the presence of software bug in any layer of ICSs. The *BayesMem* deploys a malicious co-located Virtual Private Server (VPS) in clouds near the victim VPS and injects false commands into the target ICS. To the best of our knowledge, we are the first to point out that the .bss section of the target control DLL file of cloud protocols can be duplicated by the attacker, and we introduce a new technique to duplicate the .bss section using the *Bayesian estimation* that results in less memory (i.e., 4 KiB compared to GiB) and time (i.e., 13 minutes compared to hours) compared to other works. We point out that ICSs can be expressed as state-space models; hence, the *Bayesian estimation* is an ideal choice to be combined with the memory deduplication in cloud settings. We create a Hardware-in-the-Loop (HIL) testbed using the Beremiz softPLC and use a scaled-down model of a practical engine cooling system as a target ICS in our HIL testbed. We demonstrate that the *BayesMem* can predictively inject false commands in the HIL testbed that can change the coolant flow of the engine cooling system between 0% to 100% without any prior notice. Moreover, we show that the *BayesMem* is capable of adversarial control over the target ICS that may result in severe consequences, such as an unplanned shutdown of the target ICS.

CCS CONCEPTS

• **Security and privacy → Systems security**.

KEYWORDS

industrial control systems, .bss section of DLL file, bayesian estimation, memory deduplication, rowhammer, adversarial control

1 INTRODUCTION

Historically, Industrial Control Systems (ICSs) follows the ANSI/ISA95 model [63] where disconnected computer systems and isolated sensor frameworks were used to screen various operations and tasks in lower *levels* of the *automation pyramid* [18]. As we enter the fourth industrial revolution (Industry 4.0), the ANSI/ISA95 model of ICSs is going under different transformations. These transformations include the creation of vertically/horizontally-connected, and decentralized Cyber-Physical Systems (CPSs) in all levels of the *automation pyramid* for flexible monitoring and automation. To accelerate this transformation, industrial sectors have planned to commit US$ 907 billion per annum to Industry 4.0 [34]. Despite the unprecedented growth, the rosy prospects of ICSs in Industry 4.0, however, comes laden with various security threats. Examples of such threats on ICSs include cyberattacks on Ukrainian power systems [33], Stuxnet malware attack on Iran's nuclear facilities [45], cyberattack on German still mill [49], TRISIS/TRITON attack on an oil facility [29], etc. The consequences of these attacks are very serious, resulting in unplanned shutdowns, lost production, possible equipment damage, and monetary losses [35, 60]. In this paper, we also demonstrate a new attack-*BayesMem*, on ICSs that may result in similar consequences.

Due to the movement to Industrial Internet of Things (IIoT) trends, Programmable Logic Controllers (PLCs) are connected with clouds in modern ICSs [46]. Moreover, to support multiple PLC controllers and supervisory platforms, today's ICSs use multiple Virtual Private Servers (VPSs) in a single cloud platform [36]. In this typical ICS platform, the user sends control programming and supervisory commands from VPSs using cloud protocols (i.e., MQTT, AMQP) to PLCs [47], and a specific DLL file of cloud protocols transports these commands. We call this specific DLL file as *target control DLL* file, and the .bss section of the target control DLL file contains the control commands. Section 5.1 covers a detailed description of this.

In this paper, we show how our attack model-*BayesMem* uses a malicious co-located VPS to inject false control programming and supervisory control commands into the connected PLCs with *adversarial control*. The injected false commands propagate from the VPS to the PLC and eventually may cause a Denial-of-Service (DoS) attack on the target ICS. The *BayesMem* duplicates the .bss section of the target control DLL file using a newly introduced *Bayesian estimation* technique. *As ICSs can be expressed as state-space models [31], our BayesMem exploits the Bayesian estimation technique to accurately predict the current state of the industrial controller and use the prediction of the current state to accurately duplicate the .bss section of the target control DLL.* Then, the *BayesMem* uses the underlying memory deduplication feature of the host cloud to merge the .bss section of the target control DLL with the attacker's duplicated .bss section of the target control DLL. After merging the attacker's duplicated .bss section with the .bss section of the specific target control DLL, the *BayesMem* initiates the Rowhammer attack on the merged/deduplicated .bss section from a co-located malicious VPS. By using the Rowhammer attack, the *BayesMem* can arbitrarily cause a bit-flip in the .bss section of the target control DLL file. As the .bss section of the target control DLL file contains the control commands, a bit flip in this section may cause corruption in the actual command. This method can be termed as *false command injection* into the target PLC and can cause a DoS attack on the target ICS. To the best of our knowledge, the *BayesMem* is the first work that successfully merges the idea of *Bayesian estimation* of the

Anonymous Submission to ACM CCS, 2021,
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

state-space models of ICSs with the memory deduplication and the Rowhammer bug in cloud settings in the context of ICSs. This type of attacks are not yet explored in depth by the security community and will be relevant soon in the era of Industry 4.0.

Technical Contributions: Our contributions are listed as follows:

- We present a new attack model-*BayesMem* that can *stealthily* breaks the *availability* and *integrity* [24, 34] of ICSs in cloud settings.
- To the best of our knowledge, we are the first to point out how the .bss section of the target control DLL file of cloud protocols can be exploited by using the memory deduplication feature in modern ICSs in absence of software bugs and with all defenses turned on.
- We introduce a new technique to duplicate the .bss section of the target control DLL file. This new technique is the first of its kind that involves the Bayesian estimation. This technique requires less memory and time compared to recent works [17, 23, 59].
- We create a Hardware-in-the-Loop (HIL) testbed with a scaled-down model of a practical engine cooling system of thermo-electric plants as an example of ICS. We use the Beremiz softPLC to create the automation platform and connect the softPLC to clouds using industry-standard cloud protocols.
- We evaluate our attack model in the HIL testbed to prove its strength and severe consequences in the context of ICSs. We consider five variants of industry standard cloud protocols to generalize our attack model in cloud settings.
- We show adversarial control using our attack model over the example engine cooling system in our practical HIL testbed. The demonstration of the *BayesMem* is shown in the following link: https://sites.google.com/view/bayesmem/home.

2 RELATED WORK

Attacks on ICSs: The attacks on ICSs can be broadly classified as attacks on physical hardware (e.g., PLCs, control modules, etc.), attacks on communication networks, and attacks on sensing side.

Abbasi et al. [11] demonstrated an attack on PLCs by exploiting pin control operations of certain input/output pins resulting in abnormal hardware interrupt in PLCs. Garcia et al. [32] presented a malware-PLC rootkit that can attack PLCs using the physics of the underlying systems. Bolshev et al. [22] showed an attack on the physical layer (i.e., analog-to-digital converter), resulting in false data injection into PLCs. Spenneberg et al. [66] developed a PLC worm-PLC Blaster that independently searches any network for S7-1200v3 devices and attacks them when the protective mechanisms are switched off. *Compared to our attack model, these attacks on PLCs lack the presence of adversarial control over PLCs and do not provide any means of stealthiness with respect to the monitoring entity.*

Klick et al. [44] showed that internet-facing controllers act as an SNMP scanner or SOCKS proxy, and their protocols can be misused by an adversary to inject false codes into PLCs, which are not directly connected to the internet. Basnight et al. [20] presented an attack on firmware exploiting communication protocols of PLCs. Beresford et al. [21] discovered vulnerabilities in Siemens S7 series communication protocol and showed replay attack on ICSs. *Compared to these attacks, our attack model does not need any vulnerabilities in the communication protocol and does work without any presence of software bug in any level of the system.*

Barua et al. [19], Liu et al. [50], and McLaughlin et al. [51] showed *false data injection* attack on different sensing nodes of ICSs leading to abnormal behaviour of the underlying system. *Compared to these attacks, our attack model is capable of false command injection from a remote location with adversarial control leading to DoS in ICSs.*

Attacks using memory deduplication and/or Rowhammer: Bosman et al. [23] demonstrated memory deduplication based exploitation vector on Windows using Microsoft Edge. This is the first paper that demonstrates how the Rowhammer bug can be merged with memory deduplication to cause a bit flip in the memory. Barresi et al. [17] exploited the memory deduplication in a virtualized environment to break ASLR of Windows and Linux. This attack uses brute force to duplicate the target page in the memory. Razavi et al. [59] provided Flip Fleng Shui (FFS) to break cryptosystems using both the memory deduplication and Rowhammer. **There are fundamental differences between our work and [17, 23, 59].** ***First****, our attack model exploited the .bss section of cloud protocols that is more impactful and realistic in ICSs.* ***Second****, our attack uses the Bayesian estimation to duplicate the target page compared to brute force approach in [17, 23, 59]. This results in significantly less memory usage (i.e., in KiB compared to GiB) and time (i.e., in minutes compared to hours) to duplicate the target page. This makes our attack model more feasible.* ***Third,*** *our attack model demonstrates adversarial control over the target ICS that is completely absent in [17, 23, 59].*

Seaborn et al. [64] exploited CPU caches to read directly from DRAM using the Rowhammer bug. Gruss et al. [39] used cache eviction sets and Transparent Huge pages (THP) for a successful double-sided Rowhammer. Tatar et al. [69] used Rowhammer attacks over the network to cause bit-flips using Remote DMA (RDMA). *Compared to these works, our work uses memory deduplication to skip the knowledge of physical memory location and uses single-sided Rowhammer on the target cloud memory. Moreover, our attack does not require any RDMA to happen that makes our attack more flexible in the context of ICSs.*

3 BACKGROUND

This section provides background on the synergy among various components of ICSs in cloud settings. This knowledge is also required to properly understand our attack model.

3.1 Connecting PLCs with clouds

Nowadays, the Supervisory Control and Data Acquisition (SCADA) systems are threatened by the growth of cloud IIoTs because PLCs can upload acquired data directly to clouds [62]. PLCs can be connected to clouds normally in two ways: using a cloud adapter, or directly using a standard protocol. A cloud adapter is simply an IoT gateway that can provide direct communication to clouds using master-slave protocols. Standard cloud protocols, such as MQTT and AMQP can be used by PLCs to directly communicate with clouds. In addition, cloud can enable a *bidirectional* and *event-based* data transmission between PLCs and management systems. Management systems can modify or change control functions of PLCs in run-time by flashing new *control programs* to PLCs from clouds. A basic discussion on the structure of the PLCs control programming in cloud is discussed next to facilitate the reader's understanding.

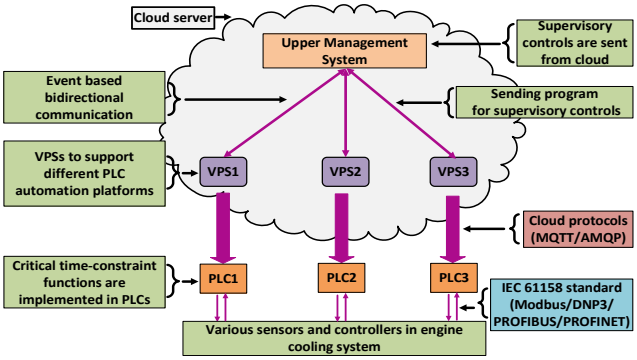

Figure 1: Different components of an ICS in cloud settings.

3.2 Programs for supervisory controls

The IEC 61131 programming standard [70] is used to program control functions of PLCs. Control programs can be broadly divided into three categories: (i) programs for basic functions, (ii) programs for supervisory controls, and (iii) programs for critical time-constraint functions (e.g., security and real-time response, etc.). Traditionally, all these three categories of control programs were implemented in PLCs in industrial premises. However, with the new trend in Industry 4.0, nowadays, only the programs for critical time-constraint functions are implemented in PLCs. Programs for basic functions and supervisory controls are not implemented in PLCs; rather, they are implemented in clouds or in web-server. For example, basic functions and supervisory control programs are outsourced as web services to a cloud or to a server for class C33 PLC controller [47]. *This gives more flexibility to upper managements as they can change programs remotely in run-time to tackle abruptly changing situations.*

3.3 Use of VPSs with PLCs

ICSs are becoming more complex in Industry 4.0. Hence, ICSs occasionally need to support multiple automation platforms that may conflict with each other. Moreover, multiple PLC controllers and supervisory platforms may need multiple software packages that may require multiple operating systems. Also, introducing web-servers and clouds to ICSs (Sections 10.2 and 3.1) increases the necessity of using multiple private-servers. As using multiple separate physical machines to support multiple automation platforms, or operating systems, or private servers is one of the available solutions, industries evidently use VPSs to reduce the number of required physical machines to reduce cost [61]. Moreover, modern cloud environments and web-servers offer cheap access to VPSs by sharing server hardware to run multiple operating system instances on a single machine using virtualization software [9].

3.4 A motivational example of an ICS

A motivational example is given in Fig. 1 to bind the above-discussed concepts together. We consider an engine cooling system of a thermo-electric plant as our example shown by a block in Fig. 1. We elaborate more on this in Section 6 while demonstrating our attack model on this. Multiple PLCs having different platforms are supported by a single cloud using multiple VPSs. Upper managements located in a single cloud send programs for supervisory controls from VPSs to PLCs using cloud protocols (i.e., MQTT/AMQP). PLCs communicate with the underlying sensors and controllers using IEC 61158 standard protocols (e.g., Modbus, DNP3, PROFINET, etc.). Given this motivation, a smart attacker can perturb the commands for supervisory controls (i.e., false command injection) in the example cooling system and hamper the temperature regulation process remotely using our attack model. In the next section, we introduce our attack model in detail.

4 ATTACK MODEL

Fig. 2 shows the attack model-*BayesMem* in ICSs in the era of Industry 4.0 in cloud settings. The essential components of the *BayesMem* are going to be described below.

Attacker's intent: The attacker wants to cause an unplanned shutdown, possible equipment damage, or monetary losses in the target ICS (e.g., engine cooling system in a thermo-electric plant).

Target system: We consider an infrastructure here that is common in today's ICSs in Industry 4.0 [37]. PLCs are connected with clouds for maintenance and control programming, and multiple Virtual Machines (VMs) act as VPSs in clouds to support multiple automation platforms sharing the same hardware. The hypervisor or the Virtual Machine Monitor (VMM) system of the host machine (i.e., cloud server) uses different containers to separate each VPSs from each other, ensuring separation of resources among VPSs. As different VPSs in a cloud use the same physical hardware, a malicious attacker can exploit this infrastructure. The attacker can deploy a *malicious co-located VPS* to attack a victim VPS in the same cloud to corrupt program for supervisory controls.

Attacker's capabilities: Let us consider a scenario where a user gives commands from his proprietary VPS to a PLC to do control programming and supervisory controls. A few specific DLL files (i.e., target control DLL) of the cloud protocols are responsible for handling this message transportation from VPS to PLCs. These DLL files are organized into different *sections*. Each section can be writable or read-only, and can encapsulate executable (i.e., code) or non-executable (i.e., data) information. The section, which encapsulates uninitialized data, is known as *.bss* section. The .bss section of the target control DLL contains control programming and supervisory control specific information/data, which is coming from the user as commands. This .bss section is page aligned in virtual memory as well as in physical memory. Let us denote this as *victim page*. If an attacker can duplicate the *victim page*, the attacker can use this duplicated *victim page* (a.k.a., *attack page*) to trigger memory deduplication. The memory deduplication merged the *victim page* with the attacker's provided *attack page*. In this way, the attacker maps the *victim page* in his address space. Now, the attacker can initiate the Rowhammer attack on the *attack page* from his address space and can *arbitrarily* flip random bits in the *attack page*. As the *attack page* is the page-aligned version of the .bss section of the target control DLL, bit-flips in *attack page* causes bit-flip in the .bss section of the target control DLL.

Outcomes of the attack: This random bit flip in the .bss section of the target control DLL corrupts memory contents. This memory corruption could be harmful for PLCs as the .bss section contains important data dedicated for control programming and supervisory controls. This can potentially crash the PLC controller that may eventually lead to cascaded failure in ICS. For example, a similar attack on the industrial engine cooling system, which we mentioned

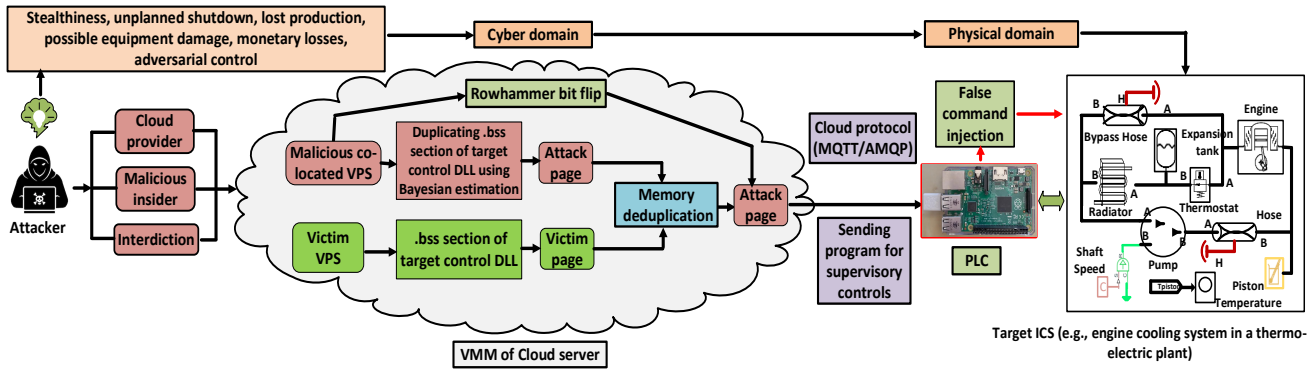

Figure 2: An overview of our end-to-end attack model-*BayesMem* on modern industrial control systems in cloud settings.

earlier as a motivational example in Section 3.4, can compromise specific PLCs. Let us assume the attacker corrupts a location in the .bss section that stores the temperature threshold of the cooling system. This can abnormally turn on/off the coolant flow without any prior notice. As a result, the compromised PLC may behave as a weird machine that eventually hampers the flow of coolant in hoses and pipes. This, in effect, may damage the temperature regulation of the entire cooling system. If the cooling system is a part of a thermo-electric power plant or a ship, this attack may cause serious system failure that may lead to system shutdown (i.e., DoS).

Attacker's access level: Our attack model requires the deployment of a malicious co-located VPS in the cloud near the victim VPS. This requires access to clouds of the victim ICS. As public clouds are not common in ICSs, the clouds in ICSs can be either private or hybrid. The access needed to private or hybrid clouds, where VPSs reside, can be possible in at least three scenarios.

In the first scenario, the attack can be originated from the cloud provider targeting VPS of cloud users [58]. As cloud providers provide software, platform, and infrastructure as service [13], they have physical access to target clouds where the victim VPS resides. This can give him access to deploy the malicious co-located VPS in the same cloud near the victim VPS.

In the second scenario, a malicious insider [26, 73], which can be a disgruntled employee, can use his insider knowledge of the system to deploy the malicious co-located VPS in the same cloud near the victim VPS. A similar incident is found in the literature. A disgruntled ex-employee of an ICS in Texas posted a note in a hacker journal indicating that his insider knowledge of the system could be used to shut down that ICS [67].

The third scenario is interdiction, which has been rumored to be used in the past [14, 65, 71] and has been recently proven to be practically feasible [68]. In this scenario, during interdiction, a competitor can intercept the installation of VPS in clouds while providing service and may deploy the malicious co-located VPS in the same cloud near the victim VPS.

End-to-end stealthy attack: As our attack model originates in the cyber domain (i.e., clouds) and impacts the physical domain (i.e., PLCs), this is a novel classic example of an *end-to-end attack* in the context of ICSs. Most of the works in security literature focus on hardening protocols existing in different layers of ICSs as a way of safety measures against cyberattacks on ICSs. Our attack model is unique in a sense that this attack model can still be valid in the case of hardened protocols in any level of ICSs. In other words, robust Fieldbus, CIP, PROFIBUS/PROFINET, or MQTT protocols can not prevent our attack from happening. This end-to-end attack model is originated from a co-located VPS in the same cloud where the victim VPS already resides. Hence, if the authorities of the victim VPS may not be aware of this type of attack model, they would possibly not detect the source of this type of attack. In this sense, our end-to-end attack model is stealthy, and has the capability to alter the normal behavior of PLCs, and can remain unidentified.

Attacker's cost: As mentioned earlier, our attack model requires to duplicate the .bss section of the target control DLL of cloud protocols. Most of these specific DLLs of cloud protocols (i.e, MQTT, AMQP) are available as open-source, and very few are proprietary. To acquire the DLL files of the open-source cloud protocols, the attacker does not need to invest any resource; hence, the cost is zero. To acquire the DLL files of the proprietary cloud protocols, the attacker just needs to buy a basic commercial license that may cost a minimum of $100 [1]. Most of the proprietary cloud protocols have a free evaluation for few days, and the attacker can use this free evaluation period to access the .bss section of a specific DLL.

5 ATTACK MODEL DESIGN

To design the *BayesMem*, the attacker needs to accomplish the following three steps sequentially.

Step i. The attacker needs to duplicate the .bss section of the target control DLL file.

Step ii. After duplicating the .bss section of the target control DLL, the attacker needs to merge his duplicated .bss section with the victim's .bss section of the same target control DLL.

Step iii. After merging, the attacker can use the Rowhammer attack on the merged/deduplicated .bss section from his co-located malicious VPS in the same cloud.

5.1 Duplicating the .bss section of DLLs

5.1.1 ***Target control DLL file.*** The attacker needs to access a specific target control DLL file of cloud protocols (i.e., MQTT, AMQP) that transports the control programming and supervisory control commands from the VPS to PLCs. The name of the specific DLL file depends upon the cloud protocol's implementation variants. For example, the name of a popular implementation of MQTT is Mosquitto, and the specific target control DLL file for this variant to

access by the attacker is mosquitto.dll. There are also other variants in the market. We tabulate five popular variants of MQTT and their target control DLL files in Table 1. The same approach is equally applicable to other cloud protocols.

Table 1: Target control DLL file of cloud protocol variants

Sl.	Cloud protocol variants	Target control DLL
1	EMQ X Broker [4]	erlexec.dll
2	Mosquitto [7]	mosquitto.dll
3	MQTT-C [8]	mqtt_pal.dll
4	eMQTT5 [5]	MQTT_client.dll
5	wolfMQTT [10]	MqttMessage.dll

5.1.2 ***Format of target control DLL files.*** In 64-bit Windows, DLL files follow Portable Executable 32+ (PE32+) format. In high level, PE32+ has a number of *headers* and *sections* (Fig. 3). The header consists of DOS header, PE header, optional header, section headers, and data directories. These headers have *Image base Address* and relative virtual address (RVA) of every section that tells the dynamic linker how to map every section of the DLL file into physical memory. There are different sections placed after headers in DLL. Among different sections in DLLs, we want to mention four sections, namely .rdata section, .data section, .text section, and .bss section. The .rdata section contains string literals, the .data section contains global/static initialized variables, the .text section contains machine code of the program, whereas the .bss section contains zero-initialized variables. It is important to note that all these sections are *page aligned* [57]. This means that these sections must begin on a multiple of a page-size in both virtual and physical memory. These sections of DLL files are mapped to pages in physical memory after the *base-relocation* [57]. The base-relocation is randomized, and the ASLR technique is used to map these sections to pages in physical memory at load time by the operating system.

5.1.3 ***Reasons for choosing the .bss section.*** The intention of the attacker is to find a section in the DLL file that has less entropy, which leads to a successful guess of the section. As the .rdata, .data, and the .text sections consist of different unknown data and addresses, the pages in physical memory corresponding to these three sections have higher entropy. Hence, the estimation of these pages by the attacker requires a large amount of memory and time [17] that is not computationally feasible.

On the other hand, we examine that the .bss section of a target control DLL file (i.e., file responsible for transporting control programming and supervisory control related data) of cloud protocols (i.e., MQTT, AMQP) are static in the very beginning. The .bss section contains different uninitialized global/static variables. When a user gives any control data/command from the VPS, a specific uninitialized variable of the .bss section of the target control DLL file gets initialized by the value of the control data/command. Then, the .bss section of the control DLL file transports the given data/command from the VPS to PLCs using cloud protocols. The initialization of the specific uninitialized variable in the .bss section of the control DLL file increases entropy, and it provides a challenge to the attacker to successfully duplicate the .bss section of the control DLL file. Thankfully, this challenge can be handled by using the Bayesian estimation of specific command data in the .bss section. This process is discussed in the next section.

5.1.4 ***Bayesian estimation of the .bss section.*** To estimate the specific uninitialized variable of the .bss section of a target control DLL file, the attacker can use the Bayesian estimation. At first, let us mathematically formulate the Bayesian estimation (i.e., Chapman-Kolmogorov equation) here.

An ICS is dynamic in nature and can be expressed as a discrete-time state-space model [31]. In the discrete-time state-space model, a control system in ICS (e.g., cooling system mentioned earlier, etc.) can be expressed by a state vector, x_k, and a measurement vector, y_k, where k denotes a discrete-time index. Here x_k is a parameter of interest and y_k is the measurement at time k. The term x_k and y_k can be expressed as follows:

$$x_k = f_{k-1}(x_{k-1}, q_{k-1}) = p(x_k|x_{k-1}) \tag{1}$$

$$y_k = h_k(x_k, r_k) = p(y_k|x_k) \tag{2}$$

where q_{k-1} and r_k are state noise and measurement noise vector respectively, and they are independent of each other (i.e., mutually exclusive). Please note that both x_k and y_k are stochastic processes and Eqn. 1 implies that current state x_k at time index k depends on only the previous state x_{k-1} at time index $k-1$. This implies that x_k is a Markov process that simplifies calculations in Eqn. 3.

The attacker can predict the current state x_k at time k if the attacker has information on previous state, x_{k-1}, and previous measurements, $y_{1:k-1}$, (Fig. 3). Here, $y_{1:k-1}$ consist of all previous measurement data $[y_1\ y_2 \ldots y_{k-1}]$ up-to time $k-1$. Let us consider a scenario to clear this concept. Let us denote the *states* of solenoid (i.e., a switching device) in our example of cooling system, as x_k at time k. Let us consider the solenoid can be in one of five states, $x_k \epsilon \{0\%$ flow, 25% flow, 50% flow, 75% flow, 100% flow$\}$. The activation of solenoid in each state depends on the temperature measurement at different hoses and pipes of the cooling system. This temperature measurement can be expressed by y_k at time k in our model. If the attacker knows previous state, x_{k-1}, of the solenoid and previous temperature measurements, $y_{1:k-1}$, then the attacker can use these data to accurately estimate the current state, x_k, at time k by using Eqn. 3 (i.e., Chapman-Kolmogorov equation). The left hand side of Eqn. 3, $p(x_k|y_{1:k-1})$, is a conditional estimation of current state, x_k, while previous measurements, $y_{1:k-1}$, are given. The right hand side of Eqn. 3 depicts that $p(x_k|y_{1:k-1})$ is a function of previous state, x_{k-1}, that is an indication of Markov process.

$$\begin{aligned} p(x_k|y_{1:k-1}) &= \int p(x_k, x_{k-1}|y_{1:k-1}) dx_{k-1} \\ &= \int p(x_k|x_{k-1}) p(x_{k-1}|y_{1:k-1}) dx_{k-1} \end{aligned} \tag{3}$$

It is important to note that state information, x_k, is a part of control programming and supervisory commands and originates in the VPS. The .bss section of the control DLL file of the cloud protocol transports this state information from VPS to PLC. To transport this state information, a specific uninitialized variable in the .bss section must be initialized first with specific state information, x_k. If the attacker knows the previous state, x_{k-1}, and previous measurements, $y_{1:k-1}$, the attacker can accurately estimate the current state x_k using Eqn. 3. As a result, the attacker can accurately estimate the value of the specific initialized variable and can use this knowledge to successfully duplicate the physical page of the corresponding .bss section. For example, the name of a specific

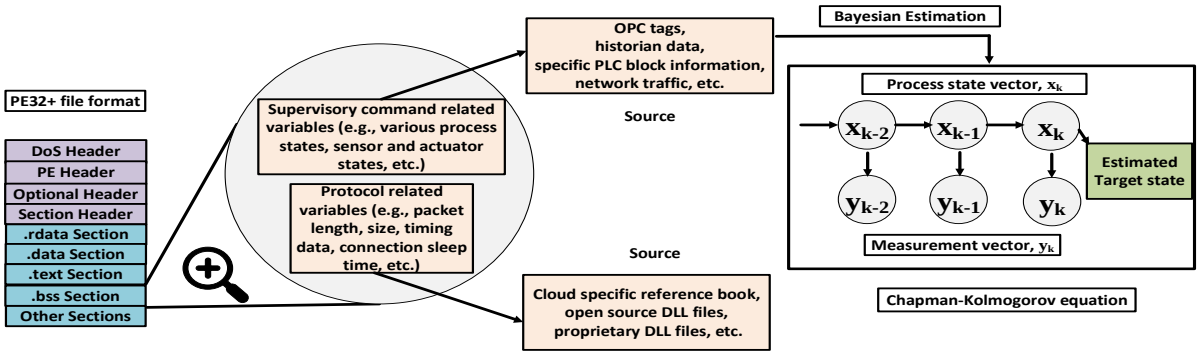

Figure 3: An overview of duplicating the .bss section of the target control DLL file.

uninitialized variable in the .bss section of the mosquitto.dll is `solenoidstate`. The attacker can use Eqn. 3 to estimate the value of `solenoidstate` and can create a duplicated .bss section. If the .bss section contains multiple specific uninitialized variables originating in the VPS, the attacker can use the similar technique to successfully estimate the values of all uninitialized variables.

5.1.5 ***Feasibility of the estimation of the .bss section.*** It is important to note that the .bss section contains different uninitialized global/static variables that can be broadly divided into two categories, namely the control programming and supervisory commands related variables and protocol related variables (Fig. 3).

As mentioned earlier, the values of the control programming and supervisory commands related variables depend on the current state, x_k, and to estimate, x_k, the attacker needs to get the previous state, x_{k-1} and previous measurement values, $y_{1:k-1}$. *The attacker can gather x_{k-1} and $y_{1:k-1}$ from OPC tags, historian data, specific PLC block information, or network traffic [26]. Moreover, as mentioned in Section 4, the cloud provider, or a malicious insider, or an interdiction method can make it possible to get x_{k-1} and $y_{1:k-1}$ from these sources.*

The protocol-related variables are specific to cloud protocols, and hence, are fixed and initialized at the load time of the control DLL file and do not originate in the VPS. The attacker can get the list of all the protocol-related variable names and their values from the reference-book of a specific cloud protocol. Moreover, as mentioned in Section 4, most of the target control DLLs of cloud protocols are available as open-source, and very few are proprietary. The open-source files are easily accessible, and the proprietary files are accessible by a basic commercial license (cost less than $100 [1]).

5.1.6 ***Entropy in the .bss section.*** The size of the specific control variable used in the target control DLL file for handling control programming and supervisory control related command can be a maximum of 64 bits in a 64-bit machine. Therefore, we have an entropy of 64 bits, and there could be 2^{64} possible values of the specific control variables. In other words, in our example, the variable named `solenoidstate` located in the .bss section of the control DLL ideally could have 2^{64} values. But in real world implementation, the control programming and supervisory control related variables are problem specific, and they have very few key values, which are also problem specific. For example, as mentioned earlier in Section 5.1.4, the state variable, `solenoidstate`, has five possible key values: {0% flow, 25% flow, 50% flow, 75% flow, 100% flow}. Though the variable `solenoidstate` has a maximum size of 64 bits, the variable `solenoidstate` can only have five key values. So the entropy of the control variable `solenoidstate` is not 2^{64}; instead, the entropy is only five for our example. Moreover, these key values are declared in the header files of the program codes, and programmers, as a good practice, generally use user-defined data types, such as *Enumeration (enum)* type to declare these key values. The use of enum data type by the programmer makes the declared control variable (e.g., `solenoidstate`, etc.) more predictable. For example, after careful examination of the source code of control programming and supervisory control-related application codes that are running on top of cloud protocols, we find the following code snippet that supports our observation:

```
enum statepool {0flow,25flow,50flow,75flow,100flow};
enum statepool solenoidstate;
```

This indicates that the attacker does not need to predict `solenoidstate` from 2^{64} possible combinations; instead, the attacker can predict the range of possible values of `solenoidstate` using intuition of enumeration data type. In this example, this range is 0 to 4. After estimating the current state, x_k (i.e., $x_k \epsilon$ {0% flow, 25% flow, 50% flow, 75% flow and 100% flow}) by using the Bayesian estimation technique (explained in Section 5.1.4), the attacker can accurately estimate the value of x_k, which is within 0 to 4 (i.e., $x_k \epsilon \{0, 1, 2, 3, 4\}$). *In this way, the attacker can reduce the entropy and successfully duplicate the .bss section of the target control DLL using a single page of size 4KiB.*

5.2 Merging the duplicated .bss section

So far, we discuss how the attacker can create a duplicated copy of the .bss section of the target control DLL file. In this section, we discuss how this duplicated .bss section can be merged with the target/victim's .bss section of the target control DLL. The attacker exploits the memory deduplication across VPSs to achieve this.

5.2.1 ***Memory deduplication and KVM.*** Memory deduplication or content-based page sharing is a process that combines/merges identical pages in the physical memory into one page. When the same/similar operating systems or applications are running in co-located VPSs, lots of redundant pages with same contents are created on the host system. The amount of redundant pages can be as high as 86% depending on the operating system and workload [25], and about 50% of the allocated memory can be saved through memory deduplication [40]. Memory deduplication is a feature in Windows 8.1, Windows 10, and Linux distribution. Due to more reliability, high security, stability, and less cost, Linux is

more preferable over Windows in ICSs [56]. That is why here we consider Linux as our implementation platform for memory deduplication, and the idea is similarly applicable to Windows as well. Let us consider that the cloud in our discussion of ICS runs in the Linux platform. To allocate multiple VPSs in the same cloud, Kernel-based Virtual Machine (KVM) has been introduced in the Linux kernel since 2.6.20. Memory deduplication is implemented as Kernel Samepage Merging (KSM) in KVM. Next, we discuss how KSM is used in our attack model to merge the duplicated .bss section.

5.2.2 ***Kernel Samepage Merging (KSM).*** When a VPS is started, a process named `qemu-kvm` of the KVM hypervisor allows KSM to merge identical pages in the memory. KSM has a specific daemon named `ksmd` that periodically scans a specific region of the physical memory of an application. The daemon `ksmd` can be configured in `sysfs` files in /sys/kernel/mm/ksm location. The `sysfs` files contain different configurable parameters. Among them, we need to mention two parameters: `pages_to_scan`, and `sleep_millisec`. The parameter `pages_to_scan` defines how many pages to scan before `ksmd` goes to sleep, and `sleep_millisec` defines how much time `ksmd` daemon sleeps before the next scan. If `sleep_millisec` = 500, and `pages_to_scan` = 100, then KSM scans roughly 200 pages per second. These numbers depend upon workload and are configured by the cloud provider accordingly. The values of `sleep_millisec` and `pages_to_scan` have a significant influence on the *attack time*. This is discussed in Section 7.6.

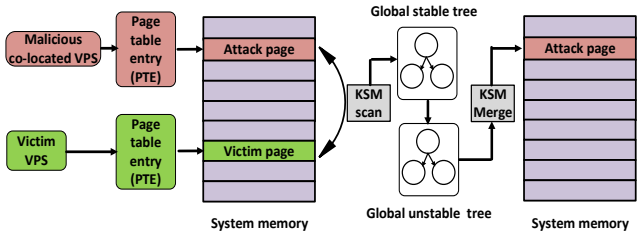

Figure 4: Merging *attack page* with the *victim page*.

KSM data structure: The daemon `ksmd` periodically scans registered address space and looks for pages with similar contents. KSM reduces excessive scanning by sorting the memory pages by their contents into a data structure, and this data structure holds pointers to page locations. Since the contents of the pages may change anytime, KSM uses two data structures in *red-black* tree format, namely unstable tree and stable tree. Moreover, there are three states of each page in the memory: frequently modified state, sharing candidate yet not frequently modified state, and shared/merged state. The page which is frequently modified is not a candidate to be loaded in a stable or unstable tree of KSM. The page which has similar contents yet not frequently modified (i.e., unchanged for a period of time) is a candidate to be loaded in unstable tree first. The pages in the unstable tree are not write-protected and liable to be corrupted as their contents are modified. The stable tree contains pointers of all shared/merged pages (i.e., ksm pages), and these pages are sorted by their contents in the stable tree. Each page in the stable tree is write-protected. Hence, whenever any process tries to write in the merged/shared page of the stable tree, a private copy of the page corresponding to that particular process is created first and mapped into the page-table-entry (PTE) of that particular process. Then the process writes in that private copy of the page. This is known as copy-on-write (CoW). As CoW involves the creation of a private copy of the shared/merged page of the stable tree first and then writes to that private page, CoW operation is expensive. Therefore, this takes a longer time compared to a write to a regular page. In other words, a longer write time on a page probably indicates that the page is already merged/shared in the stable tree by `ksmd` daemon. This longer write time in CoW process works as a side channel [23] and provides an indication that the page is already merged with another page having similar contents.

5.2.3 ***Process of merging the duplicated .bss section.*** The process of merging the duplicated .bss section is shown in Fig. 4. As discussed earlier, the .bss section of the target control DLL is page aligned and is mapped to a page in the physical memory. Let us denote this page as the *victim page*. Similarly, the duplicated .bss section of the target control DLL file is also mapped to a different page in the memory. Let us denote this page as the *attack page*. The *attack page* and *victim page* both have same contents. The only difference between them is that the *attack page* is provided by the attacker, whereas the *victim page* is coming from the victim VPS.

The daemon `ksmd` of the KVM checks the contents of the *attack page* and the *victim page* in the registered address space. Either the *attack page* or the *victim page* is available to the daemon `ksmd` depending upon their order of arrival in the memory. If the *victim page* arrives first, the daemon `ksmd` marks this page as a candidate page to be merged. At first, this candidate page is searched in the stable tree using `memcmp()`. As this candidate page is not still available in the stable tree, it is then searched in the unstable tree by recalculating the checksum over the candidate page. If the checksum has not been changed, the daemon `ksmd` searches the unstable tree for this candidate page (`unstable_tree_search()`). In this case, as the occurrence of the candidate page (i.e., *victim page*) is first in the unstable tree, this candidate page cannot be found in the unstable tree. As a consequence, a new node is created in the unstable tree for this candidate page (i.e., *victim page*). In the next step, when the *attack page* arrives in the memory, the daemon `ksmd` marks this page again as the candidate page and searches this page in the unstable tree. As the content of the candidate page (i.e., *attack page*) is same as the *victim page*, this candidate page (i.e., *attack page*) will be merged with the similar node (i.e., *victim page*), which is created in the prior step, in the unstable tree. Then this node of the unstable tree will be merged into the stable tree. If a new candidate page arrives in the memory, this process iterates again.

5.2.4 ***More practical, stronger, and reliable attack model.*** The explanation in Section 5.2.3 clearly indicates that if the *attack page* arrives first in the memory, the node of the unstable/stable tree will be updated first with the *attack page*. Therefore, if the *victim page* comes later, the *victim page* is merged with the *attack page*, and the *victim page* shares the memory location of the *attack page*. In this way, the attacker can control the memory location of the target *victim page*. By controlling the location of the *victim page*, the attacker can trigger a Rowhammer attack on that page.

To ensure that the *attack page* arrives first in the memory, the attacker's VPS should start first before the victim VPS. Recent works [23], [59] use this technique along with creating two copies of target pages to place the *attack page* in the stable/unstable tree before the

target *victim page*. These techniques explained in [23], [59] require more control over the victim VPS by the attacker that may not be possible in reality in practical ICSs. For example, the attacker does not have any way to know when the victim VPS is started.

Our attack model is stronger in the sense that the attacker does not need to start his malicious VPS before the victim, or does not need to create two copies of target pages. Thanks to the Bayesian estimation of the *victim page*. Referring to Section 5.1.4, if the attacker has information on the previous state, x_{k-1} and previous measurements $y_{1:k-1}$, he can predict the current state, x_k, at time k. If the attacker can predict the current state x_k, this means that he actually can predict the *victim page* before time k. Therefore, the attacker can provide a copy of the *victim page* to the stable/unstable tree before the actual *victim page* arrives in the memory. As the attacker has the predicted *victim page*, the attacker can provide this predicted *victim page* to the KSM at any time. Hence, the attacker does not need to start his VPS before the victim or does not need to create two copies of target pages in our attack model. This makes our attack model more practical and reliable in the context of ICSs.

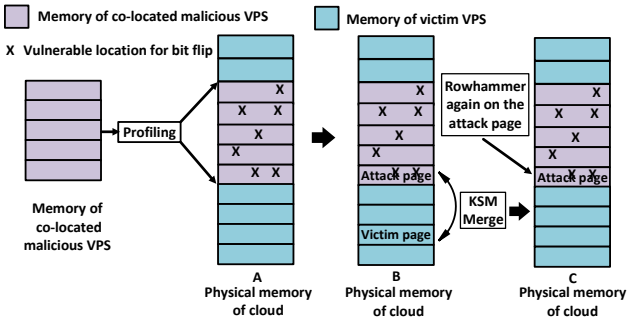

Figure 5: (A) Profiling the memory of the cloud. (B) Placing *attack page* in the vulnerable location. (C) After KSM merging, *victim page* is backed by the *attack page*.

5.3 Rowhammering on the merged .bss section

In Section 5.2, we discuss that how the target *victim page* is merged with the *attack page* using the memory deduplication technique. Note that the attacker cannot simply write to his attack page (i.e., deduplicated page) to change any data, as simply writing to the deduplicated page by the attacker triggers a CoW (Section 5.2.2) event to isolate the attack page from the victim page, and the main goal of the KSM may become invalid. That is the reason why the attacker needs something else to corrupt the deduplicated page without triggering the CoW event. Thanks to the Rowhammer bug present in DRAM, Rowhammer can be used to flip bits directly on the DRAM without triggering any CoW event.

Rowhammer [43] is a widespread vulnerability in recent DRAM devices in which repeatedly accessing a row of DRAM can cause bit flips in adjacent rows. To reliably craft our Rowhammer exploit on the deduplicated page, we have to overcome many challenges. The detail of these challenges is explained as follows.

5.3.1 ***Profiling the vulnerable locations of physical memory.*** A property of the Rowhammer is that the Rowhammer induced bit-flips tend to be repeatable. A memory location, where a bit flip occurs for the first time, there is a high chance that bit-flips will be reproducible in that location again. Therefore, it is possible to estimate whether a memory location of a DRAM tends to flip. This knowledge of exploitable bit locations is critical for the attacker to successfully exploit the Rowhammer bug from the co-located malicious VPS.

Therefore, the first step to initiate the Rowhammer attack is to find the *aggressor/victim* addresses in the physical memory of the running system. We name this step as profiling (Fig. 5(A)). The *aggressor* addresses are the memory locations within the process's virtual address space that are hammered, and the *victim* addresses are the memory locations where the bit flips occur. For a successful Rowhammer bit flip, the aggressor rows and the victim rows should be located in different rows but within the same bank of the DRAM chip. If the aggressor rows and the victim rows are located in the different banks of the DRAM chip, the Rowhammer exploit may only read/write from those bank's *row-buffers* without activating aggressor rows repeatedly. This may not cause any bit-flip in the physical location of the DRAM chip. Therefore, before starting the profiling step, the attacker must ensure that aggressor rows satisfy the *"different rows, same bank"* requirement for the Rowhammer.

Refining the profiling step: To ensure different rows but same bank location of the aggressor rows, there are different methods. One method is to use physical addresses of the DRAM rows using an absolute physical address or relative physical address information. The absolute physical address information may not be available by the malicious VPS of the attacker. The relative physical address information can be achieved by using *large pages* [53] in Windows VPS. To use the large page support in Windows, the large page option should be activated first in the victim VPS, but it may not be explicitly turned on in the victim VPS. Therefore, double-sided Rowhammering is not a suitable way for the profiling step in the context of ICSs [64]. Another method is to use random address selection. This is a simpler approach, and the attacker does not need to know the absolute physical address or relative physical address of DRAM. To keep the attack model simpler and easily exploitable, the *BayesMem* uses this random address selection approach for profiling the bit-flippable memory locations of the physical memory. This approach also falls in the category of single-sided Rowhammering.

In the random address selection approach, the attacker allocated a large block of memory of 2 GiB using a large array filled with doubles. A value of $1.7976931348623157 \times 10^{308}$ is stored as double that gives 1 in memory locations. Next, the attacker randomly picks virtual aggressor addresses from each page of this large memory block and reads 2×10^6 times from each random aggressor address of that page. Then the attacker moves to the next page and repeats the same steps. As the attacker can know the number of banks of the running system from his VPS, he can calculate his chance of hammering addresses in the same bank. For example, in our experimental setup, the machine has 2 Dual Inline Memory Modules (DIMMs) and 8 banks per DIMM. Therefore, the machine has 16 banks, and the attacker has 1/16 chance to hit aggressor rows in the same bank. This 1/16 chance is high for the attacker. Moreover, the attacker hammers 4 aggressor rows in the same iteration that increases the chance of having successful Rowhammering.

After finishing hammering the entire block of memory, the attacker checks the array for possible bit flips. If any bit-flip occurs on any page, the attacker records that page and the offset. In this way, the attacker profiles the memory for vulnerable page/location,

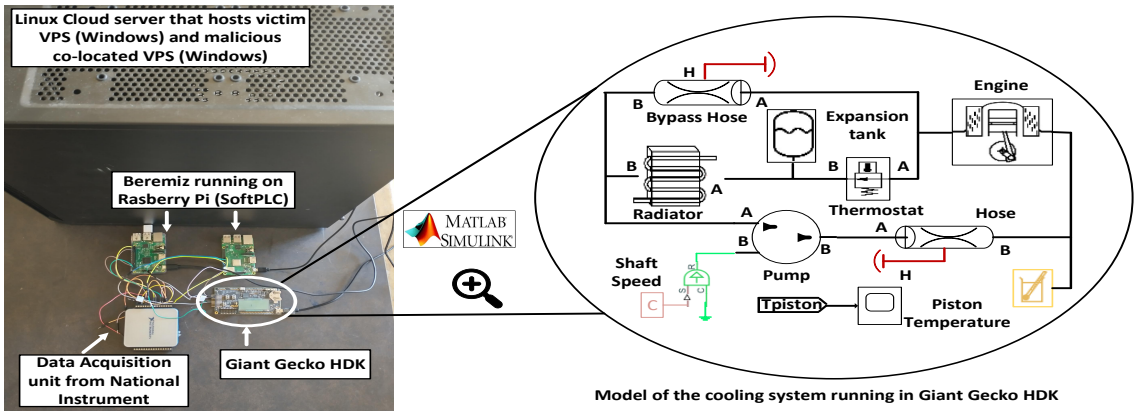

Figure 6: Hardware-in-the-Loop (HIL) testbed to evaluate the attack model-*BayesMem* in a typical engine cooling system.

where a bit flip is more probable. After profiling, the attacker has aggressor/victim addresses in hand.

The next step is to place the target *victim page* (i.e., page aligned .bss section of the target control DLL) in one of these vulnerable pages. This memory placement must be done for a successful bit-flip in the target *victim page*. This process is discussed next.

5.3.2 ***Placing the target victim page in the vulnerable location.*** As the attacker has aggressor/vulnerable addresses from the profiling step, the attacker places the *attack page* in the vulnerable addresses first (Fig. 5(B)). When the target victim VPS starts, the target victim page is merged with the attacker's provided *attack page* using the memory deduplication process (Section 5.2). Therefore, after merging with the *attack page*, as the *attack page* is used to back the memory of the *victim page*, then, in effect, the attacker controls the physical memory location of the *victim page*. As the *attack page* is placed in the vulnerable addresses for possible bit-flip, then, in effect, the target *victim page* is also placed in the same vulnerable location for possible bit-flip ((Fig. 5(C)).

5.3.3 ***Rowhammering on the aggressor rows.*** From the profiling step, the attacker knows the aggressor rows for the vulnerable memory locations. After placing the *attack page* in one of the vulnerable locations, the attacker hammers again on the aggressor rows corresponding to that vulnerable location ((Fig. 5(C)). This results in bit-flips in the *attack page* that in effect changes the value of the control programming and supervisory control related variables in the .bss section of the target control DLL.

6 HARDWARE-IN-THE-LOOP TESTBED

In this section, we prepare a testbed to evaluate the *BayesMem* on a practical ICS. We choose a scaled-down model of a practical engine cooling system of a thermo-electric plant as our target ICS. To avoid safety concerns related to high-speed rotary components and toxic chemicals of a cooling system, we create a Hardware-in-the-Loop (HIL) testbed of the engine cooling system in the lab setup (Fig. 6). A brief introduction of a typical engine cooling system is already given in Section 3.4. The steps of creating the HIL testbed are discussed below.

6.1 Modelling a cooling system

A model of a typical engine cooling system [6] is shown in Fig. 6. A typical engine cooling system has pumps, hoses, bypass hoses, radiator, thermostat, and temperature sensors. We first model the cooling system in MATLAB/Simulink. Next, we use *Simulink Coder*^{TM} to generate standalone C and C++ code from the Simulink model of the cooling system. After generating C and C++ code, we deploy the cooling system in an ARM Cortex-M3 CPU based platform having a 32 MHz clock. We use Giant Gecko Hardware Development Kit (HDK) by Silicon Labs as the implementation platform and use C/C++ compiler toolchain (i.e., Simplicity Studio) for building machine code specific to the HDK.

The modeled cooling system running on the HDK requires the following three variables for control programming and supervisory controls as inputs, namely `solenoidstate_Bypasshose`, `solenoidstate_Hose`, and `state_Thermostat`. The variables `solenoidstate_Bypasshose` and `solenoidstate_Hose` control the coolant flow through the two hoses, and the variable `state_Thermostat` controls the state of the thermostat. The possible key values of the `solenoidstate_Bypasshose` and `solenoidstate_Hose` are {0% flow, 25% flow, 50% flow, 75% flow, 100% flow} and, the possible key values of the `state_Thermostat` is {on, off}. These state variables can be changed by using our attack model (see Section 7).

6.2 Connecting the cooling system with PLC

The modeled cooling system in the HDK is connected with a PLC using IEC 61158 standard protocols, such as Modbus. The PLC is created using Beremiz [72] on Rasberry Pi. Beremiz is an integrated development environment for machine automation. It conforms to IEC-61131 standard and turns Rasberry Pi into a softPLC. Beremiz includes tools to create HMI and to connect softPLC to the existing HDK and clouds for supervision. It is important to note that Beremiz's real-time engine performance is compared to CODESYS real-time engine. Hence, Beremiz's real-time engine performance is enough to use in real field. Therefore, the use of softPLC in our HIL testbed highlights the real world scenario of a practical ICS [27].

6.3 Connecting the PLC with clouds

The cloud server has Intel CPU i7-6900K with 8 cores and 64GiB of RAM. The cloud server runs on Ubuntu Server 14.04.2 LTS x86_64.

The KVM is kept at its default configuration. The parameters for KSM (see Section 5.2.2) are also kept at their default settings. *All VPSs run with Windows 10* and have 2 GiB of main memory. *The idea of BayesMem is equally applicable to the Linux VPSs with .so file [17] of cloud protocols.* The victim VPS is using cloud protocol (i.e., MQTT) to communicate with the PLC using CODESYS.

7 ATTACK MODEL EVALUATION

In this section, we evaluate our proposed *BayesMem* in our HIL testbed and discuss the outcomes of the attack.

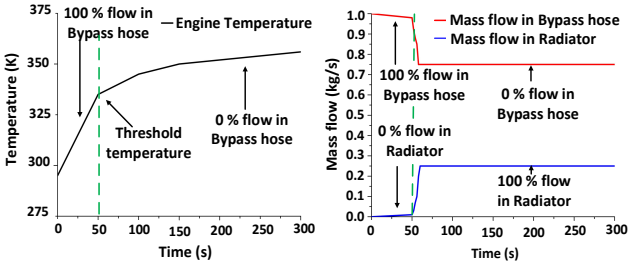

Figure 7: Normal operating condition of a cooling system.

7.1 Attacking the `solenoidstate_Bypasshose`

Fig. 7 shows the normal operating condition of the engine cooling system before the *BayesMem* is activated. In normal operating conditions, the temperature sensor of the HIL cooling system periodically sends the temperature information of the engine to the connected PLC, and the PLC sends the information to the victim VPS in the cloud for supervision. If the temperature of the engine is below a threshold, the victim VPS sends two control programming and supervisory commands to the cooling system through the .bss section of the target control DLL file of cloud protocols. These two commands are: `state_Thermostat = off` and `solenoidstate_Bypasshose = 100% flow`. These two commands dictate that maximum amount of water coolant (i.e., 100% flow) flows through the bypass hose, and minimum amount of water coolant flows through the radiator (i.e., 0% flow). This is shown in Fig. 7 before the green dashed line. When the temperature exceeds the threshold, the victim VPS sends again two commands to the cooling system through the .bss section of the target control DLL file of cloud protocols. These two commands are: `state_Thermostat = on` and `solenoidstate_Bypasshose = 0% flow`. These two commands dictate that minimum coolant flows through the bypass hose, and maximum coolant flows through the radiator. This is shown in Fig. 7 after the green dashed line. When the water coolant starts flowing through the radiator, the temperature of the engine starts decreasing. This process keeps the engine cool and maintains the normal engine condition by keeping its temperature within the limit. In case of any malfunction of the cooling system may increase the engine temperature, and as a result, the heated engine may shut down itself (i.e., DoS) to protect it from unwanted damage.

Fig. 8 shows the consequences of our attack model on the engine cooling system. After successfully duplicating the .bss section of the target control DLL file, the attacker can use his malicious co-located VPS to merge his *attack page* with the *victim page* using default on memory deduplication feature of the KVM. After a successful deduplication process, the attacker can initiate the Rowhammer on

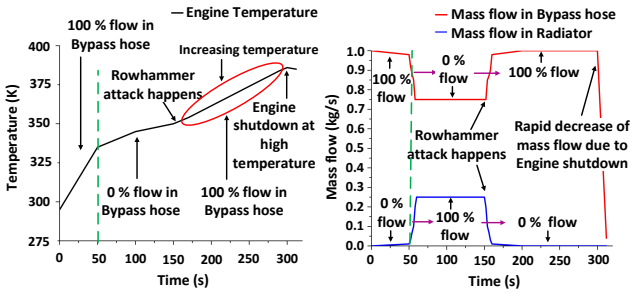

Figure 8: Causing engine shutdown using our *BayesMem*.

the *attack page*, which contains the information of the state variable `solenoidstate_Bypasshose`. We know from Section 5.1.6 that the values of `solenoidstate_Bypasshose` can be within {0,1,2,3,4}, where {0 = 0% flow, 1 = 25% flow, 2 = 50% flow, 3 = 75% flow, 4 = 100% flow}. A single bit-flip (i.e., $0 \longrightarrow 1$, or $1 \longrightarrow 0$) by the Rowhammer attack can change the value of `solenoidstate_Bypasshose` from {0 = 0% flow} to either {1 = 25% flow}, or {2 = 50% flow}, or {4 = 100% flow}. For example, Fig. 8 shows that the Rowhammer bit-flip from $0 \longrightarrow 1$ changes the `solenoidstate_Bypasshose` from {0 = 0% flow} to {4 = 100% flow} between 150 to 200 seconds. This causes maximum of the coolant (i.e., 100% flow) to flow through the bypass hose, and as a result, no water flows through the radiator. This causes the obstruction of radiation from the water coolant and the temperature of the engine starts increasing. When the temperature exceeds more than a threshold value, the engine shuts down itself to prevent any further damage. For a thermo-electric power plant, this can cause system shutdown resulting in system failure (i.e., DoS). This indicates how severe the consequences of the *BayesMem* could be in the context of ICSs.

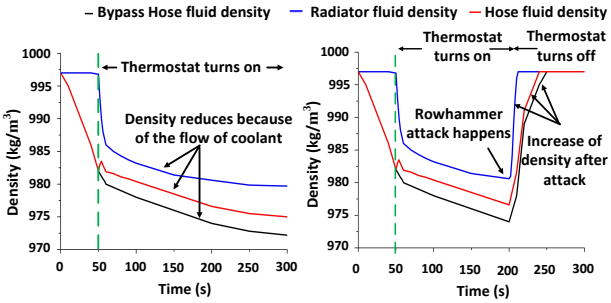

Figure 9: Changing fluid density using the *BayesMem*.

7.2 Attacking the `state_Thermostat`

Similar to attacking the parameter `solenoidstate_Bypasshose` in the .bss section of the target control DLL, the attacker can attack the `state_Thermostat` using the *BayesMem*. The variable `state_Thermostat` has two values: {0, 1}, where 0 = off, and 1 = on, or vice-versa. By randomly flipping bits using our attack model, the attacker can turn on or off the thermostat. An outcome of Rowhammering the parameter `solenoidstate_Bypasshose` is shown in Fig. 9. In normal operating condition (i.e., before the green dashed line in Fig. 9(left)), when the thermostat is off, the coolant is not flowing through the radiator. Hence, radiator fluid density is high. When the temperature of the engine exceeds a threshold (i.e., after the green dashed line in Fig. 9(left)), the thermostat turns on, and

the coolant starts flowing through the radiator. Hence, the fluid density of the radiator starts to reduce. In Fig. 9(right), the Rowhammer bit-flip (i.e., 1 → 0) happens around 200 seconds that turns off the thermostat when the thermostat is supposed to be turned on. This causes the water coolant to stop flowing through the radiator. As a result, the fluid density in the radiator is increased along with the increase of the fluid density in the bypass hose and hose. This, in effect, causes an increase of temperature in the engine in a similar way described in Section 7.1. This can eventually lead the system to shutdown state while causing system failure.

7.3 Evaluation for different cloud protocols

In this section, we evaluate the *BayesMem* for different variants of cloud protocol, and the results are tabulated in Table 2. As our attack model does not require any software bug present in the implementation of cloud protocols, state-of-the-art variants of cloud protocols should be vulnerable to our attack model. To support this claim, we implement a total of five variants of the MQTT protocol in our testbed and find that all are equally vulnerable that proves the generalization of our attack model in ICSs.

Table 2: Cloud protocol variants vulnerable to *BayesMem*

Sl.	Cloud protocol variants	Vulnerability
1	EMQ X Broker [4]	✓
2	Mosquitto [7]	✓
3	MQTT-C [8]	✓
4	eMQTT5 [5]	✓
5	wolfMQTT [10]	✓

7.4 Adversarial control using BayesMem

As the *BayesMem* can predictively cause random bit-flip in the .bss section of the target control DLL, the attacker can use our *BayesMem* for adversarial control that is shown in Fig. 10. The attacker can change the flow through the bypass hose from {4 = 100% flow} to {0 = 0% flow} by 1 → 0 bit flip, from {0 = 0% flow} to {1 = 25% flow} by 0 → 1 bit flip, and from {1 = 25% flow} to {3 = 75% flow} by 0 → 1 bit flip. The adversarial control over the flow of coolant through bypass hose in effect gives control over the engine temperature. If the attacker can predictively control the engine temperature using our *BayesMem*, the attacker can shut down the cooling system at a specific time. This adversarial control makes the *BayesMem* stronger compared to [17, 59] in ICSs.

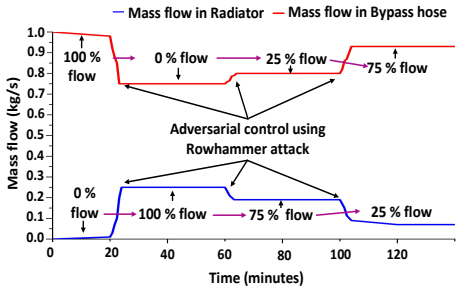

Figure 10: Profiling time for different number of VPSs.

7.5 Profiling time in our testbed

We evaluate the profiling time (see Section 5.3) for different number of VPSs in the cloud (Fig. 11). Searching for more vulnerable locations for the Rowhammer requires more time. With the increase of VPSs, this profiling time increases due to more memory pressure in the system memory. Fig. 11 shows the profiling time for 1, 3, and 6 VPSs in the same cloud.

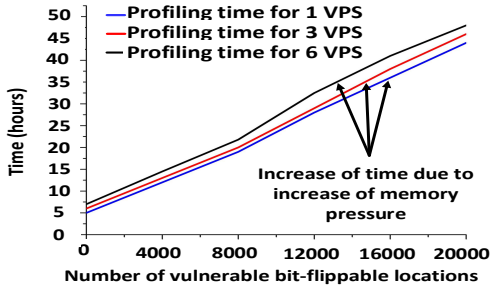

Figure 11: Profiling time for different number of VPSs.

7.6 Attack time

Here, we define attack time as how much time it takes to cause a bit flip in the .bss section of the target control DLL. Attack time is the summation of the memory deduplication time and the Rowhammer implementation time. The exact time required for memory deduplication can be calculated using the timing side-channel [23]. Moreover, the maximum value of memory deduplication time is the amount of time needed to scan all the memory of the co-located VPSs in the cloud. Here, for simplicity, we assume that deduplication happens within this maximum time frame, and hence, we consider this maximum time as the memory deduplication time. The memory deduplication time depends upon the parameters `pages_to_scan` and `sleep _millisec`. In default configuration, `pages_to_scan` = 100 and `sleep _millisec` = 20. Therefore, Linux/KSM can scan 1000 pages/second, which results in a total scan time of almost 5 minutes per 1GiB of main memory [54]. As the victim VPS has a main memory of 2 GiB, it should take approximately 10 minutes to scan all the pages in main memory of a VPS. In our testbed, the memory deduplication takes approx. 13 minutes, and the Rowhammering process takes approx. 51.45 seconds to complete single-sided Rowhammer for each target row. Therefore, after summing up these two figures, the total attack time is approximately 13 minutes and 52 seconds for 1 target VPS in the cloud.

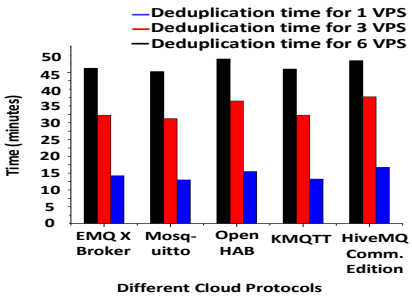

Figure 12: Deduplication time for different protocols.

Fig. 12 shows the memory deduplication time for five variants of MQTT cloud protocol for 1, 3, and 6 VPSs. This figure indicates that all five variants of the cloud protocol give almost equal deduplication time. As the addition of a VPS increases the scannable memory locations, the deduplication time increases with the number of co-located VPS in the cloud. The Rowhammer implementation time for a target row is almost the same for all five protocol variants.

Therefore, if we add the deduplication time from Fig. 12 with the Rowhammer implementation time (i.e., ≈ 52 seconds), we get the attack time for our attack model.

7.7 Probability of bit-flip

To increase the chance of getting bit-flip in the target row, it is a good idea to place the duplicated *attack page* in multiple vulnerable locations. This increases the probability of getting bit-flip in the *attack page*. Fig. 13 shows that placing 1 *attack page* in the vulnerable location causes around 30% probability of bit-flip, whereas 50 *attack pages* at different vulnerable locations increases the probability to 90%. We also compare the bit-flip probability among 3 variants of cloud protocols in Fig. 13. It shows that protocol variation does not impact the probability of bit-flip in the attack model.

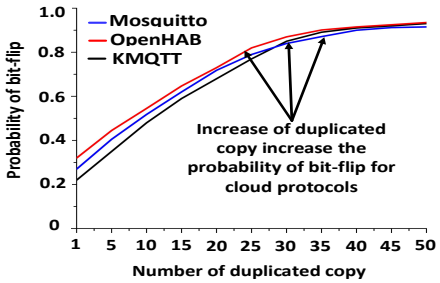

Figure 13: Probability of bit-flip with different number of duplicated copy.

8 DEFENSE

As our attack model does not require any software bug in any level of cloud protocols, a reliable defense against this type of attack should involve all of the following mitigations.

Increasing entropy in the .bss section: The most important reason that makes the attack possible is the presence of low entropy in the .bss section of the target control DLL file of cloud protocols (i.e., MQTT, AMQP). Most of the uninitialized variables of the .bss section of the target control DLL are available from cloud protocols, and control programming and supervisory controls are predictable using the Bayesian estimation. These are reasons why the attacker can successfully duplicate the .bss section with less memory and effort. To prevent the attack model to happen, the first method could be to increase entropy in the .bss section. This can be done using a random variable as an authentication variable in the .bss section. This may increase entropy, and the attacker requires a significant amount of memory and time to break this signature variable [17].

Securing cloud server from the malicious VPS: As ICSs need to support multiple automation platforms, the use of multiple VPSs cannot be avoided in Industry 4.0. However, it has to be kept in mind that unnecessary access to the cloud server, where all the VPSs are co-hosted, needs to be prevented. Any unauthorized cloud provider, or personnel, or visitor should not access the cloud server without the presence of any authorized personnel. Periodic screening by authorized personnel of the individual industry needs to be carried out to look for the presence of any unauthorized co-hosted VPS in the cloud. Any co-located VPS, which seems unnecessary or suspicious, should be considered as a security breach and should be immediately contained in the cloud.

Turning off the KSM: As our attack model depends on KSM, a basic defense could be to turn off the KSM feature. In response to prevent memory deduplication based attacks, Microsoft already disables [3] memory deduplication. Similarly, KSM is off by default in recent Linux kernel [2]. However, the KSM service, which is included in the `qemu-kvm` package, is turned on by the KVM host. The KSM can be turned off using the `ksm/ksmtuned` services in the KVM host. However, turning off the KSM may increase of memory usage in clouds. Therefore, it is not favorable by the cloud user where memory workloads are very high in ICSs [41].

Preventing Rowhammer in DRAM: If the KSM cannot be turned off, the next way to prevent our attack is to preventing bit-flip in DRAM. DRAM has built-in error-correcting codes (ECC) to prevent single bit-flip in 64-bit words [28]. But the ECC may not be enough where the Rowhammer causes multiple bit-flips in the memory [12, 48]. While only modern AMD Ryzen processors support ECC RAM in consumer hardware, Intel restricts its support to server CPUs, thus, making it unavailable in clouds [38]. To prevent Rowhammer in the DRAM, different methods can be adopted. One method is to increase (e.g., double) the refresh rate in DRAM chips [55]. This can reduce the probability of multiple bit-flips in DRAM, but causes more energy consumption and more overhead in the memory [30, 43]. Another method is to probabilistically open adjacent or non-adjacent rows, whenever a row is opened or closed [42]. An introduction of a redundant array of independent memory system (i.e., RAIM) in the server hardware can make the Rowhammer attack infeasible [52]. Another solution could be to use ANVIL [16] that relies on Intel's performance monitoring unit (PMU). Moreover, replacing traditional DRAM with LPDDR4 [15] can prevent the Rowhammer attack on future cloud networks.

9 CONCLUSION

We present an attack model-*BayesMem* that can hamper the availability and integrity of an ICS in cloud settings. We are the first to point out how the .bss section of the target control DLL file of cloud protocols is vulnerable in ICS. The *BayesMem* exploits the memory deduplication feature of the cloud that merges the attacker's provided attack page with the victim page. To create the attack page, the *BayesMem* uses a new technique that involves the *Bayesian estimation*, which results in less memory and time compared to recent works [17, 23, 59]. We show that as ICSs can be expressed as state-space models; hence, the *Bayesian estimation* is an ideal choice to be combined with the memory deduplication in cloud settings. We prepare a HIL testbed of a scaled-down model of an engine cooling system as an example of target ICS and demonstrate our attack model on this HIL testbed. We show that our attack model is effective on different variants of cloud protocols, and does not need any vulnerabilities in the cloud protocol, and works without any presence of software bug in any level of the system that proves a generalization of our attack model. We show that the *BayesMem* is capable of *adversarial control* that can cause severe consequences through *system shutdown (i.e., DoS)* in ICS. Our work is an example of an end-to-end attack that originates in the cyber domain and finally impacts the physical domain. This type of cross-domain attack draws attention to the security community for further research.

REFERENCES

[1] [n.d.]. Affordable MQTT Broker Pricing. https://www.bevywise.com/mqtt-broker/pricing.html.

[2] [n.d.]. Chapter 7. KSM. https://docs.fedoraproject.org/en-US/Fedora/18/html/Virtualization_Administration_Guide/chap-KSM.html.

[3] [n.d.]. CVE-2016-3272. Microsoft Security Bulletin MS16-092- Important. https://www.microsoft.com/en-us/msrc?rtc=1.

[4] [n.d.]. EMQ X Broker. https://www.emqx.io/downloads#broker.

[5] [n.d.]. eMQTT5. https://github.com/X-Ryl669/eMQTT5.

[6] [n.d.]. Engine Cooling System. https://www.mathworks.com/help/physmod/simscape/ug/engine-cooling-system.html.

[7] [n.d.]. mosquitto. https://mosquitto.org/.

[8] [n.d.]. MQTT-C. https://github.com/LiamBindle/MQTT-C.

[9] [n.d.]. Siemens: How to connect to a PLC with TIA Portal in a Virtual Machine. https://web.awc-inc.com/siemens-how-to-connect-to-a-plc-with-tia-portal-in-a-virtual-machine/.

[10] [n.d.]. wolfMQTT. https://github.com/wolfSSL/wolfMQTT.

[11] Ali Abbasi and Majid Hashemi. 2016. Ghost in the plc designing an undetectable programmable logic controller rootkit via pin control attack. *Black Hat Europe* 2016 (2016), 1–35.

[12] Barbara Aichinger. 2015. DDR memory errors caused by Row Hammer. In *2015 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–5.

[13] S Annapoorani, B Srinivasan, and GA Mylavathi. 2018. Analysis of various virtual machine attacks in cloud computing. In *2018 2nd international Conference on Inventive Systems and Control (ICISC)*. IEEE, 1016–1019.

[14] J.R. Appelbaum, L. Poitras, M. Rosenbach, C. Stöcker, J. Schindler, and H. Stark. 2013. Inside TAO : documents reveal top NSA hacking unit. *Der Spiegel* (29 12 2013).

[15] JEDEC Solid State Technology Association et al. 2015. Low Power Double Data 4 (LPDDR4). *JESD209-4A, Nov* (2015).

[16] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. 2016. ANVIL: Software-based protection against next-generation rowhammer attacks. *ACM SIGPLAN Notices* 51, 4 (2016), 743–755.

[17] Antonio Barresi, Kaveh Razavi, Mathias Payer, and Thomas R Gross. 2015. {CAIN}: Silently Breaking {ASLR} in the Cloud. In *9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15)*.

[18] Christoph Jan Bartodziej. 2017. The concept industry 4.0. In *The concept industry 4.0*. Springer, 27–50.

[19] Anomadarshi Barua and Mohammad Abdullah Al Faruque. 2020. Hall Spoofing: A Non-Invasive DoS Attack on Grid-Tied Solar Inverter. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 1273–1290.

[20] Zachry Basnight, Jonathan Butts, Juan Lopez Jr, and Thomas Dube. 2013. Firmware modification attacks on programmable logic controllers. *International Journal of Critical Infrastructure Protection* 6, 2 (2013), 76–84.

[21] Dillon Beresford. 2011. Exploiting siemens simatic s7 plcs. *Black Hat USA* 16, 2 (2011), 723–733.

[22] Alexander Bolshev, Jason Larsen, Marina Krotofil, and Reid Wightman. 2016. A rising tide: Design exploits in industrial control systems. In *10th {USENIX} Workshop on Offensive Technologies ({WOOT} 16)*.

[23] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2016. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 987–1004.

[24] Alvaro Cardenas, Saurabh Amin, Bruno Sinopoli, Annarita Giani, Adrian Perrig, Shankar Sastry, et al. 2009. Challenges for securing cyber physical systems. In *Workshop on future directions in cyber-physical systems security*, Vol. 5. Citeseer.

[25] Chao-Rui Chang, Jan-Jan Wu, and Pangfeng Liu. 2011. An empirical study on memory sharing of virtual machines for server consolidation. In *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 244–249.

[26] Seungoh Choi, Jongwon Choi, Jeong-Han Yun, Byung-Gil Min, and HyoungChun Kim. 2020. Expansion of {ICS} testbed for security validation based on {MITRE} atT&Ck techniques. In *13th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 20)*.

[27] S Chu, E Shin, S Lee, K Lee, and K Nam. 2015. Can we use beremiz real-time engine for robot programmable logic controller?. In *2015 International Automatic Control Conference (CACS)*. IEEE, 114–118.

[28] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. 2019. Exploiting correcting codes: On the effectiveness of ECC memory against Rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 55–71.

[29] Alessandro Di Pinto, Younes Dragoni, and Andrea Carcano. 2018. TRITON: The first ICS cyber attack on safety instrument systems. In *Proc. Black Hat USA*. 1–26.

[30] Philip G Emma, William R Reohr, and Mesut Meterelliyoz. 2008. Rethinking refresh: Increasing availability and reducing power in DRAM for cache applications. *IEEE micro* 28, 6 (2008), 47–56.

[31] Bernard Friedland. 2012. *Control system design: an introduction to state-space methods*. Courier Corporation.

[32] Luis Garcia, Ferdinand Brasser, Mehmet Hazar Cintuglu, Ahmad-Reza Sadeghi, Osama A Mohammed, and Saman A Zonouz. 2017. Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit.. In *NDSS*.

[33] Ang Chee Kiong Gary and Utomo Nugroho Prananto. 2017. Cyber security in the Energy World. In *2017 Asian Conference on Energy, Power and Transportation Electrification (ACEPT)*. IEEE, 1–5.

[34] Reinhard Geissbauer, Jesper Vedso, and Stefan Schrauf. 2016. Industry 4.0: Building the digital enterprise. *Retrieved from PwC Website: https://www.pwc.com/gx/en/industries/industries-4.0/landing-page/industry-4.0-building-your-digital-enterprise-april-2016.pdf* (2016).

[35] Andrew Ginter. 2017. The top 20 cyber attacks against industrial control systems. *White Paper, Waterfall Security Solutions* (2017).

[36] Omid Givehchi, Jahanzaib Imtiaz, Henning Trsek, and Juergen Jasperneite. 2014. Control-as-a-service from the cloud: A case study for using virtualized PLCs. In *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*. IEEE, 1–4.

[37] Thomas Goldschmidt, Mahesh Kumar Murugaiah, Christian Sonntag, Bastian Schlich, Sebastian Biallas, and Peter Weber. 2015. Cloud-based control: A multi-tenant, horizontally scalable soft-PLC. In *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 909–916.

[38] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom. 2018. Another Flip in the Wall of Rowhammer Defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*. 245–261. https://doi.org/10.1109/SP.2018.00031

[39] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer. js: A remote software-induced fault attack in javascript. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 300–321.

[40] Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage, Alex C Snoeren, George Varghese, Geoffrey M Voelker, and Amin Vahdat. 2010. Difference engine: Harnessing memory redundancy in virtual machines. *Commun. ACM* 53, 10 (2010), 85–93.

[41] Gangyong Jia, Guangjie Han, Joel JPC Rodrigues, Jaime Lloret, and Wei Li. 2015. Coordinate memory deduplication and partition for improving performance in cloud computing. *IEEE Transactions on Cloud Computing* 7, 2 (2015), 357–368.

[42] Dae-Hyun Kim, Prashant J Nair, and Moinuddin K Qureshi. 2014. Architectural support for mitigating row hammering in DRAM memories. *IEEE Computer Architecture Letters* 14, 1 (2014), 9–12.

[43] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 361–372.

[44] Johannes Klick, Stephan Lau, Daniel Marzin, Jan-Ole Malchow, and Volker Roth. 2015. Internet-facing PLCs-a new back orifice. *Blackhat USA* (2015), 22–26.

[45] David Kushner. 2013. The real story of stuxnet. *ieee Spectrum* 3, 50 (2013), 48–53.

[46] Reinhard Langmann and Leandro F Rojas-Peña. 2016. A PLC as an Industry 4.0 component. In *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. IEEE, 10–15.

[47] Reinhard Langmann and Michael Stiller. 2019. The PLC as a smart service in industry 4.0 production systems. *Applied Sciences* 9, 18 (2019), 3815.

[48] Mark Lanteigne. 2016. How rowhammer could be used to exploit weaknesses in computer hardware. *SEMICON China* (2016).

[49] Robert M Lee, Michael J Assante, and Tim Conway. 2014. German steel mill cyber attack. *Industrial Control Systems* 30 (2014), 62.

[50] Yao Liu, Peng Ning, and Michael K Reiter. 2011. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)* 14, 1 (2011), 1–33.

[51] Stephen McLaughlin and Saman Zonouz. 2014. Controller-aware false data injection against programmable logic controllers. In *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 848–853.

[52] Patrick J Meaney, Luis Alfonso Lastras-Montaño, Vesselina K Papazova, Eldee Stephens, JS Johnson, Luiz C Alves, James A O'Connor, and William J Clarke. 2012. IBM zEnterprise redundant array of independent memory subsystem. *IBM Journal of Research and Development* 56, 1.2 (2012), 4–1.

[53] Microsoft. [n.d.]. Large-Page Support. https://docs.microsoft.com/en-us/windows/win32/memory/large-page-support.

[54] Konrad Miller, Fabian Franz, Marc Rittinghaus, Marius Hillenbrand, and Frank Bellosa. 2013. {XLH}: More effective memory deduplication scanners through cross-layer hints. In *2013 {USENIX} Annual Technical Conference ({USENIX}{ATC} 13)*. 279–290.

[55] Onur Mutlu. 2017. The RowHammer problem and other issues we may face as memory becomes denser. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 1116–1121.

[56] Nicholas Petreley. 2004. Security report: Windows vs linux. *The Register* 22 (2004), 1–24.

[57] Matt Pietrek. 1994. Peering inside the PE: a tour of the win32 (R) portable executable file format. *Microsoft Systems Journal-US Edition* 9, 3 (1994), 15–38.

[58] Noëlle Rakotondravony, Benjamin Taubmann, Waseem Mandarawi, Eva Weishäupl, Peng Xu, Bojan Kolosnjaji, Mykolai Protsenko, Hermann De Meer, and Hans P Reiser. 2017. Classifying malware attacks in IaaS cloud environments. *Journal of Cloud Computing* 6, 1 (2017), 26.

[59] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. 2016. Flip feng shui: Hammering a needle in the software stack. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 1–18.

[60] Johannes Reichl, Michael Schmidthaler, and Friedrich Schneider. 2013. The value of supply security: The costs of power outages to Austrian households, firms and the public sector. *Energy Economics* 36 (2013), 256–261.

[61] Jarno Ruotsalainen. 2018. *Hardening and architecture of an industrial control system in a virtualized environment.* Master's thesis.

[62] Anam Sajid, Haider Abbas, and Kashif Saleem. 2016. Cloud-assisted IoT-based SCADA systems security: A review of the state of the art and future challenges. *IEEE Access* 4 (2016), 1375–1384.

[63] Bianca Scholten. 2007. *The road to integration: A guide to applying the ISA-95 standard in manufacturing.* Isa.

[64] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. *Black Hat* 15 (2015), 71.

[65] Bill Snyder. 2014. Snowden: The NSA planted backdoors in cisco products. *InfoWorld* 15 (2014).

[66] Ralf Spenneberg, Maik Brüggemann, and Hendrik Schwartke. 2016. Plc-blaster: A worm living solely in the plc. *Black Hat Asia* 16 (2016), 1–16.

[67] Jonathan Stidham. 2001. Can hackers turn your lights off: The vulnerability of the US power grid to electronic attack. *SANS Institute InfoSec Reading Room* (2001).

[68] Pawel Swierczynski, Marc Fyrbiak, Philipp Koppe, Amir Moradi, and Christof Paar. 2017. Interdiction in practice—Hardware Trojan against a high-security USB flash drive. *Journal of Cryptographic Engineering* 7, 3 (2017), 199–211.

[69] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Throwhammer: Rowhammer attacks over the network and defenses. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 213–226.

[70] Michael Tiegelkamp and Karl-Heinz John. 1995. *IEC 61131-3: Programming industrial automation systems.* Vol. 14. Springer.

[71] Lonneke Van der Velden. 2015. Leaky apps and data shots: Technologies of leakage and insertion in NSA-surveillance. *Surveillance & Society* 13, 2 (2015), 182–196.

[72] Jiang Wan, Arquimedes Canedo, and Mohammad Al Faruque. 2015. Model-based design of time-triggered real-time embedded systems for digital manufacturing. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. 295–296.

[73] Ercan Nurcan Ylmaz, Bünyamin Ciylan, Serkan Gönen, Erhan Sindiren, and Gökçe Karacayılmaz. 2018. Cyber security in industrial control systems: Analysis of DoS attacks against PLCs and the insider effect. In *2018 6th International Istanbul Smart Grids and Cities Congress and Fair (ICSG)*. IEEE, 81–85.

10 APPENDIX

10.1 PLCs and Industry 4.0

As Programmable Logic Controllers (PLCs) are one of the key ingredients of ICSs, Industry 4.0 drives new approaches in the PLC design [46]. Historically, PLCs were originally designed to support three main concepts, namely programmability, reliability and, real-time response. Different programmable platforms, such as microprocessors, FPGAs, Hard Processor Systems (HPS) are chosen to support programmability in PLCs, as these hardware are programmable in run time in onsite industrial premises following the IEC 61131 key programming standard. Moreover, the standard IEC 61131 is developed in such a way to ensure reliability and real-time response by treating PLCs as logically independent with its own, individual configuration.

An architecture like this may provide predictable outcomes with a low likelihood of failure, but on the flip-side, it turns out to be progressively lumbering when confronted with developments in IIoTs that require noteworthy adaptability. The IIoTs require the cooperation of individual PLCs on a much deeper level. Moreover, individual PLCs likewise need to work considerably more closely with each other within the industry and remotely, to the web-server and cloud, for instance.

10.2 PLCs interface for basic web technologies

Todays PLCs have an interface that can be connected to a web-server via a device gateway. The device gateway is integrated into the existing PLC controllers that can support web-compatible protocol required for communication with the IP network. The web-server can connect to the PLC controller using HTML pages that enables a browser-based communication and diagnosis of the PLCs. The web-server can read and write control variables and collect measurement data from PLCs, with restrictions. Sometimes, this web-server is referred to as a "thin server" having enough computing resources to support local client/server network architecture.

10.3 Implemented protocols

Different protocols exist in different layers of ICSs. Typically IEC 61158 standard protocols are used in communication between PLCs and sensors. Here PLCs act as master, and sensors act as slaves. IEC 61158 standard contains a total of nine protocols: Fieldbus, Common Industrial Protocol (CIP), PROFIBUS/PROFINET, P-NET, WorldFIP, INTERBUS, HART, CC-Link, and SERCOS. These same protocols can be used between PLCs (master) and cloud adapters (slave). RS-232 or RS-485 based Fieldbus has multiple variants. Modbus and DNP3 are two of the most popular variants. They are widely adopted as a de facto standard and has been modified further over the years into several distinct variants. Moreover, Ethernet-based protocols, such as PROFINET, CC-LINK, SERCOS have lower latency than

the Fieldbus protocols. Hence, these are preferred over Fieldbus in today's ICSs.

As already discussed in Section 3.2, the program for basic functions and supervisory controls are implemented in clouds or in web-server. These control programs are implemented using service functions in PLC controllers. A standardized protocol named Device Protocol for Web Services (DPWS) enables service-based access to PLC controllers. As mentioned earlier in Section 3.1, MQTT and AMQP are used to communicate with PLCs from clouds using an IoT gateway.