Artifact Evaluated
NDSS
Available
Functional
Reproduced

Compromising Industrial Processes using Web-Based Programmable Logic Controller Malware

Ryan Pickren, Tohid Shekari, Saman Zonouz, Raheem Beyah
Georgia Institute of Technology
{rpickren3, tshekari3, saman.zonouz, ab207}@gatech.edu

***Abstract*—We present a novel approach to developing programmable logic controller (PLC) malware that proves to be more flexible, resilient, and impactful than current strategies. While previous attacks on PLCs infect either the control logic or firmware portions of PLC computation, our proposed malware exclusively infects the web application hosted by the emerging embedded webservers within the PLCs. This strategy allows the malware to stealthily attack the underlying real-world machinery using the legitimate web application program interfaces (APIs) exposed by the admin portal website. Such attacks include falsifying sensor readings, disabling safety alarms, and manipulating physical actuators. Furthermore, this approach has significant advantages over existing PLC malware techniques (control logic and firmware) such as platform independence, ease-of-deployment, and higher levels of persistence. Our research shows that the emergence of web technology in industrial control environments has introduced new security concerns that are not present in the IT domain or consumer IoT devices. Depending on the industrial process being controlled by the PLC, our attack can potentially cause catastrophic incidents or even loss of life. We verified these claims by performing a Stuxnet-style attack using a prototype implementation of this malware on a widely-used PLC model by exploiting zero-day vulnerabilities that we discovered during our research[1]. Our investigation reveals that every major PLC vendor (80% of global market share [1]) produces a PLC that is vulnerable to our proposed attack vector. Lastly, we discuss potential countermeasures and mitigations.**

I. Introduction

Industrial Control Systems. Industrial control systems (ICSs) can be abundantly found in many critical infrastructure sectors including the electric grid, pharmaceutical, and manufacturing industries [2]. ICSs integrate IT capabilities such as monitoring and communication with physical system control [3]. This integration has resulted in today's "smart" industrial technologies such as the smart electric grid and smart manufacturing, which provide operational convenience and increased sustainability [2]. Unfortunately, the rise of smart industrial technologies has also expanded the ICS attack surface. The ICS cybersecurity market is projected to grow from $16.7B USD in 2022 to $23.7B USD by 2027 [4].

Programmable Logic Controllers. Programmable logic controllers (PLCs) are considered the core component of ICSs because they monitor sensors and manipulate actuators using local automatic control. PLCs take raw data from sensors, perform calculations based on control logic, and send commands to physical actuators to control the real-world processes [5]. Process Engineers are responsible for programming PLCs using an IEC 61131-3 compliant control language such as ladder diagram (LD) through proprietary engineering software on an engineering workstation (EWS). These EWSs compile the written PLC programs into binary executables that can be run by the processors of the PLCs in a user-code sandbox. PLCs also utilize a firmware layer to provide the low-level interface between the hardware and the control logic.

In recent years, this firmware layer has also begun to include a customizable embedded webserver, which provides customers with a convenient method for accessing both administrative configurations and physical process monitoring and control via standard web browsers. This emerging trend has transformed the ICS ecosystem in profound and irreversible ways. Unfortunately, our research has uncovered that this transformation has also introduced new web-oriented security concerns that are specific to ICS environments. These security concerns are not simply the standard baggage caused by web technology in the IT domain, but rather are issues unique to the conditions caused by industrial control environments and hierarchical network architecture (i.e. Purdue Enterprise Reference Architecture [PERA]).

Real-World Attacks. While it may seem that ICSs, and PLCs in particular, are impossible targets for attackers because they are mostly disconnected from the public Internet, this notion is simply not true as demonstrated by the emergence of recent severe attacks in this domain [6]–[11]. The Stuxnet [7], [8] worm targeted Iranian Uranium enrichment facilities in 2010. A few years later in 2015 and 2016, the Ukrainian power grid experienced two widespread blackouts caused by the BlackEnergy 3 malware [9], [10]. More recently, Triton (i.e., "the world's most murderous malware") [11] targeted a Saudi petrochemical plant in 2017, where the malware disabled safety instrumented systems (SISs) of the plant to cause sabotage in the underlying physical process. These real-world examples of successful ICS attacks show that persistent bad actors are able to infiltrate segregated industrial networks

[1] These issues were disclosed to the vendor and fixed as CVE-2022-45137, CVE-2022-45138, CVE-2022-45139, and CVE-2022-45140.

Network and Distributed Systems Security (NDSS) Symposium 2024
26 February – 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23049
www.ndss-symposium.org

using a variety of techniques (e.g., Out of Band malware infections, malicious USB drives, insider threats, etc).

Existing PLC Malware and Shortcomings. The ultimate goal of many ICS attacks is to somehow infect PLCs with malicious software (i.e., "PLC malware"). This is usually done via the final payload of an advanced ICS attack (e.g., Stuxnet [7]). PLCs are typically thought to only run software at two different levels: firmware and control logic. This preconception has inspired numerous research works and real-world attacks exploring malware implemented at both levels with firmware rootkits (e.g., HARVEY [12]) implemented in assembly code and control logic malware (e.g., LLB [13]) implemented in LD or another PLC control language. Fortunately for ICSs, both of these approaches have substantial drawbacks that make them impractical for casual adversaries. Such drawbacks include infection difficulty (e.g., requiring physical or network access), fully-offline operation (e.g., trapped in segregated industrial networks), platform dependence (e.g., requiring model-specific payloads), and low-persistence (e.g., trivially erased with factory resets).

Proposed Web-Based PLC Malware. In this paper, we introduce a new strategy for developing PLC malware that infects the front-end web layer with malicious JavaScript code. This malware, which we call Web-Based (WB) PLC malware, is fundamentally different than prior approaches and overcomes many of the drawbacks of those strategies. Our WB PLC malware resides in PLC memory, but ultimately gets executed client-side by various browser-equipped devices throughout the ICS environment. From there, the malware uses ambient browser-based credentials to interact with the PLC's legitimate web APIs to attack the underlying real-world machinery. Our paper demonstrates that this type of malware is much easier to deploy against a real-world ICS, is capable of online operations, is largely platform independent, and achieves extremely high levels of persistence.

Contribution. Our main contributions are as follows:

1) We introduce the concept of WB PLC malware, which oftentimes proves to be more flexible, resilient, and impactful than prior PLC malware infections (control logic or firmware);
2) We developed a cross-platform framework that outlines how methodically compromising embedded PLC webservers can sabotage industrial processes;
3) We implemented a prototype of WB PLC malware, dubbed *IronSpider*, on a widely-used PLC model to show its effectiveness compared to existing PLC malware techniques in a Stuxnet-style attack;
4) We propose practical countermeasures to mitigate the risk of the developed attacks or significantly reduce their damaging consequences;

Furthermore, we experimentally verified that every PLC model included in our study, namely Siemens S7-1200, Schneider TM241C, Allen-Bradley MicroLogix 1400, Mitsubishi MELSEC-F, GE/Emerson RX7i, and WAGO 750 (these vendors account for over 80% of global PLC market share [1]), is vulnerable to some sort of WB PLC malware. The rest of this paper is organized as follows. Section II discusses background information about ICS networks and PLCs operations. Section III introduces our proposed WB PLC malware and compares it to related work. The details about our multi-stage attack method are given in Section IV. Section V presents our experimental results and performance evaluations. Finally, Section VI is the conclusion of the paper.

II. Background

IoT vs PLC Embedded Webservers. Prior work has shown that consumer "Internet of Things" (IoT) devices such as printers and home routers may also incorporate embedded webservers for ad-hoc administrative control [14]. While these household embedded webservers do introduce their own security concerns, they are primarily limited to basic entry-point attacks such as weak authentication and default passwords because these webservers typically only host simplistic vendor-authored setup wizards used for an initial 1-time configuration [15]. On the contrary, PLC embedded webservers are used for continuous monitoring and control via programmable web applications consumed by dedicated client hardware (e.g., WAGO e!DISPLAY 7300 Microbrowser). This unique utilization of embedded web technologies introduces a new attack vector not applicable to consumer devices - *persistent and covert front-end code execution*. In the ICS domain, malicious front-end code can be pushed to a programmable controller through the legitimate channels discussed in Section IV-B and perpetually executed on a multitude of browser-equipped devices throughout the industrial network [16]. Table I summarizes the key differences between embedded web technology in the IoT devices vs PLCs and illustrates how PLCs are uniquely susceptible to web-based malware attacks.

TABLE I: Embedded Webservers in IoT Devices vs PLCs

	Webserver Purpose	**Front-End Code Author**	**Web Client**	**Web Attack Vector**
IoT	Initial 1-time Setup	Device Manufacturer	Browser on Personal Device	Standard Web Vulnerabilities
PLC	Continuous Monitoring & Control	Customer & Device Manufacturer	Dedicated Hardware	Persistent and Covert Code Execution

Edge Device ICS Webservers. Recently, Sasaki et al. demonstrated that edge device ICS webservers, specifically *Remote Management Devices*, are often Internet-facing and lack basic security measures [17]. In this work, Sasaki et al. scanned the public Internet to find Remote Management Device web portals (these servers are often intentionally Internet-facing since they are meant to be used *remotely*) and performed successful penetration tests on interfaces made by various mid-sized vendors. Their work suggests that the ICS industry is adopting web-based technologies without a solid understanding of their exposed attack surface. Our work builds on their paper by exploring how the customizable front-end of PLC webservers can be a surprisingly ideal environment to run ICS malware. As discussed below, PLCs differ from Remote Management Devices in that they operate deep within the private ICS network and their programmable front-ends are perpetually rendered inside the ICS network. Remote Management Devices, on the other hand, are edge devices that operate on the perimeter of ICS networks and host static front-ends that get consumed outside the network (making them a great entry-point into the network, as shown by Sasaki et al., but an infeasible environment for web-based malware).

ICS Network Architecture. A basic understanding of ICS network architecture is needed to appreciate the unique characteristics of our proposed malware. Figure 1.A shows the most common architecture, PERA, which represents the various layers of ICS networks, separated into functionally distinct groups [18]. At the top, we have Level 4/5 where the primary business functions occur. This layer is typically contained within the IT/business network and is connected to the public Internet through a firewall. This level provides business direction and orchestrates manufacturing operations. In Level 3, the production workflow is managed in remote control centers. This layer consists of data historians to record operations data as well as EWSs and remote Human-Machine Interfaces (HMIs) that program and monitor local controllers (e.g., PLCs). This layer poses a significant challenge to network isolation because it often contains "dual-homed" devices with simultaneous connectivity to both the IT/Business network and the segregated industrial network [19].

In Level 2, Supervisory Control and Data Acquisition (SCADA) software and local HMIs are located within a geographically close distance to the physical plant. These HMIs are again used to monitor and control the underlying physical processes of PLCs. Devices in this layer and below are exclusively connected to the industrial network, thus typically do not have any connection to the public internet. In Level 1, local controllers such as PLCs perform sensing and manipulation of physical processes using sensors and actuators with a closed-loop control structure. Finally, Level 0 defines the actual physical processes.

Emerging PLC Web Applications. PLCs run a variety of network services in their firmware layer such as a Modbus TCP Server, a DHCP client, and an SNMP agent. Some of these services are utilized by complimentary ICS systems (e.g., SCADA) while others are available for basic networking configuration (to establish an IP address, etc).

Additionally, modern PLCs also use an embedded webserver to host deeply customizable web applications that utilize a suite of web-based APIs to manage nearly all PLC operations, including physical process monitoring and control. This functionality has become so ubiquitous in the ICS ecosystem that virtually every major PLC vendor today includes an embedded webserver in their flagship product [20], and these webservers tend to gain additional capabilities with every firmware update [21], [22]. We independently confirmed this trend using an empirical study in Appendix I-A. A clear advantage of this web-based architecture is that any browser-equipped device can now configure and control the PLC (a task that previously required proprietary engineering software and clunky HMI clients [16]). This design has resulted in web browsers being abundantly found in all layers of modern ICS environments [23]. Legacy ICS equipment that communicated over serial protocols have been replaced with single-purpose microbrowser touch screens and even tablets or smartphones. Many of these devices render the web application 24/7 using a mounted display panel [24].

III. Related Work & WB Architecture

This section compares our proposed WB PLC malware to existing PLC malware categories. We aim for this paper

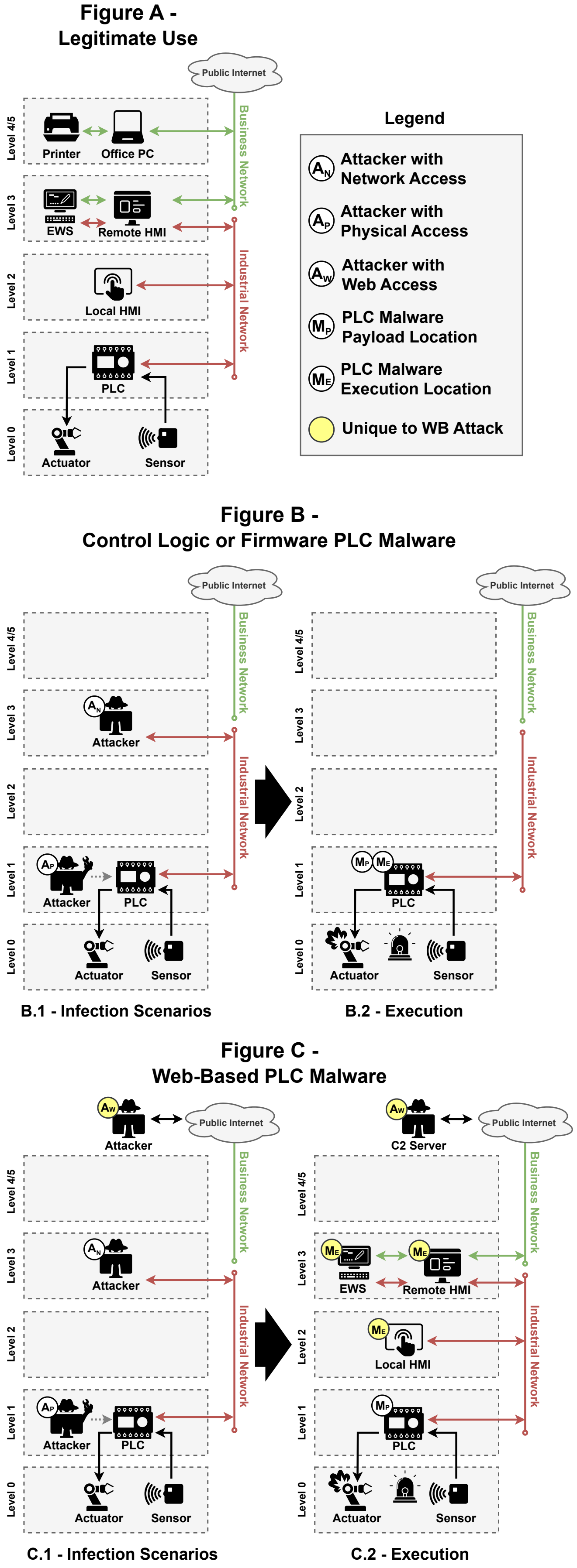

Fig. 1: PLC malware in the PERA Model

TABLE II: Example Infections per Malware Category per Access Level

		Example Infection	Access Needed	PERA Level	Prerequisite	Tested Device
New	**WB #1**	CORS Misconfiguration to override UWP	Web Access	N/A*	Vulnerability**	WAGO 750
	WB #2	rXSS to Restore from Malicious Backup	Web Access	N/A*	Vulnerability***	Siemens S7-1200
	WB #3	Push Malicious UWP	Network Access	1-3	FTP Password	Emerson RX7i
	WB #4	Hijack GUI via MiTM	Network Access	1-3	Insecure Protocols	Schneider TM241
	WB #5	ICS XCS (over SNMP)	Network Access	1-3	Vulnerability**	Allen Bradley MicroLogix 1400
	WB #6	Malicious UWP via SD Card	Physical Access	1	Insider Threat	Mitsubishi MELSEC-F
Traditional	**CL #1**	Push Malicious CL Program	Network Access	1-3	PLC Password	Siemens S7-1200
	CL #2	Hijack CL Update via MiTM	Network Access	1-3	Insecure Protocols	Schneider TM241
	CL #3	Malicious CL Program via SD Card	Physical Access	1	Insider Threat	WAGO 750
	FW #1	Firmware Update w/ Corrupted Image	Network Access	1-3	Vulnerability***	Allen Bradley MicroLogix 1400
	FW #2	Inject Malicious Binary via JTAG Port	Physical Access	1	Insider Threat	Allen Bradley MicroLogix 1400

* No system-level compromise inside the network is needed, but an attacker-controlled website must be *viewed* in 1-3;

** Our team discovered 0day vulnerabilities in latest firmware (confirmed and fixed by vendors); *** Our team used known vulnerabilities in older firmware;

to provide compelling evidence that due to the emergence of powerful PLC web services, system-level compromise of the PLC is no longer necessary to successfully attack ICSs.

Traditional PLC Malware and Shortcomings. We use the term "traditional PLC malware" to describe malicious PLC control logic (CL) programs (e.g., LLB [13], LogicLocker [25], PLC-Blaster [26], ICS-BROCK [27]) and malicious PLC firmware (FW) images (e.g., HARVEY [12], Durin [28]). As discussed in Section I, these two strategies are the only publicly known methods of infecting a PLC with malicious software. Figure 1.B illustrates traditional PLC malware's infection scenarios and execution environment in the context of the PERA model.

Traditional PLC malware infections are possible from two distinct vantage points - *network access* (levels 1-3) and *physical access* (level 1), as shown in Figure 1.B.1. For example, a malicious control logic program may be downloaded via a compromised EWS (à la Stuxnet) or a malicious firmware update may be initiated using physical access to an exposed JTAG port (à la HARVEY [12]). Both of these scenarios require sizable prerequisites to be successful in-practice (e.g., coupled with Windows malware or launched by human assets). Table II lists example CL and FW malware infection methods.

Once the target PLC has become infected with traditional PLC malware, the code is both *stored and executed* on the PLC device within level 1, as shown in Figure 1.B.2. This constraint requires traditional PLC malware to abide by the strict hardware requirements of a real-time operating system (RTOS) with a modest CPU and limited network connectivity. For example, the firmware malware, HARVEY, required tedious model-specific firmware reverse engineering and binary instrumentation to carefully inject instructions in subroutines outside of the time-critical scan cycle and conform to real-time expectations of the control loop [12]. Even more restricted, control logic malware runs as user-code contained in an execution sandbox (often referred to as "jail") and only has access to specific memory regions and limited control logic APIs provided by the vendor [29]. In either case, the code exclusively runs in level 1, trapped in the segregated industrial network without a public internet connection.

Proposed WB PLC Malware and Benefits. Following the recent growing trend of web-based PLC functionalities, we present a new method for infecting PLCs with malicious software that results in a radically different type of PLC malware than the previous approaches. This malware, which we call *Web-Based (WB) PLC malware*, compromises the web application hosted by PLCs' embedded webservers with malicious JavaScript code. This code ultimately gets executed client-side by various browser-equipped devices throughout the ICS environment (not the PLC itself as in the case for CL and FW malware). During execution, the malware uses ambient browser-based credentials to interact with the PLC's legitimate web APIs to attack the underlying real-world machinery. Figure 1.C illustrates WB PLC malware's infection scenarios and execution environment in the context of the PERA model.

As shown in Figure 1.C.1, WB malware introduces a new infection scenario not possible with previous attacks. In this scenario, which we call "*Web Access*," the attacker lures a dual-homed ICS operator within level 3 to view a malicious website. This scenario does not require the EWS to be compromised (i.e., running a malicious binary) but rather simply *viewing* an attacker-controlled website. This scenario originates from the public internet, above level 4/5, and uses cross-origin web requests to pivot into the private industrial network. An example attack from this scenario is a malicious website that exploits a Cross-Origin Resource Sharing (CORS) misconfiguration vulnerability to transfer a malicious User-defined Web Page (UWP) to the PLC's embedded web server. A key observation is that traditional vulnerabilities impacting PLCs (e.g., broken authentication over Modbus [30], denial-of-service over Profinet [31], etc.) are not exploitable from the web environment, since browsers only allow websites to use web-oriented protocols (e.g., HTTP(s), websockets, etc.). Therefore, PLC web vulnerabilities are more exploitable to remote adversaries than previous classes of security issues, further highlighting the practicality of WB malware.

Additionally, the two access levels used by traditional PLC malware (*network* & *physical*) are also viable access levels for WB PLC malware. For example, a malicious UWP can be downloaded via an ICS protocol or a malicious web-based GUI may be installed via an SD Card. We experimentally verified several different example infection methods from each access level on every PLC device included in the study. The results from this experiment, as well as the specific prerequistes for each example attack, are included in Table II. Note that this table is not meant to be an exhaustive list of all possible infection methods, but rather is presented to provide the reader with concrete examples from real-world tests. Technical details about WB infection methods are discussed in Section IV-B.

Figure 1.C.2 illustrates a fundamental difference between our proposed WB malware and traditional PLC malware - WB

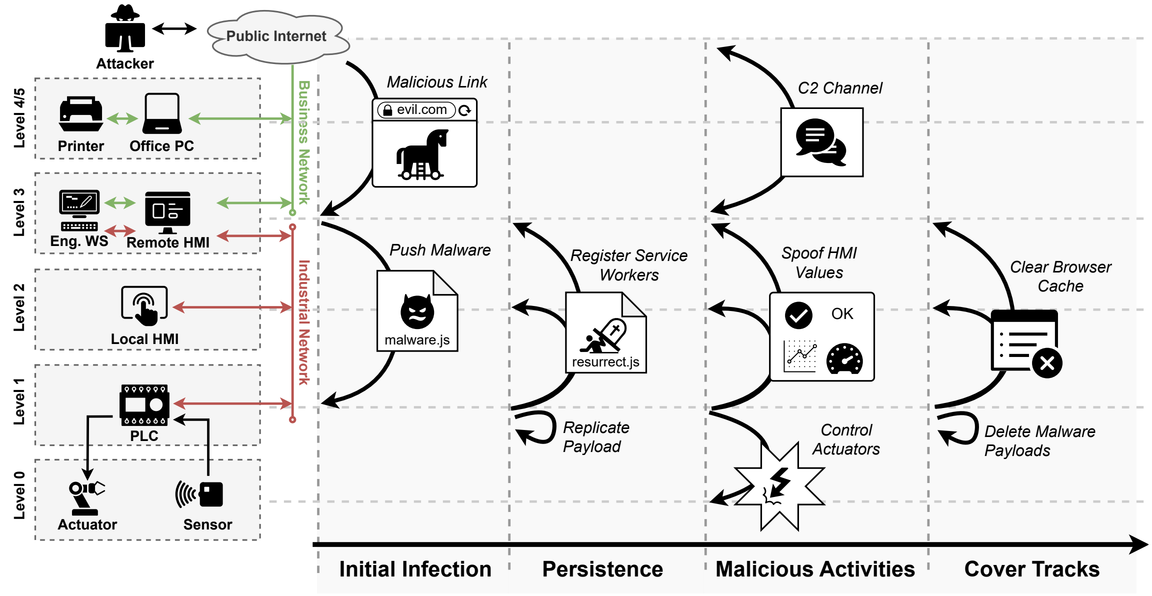

Fig. 2: Lifecycle of WB PLC Malware

malware *decouples where the malware resides and where it executes*. WB malware resides in PLC memory but executes in web browsers (e.g., Microsoft Edge in EWSs, Chromium Microbrowsers in local HMIs, etc) located in levels 2 and 3, physically detached from the PLC. These level 3 browsers are also often connected to the business network to enable public internet access [19]. As a result from this architecture, WB malware can oftentimes utilize a web-based Command & Control (C2) connection, where all C2 communication, initiated from level 3 browsers, traverses the business network in level 4/5 and escapes to the public internet (see Section IV-D).

IV. WB PLC Malware Stages

This section introduces the stages of our proposed WB PLC malware using a vendor-agnostic framework as well as an example implementation on a widely-used PLC model in a real-world Stuxnet-style attack.

Vendor-Agnostic Framework. We developed a general-purpose framework for building and analyzing WB PLC malware. This framework explains the malware lifecycle using four distinct stages, as shown in Figure 2: *Initial Infection*, *Persistence*, *Malicious Activities*, and *Cover Tracks*. This framework explores each stage using widely applicable strategies that can be used against most modern PLC models and presents an overview of how malicious front-end code can subvert the integrity of ICS environments by methodically compromising PLCs' web properties. This framework can be used as a benchmark in future studies across any PLC vendor and model. Note that this framework covers many different strategies and not all of them are applicable to every PLC in every threat model. For our example WB malware, *IronSpider*, we only implemented the strategies that are applicable to the threat model of our testbed (an intentionally realistic and restrictive environment). Other, less restrictive, threat models may allow for other strategies outlined in our framework.

Example Implementation. We implemented each step of this framework using an example malicious program, which we call *IronSpider*. This program was designed to illustrate the effectiveness of the WB malware by performing a Stuxnet-style attack on a popular PLC model (WAGO 750) in a real-world ICS testbed. The testbed's main objective is to precisely spin a three-phase 220VAC industrial motor, representative of the ones used to power gas centrifuges during the uranium enrichment process. We used this testbed to demonstrate the core functionality of *IronSpider*, however modern ICSs of any size and complexity are equally as susceptible to this emerging threat. A detailed description of the testbed equipment and configurations can be found in Section V. Note that *IronSpider* was specifically crafted to attack the testbed under the assumptions listed in the threat model from subsection IV-A.

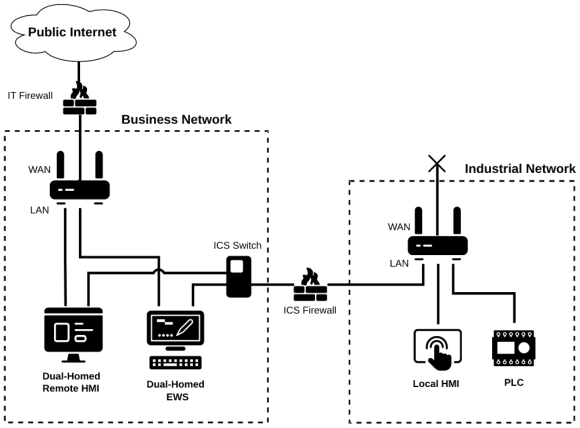

Fig. 3: ICS Network Topology

A. Threat Model & Assumptions

In this attack, we assume that ICS operators use EWSs that have simultaneous access to both the business network and the industrial network and that both networks are secured using tightly configured firewalls. Figure 3 gives a detailed view of the network topology, which is a typical implementation of the PERA model in critical infrastructure environments according to surveys conducted by the Centre for the Protection of

National Infrastructure [19]. We also assume that the EWSs are secured using standard IT best-practices (e.g., up-to-date operating system and browser, sufficiently strong passwords, continuous anti-virus scans, etc.) and that the PLC is using the most secure settings possible (e.g., password-protected services, encrypted protocols, up-to-date firmware, etc.).

We assume that this is a targeted, goal-oriented attack, where the adversary has some basic knowledge about the physical process being performed by the victim (i.e. that it is precisely spinning an industrial motor), but does not know any details of the physical configuration (e.g., plant layout, I/O pin configuration, speed controller settings, etc.). The attacker also knows the specific PLC model being used in the ICS, however does not know its location in the network nor any details about other ICS networking devices.

In this scenario, the attacker has the goal of maliciously controlling the PLC actuators by injecting JavaScript code into a context where it executes under the PLC's web origin inside the private network. The attacker does not wish to perform any system-level compromise of any device within the network (to evade detection) and is not aware of any browser zero-days. Therefore, the attacker's malicious JavaScript must legitimately execute within all devices' browsers and conform to all browser-based rules (e.g., remain in the JS execution sandbox, abide by Same-Origin Policy, respect Site-Isolation protections, etc). The adversary will consider the attack successful if they are able to covertly change the motor speed set-point to a value above the critical safety threshold, thus causing physical damage to the operation.

B. Initial Infection

The initial infection stage of our framework is when the attacker successfully deploys malicious JavaScript in a context where it will be executed in the same web origin as the PLC's admin portal (oftentimes referred to as the "system website").

1) Initial Infection - Framework: Injecting code into the system website can be accomplished by many different methods as shown in Table II. This section provides a technical explanation of the strategies unique to the ICS domain, namely malicious User-defined Web Pages (UWPs), hijacked PLC GUI files, and ICS Cross Channel Scripting (XCS). Each injection mechanism has its own strengths and weaknesses with varying degrees of practicality. The numerous injection mechanisms, spanning various technologies, is one of the reasons why we claim this malware is so flexible.

Malicious User-defined Web Page (UWP). PLC vendors such as Siemens, Allen Bradley, and Mitsubishi allow customers to write their own HTML code to augment the web application hosted by the PLC's embedded web server [32]–[34]. These custom HTML files are referred to as "User-defined Web Pages" (UWPs) and are leveraged to create specialized HMI dashboards. These UWPs are considered a distinct web property from the system website and have a limited set of advertised capabilities. UWPs typically have read-only access to PLC inputs/outputs, and depending on the vendor, may also have limited write-access to a subset of control logic variables.

The supposed restrictions imposed on UWPs would make most users assume that the impact of a malicious UWP is quite limited, however due to an unintended consequence of Same-Origin-Policy (SOP), the permission boundary of UWPs is actually defined by the *user* viewing the UWP and not an intrinsic property of the UWP itself, despite official vendor documentation stating otherwise [35]. This (seemingly misunderstood) relationship between the system website and UWPs allows a malicious UWP, when viewed by an administrator, to take full administrative control over the PLC. Thus, a malicious UWP is a viable injection mechanism to plant WB PLC malware. To help mitigate the impact of malicious UWPs, PLC vendors should consider *sandboxing* untrusted front-end code to an isolated origin (e.g., Facebook's *fbsbx.com* [36]). A detailed explanation of how this potential mitigation strategy can be adapted to the ICS domain is presented in Section V-D.

UWPs can be downloaded to PLCs via proprietary ICS protocols (e.g., CIP PCCC for Allen Bradley [37] and ISO-TSAP for Siemens [38]) or via non-ICS download methods (e.g., FTP for GE [39] or SD card for Mitsubishi [34]). Furthermore, some vendors allow a full project image, including any UWPs, to be downloaded to the PLC over HTTP(s) via a "Restore from Backup" web API exposed by the system website. The flexibility of download methods gives an attacker multiple paths to planting a malicious UWP. Generally speaking, pushing a new UWP to the target PLC requires either a vulnerability, insecure victim settings, or compromised credentials. These are the same prerequisites needed to push CL malware, as discussed in prior work [25].

In lieu of personally downloading the malicious UWP, a bad actor can also trick an authorized user into installing a trojan UWP, as many UWPs are actually authored and sold by third-parties (e.g., Elmi Elettromeccanica [40]). The lack of web subject-matter expertise by ICS operators combined with the misconception about the impact of malicious UWPs makes this injection path particularly feasible. This is a compelling argument for enforcing domain sandboxing because without it, a UWP is equally as enticing of a target for attackers as the system website. Furthermore, PLC vendors cannot guarantee the security of UWPs (as they are not authoring them), so the only way to mitigate their compromise is to isolate it from the administrative portion of the web application.

Hijacked PLC GUI Files. Instead of manually writing HTML code, WAGO and Schneider customers can choose to use software that generates web-based graphical user-interface (GUI) from a high-level visual description [41], [42]. The most common example of such software is the *WebVisu* application licensed from *CODESYS* [43]. This software allows an operator to drag-and-drop GUI elements to build an interface that gets transpiled into front-end files (HTML, JavaScript, CSS). This software helps ICS operators build rich, although less customized, HMI dashboards using pre-build elements, without needing any web subject matter expertise. If these files are hosted on the same embedded web server as the system website (without any sandboxing considerations), they may be a feasible infection mechanism for WB malware. In most cases, simply modifying the transpiled files in-transit during the download process or overwriting them in the filesystem after download is all that is needed to compromise the device. We verified this technique by SSH'ing into a WAGO 750 PLC in our lab and overwriting the transpiled front-end files. In general, hijacking GUI files requires either a vulnerability, insecure protocols, or compromised credentials (same prerequisites as when hijacking CL programs [13]).

ICS Cross-Channel Scripting. In addition to the two legitimate channels for pushing front-end code to the PLC discussed above, an attacker may also be able to exploit a Cross Channel Scripting (XCS) vulnerability to inject WB malware. XCS is an obscure variant of Cross-Site Scripting (XSS) where the malicious payload is transferred to the webserver via a non-web protocol such as SNMP or FTP [44]. We discovered that this vulnerability classification is particularity common in the ICS domain because real-time constraints force industrial equipment to utilize low-latency proprietary protocols. While investigating this project, we observed that these protocols provide an effective method for sending malicious JavaScript payloads to the embedded web servers inside PLCs. We believe that XCS is an understudied vulnerability in the ICS domain as the analysis by our team revealed multiple zero-day vulnerabilities across several different vendors. Note that in order for an attacker to successfully exploit a XCS vulnerability, they need the ability to communicate over the payload-delivery protocol (which may or may not require authentication). In our case, we discovered and exploited an unauthenticated zero-day Allen Bradley SNMP XCS vulnerability (CVE-2022-46670) to push WB malware to our MicroLogix 1400.

Discovering these injection bugs required manual effort with custom-written clients because traditional IT-oriented web scanning tools (e.g., Burp Suite [45]) are unable to inspect the industrial protocols that PLCs regularly utilize to accept user-input (e.g., CIP, Modbus, EIP, etc). Our research shows that PLCs are uniquely difficult to protect from JavaScript injection because their webservers often render user-input that was ingested from a plethora of specialized non-web protocols.

2) Initial Infection - Example Implementation: After approximately one week of testing, our team identified four zero-day vulnerabilities in WAGO 750's latest firmware, three of which allowed us to automatically deploy *IronSpider* from the *Web Access* level (i.e. when a PERA 1-3 operator views our website). Specifically, we leveraged a CORS misconfiguration vulnerability (CVE-2022-45139) in conjunction with an authentication bypass vulnerability (CVE-2022-45138) to perform a cross-origin HTTP API call that exploited an arbitrary file upload vulnerability (CVE-2022-45140). This chain enabled our third-party website to override the transpiled PLC GUI file (*/home/codesys_root/PlcLogic/visu/webvisu.htm*) with a duplicate page that contained *IronSpider*. We emphasize that these vulnerabilities resided in the PLC web application, not in the EWS browser (Google Chrome) or EWS operating system (Microsoft Windows). Exploiting these vulnerabilities did not break any browser-based rules such as SOP or Site-Isolation because the misconfigured CORS header inadvertently instructed the browser to relax SOP and the authentication bypass payload circumvented the need for accessing PLC session data when sending the request. From the perspective of the browser and operating system, our attack is a legitimate website abiding by all W3C specifications.

These issues were disclosed to the vendor and fixed in a subsequent firmware update. The methodology and testing procedures used to identify these issues are outside the scope of this paper, however we believe that similar vulnerabilities can be found in other PLC admin portal web applications. Our malicious website also used basic JavaScript reconnaissance methods (e.g., websocket Favicon sweeping internal IP ranges [46]) to automatically locate the WAGO PLC within the private industrial network. Partial exploit code is included in Listing 1.

We emailed a link to this website to the EWS in our testbed and manually opened it in the default web browser. Alternatively, an attacker looking to indiscriminately launch the attack at-scale can perform a *watering hole attack* [47] by simply purchasing an ad banner on a popular PLC help forum. Note that the attacker did not need any prior knowledge of the EWS hardware, operating system, or web browser. After 3.7 seconds of viewing the webpage, *IronSpider* was successfully downloaded to the target PLC without any user notification or firewall intervention. We emphasize that *this attack did not compromise the EWS*, but rather simply used it as a pivot point to gain network access to the PLC. None of the typical EWS security measures such as anti-virus scans and patch management were able to prevent this attack.

```
/*
Kill Chain to deploy Iron Spider into the transpiled PLC GUI file:
  1) CVE-2022-45139 - CORS Misconfiguration - adding "/x.pdf" to any
       API endpoint will trick the webserver into responding with a
       wildcard "Access-Control-Allow-Origin," allowing it to be
       called cross-origin
  2) CVE-2022-45138 - Authentication Bypass - intentionally leaving
       off cookies and adding "renewSession:true" will force the
       webserver to utilize a guest user account, which accidently
       has permission to call several APIs
  3) CVE-2022-45140 - Arbitrary File Upload - the "network_config" API
        can be tricked into writing arbitrary content at an arbitrary
        location using root privileges via the undocumented "--error-
       msg-dst" argument
*/
async function exploit(wagoIP,filepath,content){
  let resp = await fetch(
    "https://"+wagoIP+"/wbm/php/parameter/configtools.php/x.pdf",
    {
      method:"post",
      body: JSON.stringify(
        {"aDeviceParams":[{"name": "network_config","parameter": ["--
             restore",content,"--error-msg-dst",filepath],"multiline"
             : false}],"renewSession":true})
  });
  if (resp.ok) {
    let j = await resp.json();
    return j.aDeviceResponse[0].status == 2
  }
  return false
}
/*
    Usage: Call exploit() with the path of the WebVisu GUI file and
         the source code of the WB PLC Malware
*/
exploit(
  wagoIPAddress,
  "/home/codesys_root/PlcLogic/visu/webvisu.htm",
  ironSpiderCode
).then((success)=>{
  if (success) { console.log("Exploit Successful") }
})
```

Listing 1: JavaScript code to deploy *IronSpider* into the PLC

C. Persistence

The next stage of our framework is *persistence*. In this stage, the malware will employ several techniques to hide its presence and become resilient to typical removal methods.

1) Persistence - Framework: In our experiments, we found that the exact steps needed to accomplish persistence will vary depending on the model of the infected PLC, however we included the most common strategies below.

Resurrection Code in HMIs and EWSs. A shockingly effective strategy to achieve persistence in the ICS domain is to leverage a web service worker to cache "resurrection code" in multiple web browsers throughout the ICS network (levels 2 and 3 in the PERA model). Service workers are a relatively new addition to the HTML5 specification that lets scripts run in the background, detached from any single web page. This powerful feature of web browsers is used to build rich offline experiences that include functionality such as push notifications and background sync [48]. We propose that this functionality be repurposed in the ICS domain to secure a foothold in the segregated industrial network by caching secondary malware payloads throughout the ICS environment. These secondary payloads will execute directly from EWS/HMI browser cache for up to 24 hours [49] after the primary malware payload has been completely removed from the PLC device. These service workers can periodically check for such removal, and when detected, use various web strategies to re-infect the device, as shown in Figure 4. These strategies are similar to the initial infection methods from the *Web Access* level, as discussed in Section III, however they are performed from within the PLC's web origin (not cross-origin) and will therefore be able to directly access all authenticated web-based APIs without restriction. For example, the malicious service worker could legitimately call APIs to upload new UWPs or edit GUI files to complete the re-infection process.

While service workers do have a limited set of capabilities compared to normal JavaScript execution [50] (e.g., no DOM access, localStorage, or other synchronous APIs), these limitations are unrelated to security and do not impact our attack methodology. Service workers run in a renderer process that is associated with the page's origin [49], which allows it to abide by SOP when interacting with the reset PLC. Furthermore, a malicious service worker can trivially bypass all limitations by simply injecting itself into the page by attaching an event listener to the fetch API and modifying the proxied traffic. Specifically, the service worker can wait for a legitimate request for a JavaScript file and use the *event.respondWith* [51] method to replace it with a mocked *Response()*, thus pushing new code into a regular *<script>* tag. This strategy may be useful if the resurrection process requires authentication secrets stored in localStorage (e.g., to call a *Restore from Backup* web API) or CSRF tokens stored in the page's DOM (e.g., to submit POST requests that exploit sXSS). Example resurrection code is included as an artifact to this paper.

In addition to re-infecting a factory-reset PLC, the proposed strategy also allows the malware to infect any replacement PLC, thus giving the malware the ability to survive even after the PLC hardware has been completely rebuilt. This advanced "resurrection" technique helps the WB malware withstand even the most stringent eradication steps set by the Cybersecurity and Infrastructure Security Agency's (CISA) "Cybersecurity Incident & Vulnerability Response Playbooks" (i.e., reimage PLC to "gold" source and rebuild PLC hardware) [52]. Note that this malicious utilization of service workers is unique to the ICS domain because securing a web-based foothold in a segregated private network is not needed to communicate with a webserver in the IT domain.

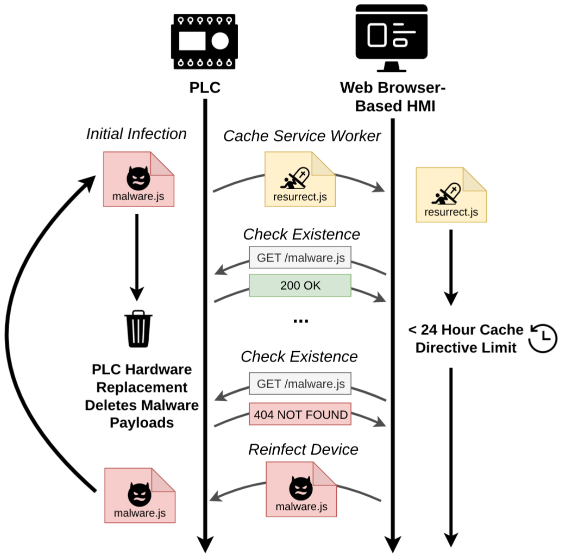

Fig. 4: Service Worker in WB Malware Resurrection

Self-Replication via Downgraded PLC Firmware. As discussed in Section IV-B, there are numerous infection mechanisms for WB malware. This is especially true if the malware uses the system website APIs to downgrade the firmware version to re-introduce known security issues. The malware may utilize these different infection mechanisms to provide redundancy (i.e., store copies of itself in different sections of PLC memory), which enables the malware to survive intentional, or accidental, actions by the PLC operator that may delete the payload (e.g., updating configuration settings or power-cycling the device). We experimentally verified with Siemens S7-1200 that using web APIs to downgrade the firmware version then exploiting known file upload vulnerabilities was an effective method of self-replication in that PLC model. Note that the technique discussed here is unique to the ICS domain, as front-end code in the IT domain does not typically have the ability to control versioning of server-side code.

2) Persistence - Example Implementation: Recall that *IronSpider* was deployed to the transpliled PLC GUI file in the previous stage. In this stage, *IronSpider* employed various strategies to ensure continuous execution in the ICS environment. Firstly, *IronSpider*, utilized the same zero-day bugs from the Initial Infection step to overwrite the system website files, thus spawning a new execution process in the EWS browser. Next, *IronSpider* registered service workers in all three of the browsers rendering its payload (Microsoft Edge on EWS, Chromium on remote HMI, WAGO microbrowser on local HMI). These service workers became cached in the browsers and periodically checked for the existence of the main malware payload. If/when the main payload was found to be missing, the service worker used the same methods from the first step to re-infect the device. Specifically, our resurrection code exploited our zero-day vulnerability, CVE-2022-45140, to re-upload the main malware file to the embedded web server's root directory. We emphasize that this strategy enables *IronSpider* to survive PLC hardware replacement.

D. Malicious Activities

The impact of the WB malware can be measured by its ability to sabotage real-world machinery, abuse admin settings, and exfiltrate data. Recall from Section III that traditional PLC malware is trapped within segregated industrial network without an internet connection and that CL malware runs in a user-code sandbox with limited functionality. For these reasons, we claim that WB malware is often capable of performing more impactful malicious activities than prior work. Table III compares the capabilities of each PLC malware category.

TABLE III: Malicious Capabilities per Malware Category

	Web-Based	**Control Logic**	**Firmware**
Admin Settings	✓		✓
Sabotage	✓	✓	✓
Exfiltration	✓		

✓= Possible Malicious Activity

1) Malicious Activities - Framework: The malicious capabilities of the malware are directly mapped to features of the system website. All actions capable of being performed by a human using the system website normally can be accomplished programmatically by WB malware (e.g., virtually *"click"* buttons, virtually *"type"* into forms, and utilize all legitimate HTTP APIs). Therefore the impact of WB malware will depend on which PLC model has been infected. The following sections contain the general approach that can be applied to most models.

Sabotage Machinery for Physical Damage. A key component of any PLC malware is its ability to influence real-world physical events. WB malware is capable of performing such control by utilizing the legitimate system website APIs to sabotage the industrial processes. Because WB malware executes on the same web origin as the PLC's system website, it can leverage the ambient browser credentials (e.g., cookies) needed to interact with authenticated web APIs. This can be done directly via JavaScript-initiated network requests (e.g., *fetch()* or *XMLHttpRequest*) or indirectly via JavaScript-initiated simulated user input (e.g., virtually "clicking" buttons in the UI). The specific steps needed to sabotage the real-world machinery will depend on the functionality provided by the vendor of the infected PLC. Some PLCs, such as the Allen Bradley MicroLogix 1400, expose web APIs to directly modify I/O values by overwriting the data stored in CPU memory addresses via the "*Editable Data Table Memory Map*" (even when the data itself is not tied to an HMI-controlled variable [33]). Other PLCs, such as the Schneider TM241, expose web APIs to overwrite control logic variables with arbitrary data [42]. Furthermore, some PLCs, such as Siemens S7-1200, even allow the entire PLC project overwritten via the *"Restore from Backup"* web API [53]. This backup can contain new set points, user configuration, and safety settings. A bad actor can abuse these powerful web APIs to maliciously control actuators and cause catastrophic damage to the underlying physical processes. Note that modifying control logic variable values via web-based APIs does not recompile the control logic binary and will therefore not trigger any attestation systems such as *PLCDefender* [54], further illustrating how this attack is materially different than CL malware.

Due to the intentional human-readability of web-based HMIs, little-to-no prerequisite information regarding the underlying physical domain (level 0) is needed to launch a successful sabotage attack. An adversary can deduce unsafe states by visually inspecting screenshots of the HMI UI (exfiltrated using the techniques discussed later in this section) and modifying the controls accordingly (e.g. virtually "turn the knob" to change motor speed set points). This type of casual control is not possible using traditional PLC malware, which requires intimate knowledge of I/O pin configuration and downstream actuator settings. Thus, physical sabotage via WB malware requires significantly less reverse engineering effort and prerequisite intelligence compared to existing strategies.

In addition to compromising the PLC actuators, WB malware can also sabotage industrial processes by spoofing values displayed in the system website and web-based HMIs. This can be accomplished by simply modifying the DOM of the displayed webpage using the standard JavaScript interface (e.g., *document.body.innerHTML*) or by overlaying fabricated displays (e.g., adding a screenshot to a top layer full-page *img* tag). For example, stealthy WB malware may record sensor values in browser storage (e.g. local storage [55]) and display them later during the actuator compromise to hide the attack.

Abuse Admin Settings for Further Compromise. Another malicious action that the WB malware can perform is to modify the administrative PLC configuring via the web-based APIs exposed by the system website. These APIs allow an operator to control admin settings on the device through a feature called "*Web-Based-Management (WBM)*" [56] [57]. An adversary can abuse these APIs to aid in future attacks or enable further compromise of the device. The specific settings available for modification depend on the PLC vendor and firmware version, however a typical attack may include editing the on-device firewall, creating new users, and enabling/disabling certain network services. We emphasize that this type of control is not possible with CL malware due to the user-code sandbox and is extraordinarily difficult with FW malware due to the tedious nature of binary instrumentation in a real-time embedded device (and is sometimes not possible at all depending on the chipset isolation in the motherboard [12]). With WB malware, this control is easily accomplished by simply calling the legitimate HTTP APIs with JavaScript. Note that our malicious JavaScript code leverages the same authentication mechanism as the user rendering its payload, which in the case of the EWS, will likely be the cookies belonging to the PLC system administrator because the primary purpose of the EWS is to perform administrative device configuration [58]. Table IV lists common WBM admin settings and example consequences of their malicious misuse.

TABLE IV: Consequences of Misusing WBM Admin APIs

Admin Setting	**Example Misuse**
On-Device Firewall	Modify EWS MAC Address whitelist to allow rogue connections to arbitrary hosts.
User Management	Create new "backdoor" user to ensure continuous control.
Network Configuration	Modify IP address to circumvent downstream firewall routing rules.
Network Services	Disable SNMP to prevent network operators from noticing IP reconfiguration.
Image Backup	Create and exfiltrate a full project backup, including current CL programs, device metadata, and historical logfiles to aid in espionage.
Security Settings	Disable IPsec, OpenVPN, and TLS to let other devices to MiTM PLC traffic.

Data Exfiltration for Industrial Espionage. As mentioned in Section II, the unique execution characteristics of our proposed malware allows it to utilize a public Internet connection even when the PLC itself is located in an isolated private network. Web browsers in which the malware executes (e.g., Microsoft Edge on an EWS and/or Chromium-based Microbrowsers in remote HMIs) are typically simultaneously connected to both the target PLC network and other less-critical networks [19]. For example, EWSs are usually connected to both the level 2 network (to communicate with the PLC) and the level 3 network (to perform online tasks like sending emails and viewing forum websites to troubleshoot devices, e.g., http://support.industry.siemens.com [59]). This simultaneous connectivity can be accomplished using a variety of networking setups common in large ICSs such as dual NICs, VLAN tagging, and firewall-managed enclaves. The perimeter of these networks are often secured using domain-specific firewalls that attempt to block abnormal traffic [19]. Our WB malware can bypass this scrutiny by only utilizing protocols allowed in level 3 traffic, such as DNS or HTTPS, when communicating to the C2 server and only using the legitimate PLC APIs when communicating to the PLC.

Performing this exfiltration using WB malware does not require any intimate familiarity with the target PLC or the underlying physical process. Generic, but powerful, web-based exfiltration strategies such as canvas screenshots, event-listener keylogging, and full DOM dumps can be applied to virtually all PLC models across every vendor. These techniques allow WB malware to intercept sensitive ICS information such as physical process characteristics, plaintext usernames and passwords, and plant configuration details. This stolen data can be covertly exfiltrated using front-end network requests such as JavaScript *fetch()* and URI parameters to an HTML *img* tag *src*. Browser-based exfiltration strategies like this are notoriously difficult to detect or prevent using firewalls [60] because benign web applications often communicate with a variety of third party servers via encrypted protocols [61] (e.g., HTTPS and WSS), which commonly causes nefarious browser-based connections to go unnoticed, especially if the C2 infrastructure is built on top of a reputable third party web service such as Google Analytics or Facebook pixels [62]. Note that the C2 stream will be completely unaffected by any *on-device PLC firewalls* because the PLC-to-browser communication is utilizing the legitimate APIs provided by the PLC vendor. We emphasize that neither CL nor FW PLC malware can perform real-time data exfiltration in-practice because they typically execute in segregated industrial networks and cannot utilize a public internet connection.

2) Malicious Activities - Implementation Example: The goal of *IronSpider* was to perform a Stuxnet-style attack using modern web technologies. To prepare for the attack, a real-time websocket C2 channel was established by all three execution environments. While only the EWS and remote HMI had direct internet access, the local HMI process was still able to achieve C2 communications by using a covert channel within the PLC device as a proxy. During the attack, *IronSpider* modified the web-based HMIs' DOMs to display falsified sensor values (recorded the previous day and saved in browser local storage). Next, the malware virtually interacted with the HMI's UI to covertly change the set-point for the motor's speed control to the highest possible value (recall that the attacker's goal is to spin the motor at any level above the critical safety threshold). Our testbed used an emergency light to indicate that a critical failure occurred (i.e., the attack was successful and it is too late to intervene). Approximately 7.4 seconds after the attack began, the emergency system tripped indicating that the centrifuge was damaged. The HMIs displayed fake readings throughout the entire attack.

E. Cover Tracks

The final stage of the framework is to remove any traces of the infection. This stage is aimed to impede incident response and forensics postmortem efforts. We emphasize that this self-contained removal strategy is not possible using CL malware due to the user-code sandbox in which it executes.

1) Cover Tracks - Framework: The following section contains the general approach to removing the payload and resetting the device.

Delete PLC Malware Payload. Once the malicious activities have been accomplished and the malware is no longer needed, it should attempt to remove the payload from PLC storage. The exact removal techniques will depend on how the payload was initially implanted onto the device (see Section IV-B), however in general, the process will involve repeating stage 1 using a blank, or otherwise benign, payload to overwrite the malware file. Additionally, the malware will need to invalidate any resurrection code by *unregistering* the service workers using the exposed HTML5 browser APIs.

Restore from PLC Backup Image. Even after the payload has been removed from PLC storage, traces of its existence may still reside in access logs and/or obscure memory caches. To finish the removal process, WB PLC malware may use the legitimate system website APIs to restore the PLC program from a prior backup image. Doing this will overwrite the set points, reset all configurations, and flush all caches. This drastic step gives the malware full control of the current PLC image, which is likely where forensics teams will begin their investigation. This type of absolute control over raw server-side memory states is unique to the ICS domain and allows WB PLC malware to self-destruct in a much more complete manner than malicious JavaScript in the IT domain.

2) Cover Tracks - Implementation Example: After *IronSpider*'s successful attack, it began the process of cleansing the PLC of any traces of the infection. This was a non-trivial exercise because our zero-day vulnerabilities (CVE-2022-45137, CVE-2022-45138, CVE-2022-45139, and CVE-2022-45140) planted the malware payload in a section of the PLC's filesystem that was unaffected by factory resets. *IronSpider*'s first step in covering its tracks was to re-use these vulnerabilities to overwrite both the system website homepage and transpiled GUI files back to their original content. Then, the malware (while still executing on any open tabs) unregistered the service workers and flushed browser cache. Lastly, the malware refreshed all pages, thus killing all execution processes.

V. Evaluation

As demonstrated in the previous section, *IronSpider*, is capable of sabotaging industrial processes using a more modern approach than the ones used by existing PLC malware. Our malware can exfiltrate sensitive data, spoof HMI displays, maliciously control PLC actuators, and self-destruct all without system-level compromise. Recall that implementation details for *IronSpider* were included in each subsection of Section IV. This section provides an analysis of the experimental results obtained from running *IronSpider* in a real-world ICS testbed to perform a Stuxnet-style attack. This section also discusses WB generalizability, limitations, and countermeasures.

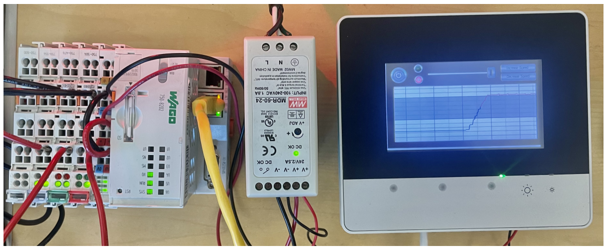

Fig. 5: PLC, Power Supply, and Microbrowser HMI

A. Experimental Setup

The previous section explained how *IronSpider* implemented each stage of our WB PLC malware framework in a real-world ICS testbed constructed to precisely spin a three-phase 220VAC industrial motor. This scenario was inspired by the real-world configuration for controlling uranium enrichment centrifuges during the Stuxnet attack [63]. This subsection outlines the key hardware components and networking details that comprised our testbed. Figure 5 shows a photo of the PLC, power supply, and Microbrowser HMI.

Networking Details. We developed a real-world ICS network environment by segregating the industrial network from the business network, as shown in Figure 3. The industrial network consisted of a *WAGO 750* PLC, a *WAGO e!Display 7300* local HMI, and a dedicated LAN port to connect to certain devices in the business network. Firewall rules only permitted legitimate PLC traffic to traverse the industrial network and the WAN port was sealed to prevent direct routable access to other networks. The business network consisted of a Raspberry-Pi remote HMI and a Microsoft Windows EWS. The business network was connected to the public Internet through a firewall that only allowed standard office traffic. The remote HMI and EWS were dual-homed to enable simultaneous access to both the business and industrial networks.

PLC Configuration. The WAGO 750 PLC controlled a *Dayton 11W366* industrial motor with a 0-10v analog signal to a *Schneider ATV12* Variable Frequency Drive. The PLC also read the actual rotor speed using a *Compact Instruments Tachoprobe A2108* tachometer, which outputted a 0-6V analog signal. A web-based HMI was developed using WAGO's *WebVisu* integration that allowed operators to both view the tachometer readings and change the setpoint for the motor speed. This HMI was displayed in the *WAGO e!Display 7300* Microbrowser local HMI and in Chrome on the remote HMI.

Failure Indication. Lastly, an emergency light was configured to trip if the tachometer read values over a certain threshold. This system was designed to indicate that a critical failure has occurred, similar to a standard smoke detector.

B. Execution & Results

IronSpider performed Stuxnet-style sabotage by orchestrating an end-to-end attack where the EWS, HMIs, and PLC all worked together to covertly set the motor speed setpoint to a dangerous level. This subsection summarizes the data flow during the attack and analyzes the outcome from the sabotage.

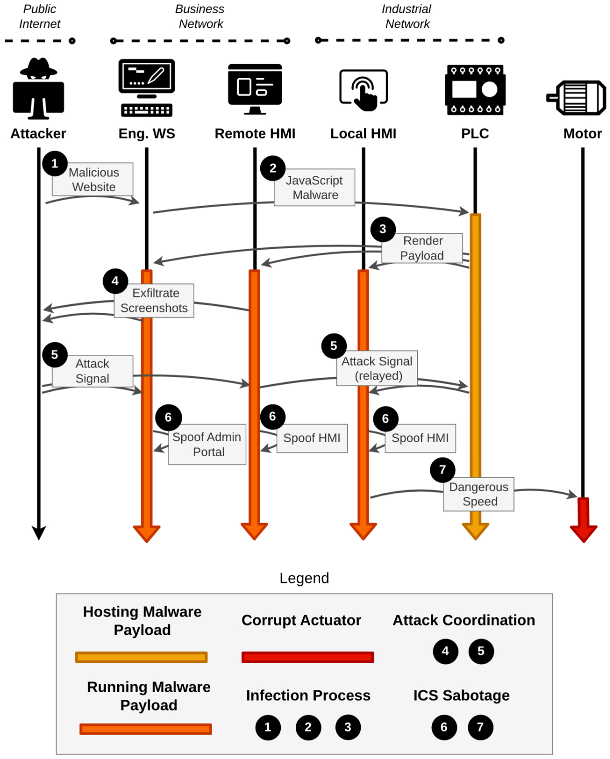

Fig. 6: Overview of *IronSpider*'s Data Flow

Execution. During this attack, *IronSpider*, covertly interacted with several components of the ICS environment. The EWS was used as a pivot point to gain entry into the industrial network through cross-origin network requests by the malicious website. The PLC was used to host the malware payload, relay C2 messages to the segregated local HMI, and physically interact with the sensors/actuators. Both the HMIs and the EWS were used to execute the malware payload in their respective browsers. And of course, the motor was used to sabotage the industrial process. Figure 6 illustrates the high-level data flow that occurred during this attack. We emphasize that neither CL nor FW malware alone could perform data exfiltration in our testbed due to the realistic network segregation controls.

Results. Stuxnet sabotaged Iranian nuclear facilities by modifying the analog output signal to variable-frequency drives that controlled uranium enrichment centrifuges [63]. A direct result from this sabotage was the physical destruction of over 1,000 centrifuges and a 30% reduction in operational capacity at the facilities [64]. Our prototype malware, *IronSpider*, was able to achieve a fundamentally similar attack using a drastically different approach. Stuxnet attacked PLCs via CL malware that it deployed via compromised EWS (these EWS were compromised using a Microsoft Windows worm and Trojan Step7 DLLs [65]). *IronSpider*, however, used WB malware that it deployed using a malicious website without needing to compromise any peripheral systems. While both attacks achieved the same outcome (sabotaged motor), *IronSpider*'s approach has all of the advantages discussed in Section III.

Figure 7 shows the true tachometer reading during the *IronSpider* attack. Recall that a spoofed tachometer reading, with incorrect values, was visibly shown on the HMIs while the attack took place. As displayed in this figure, shortly after the attack began, the industrial motor from the testbed started spinning at a speed above the critical safety threshold[2].

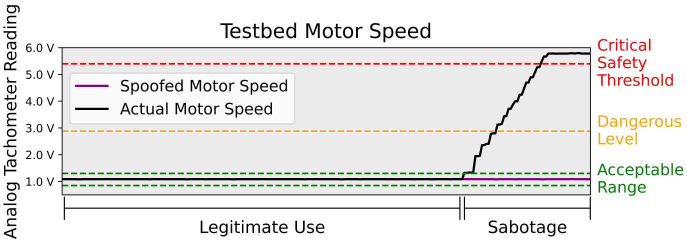

Fig. 7: Tachometer Readings During the *IronSpider* Attack

Analysis. As discussed in Section III, WB malware uses an unconventional architecture, where the payload is stored in PLC memory but executes in browsers belonging to EWSs and HMIs. In this subsection, we examine the performance impact and overhead associated with such a design. *IronSpider* is comprised of two standalone files (*malware.js* and *resurrect.js*) and installation is accomplished via the addition of a single line of HTML code (i.e. *<script src="/malware.js"></script>*). The standalone files are relatively small and consumed a negligible amount of storage in the PLC's filesystem (<0.01%), as broken down by functionality in Table V. Since script-based (non-compiled and platform independent) malware is modular by nature, an adversary can add or remove functionality as needed to accomplish their goal and remain undetected. For example, a reasonable tradeoff may entail omitting the screenshot capturing functionality (193KB) if the victim's information is text-based and can be readily exfiltrated from the DOM interface. Note that all functionality is necessarily duplicated in the resurrection payload, meaning that any filesize reduction is effectively doubled in the filesystem.

TABLE V: *IronSpider* Overhead in PLC Storage

Stage	Code Snippet Description	Size (B)	Overhead (%)
Persistence	Install Service Worker	55	1.3E-6
	Resurrection Bootstrap	448	1.0E-5
	Resurrection Payload	198,107	4.6E-3
Malicious Activities	Websocket-Based C2	296	6.9E-6
	Canvas Screenshot Capturer	193,650	4.5E-3
	Keylogger	412	9.6E-6
	Local Storage Usage	62	1.4E-6
	Covert-Channel PLC Proxy	1011	2.4E-5
	Programmatic WebVisu	985	2.3E-5
	Create New Backdoor User	699	1.6E-5
Cover Tracks	Uninstall Service Worker	94	2.2E-6
	Override Payloads	395	9.2E-6
	Total:	396,214	0.0092

Since WB malware executes alongside the real PLC JavaScript (inside the same browser-based sandbox), we must confirm that there is no noticeable performance impact to the legitimate web-based operations. Scrutinizing this aspect is critically important because any visual lag in the HMI or WBM may tip-off the ICS operator. We used the following industry-standard metrics to measure website performance both before and after the infection - 1) Frame Rate, 2) First Input Delay, 3) CPU Utilization, 4) Memory Footprint, and 5) Google Lighthouse Score (a popular SEO score calculated using a variety of visual and speed related indicators). We configured *IronSpider* to aggressively capture and exfiltrate full screenshots every 60 seconds, record all keystrokes, and randomly interact with the WebVisu HMI. This is an intentionally "noisy" configuration to serve as an upper-bound of performance impact. The results of our test can be found in Table VI, which show that the performance impact is minimal, with the average variation across metrics being 3.57%. More importantly, we confirmed that even in this aggressive configuration, the infection is virtually unnoticeable to human operators by manually using the web-HMI without issue.

TABLE VI: Client-Side Performance Impact of *IronSpider*

	Pre-Infection	Post-Infection	Percent Change
Frame Rate	112.4fps	108.9fps	(3.11%)
First Input Delay	0.99s	0.99s	0.00%
CPU Utilization	4.6	4.7	2.17%
Memory Footprint	112.3	121.1	7.84%
Lighthouse Score	96%	95%	(1.04%)

C. Discussion

Generalizability Across PLCs and Testbeds. We tested aspects of WB PLC malware against PLCs from every major ICS vendor and confirmed that they are all indeed susceptible to WB infections, either through legitimate means (e.g., using the FTP password to push a new UWP) or through illegitimate exploitation (e.g., using zero-day vulnerabilities), as shown in Table II. We also verified that much of *IronSpider*'s core functionality (e.g., C2 comms, remote DOM interactions, etc.) were functional against every target device, thus confirming the generalizability of this new class of PLC malware. In this section, we provide the reader with concrete end-to-end examples by briefly exploring two new ICS testbeds controlling different physical processes, separate from the threat model and testbed used by *IronSpider*, as shown in Figure 8.

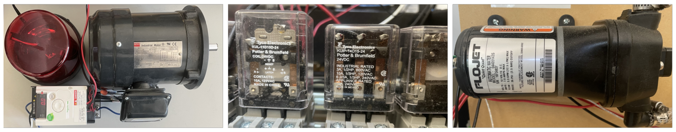

Fig. 8: Industrial Motor, Power Relay, Water Pump

Testbed #2) An attacker compromised the ICS network at a manufacturing facility using a vulnerable edge device (e.g., industrial router) and is targeting a password-protected Allen-Bradley MicroLogix 1400 PLC controlling a conveyor belt system through a series of industrial panel plug-in power relays. This modern ICS uses a custom UWP for daily monitoring and control. From this network vantage point, we were able to discover and exploit a zero-day unauthenticated XCS vulnerability to inject JavaScript via the SNMP protocol (fixed in a subsequent firmware update as CVE-2022-46670). This bug allowed our team to deploy WB malware into the

[2] As a safety precaution, we set the threshold of our testbed motor to a modest speed that did not actually cause any physical damage.

TABLE VII: Proposed Countermeasures to Defend Against PLC WB Malware

Prevention Strategy	Protections Provided	Responsible Party	Practicality
Private Network Access	Increase difficulty of *Web Access* infections	Browser developers	Medium; may disrupt some legitimate traffic
CSP Confidentiality Directive	Increase difficulty of web-based C2 channel	Browser developers and PLC vendors	High; minor server-side configuration for PLC vendors
ICS Domain-Sandboxing	Increase difficulty of *Network Access* infections such as malicious UWP and hijacked GUIs	PLC vendors	Medium; Requires separate auth scheme and server-side reconfiguration
Real-Only CDN w/ CSP and SRI	Increase difficulty of all infections mechanisms	PLC vendors	Low; Requires substantial front-end restructure and CDN management
PLC-configured WAF	Increase difficulty of *Network Access* infections such as ICS XCS	Third-parties	Medium; may add some overhead to real-time ICS protocols

PLC device, which was later rendered on the dedicated HMI machine for viewing the UWP. From there, our malicious JavaScript was able to sabotage the industrial process by virtually interacting with the web-based UI for controlling the relay states. While this WB malware successfully performed sabotage, it was not able to achieve persistence using the service worker resurrection method discussed in Section IV-C because this particular XCS vulnerability did not create new JavaScript files, but rather embedded JavaScript snippets directly into the existing HTML.

Testbed #3) An attacker has full system-level compromise of an EWS and is targeting a Siemens S7-1200 at a security conscious water treatment plant. This modern ICS uses a custom UWP for daily monitoring and control. In this scenario, the attacker infiltrated an authenticated workstation (perhaps via compromised TeamViewer credentials, similar to the alleged 2021 Florida water treatment hack [66]), however the ICS uses state-of-the-art control logic attestation (e.g., *PLCDefender* [54]) that will notify them of any altered CL. In this scenario, the attacker is able to use the legitimate Siemens Step7 software to authenticate against the PLC and push a replacement malicious UWP. This technique does not modify any compiled CL code, therefore keeping the CL digital signature in-tact. This malicious UWP is then rendered in a dedicated HMI machine and can covertly control the industrial water pump. Fortunately for the attacker, this approach is able to use the service worker resurrection method, since UWPs can create standalone JS files in Siemens devices [32].

Limitations. The largest, and most obvious, limitation is that this new class of malware can only be used against modern ICSs that enable and regularly use the embedded webserver (an emerging trend as demonstrated with an empirical study in Appendix I-B). Older industrial plants that exclusively use legacy equipment and antiquated protocols will likely be immune to this new threat. Another limitation is that the capability of WB malware is directly mapped to the functionality of the embedded web application. Older firmware versions with less powerful web-based APIs will inherently have a lower blast radius compared to feature-rich newer firmware versions. Additionally, some APIs may only be available to specific users (e.g., the *Restore from Backup* API may only be available to the plant foreman account). Since WB malware utilizes the authorization of the currently signed-in user, this limitation will require the malware to wait and only perform some actions when certain operators use the web application. Finally, the display spoofing strategy (i.e. modifying the DOM of the web-based HMI) is not applicable to non-web systems that also ingest and process measurements using traditional ICS protocols (e.g., legacy data historians or SIS).

D. Countermeasures

Prevention Mechanisms. Recall from Section III that WB infections can occur from three different access levels - *Web*, *Network*, and *Physical*. Each of these levels pose different challenges from an infection prevention standpoint. Successful *Web Access* infections generally require a vulnerability in the target PLC web application, meaning that, barring zero-days, keeping the PLC firmware up-to-date should provide adequate protection. Additionally, training operators to avoid suspicious websites when using dual-homed devices is also wise, however this strategy is not a comprehensive protection since even ad-banners on trusted websites could launch the attack. *Network Access* infections generally occur due to compromised credentials (e.g., stolen FTP password), a vulnerability in the PLC web application (e.g., XCS), or insecure settings (e.g., unencrypted protocols). The best way to prevent this type of attack is to again keep PLC firmware up-to-date, safeguard credentials, and employ the best-practices set by CISA [52]. Finally, *Physical Access* infections generally use the legitimate, trusted, process for updating front-end files (e.g., SD card) and therefore typically only occur from supply chain compromise or insider threats. Traditional physical protections are the best defense for this type of attack.

Detection Mechanisms. Unfortunately, since WB PLC malware circumvents the need for system-level compromise, most state-of-the-art PLC intrusion detection systems proposed in recent academic papers are unable to detect the infection. This includes defenses such as PLC intrusion prevention systems (e.g., *Reditus* [67]), PLC control logic attestation systems (e.g., *PLCDefender* [54]), and PLC control logic formal verification (e.g., *TSV* [68]). Generally speaking, most PLC malware detectors today focus on scrutinizing the control logic and/or firmware, which are unmodified during our attack.

Detecting WB malware at the browser-level is also challenging, since the malicious JavaScript is intertwined with the legitimate PLC JavaScript (executing under the same origin, within the same sandbox) and conforms to all browser-based rules. Furthermore, since PLC webservers are programmable (i.e. intentionally running customer-authored front-end code), we cannot simply allowlist the vendor-authored JavaScript files that came with the firmware image. We believe WB PLC malware detection is an open problem since it would require differentiating benign customer-authored JavaScript and malicious attacker-authored JavaScript, which is an active (and currently unsolved) field of research [69]. We ran *IronSpider* against the top 4 state-of-the-art JS malware detectors (Cujo [70], Zozzle [71], JaSt [72], and JStap [73]), all of which incorrectly marked it as benign, further illustrating that this is a non-trivial challenge (full details in Appendix I-C).

Proposed Countermeasures. Fortunatly, aspects of WB PLC malware can still be partly mitigated using a combination of protections implemented by browser developers, PLC vendors, and third-party products. These protections can serve as layers for a defence-in-depth strategy to reduce the likelihood and impact of an attack. Table VII lists the proposed countermeasures, the protections provided by each defense, and the practicality of deploying them in real-world ICSs. Notable W3C-draft/proposed browser improvements include "*Private Network Access*" and the "*Content-Security-Policy (CSP) Confidentiality Directive*." Potential protections by PLC vendor include a read-only CDN used with the *CSP src-script Directive* and *Subresource Integrity (SRI)* as well as domain-sandboxing for untrusted JavaScript code (i.e., isolating customer-authored UWPs and GUIs from the system website). Lastly, third parties could offer PLC-configured Web App Firewalls (WAFs) (i.e., configured to inspect non-web PLC protocols such as SNMP, Modbus, and CIP). We included further discussions about these protections in Appendix I-D.

VI. Conclusion

Contrary to popular belief, firmware and control logic are not the only levels of PLCs computation. Modern PLCs now contain a programmable embedded webserver, where custom client-side JavaScript code uses increasingly powerful APIs to monitor and control physical processes. This environment offers a new, and surprisingly ideal, platform to run PLC malware, which poses an emerging threat to industrial control systems.

References

[1] A. Advisory and Intelligence, "Plc market - global outlook and forecast 2020-2025." [Online]. Available: https://www.arizton.com/market-reports/plc-market-analysis

[2] K. Stouffer, S. Lightman, V. Pillitteri, M. Abrams, and A. Hahn, "NIST special publication 800-82, revision 2: Guide to industrial control systems (ICS) security," *National Institute of Standards and Technology*, 2014.

[3] M. Barrère, C. Hankin, D. G. Eliades, N. Nicolau, and T. Parisini, "Assessing cyber-physical security in industrial control systems," *arXiv preprint arXiv:1911.09404*, pp. 1–10, 2019.

[4] , "Industrial control systems (ICS) security market by solution (firewall, antimalware/antivirus, IAM, encryption, whitelisting, security configuration management, DDoS, and IDS/IPS), service, security type, vertical, and region - global forecast to 2023." [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/industrial-control-systems-security-ics-market-1273.html

[5] E. D. Knapp and J. T. Langill, *Industrial Network Security: Securing critical infrastructure networks for smart grid, SCADA, and other industrial control systems*. Syngress, 2014.

[6] , "Five myths of industrial control systems security." [Online]. Available: https://media.kaspersky.com/pdf/DataSheet_KESB_5Myths-ICSS_Eng_WEB.pdf

[7] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[8] N. Falliere, L. O. Murchu, and E. Chien, "W32. Stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011.

[9] Defense Use Case, "Analysis of the cyber attack on the Ukrainian power grid," pp. 1–29, 2016.

[10] R. Khan, P. Maynard, K. McLaughlin, D. Laverty, and S. Sezer, "Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid," in *4th Int. Symp. ICS & SCADA Cyber Security Research*, 2016, pp. 53–63.

[11] A. Di Pinto, Y. Dragoni, and A. Carcano, "TRITON: The first ICS cyber attack on safety instrument systems," in *Proc. Black Hat USA*, 2018, pp. 1–26.

[12] L. Garcia and S. A. Zonouz, "Hey, my malware knows physics! attacking PLCs with physical model aware rootkit." in *Network and Distributed System Security (NDSS) Symp.*, 2017, pp. 1–15.

[13] N. Govil, A. Agrawal, and N. O. Tippenhauer, "On ladder logic bombs in industrial control systems," in *Computer Security*, 2017, pp. 110–126.

[14] A. Costin, A. Zarras, and A. Francillon, "Automated dynamic firmware analysis at scale: a case study on embedded web interfaces," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 437–448.

[15] B. Gourdin, C. Soman, H. Bojinov, and E. Bursztein, "Toward secure embedded web interfaces," in *20th USENIX Security Symposium (USENIX Security 11)*, 2011.

[16] Schweitzer Engineering Laboratories Inc., "Interface HMI touchscreens to SEL devices using modbus protocols." [Online]. Available: https://cms-cdn.selinc.com/assets/Literature/Publications/Application%20Notes/AN2013-24-20130701.pdf?v=20211013-235342

[17] T. Sasaki, A. Fujita, C. H. Gañán, M. van Eeten, K. Yoshioka, and T. Matsumoto, "Exposed infrastructures: Discovery, attacks and remediation of insecure ics remote management devices," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 2379–2396.

[18] T. Williams, "The Purdue enterprise reference architecture and methodology (PERA)," *Handbook of life cycle engineering: concepts, models, and technologies*, vol. 289, 1998.

[19] Centre for the Protection of National Infrastructure, "Firewall deployment for SCADA and process control systems." [Online]. Available: https://www.energy.gov/sites/prod/files/Good%20Practices%20Guide%20for%20Firewall%20Deployment.pdf

[20] statista, "Global PLC market share as of 2017, by manufacturer." [Online]. Available: https://www.statista.com/statistics/897201/global-plc-market-share-by-manufacturer/

[21] WAGO, "WAGO PFC Firmware Releases," 2022. [Online]. Available: https://github.com/WAGO/pfc-firmware/releases

[22] Siemens, "Firmware update for CPU 1214C, DC/DC/DC, 14DI/10DO/2AI." [Online]. Available: https://support.industry.siemens.com/cs/document/107539750/firmware-update-for-cpu-1214c-dc-dc-dc-14di-10do-2ai?dti=0&lc=en-US

[23] Schweitzer Engineering Laboratories Inc., "Web-based HMI: An emerging trend?" [Online]. Available: https://www.automation.com/en-us/articles/2003-1/web-based-hmi-an-emerging-trend

[24] Spider Control, "(Micro-) Browser Solution." [Online]. Available: https://spidercontrol.net/spidercontrol-products/micro-browser-solution/?lang=en

[25] D. Formby, S. Durbha, and R. Beyah, "Out of control: Ransomware for industrial control systems," in *RSA conference*, vol. 4, 2017.

[26] R. Spenneberg, M. Brüggemann, and H. Schwartke, "Plc-blaster: A worm living solely in the plc," *Black Hat Asia*, vol. 16, pp. 1–16, 2016.

[27] Y. Zhang, Z. Sun, L. Yang, Z. Li, Q. Zeng, Y. He, and X. Zhang, "All your plcs belong to me: Ics ransomware is realistic," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2020, pp. 502–509.

[28] T. H. Ali Abbasi, Tobias Scharnowski, "Doors of durin: The veiled gate to siemens s7 silicon." [Online]. Available: https://i.blackhat.com/eu-19/Wednesday/eu-19-Abbasi-Doors-Of-Durin-The-Veiled-Gate-To-Siemens-S7-Silicon.pdf

[29] T. Keren, "The race to native code execution in PLCS." [Online]. Available: https://claroty.com/2021/05/28/blog-research-race-to-native-code-execution-in-plcs/

[30] CISA, "Schneider Electric Modicon Modbus Protocol." [Online]. Available: https://www.cisa.gov/news-events/ics-advisories/icsa-17-101-01

[31] ——, "Siemens PROFINET Devices (Update L)." [Online]. Available: https://www.cisa.gov/news-events/ics-advisories/icsa-19-283-02

[32] Siemens, "Creating userdefined web pages on s7-1200 / s7-1500." [Online]. Available: https://cache.industry.siemens.com/dl/files/496/

68011496/att_917318/v3/68011496_S7-1200_1500_Webserver_DOC_v22_en.pdf

[33] "Micrologix 1400 embedded web server." [Online]. Available: https://literature.rockwellautomation.com/idc/groups/literature/documents/um/1766-um002_-en-p.pdf

[34] "Web server function application guide." [Online]. Available: https://dl.mitsubishielectric.co.jp/dl/fa/document/catalog/plcf/l08643/l08643-a.pdf

[35] R. Automation, "Micrologix 1400 embedded web server." [Online]. Available: https://literature.rockwellautomation.com/idc/groups/literature/documents/um/1766-um002_-en-p.pdf

[36] Facebook, "Out of scope & false positive XSS." [Online]. Available: https://www.facebook.com/whitehat/education/false-positives/

[37] Rockwell Automation, "Delivery of CIP over RA serial DF1 links." [Online]. Available: https://www.rockwellautomation.com/content/dam/rockwell-automation/sites/downloads/pdf/CIPandPCCC_v1_1.pdf

[38] Siemens, "Basic examples for open user communication: ISO-on-TCP." [Online]. Available: https://cache.industry.siemens.com/dl/files/710/109747710/att_923140/v6/109747710__IsoOnTcp_BaseComm_V1_en.pdf

[39] GE, "PACSystems* RX7i & RX3i TCP/IP Ethernet Communications User Manual." [Online]. Available: https://www.manualslib.com/manual/1258748/Ge-Rx3i.html?page=22

[40] Elmi Elettromeccanica, "PLC siemens TIA portal - controller per turbine eoliche - automazione e robotica windmill." [Online]. Available: http://www.elmielettromeccanica.it

[41] WAGO, "Wago webvisu app." [Online]. Available: https://www.wago.com/us/software/webvisu

[42] Schneider, "How to configure web visualization in tm241 using ecostruxture machine expert?" [Online]. Available: https://www.se.com/in/en/faqs/FAQ000191672/

[43] Codesys, "Codesys webvisu." [Online]. Available: https://www.codesys.com/products/codesys-visualization/webvisu.html

[44] R. Madhusudhan *et al.*, "Cross channel scripting (xcs) attacks in web applications: Detection and mitigation approaches," in *2018 2nd Cyber Security in Networking Conference (CSNet)*. IEEE, 2018, pp. 1–3.

[45] S. Rahalkar, "Extending burp suite," in *A Complete Guide to Burp Suite*. Springer, 2021, pp. 131–145.

[46] J. Lajara, "JS-Recon detailed analyzing the internal network with a XSS." [Online]. Available: https://jlajara.gitlab.io/js-recon

[47] N. Krithika, "A study on wha (watering hole attack)–the most dangerous threat to the organisation," *Int. J. Innov. Sci. Eng. Res.(IJISER)*, vol. 4, pp. 196–198, 2017.

[48] M. Gaunt, "Service workers: an introduction." [Online]. Available: https://developers.google.com/web/fundamentals/primers/service-workers

[49] W3C, "Service workers." [Online]. Available: https://www.w3.org/TR/service-workers/

[50] Google, "Service worker security FAQ." [Online]. Available: https://chromium.googlesource.com/chromium/src/+/main/docs/security/service-worker-security-faq.md

[51] Mozilla, "Fetchevent: respondwith() method." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/FetchEvent/respondWith

[52] Cybersecurity and Infrastructure Security Agency, "Cybersecurity incident & vulnerability response playbooks." [Online]. Available: https://www.cisa.gov/sites/default/files/publications/Federal_Government_Cybersecurity_Incident_and_Vulnerability_Response_Playbooks_508C.pdf

[53] Siemens, "Simatic web server." [Online]. Available: https://cache.industry.siemens.com/dl/files/560/59193560/att_898124/v1/s71500_webserver_function_manual_en-US_en-US.pdf

[54] M. Salehi and S. Bayat-Sarmadi, "PLCDefender: Improving remote attestation techniques for PLCs using physical model," *IEEE Internet of Things Journal*, vol. 8, no. 9, pp. 7372–7379, 2021.

[55] whatwg.org, "HTML living standard web storage." [Online]. Available: https://html.spec.whatwg.org/multipage/webstorage.html

[56] Siemens, "SCALANCE XM-400/XR-500 web based management (WBM)." [Online]. Available: https://cache.industry.siemens.com/dl/files/663/109798663/att_1070766/v1/PH_SCALANCE-XM-400-XR-500-WBM_76.pdf

[57] WAGO, "PFC200 controller." [Online]. Available: https://www.wago.com/us/pfc200

[58] Cybersecurity and Infrastructure Security Agency, "Control system engineering workstation." [Online]. Available: https://www.cisa.gov/uscert/ics/Control_System_Engineering_Workstation-Definition.html

[59] Siemens, "Industry Online Support." [Online]. Available: https://support.industry.siemens.com/cs/start?lc=en-US

[60] K. Born, "Browser-based covert data exfiltration," *arXiv preprint arXiv:1004.4357*, 2010.

[61] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against {Third-Party} tracking on the web," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 155–168.

[62] MITRE, "Exfiltration Over Web Service." [Online]. Available: https://attack.mitre.org/techniques/T1567/

[63] K. Zetter, "An unprecedented look at Stuxnet, the world's first digital weapon." [Online]. Available: https://www.wired.com/2014/11/countdown-to-zero-day-stuxnet/

[64] D. Albright, P. Brannan, and C. Walrond, *Did Stuxnet take out 1,000 centrifuges at the Natanz enrichment plant?* Institute for Science and International Security, 2010.

[65] ICS Alert (ICS-ALERT-14-281-01E), "Ongoing sophisticated malware campaign compromising ICS (Update E)." [Online]. Available: https://us-cert.cisa.gov/ics/alerts/ICS-ALERT-14-281-01B

[66] Wired, "A Hacker Tried to Poison a Florida City's Water Supply." [Online]. Available: https://www.wired.com/story/oldsmar-florida-water-utility-hack/

[67] S. A. Qasim, J. M. Smith, and I. Ahmed, "Control logic forensics framework using built-in decompiler of engineering software in industrial control systems," *Forensic Science International: Digital Investigation*, vol. 33, p. 301013, 2020.

[68] S. E. McLaughlin, S. A. Zonouz, D. J. Pohly, and P. D. McDaniel, "A trusted safety verifier for process controller code." in *NDSS*, vol. 14, 2014.

[69] A. Romano, D. Lehmann, M. Pradel, and W. Wang, "Wobfuscator: Obfuscating javascript malware via opportunistic translation to webassembly," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1574–1589.

[70] K. Rieck, T. Krueger, and A. Dewald, "Cujo: Efficient detection and prevention of drive-by-download attacks," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 31–39. [Online]. Available: https://doi.org/10.1145/1920261.1920267

[71] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "Zozzle: Fast and precise in-browser javascript malware detection," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. USA: USENIX Association, 2011, p. 3.

[72] A. Fass, R. P. Krawczyk, M. Backes, and B. Stock, "Jast: Fully syntactic detection of malicious (obfuscated) javascript," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 15th International Conference, DIMVA 2018, Saclay, France, June 28–29, 2018, Proceedings 15*. Springer, 2018, pp. 303–325.

[73] A. Fass, M. Backes, and B. Stock, "Jstap: A static pre-filter for malicious javascript detection," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 257–269.

[74] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," *IEEE transactions on software engineering*, vol. 17, no. 12, pp. 1284–1288, 1991.

[75] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, "Source code metrics: A systematic mapping study," *Journal of Systems and Software*, vol. 128, pp. 164–197, 2017.

[76] Shodan, "Shodan Search Engine." [Online]. Available: https://shodan.io/

[77] Aplitude, "2022 App vs. Website Trend Report." [Online]. Available: https://amplitude.com/2022-app-vs-website-report#key-takeaways

[78] V. Raychev, P. Bielik, M. Vechev, and A. Krause, "Learning programs from noisy data," *SIGPLAN Not.*, vol. 51, no. 1, p. 761–774, jan 2016. [Online]. Available: https://doi.org/10.1145/2914770.2837671

[79] @HynekPetrak, "Javascript Malware Collection." [Online]. Available: https://github.com/HynekPetrak/javascript-malware-collection

[80] W3C Community Group, "Private network access." [Online]. Available: https://wicg.github.io/private-network-access/

[81] Github Security Engineering, "GitHub's post-CSP journey." [Online]. Available: https://github.blog/2017-01-19-githubs-post-csp-journey/

[82] S. Van Acker, D. Hausknecht, and A. Sabelfeld, "Data exfiltration in the face of CSP," in *Proc. of the 11th ACM on Asia Conf. on Computer and Communications Security*, 2016, pp. 853–864.

[83] Mozilla Firefox, "Security/CSP/Confidentiality." [Online]. Available: https://wiki.mozilla.org/Security/CSP/Confidentiality

[84] Google, "Content hosting for the modern web." [Online]. Available: https://security.googleblog.com/2012/08/content-hosting-for-modern-web.html

Appendix I. Appendix

A. PLC Web Capability Trends.

We independently confirmed that embedded PLC webservers are indeed gaining functionality over time by analyzing the total Source Lines of Code (SLOC) and cyclomatic complexity [74] of the web-related codebases of multiple unpacked WAGO firmware update images. Source code size and program complexity are common metrics in software analysis to gauge the capabilities/functionality of a given application [75]. Firmware version 3.0.39 (released May 2019) contained 13,188 total SLOC (12,868 JS; 320 PHP) and an aggregate cyclomatic complexity score of 4,529 (2,922 JS; 1,607 PHP) while version 4.02.13 (released June 2023) contained 39,007 total SLOC (38,444 JS; 563 PHP) and an aggregate cyclomatic complexity score of 11,974 (9,294 JS; 2,680 PHP). This data shows that over the past several years, the web application codebase has grown by over 195% and increased in complexity by over 164%. All source code analyzed in our study is used solely by the embedded web application, thus corroborating the claim that PLC vendors are actively introducing additional functionality to their on-board web applications. The raw data from this study is included as an artifact of our paper.

B. PLC Webserver Usage Study.

We experimentally verified that customers are increasingly using PLC embedded webservers by performing a modest longitudinal survey of internet-facing devices using the Shodan [76] search engine. We analyzed the publicly-reachable population of three widely-used PLCs (WAGO 750, Siemens S7-1200, and AB MicroLogix 1400) from June 2017 to Septemeber 2022. We discovered that on average, embedded webserver usage has increased 212.66% over the past 5 years, even though overall PLC population has only increased by 12.15%. We came to this conclusion by observing the rate at which the webservers became internet-facing (discovered using web fingerprints such as SSL issuers and favicon hashes) and comparing it to the rate at which the SNMP services appeared online (discovered using keywords on the 161 UDP port). This data provides strong evidence that customers are indeed enabling and using these webservers. Our results are in-line with previously published data regarding the adoption of web technology [23], [77] and further supports the intuitive claim that PLC customers are embracing web-based app design.

C. JavaScript Malware Detectors.

While JavaScript obfuscation is outside the scope of this paper, we still decided to run *IronSpider* against state-of-the-art JavaScript malware detectors for completeness. We modeled our approach similarly to Ramano at el., 2022's recent paper [69] where they setup and trained various JS static code analysis tools on large datasets of benign and malicious JavaScript samples. Like Ramano at el., we primarily used the popular JS150k dataset [78] for benign files and the Hynek Patrak JavaScript malware collection [79] for malicious files. We also used the same static analysis tools, namely Fass et al.'s reimplementation of Cujo [70], Fass et al.'s reimplementation of Zozzle [71], JaSt [72], and JStap [73]. All four tools incorrectly classified *IronSpider* as benign. After manually inspecting the Patrak malware collection, we believe the most likely explanation for this poor performance is that most IT-oriented malware contains exceedingly aggressive indicators (e.g., mining cryptocurrency in WebAssembly, attempting to drop *.exe* files, using off-the-shelf exploit kits, etc), while *IronSpider* more-or-less just uses the DOM interface as intended. Our experiment strongly suggests that conservative and well-behaving JavaScript malware is extremely difficult to detect.

D. Countermeasures Discussion.

A potential improvement is *limiting communication between the public Internet and private Intranet*. Providing this defence will greatly reduce the possibility of a "Web Access" infection (see Section III) because it would restrict public websites' access to PLCs' embedded webservers using HTML and/or JavaScript network requests. In theory, this prevention should cause many PLC web bugs such as rXSS and CSRF to become mostly un-exploitable from malicious websites originating from the public web. Google Chrome engineers have already proposed this countermeasure in the form of the "*Private Network Access*" specification (previously known as *CORS-RFC1918*) [80], however this defence has not yet been implemented in any major browser. While a full adoption of this protection will definitely help in mitigating web-based infections, it is unfortunately not a perfect solution in all cases. Currently, the draft does not apply to local HTML files (even when delivered via a malicious USB drive). This draft also will not prevent an existing WB malware infection from spreading to other PLCs because existing WB malware will already have a private IP address, which is permitted to interact with other private IP addresses according to the spec.

Another area of potential improvement for browsers is *adding built-in exfiltration defences*. There are currently no browser mechanisms that prevent rogue JavaScript code from covertly connecting to, and exfiltrating sensitive data to, a third party remote server. This lack of protection is what enables our proposed malware to establish a C2 channel with the public internet. While some websites attempt to use Content-Security Policy (CSP) as a makeshift exfiltration defence [81], prior work has shown that this is an inadequate protection because methods such as DNS resolution, subresource prefetching, and navigation redirects easily defeat it [82]. In 2012, Firefox developers proposed a new CSP directive called "*Confidentiality*" [83] designed to fully eliminate exfiltration attacks, however it was shelved shorty after creation due to reprioritization. We hope that this paper will inspire browser developers to reevaluate that decision.

Lastly, all JavaScript code authored by the PLC customer (either in UWPs or GUI files) should be fully isolated from the front end properties authored by the PLC vendor. As mentioned in Section IV-B, the IT domain has a well-established method for performing origin isolation called "domain sandboxing." Using this method, untrusted front end files are hosted on a different web origin, usually with a different domain name, than the main website. For example, Facebook hosts customer-written code on *fbsbx.com* [36] and Google hosts customer-written code on *googleusercontent.com* [84]. This origin isolation prevents untrusted code from having access to the ambient browser credentials associated with the main origin, which are needed to access sensitive web APIs.

In ICS environments, where IP addresses are typically used instead of DNS to access network devices, a sandboxing solution may seem infeasible, however can still be accomplished using a slightly different approach. Instead of using a different domain name, PLC vendors can either host untrusted front end files on a different port of the same IP address or utilize the CSP *sandbox* directive to virtually render untrusted content on an isolated *opaque* origin.

Appendix II. Artifact Appendix

This artifact consists of four unique and self-contained experiments, as listed below:

1) (E1) WAGO 750-8XXX WBM Application Code Base study
2) (E2) WAGO Exploit Code
3) (E3) Allen Bradley Exploit Code
4) (E4) Resurrection Example

The first experiment, *(E1) WAGO 750-8XXX WBM Application Code Base study*, is used to gauge the approximate functionality increase over time of the on-board, embedded, web application hosted by WAGO 750 PLCs. This study supports the paper's central idea - that the web-related PLC attack surface has substantially grown in recent years, which makes the front-end environment a new and appealing platform to run malware. The second two experiments, *(E2) WAGO Exploit Code* and *(E3) Allen Bradley Exploit Code*, are provided to show example weaponizations of the zero-day vulnerabilities discovered in this study. We encourage readers to test these vulnerabilities on the latest operating systems and browsers to show that these attacks are indeed compatible with a modern threat model. The final experiment, *(E4) Resurrection Example*, is an example implementation of a malicious Service Worker JavaScript file, which trivially bypasses Service Worker restrictions, and may be used to resurrect WB malware from within the ICS network.

A. Description & Requirements

1) How to access:

- https://zenodo.org/record/8279954

2) Hardware dependencies:

- (E1) None; (E2) WAGO 750 PFC200 PLC; (E3) Allen Bradley MicroLogix 1400 PLC; (E4) None

3) Software dependencies:

- (E1) complexity-report (https://github.com/escomplex/complexity-report); (E1) lint_php (https://github.com/pceres/lint_php); (E1) sloccount (https://dwheeler.com/sloccount/); (E2) WAGO 750 PFC200 <FWv04.02.13; (E3) Allen Bradley MicroLogix 1400 <FWv21.007; (E4) None

4) Benchmarks:

- (E1) None; (E2-E3) Included exploit code; (E4) Included example Service Worker JavaScript file

B. Artifact Installation & Configuration

- (E1) None; (E2-E3) Appropriate PLC hardware and corresponding firmware must be installed within an ICS network. NDSS reviewers were given remote access to our testbed during the peer review process; (E4) Standard JavaScript Service Worker Installation (*navigator.serviceWorker.register('/sw.js')*), alternatively view live demo at https://ndss-2024-23049.s3.amazonaws.com/index.html.

C. Experiment Workflow

D. Major Claims

Our experiments support the following major claims presented in our paper:

1) (E1) PLC vendors are actively introducing new functionality into their embedded webservers. This is referenced in Appendix I-A.
2) (E2) The vulnerabilities we discovered in WAGO (CVE-2022-45139, CVE-2022-45138, CVE-2022-45140) allow for the deployment of web-based PLC malware. This is used in Section IV-B2.
3) (E3) The vulnerabilities we discovered in Allen Bradley (CVE-2022-46670) allow for the deployment of web-based PLC malware. This is used in Section V-C.
4) (E4) Our novel usage of Service Workers allows WB PLC malware to survive firmware/hardware factory resets. This is used in Section IV-C2.

E. Evaluation

1) Experiment (E1): [WAGO 750-8XXX WBM Application Code Base study] [1 human-hour]: Empirical study to independently confirm that PLC vendors are actively introducing functionality to their embedded webservers over time via firmware updates. We use the following metrics to gauge functionality: Source Lines of Code (SLOC) and Cylomatic Complexity. We consider a file "web-related" if it is in the language PHP (for backend files) or JavaScript (for front-end files).

[How to (E1)] In this experiment, we compare the aggregate SLOC and complexity scores from two WAGO PLC firmware versions to estimate the embedded webserver functionality increase as a percent over time. A large percent increase in SLOC and/or complexity suggests a large amount of functionality has been added. We use industry-standard tools

to examine the source code unpacked from publicly available firmware updated images.

[Preparation (E1)] Download the oldest and newest WAGO 750-8XXX firmware images (.img) from their GitHub page (https://github.com/WAGO/pfc-firmware/releases). At the time of writing, that is the following images:

1) Oldest - v03.0.39 - https://github.com/WAGO/pfc-firmware/releases/download/v3.0.39/WAGO_FW0750-8xxx_V030039_IX12_r38974.img
2) Newest - v04.02.13 - https://github.com/WAGO/pfc-firmware/releases/download/v04.02.13-24/PFC-G2-Linux_sd_V040213_24_r74297.img

[Execution (E1)]

1) Unpack Firmware images (done automtically via the Docker Image; see README for details)
2) Use *complexity-report*, *lint_php*, and *sloccount* to analyze the web-related codebases (done automtically via the Docker Image; see README for details)
3) Calculate the percent increase (done automtically via the Docker Image; see README for details)

[Results (E1)] When the outputs from these tools are aggregated, we get the following metrics: Firmware version 3.0.39 (released May 2019) contained 13,188 total SLOC (12,868 JS; 320 PHP) and an aggregate cyclomatic complexity score of 4,529 (2,922 JS; 1,607 PHP) while version 4.02.13 (released June 2023) contained 39,007 total SLOC (38,444 JS; 563 PHP) and an aggregate cyclomatic complexity score of 11,974 (9,294 JS; 2,680 PHP). This data shows that over the past several years, the web application codebase has grown by over 195% and increased in complexity by over 164%.

2) Experiment (E2): [WAGO Exploit Code] [30 human-minutes]: Example weaponization of CVE-2022-45139, CVE-2022-45138, and CVE-2022-45140 to deploy “Web-Based (WB) PLC malware” into a WAGO 750 PLC pre Firmware 24 (v04.02.13).

[How to (E2)] In this experiment, we use our zero-day vulnerabilities to deploy web-based malware from the web-access level. The exploit code is already included and creates dummy HTML and CSS files as a proof-of-concept.

[Preparation (E2)] Configure a WAGO 750 PLC in you network and flash firmware v04.02.13 to memory.

[Execution (E2)]

1) Verify that https://⟨wago_ip⟩/evil.html is not present (404 - not found)
2) Open *Exploit.html* in your default Web Browser
3) Enter the WAGO PLC IP Address
4) Click *“Attack!”* button
5) Verify that https://⟨wago_ip⟩/evil.html is now present

[Results (E2)] This exploit demonstrates that a third party website is able to add arbitrary front-end files into PLC memory without authentication. An attacker can use this chain to deploy WB PLC Malware.

3) Experiment (E3): [Allen Bradley Exploit Code] [30 human-minutes]: Example Python script to exploit CVE-2022-46670 to push Web-Based (WB) PLC Malware to a MicroLogix 1400 pre Firmware V21.007.

[How to (E3)] In this experiment, we use an ICS Cross-Channel Scripting (XCS) vulnerability to push JavaScript into the MicroLogix 1400 PLC embedded webserver over the SNMP protocol, which gets rendered on the homepage of the system website.

[Preparation (E3)] Configure a Allen Bradley MicroLogix 1400 PLC in you network and flash firmware V21.007 to memory.

[Execution (E3)]

1) Verify that http://⟨ab_ip⟩/ is present without any popup windows
2) Run *python3 exploit.py* $< ab\_ip >$
3) Refresh http://⟨ab_ip⟩/
4) See *alert()* popup, proving arbitrary JavaScript injection into the system website

[Results (E3)] This exploit demonstrates that an attacker with network access to the PLC is able to add arbitrary JavaScript code to the system website without authentication. An attacker can use this bug to deploy WB PLC Malware.

4) Experiment (E4): [Resurrection Example] [30 human-minutes]: Example Service Worker script that checks for the removal of the main WB malware infection payload, and when detected, re-infects the device using CVE-2022-45140.

[How to (E4)] In this experiment, we show how a malicious website can use a Service Worker to gain a web-based foothold in a private industrial network by caching secondary malware payloads in browsers throughout the ICS environment. This allows the website to continue execution, and potentially re-infect the device, even after the infected webserver device has been completely factory reset or replaced.

[Preparation (E4)] Host the provided *sw.js* file on an SSL-secured webserver (e.g., *https://10.0.0.1/sw.js*) next to a blank HTML file (e.g., *https://10.0.0.1/index.html*).

[Execution (E4)]

1) Inspect the *sw.js* file and see how it injects itself into the main page by modifying the proxied *fetch* API calls.
2) (optionally) Install the sw.js via $navigator.serviceWorker.register()$; from the HTML file, Refresh the page and observe SW execution, Delete the sw.js file from the server (simulating a factory reset), Refresh again and continue observing execution. From this position, the code could re-perform E2 to re-infect the device.

[Results (E4)] Observe that Service Workers can trivially bypass all WC3 limitations (e.g., no DOM access, localStorage, or other synchronous APIs) by simply injecting itself into the main page via the proxied *fetch* API. Observe that this code continues to execute, even after the removal of the SW file. Note that this code can potentially re-perform E2 to re-infect the device. Readers can view a live demo at https://ndss-2024-23049.s3.amazonaws.com/index.html.