Helper-in-the-Middle: Supporting Web Application Scanners Targeting Industrial Control Systems

Anne Borcherding[1][a], Steffen Pfrang[1][b], Christian Haas[1], Albrecht Weiche[1] and Jürgen Beyerer[1,2]
[1]*Fraunhofer IOSB, Karlsruhe, Germany*
[2]*Vision and Fusion Laboratory (IES), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*

Keywords: Industrial Control Systems, Black Box Security Testing, Web Application Scanners, Proxy, Usability.

Abstract: Web applications on industrial control systems (ICS) provide functionality such as obtaining status information or updating configurations. However, a web application possibly adds additional attack vectors to the ICS. In order to find existing vulnerabilities of web applications, automated black box web application scanners (WAS) can be used. Evaluations of existing scanners show similar limitations in their applicability. For example, ICS often crash during a scan. If the used scanner does not recognize and handle this issue, it is not able to finish the test. We present HelpMeICS which makes improvements available for different scanners without the need to adapt the specific scanner. It is implemented as a proxy-based solution which is transparent for the scanners and handles different aspects such as error-handling, authentication, and replacement of contents. Our evaluation with five different ICS shows an improvement of applicability as well as a reduction of additional limitations of WAS. As an example, our improvements increased the URL coverage from 8% to 100%. For one of the ICS, a complete scan was only made possible by HelpMeICS since the ICS crashed irrecoverably during the scans without HelpMeICS.

1 INTRODUCTION

ICS play an elementary role in making production faster, more reliable and more flexible. Corresponding device classes are, amongst others, programmable logic controllers (PLC), bus couplers, gateways, and industrial firewalls. Most ICS include a communication interface over Ethernet to transfer process data or status information, either to a local communication partner or even to a communication partner in the cloud. Due to a high priority on improving the features of ICS, security often gets neglected during the development process.

The lack of security is critical since ICS pose an interesting target for attackers. With their two communication channels, ICS possibly build a bridge for an attacker trying to attack the production process from the Internet. In a simple case – as discovered during this work – a single HTTP packet suffices to stop a production process from anywhere in the world. A more advanced case is pivoting in the automation network which may also lead to data leakage, disruption of safety systems, damage to facilities or even human harm. This situation is presented in Figure 1: The attacker uses the web application of an ICS as an entry point. Based on this, the attacker can start pivoting using, for example, unsecured industrial communication protocols such as PROFINET.

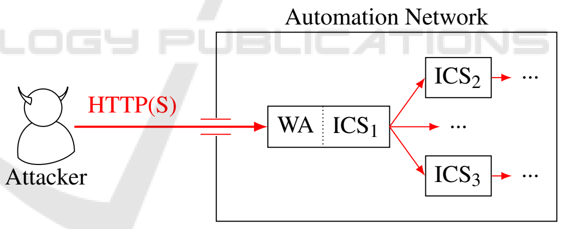

Figure 1: An attacker uses the web application (WA) of ICS_1 as an entry point to the automation network.

Security testing is a major measure to improve the security of software in general and of ICS in particular. It aims at revealing vulnerabilities in an early stage of the development life cycle as well as finding vulnerabilities in a later stage. There exist different security testing approaches, ranging from model based tests of the architecture to penetration tests of the already published product (Felderer et al., 2016). The requirement to perform security testing is also postulated in the *IEC 62443 standard* (IEC62443,

[a] https://orcid.org/0000-0002-8144-2382
[b] https://orcid.org/0000-0001-7768-7259

Borcherding, A., Pfrang, S., Haas, C., Weiche, A. and Beyerer, J.
Helper-in-the-Middle: Supporting Web Application Scanners Targeting Industrial Control Systems.
DOI: 10.5220/0009517800270038
In *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications (ICETE 2020) - SECRYPT*, pages 27-38
ISBN: 978-989-758-446-6

2019) which targets security for industrial automation and control systems.

The aim of this work is to detect vulnerabilities in ICS using automated black box WAS. WAS provide a cost-efficient approach on finding vulnerabilities of a web application. Since they run automatically, they can be used to conduct tests on a regular basis (Bau et al., 2010). However, literature shows various similar limitations of WAS, many of them concerning the methodology and applicability of WAS.

This work presents a comprehensive solution approaching many of these limitations. Note that we are not presenting a new WAS since we are convinced that there are plenty sophisticated WAS out there. Instead, we are presenting a framework to improve the methodology and applicability of existing WAS: HelpMeICS. With HelpMeICS, it is possible to make improvements transparently available for arbitrary WAS. It consists of a proxy, which is located between the WAS and the device under test (DUT), containers in which the WAS run, and an integration into the industrial security testing framework ISuTest (Pfrang et al., 2017).

In summary, the main contributions of this paper are the following:

- Review of limitations identified and improvements achieved by existing evaluations of WAS.
- HelpMeICS: A system to make improvements of methodology and applicability available for existing WAS.
- Extensive evaluation of HelpMeICS using real ICS as targets that shows reduced limitations, increased number of true positive reports as well as a higher URL coverage.
- Publication of the source code of the developed add-ons for the open source proxy *mitmproxy*, which also can be run as a stand-alone system[1,2].

Following this introduction, Section 2 provides more information on ICS and security testing. In Section 3, we present a review to show limitations identified and improvements achieved by literature. Building upon this background, we present the design of HelpMeICS in Section 4. A qualitative and quantitative evaluation is shown in Section 5. Finally, our conclusions are presented in Section 6.

[1]https://github.com/mitmproxy/mitmproxy/pull/3961

[2]https://github.com/mitmproxy/mitmproxy/pull/3962

2 BACKGROUND

Automated black box security testing of all communication interfaces plays an important role in improving ICS security. This includes web security testing if the DUT provides a web interface. The following provides some context on ICS and security testing as well as on ISuTest, a security testing framework that serves as a building block for HelpMeICS.

2.1 Industrial Control Systems

The main task of ICS is to control and monitor processes in industrial environments. In order to achieve this task or to provide additional services, ICS need to communicate to other nodes in the industrial network. In former times, industrial networks have been set up using proprietary communication protocols and corresponding wiring. Nowadays, the communication is performed using Ethernet wiring and Ethernet-based protocols. By using these Ethernet-based protocols such as PROFINET or EtherCat, requirements of industrial communication such as real-time communication can be fulfilled using standardized Ethernet hardware (Galloway and Hancke, 2012).

Next to these specialized protocols, ICS communicate using standard Internet protocols such as UDP or TCP. For an attacker, the barrier to attack these protocol stacks is especially small, since well known techniques and tools exist. In addition, the possible outcome of an successful attack is enormous. An successful attack can lead to an interruption of production, data leakage as well as physical damage to the production or humans (BSI, 2019).

Awareness for security issues in the industrial domain is rising steadily. One of the triggers for this surely was Stuxnet (Langner, 2011). By now, the International Electrotechnical Commission (IEC) has published several standards concerning the security for ICS (IEC62443, 2019). But still, the implementation and integration of security in industrial networks and systems is an ongoing process that just has started. To keep this process going, practicable solutions to find vulnerabilities need to be offered. Our work contributes to the fulfilment of this need.

2.2 Security Testing

Security testing can be categorized and arranged using various metrics and dimensions. A full description of these categories is beyond the scope of this work which is focussed on *automated black box web application scanning of ICS*. Nevertheless, the specifics of our point of view on security testing for

ICS are to be elaborated. Felderer et al. give a good overview on the categorization of security testing (Felderer et al., 2016).

Automated: This keyword refers to the meaning of automated as opposed to manual. Automated security testing is easier to integrate into a development life cycle, since tests can be run on a regular basis and the security status of the DUT can be monitored easily.

Black Box: For black box testing, only the external inputs and outputs of the DUT are considered. With this, no internal information of the DUT needs to be disclosed. Simplified, input is sent to the DUT and the following reaction of the DUT is monitored based on its output. In the case of ICS testing, the inputs are network packets and the output may be any external interface of the DUT. This includes digital or analog I/O as well as various network protocol stacks.

Web Application Scanning: Each additional feature of an ICS poses an additional attack vector. This also holds for the web applications many ICS are running for maintenance purposes. Web applications simplify maintenance tasks such as obtaining status information or updating the firmware. Additionally, they often simplify attacking the ICS. Many hacking tools are available for the web domain since web application hackers have existed almost as long as the web exists. This makes hacking web applications easier than hacking an industrial protocol for which an attacker needs special domain knowledge. It follows that the existence of web applications on ICS makes hacking ICS much easier.

2.3 ISuTest

ISuTest is a security testing framework for ICS (Pfrang et al., 2017). We use its different features for black box testing of ICS for HelpMeICS. One feature of ISuTest is to configure and start so called *test scripts* which define the parameters and specifications of test sequences. For example, a test sequence might be a fuzzing test for UDP or a test sequence probing the DUT for a specific vulnerability in its PROFINET protocol stack. During a test, the DUT is *monitored* continuously using different outputs such as digital or analog I/O as well as network protocols such as ICMP, TCP, HTTP, and PROFINET. In addition, ISuTest is able to restart a DUT by switching the power off and on. With this, ISuTest is able to reset the DUT before each test and is also able to restart the DUT after a crash. We are using these features of ISuTest as one building block for HelpMeICS.

3 REVIEW

This work addresses the following requirements: (I) reducing the limitations of existing WAS regarding their applicability, while (II) not restricting this reduction to some WAS, but by making improvements available independently of the specific WAS, and (III) making the code publicly available. To the best of our knowledge, no existing open source tool fulfills all of the formulated requirements. For the usage against web applications on industrial hardware, DUT specific changes are often necessary. Because of this, public availability of the code is a fundamental requirement. As a result, we do not include commercial solutions but focus on results from literature.

Literature includes various evaluations of WAS which show limitations as well as improvements. The following review is based on 15 papers aiming at comparing different black box WAS (Bau et al., 2010; Pfrang et al., 2019; Makino and Klyuev, 2015; Idrissi et al., 2017; Vega et al., 2017; Doupé et al., 2010; Ferreira and Kleppe, 2011; Suteva et al., 2013; Alassmi et al., 2012; Fonseca et al., 2007; Khoury et al., 2011; McAllister et al., 2008; Doupé et al., 2012; Deepa et al., 2018; Esposito et al., 2018). In total, these papers evaluated 25 WAS using 25 targets.

In summary, it shows (I) that there are many different WAS relevant enough to be considered, (II) that none of the WAS clearly stands out, and (III) that a combination of WAS leads to better results. From these insights, we derived the necessity of a system which improves and orchestrates existing WAS.

Next to the comparisons of WAS, literature shows various limitations that are similar for many WAS, as well as work to reduce these limitations. In the following, we present a classification of the limitations identified and improvements made (see Sections 3.1 and 3.2). It shows that a significant part of limitations concerns the applicability of WAS, but that there has been only little effort to reduce these limitations. Nevertheless, applicability of WAS is an important factor in their usefulness and, consequently, for their spreading. Based upon this insight, we focus on reducing the limitations of WAS regarding their applicability.

3.1 Limitations Identified

In order to give a better understanding of the limitations of WAS revealed by literature, we collected and classified them. We identified the following clusters: *target*, *vulnerability classes*, *methodology* and *applicability*. In fact, the cluster *target* includes limitations of the targets used for the evaluation instead of limitations of the WAS itself. This cluster includes the limitations that targets exhibit unintended vulnerabilities (Makino and Klyuev, 2015) and that the exhibited vulnerabilities are outdated (Suteva et al., 2013). For the sake of completeness we included them in our classification but we will not go into it any further in the following since they are not concerned with the WAS itself.

The clusters as well as the classified limitations and the corresponding papers are shown in Table 1. The cluster *vulnerability classes* includes types of vulnerabilities WAS have shown to struggle with especially. In the cluster *methodology*, limitations of general approaches of WAS are collected. For example, this includes Crawling and Fuzzing capabilities, handling of Periodically Changing Content, and Application Logic limitations. The limitation of Periodically Changing Content refers to web applications that include regularly changing content such as the current time. This content can pose a problem for WAS while they are probing the DUT for injection vulnerabilities. To find out if injections are possible, WAS first try to inject code to a web application. Second, they check if the content of the web application is different to what it had been before the injection. If the web application contains content that changes regularly, WAS assume they were able to change the content and they reason that they were able to inject code. However, in reality only the regularly changing content such as the current time has changed. Application Logic limitations include the checking and detection of application specific vulnerabilities.

Table 1: Limitations of WAS as shown by literature.

Vulnerability Classes	
Stored XSS	(Bau et al., 2010; Doupé et al., 2010; Alassmi et al., 2012)
Stored SQL	(Bau et al., 2010; Doupé et al., 2010; Khoury et al., 2011)
Remote File Inclusion	(Makino and Klyuev, 2015; Doupé et al., 2010)
Local File Inclusion	(Idrissi et al., 2017)
Path Disclosure	(Doupé et al., 2010)
Methodology	
Crawling	(Esposito et al., 2018; Doupé et al., 2010; Idrissi et al., 2017; Ferreira and Kleppe, 2011; Khoury et al., 2011)
Fuzzing	(Idrissi et al., 2017)
Periodically Changing Content	(Pfrang et al., 2019)
Attack Code Selection	(Ferreira and Kleppe, 2011; Khoury et al., 2011)
User Login	(Pfrang et al., 2019; Doupé et al., 2010; Khoury et al., 2011)
Second Order Vulnerabilities	(Bau et al., 2010; Doupé et al., 2010)
Application Logic	(Doupé et al., 2010)
JavaScript / Flash / HTML5	(Doupé et al., 2010; Idrissi et al., 2017; Doupé et al., 2012)
Categorization of Findings	(Khoury et al., 2011)
Applicability	
Parameter Understandability	(Esposito et al., 2018)
Reproducibility	(Pfrang et al., 2019)
Conduction of Single Tests	(Pfrang et al., 2019)
Load Reduction	(Pfrang et al., 2019)
Pause and Resume a Scan	(Pfrang et al., 2019)
Behavior in Case of an Error	(Pfrang et al., 2019)
Different Scanners Necessary	(Esposito et al., 2018; Pfrang et al., 2019; Idrissi et al., 2017)
Runtime	(Pfrang et al., 2019; Doupé et al., 2010; Suteva et al., 2013)

The cluster *applicability* refers to limitations of WAS concerning the practical handling of WAS. Load Reduction refers to limitations in specifying the load to be sent to the DUT. Especially when using ICS, a crash of the DUT because of a high load is probable. Behavior in Case of an Error is connected to this. If the DUT crashes and the WAS does not recognize the problem, the WAS will send its probes into the emptiness. Literature has shown that different scanners are necessary to generate a good coverage. Another important point is the Runtime of a WAS. It has a high impact on how often a test of the DUT is conducted.

Note that the presented limitations are not specific to the domain of ICS. However, some of the limitations become more prominent while using WAS against ICS.

3.2 Improvements Achieved

Building upon the revealed limitations, researchers have worked on reducing these limitations. The following describes works in this domain and correlates them to the limitations described in Section 3.1.

McAllister et al. built a WAS enhanced by techniques to detect more entry points for the scan. This includes the recording and replaying of use cases, fuzzing various forms at the same time to increase testing breadth, and stateful fuzzing. Note that the stateful fuzzing is only possible if the WAS is able to control the web application. They find that their solution is able to identify more entry points and bugs than the other evaluated WAS (McAllister et al., 2008).

Doupé et al. present a state-aware WAS. Through sending packets to the DUT and interpreting the responses, it builds a model of the internal states of the web application. Using this model, new entry points are derived and tested. Additionally, the model is used to build new payloads for the included fuzzer. Their evaluation shows that the code coverage as well as the effectiveness of vulnerability tests are improved by their system (Doupé et al., 2012).

Deepa et al. present a WAS that is using the data and control flow of a web application to build a model of its behavior. With this model, logic vulnerabilities like parameter manipulation, access control and work flow bypass vulnerabilities are targeted. The authors show that their system results in a high precision and a high true positive rate (Deepa et al., 2018).

In contrast to the solutions above, Esposito et al. present a proxy to improve the capabilities of WAS instead of building a new WAS. The proxy is used to improve the crawling capabilities as well as the authentication of the WAS against the target. Crawling is improved by injecting known URLs into places the WAS will look into for crawling information. Additionally, the proxy is able to authenticate the originally unauthenticated requests sent by the WAS. The authors show in their evaluation that the presented proxy increases the number of detected vulnerabilities without having a significant impact on the reported false positives (Esposito et al., 2018).

In summary, literature has improved WAS in their limitations of Crawling (McAllister et al., 2008; Esposito et al., 2018; Doupé et al., 2012), Fuzzing (McAllister et al., 2008; Doupé et al., 2012), Application Logic (Deepa et al., 2018), and User Login (Esposito et al., 2018; McAllister et al., 2008). All these improvements focus on the methodology of the WAS and not on their applicability. As we have learned from our practical work with WAS, the applicability of WAS is an important factor in their usefulness and spreading. That is why there should be improvements in this cluster as well.

4 OUR APPROACH

Our approach is to support and improve arbitrary WAS through a transparent helper-in-the-middle. The following presents the concept of HelpMeICS as well as the extensions we designed and implemented. The goal of HelpMeICS is to improve the following applicability limitations of WAS: Crawling, Periodically Changing Content, User Login, Reproducibility, Conduction of Single Tests, Load Reduction, Pause and Resume a Scan, Behavior in Case of an Error, and Different Scanners Necessary.

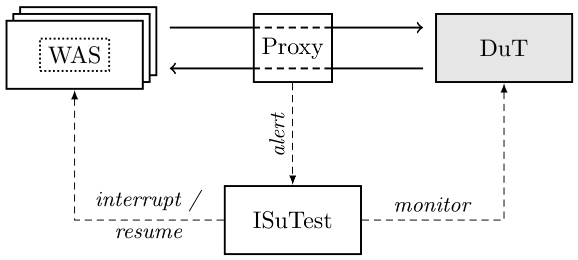

Figure 2: Overview over HelpMeICS. At a given time, one WAS running in a container can communicate with the DUT using the proxy. When ISuTest receives an alert by the watchdog of the proxy, it interrupts the currently running WAS and monitors the DUT. All components except the DUT run on one computer.

4.1 Basic Idea

The basic idea of HelpMeICS is to combine the advantages of a proxy, ISuTest, and containers to support WAS during their testing. Our concept for this combination is presented in Figure 2. The three components proxy, WAS, and ISuTest run on one computer which is called test device (TD).

The *proxy* is located in between the WAS and the DUT. With this, it is able to read, interpret and change the packets sent between the WAS and the DUT. The extensions of the proxy are implemented as add-ons which are presented in detail later in this section.

The WAS run in *containers* which results in the following two main advantages: First, it is now possible to pause and resume the WAS. Second, if the container is not updated, scans can be run in a more reproducible way since the WAS can be reset to a known state. The third component of HelpMeICS is *ISuTest*.

We use ISuTest to orchestrate the WAS and to monitor the DUT. In order to orchestrate the WAS, we added the possibility to integrate external tools into ISuTest. This is not restricted to WAS. In addition, we use the monitoring of ISuTest. The monitoring is supported by the so-called *watchdog* add-on of the proxy which alerts ISuTest if the DUT gives the impression of no longer reacting properly to HTTP requests. When the watchdog of the proxy detects a possible error of the DUT, it alerts ISuTest. ISuTest then performs a monitoring cycle and checks if the DUT indeed has an error. Refer to the description of the watchdog later in this section for more information on the monitoring.

Our solution can be used by proxy-aware WAS as well as by WAS that are not proxy-aware. For those WAS that are not proxy-aware, we implemented a *transparent proxy mode*. The challenge for the transparent proxy mode is to differentiate between the packets to be routed to the proxy and the ones that should not be routed to the proxy. This is made possible by HelpMeICS because of the containers the WAS run in. We added a virtual network bridge to the TD which redirects all traffic from and to the containers to the proxy. With this, HelpMeICS can be used even with WAS that are not proxy-aware. Note that HelpMeICS can also be used if it is not possible to run the WAS in a container. However, this reduces the impact of HelpMeICS (see Section 5) and the transparent proxy mode is not possible.

Esposito et al. present a proxy-based solution to improve WAS as well (Esposito et al., 2018). The authors show that their approach is able to improve Crawling and User Login. HelpMeICS goes beyond this approach and addresses more limitations of WAS. We present a full integration of the WAS into a security testing system without the need to adapt the WAS. This includes ISuTest, running the WAS in a container and additional add-ons for the proxy. In addition, we publish the add-ons to make them available for further research. Note that the proxy can also be used standalone. As a result, many of our improvements can be used just by installing and configuring the proxy.

4.2 Extensions

We built the containers for the WAS using LXD[3]. Our decision was mainly based on the fact that LXD provides an easy configuration and manipulation of the forwarding for virtual network bridges. This is necessary for the transparent proxy mode.

As a basis for the proxy included in HelpMeICS, mitmproxy[4] is used. We chose this proxy since it is easily extensible and its code is open source under the MIT license. In addition, it supports HTTP as well as HTTPS. With this, it builds a good base for this work. In the following, we present the additional features for mitmproxy we designed and implemented.

Watchdog. The watchdog add-on addresses the limitation of crashing DUTs during a test (*Behavior in Case of an Error*). For this, it monitors whether the DUT shows errors in the communication with the WAS using the error method provided by mitmproxy. This method is called if an error occurs during the transmission or reception of an HTTP packet. Serving as a connection between mitmproxy and ISuTest, the watchdog alerts ISuTest and such informs about the error. In Figure 2 this process is presented through dashed arrows. When receiving an alert, ISuTest interrupts the WAS. This interruption is made possible because of the container the WAS runs in. The next step for ISuTest is to conduct a monitoring cycle. A monitoring cycle consists of a check if the DUT responds to requests of different protocols. If the check shows that the DUT is indeed not responding, ISuTest restarts the DUT and checks it again. Then, the WAS is resumed and the test is continued. Note that this process is completely transparent for the WAS.

[3]https://linuxcontainers.org/

[4]https://mitmproxy.org/

Mapping. The mapping add-on aims at reducing the false positives resulting from changing content (*Periodically Changing Content*). It provides the possibility to replace or delete content of a web page, for example regularly changing content such as the current time. Content referred to by a CSS selector is replaced by a given HTML code.

User Login. We present two different approaches to address the limitation concerning user login or authentication (*User Login*). The first approach implements different authentication schemes and executes the authentication if necessary. With this, the requests of the WAS are being authenticated transparently. One limitation of this approach is that it does not have access to a browser instance. With this, it is not possible to fully retrieve and execute the web page. The second approach addresses this limitation of the first approach by using selenium[5]. Selenium includes a browser instance with which we can simulate a login by a user. Afterwards, the login information such as cookies or session IDs need to be transferred to the requests of the WAS. Both approaches share the need to detect if a request is authenticated or not. This check is performed during a request of the WAS as well as during a response of the DUT.

URL Injection. In order to improve the *Crawling* capabilities of the WAS, we present an URL injection add-on. With this add-on, URLs are injected into the responses sent by the DUT. As has been shown by Esposito et al., this improves the crawling performance of the WAS since the WAS are able to find more links and pages. We implemented the four different approaches suggested by the authors: injecting the URLs using (I) robots.txt, (II) sitemap.xml, (III) a landing page, and (IV) an index page (Esposito et al., 2018). The difference between their approach and our approach is the extraction of URLs used for the injection (*URL index*). Esposito et al. created the URL index for their evaluation using endpoint extraction of URLs. Since our work is placed in an automated black-box setting, this is not realistic. We performed the generation of the URL index automatically using HelpMeICS. We first automatically run all

[5]https://selenium.dev/

WAS against a target and recorded all requests and responses made. Afterwards, we extracted all requests to which the DUT responded with a non-error code which then built the URL index. For an evaluation, please refer to Sections 5.2 and 5.3.

Logging. To be able to narrow down a vulnerability, it is necessary to know which requests and responses have been sent between the WAS and the DUT. The logging add-on addresses this by logging each response and request seen by the proxy.

5 EVALUATION

To evaluate HelpMeICS, we conducted a qualitative evaluation as well as a quantitative evaluation. This section describes our evaluation setting (Section 5.1), presents the results of the quantitative evaluation (Section 5.2), and shows the results of the qualitative evaluation (Section 5.3). Our qualitative evaluation shows that HelpMeICS reduces the following limitations of WAS: *Crawling*, *Periodically Changing Content*, *User Login*, *Reproducibility*, *Pause and Resume a Scan*, *Conduction of Single Tests*, *Load Reduction*, *Behavior in Case of an Error*, and *Different Scanners Necessary*. The quantitative evaluation shows an increase of true positive reports as well as a higher URL coverage. For one of the used DUTs, a security test was only possible using HelpMeICS because it crashes irrecoverably during web application scans without HelpMeICS.

5.1 Evaluation Setting

For our evaluation we selected six WAS to be run against five real ICS. The aim of the evaluation is to evaluate HelpMeICS and not to evaluate the ICS used. That is why we decided not to reveal the manufacturers and versions of the ICS. Instead, we describe the device class as well as the properties of the ICS which are relevant for the evaluation. In the following, we describe the ICS, WAS, and configurations of HelpMeICS used for the evaluation.

Table 2: WAS used for the evaluation.

Scanner	Version
Arachni	1.5.1
Nikto	2.1
Skipfish	2.10
Subgraph Vega	1.0
Wapiti	commit[1] 0bf7g7
ZAP	2.8.0

[1] https://git.code.sf.net/p/wapiti/git

Web Application Scanners. In order to evaluate the impact of HelpMeICS, we used HelpMeICS with six WAS against five ICS. In Table 2 the WAS we used are shown. We selected six open source WAS which have already been used for many evaluations by literature. To be able to integrate the GUI-based WAS Vega, we created a customized version which allows controlling the conducted tests via a python API. We made this customized version of Vega publicly available[6]. Arachni differs from the other WAS since it includes a browser engine. With this, it is able to investigate web applications which include technologies like JavaScript, HTML5, DOM manipulation and AJAX. Nikto also shows a special behavior. For its tests, Nikto uses a fixed database of URLs and checks. With this, Nikto is the only one of the considered WAS which conducts highly reproducible tests.

Devices Under Test. We selected five ICS with different functions and properties as realistic targets. The first DUT, DUT1, is a PROFINET bus coupler. Bus couplers translate communication between different bus systems and architectures. For example, bus couplers can serve as a provider of additional input and output interfaces for a PLC. DUT1 provides a web application that uses HTTP-Basic-Authentication. One characteristic of this web application is that it includes a vulnerability which can be used to crash DUT1. The DUT needs to be restarted by switching the power off and on to be functional again. This poses a problem for WAS, since each WAS detecting this vulnerability will crash DUT1. The WAS are not able to detect and recover from this issue since they are not able to restart DUT1.

The second ICS, DUT2, is a gateway for OPC UA, a machine-to-machine communication protocol for industrial automation. DUT2 connects different communication protocols to OPC UA in order to combine the data from different sources. Its web application uses a form based user login. The identifiers of the input fields used for login are changed on a regular basis. WAS which are able to perform form based authentication are not able to login to the web application without help since they assume fixed identifiers.

DUT3 is a firewall which connects two networks and restricts the traffic in between. In contrast to the other DUTs, the web application of DUT3 only allows HTTPS. To login to the web application, the user needs to provide only a password but no user name.

[6]https://github.com/subgraph/Vega/pull/184

Since the WAS which are able to perform form based authentication expect a password field as well as a field for the user name, they are not able to login.

Similar to DUT1, DUT4 is a PROFINET bus coupler. The web application of DUT4 shows the special feature that it first delivers a JavaScript program to a user. Afterwards, this program loads the actual user interface. This makes it more complicated for a proxy to interpret and manipulate the packets.

DUT5 is a thermometer which distributes its measurements over different communication protocols such as MQTT, SNMP, FTP, and Mail. In addition, the measurements are presented on a web application as a dynamically updated graph as well as a XML-Feed. This data is of that kind of *Periodically Changing Content* that can mislead WAS (see Section 3.1).

Configurations. In order to be able to compare the performance of the add-ons (see Section 4), we defined different configurations of our system for the evaluation. An overview of these configurations is given in Table 3. The configuration *bare* names the execution of the WAS without HelpMeICS. The configuration using our system without the proxy is called *virtualised*. That means that the WAS are run in a container and are configured by ISuTest. All other configurations refer to the usage of all components of the system with different add-ons used by the proxy.

5.2 Qualitative Evaluation

HelpMeICS reduces the limitations of WAS by three means: proxy add-ons, containers, and ISuTest. In the following, we present our solutions for improved limitations, including potential restrictions.

Table 3: Configurations used for the evaluation. *bare* corresponds to an execution without HelpMeICS, *virtualised* to WAS running in a container, and all other configurations to an execution using a different subset of proxy add-ons.

configuration	container	proxy	logging	watchdog	user login	URL injection
bare	-	-	-	-	-	-
virtualised	✓	-	-	-	-	-
proxy	✓	✓	✓	-	-	-
proxy_W	✓	✓	✓	✓	-	-
proxy_WL	✓	✓	✓	✓	✓	-
proxy_WI	✓	✓	✓	✓	-	✓
proxy_WLI	✓	✓	✓	✓	✓	✓

Crawling. The crawling capabilities by a WAS are improved by the URL injection add-on (see also Section 5.3). Recall that the used URL index is crafted automatically based on the requests from all WAS. In a special case, this automatic approach increases the number of indexed URLs unnecessarily. Given a WAS that randomly changes a parameter in the URL and a web application that does not care about this parameter but delivers always a positive response, it follows that each of these URLs are indexed. An automatic reduction of URLs is difficult since other web applications might react to changing parameters. Nevertheless, the injection of crawled URLs increases the coverage tremendously (see Section 5.3).

Periodically Changing Content. The mapping add-on allows for replacing or removing periodically changing contents that might mislead WAS. A user needs to specify the content to be replaced using a CSS selector. The positive impact of this add-on has been confirmed by the evaluation of ZAP against DUT5. As has already been stated, the web application of DUT5 provides a XML-feed of the measured temperatures. Since this feed is updated once a minute, ZAP is convinced that its tests for SQL injection have been successful. Using the mapping add-on, the feed can be set to a fixed value. With this, ZAP no longer reports the false positive SQL injection.

User Login. On some DUTs, logged-in users can access more websites than unauthenticated users. Even though most WAS provide different authentication mechanisms, they still struggle to authenticate against some of the ICS as has been described in Section 5.1. In order to support the WAS during their scan, login add-ons have been implemented and tested. Using the selenium add-on, the WAS are now able to scan web applications only asking for a password and no user name (DUT5) as well as web applications with changing session IDs (DUT3). The user needs to record the login process only once using a selenium recorder and is able to reuse it for all WAS afterwards. Nevertheless, the add-on fails on DUTs whose websites contain logic to answer a specific call regularly in the background (DUT2 and DUT4). In order to overcome this issue, one would have to implement a DUT-specific solution.

Reproducibility. The reproducibility of tests is reached by running the WAS in containers and taking snapshots on a regular basis. However, some WAS generate their random data used for fuzzing on the fly. In this case, two test runs from the same snapshot can produce two different testing processes. This issue can only be solved by the specific WAS.

Pause and Resume a Scan. Especially when using WAS against ICS, pausing a scan might be necessary, for example to restart the DUT. The use of containers with snapshots allows for stopping the scan and resuming the scan afterwards.

Conduction of Single Tests. Using only a subset of security tests is partly supported by the use of containers. With the granularity of snapshots, only some tests can be conducted. This improves WAS which do not support the feature to run single tests.

Load Reduction. High loads of network traffic can overstrain some DUTs. The container solution allows for stopping and resuming a security test by stopping and resuming the container. For example, this helps in the case of limited TCP connections. If the proxy detects too many concurrent TCP requests, it can pause the WAS. Due to this pause, the DUT is able to handle the remaining TCP requests and is able to free enough sources to handle new requests by the WAS.

Behavior in Case of an Error. In case of a crash of the DUT which requires a restart of the device, WAS get stuck. This means that they discovered one vulnerability but cannot test for more. HelpMeICS employs the combination of ISuTest and the containers which improves this situation enormously. At first, the WAS gets stopped for the regular monitoring. If ISuTest detects that the DUT does not respond anymore, it executes a restart by switching the power off and on. Then, the container of the WAS can be resumed and thus the test can be continued. In the concrete example of DUT1, testing with HelpMeICS has two advantages: The first one is that DUT1 will be restarted after the crash and the test can be continued until it finishes. With this, only possibly a few tests in the time period between the crash and the monitoring are lost. The second advantage is that HelpMeICS logs the information about the packages sent during the test as well as generating a report about the crash. With this information, a security tester is able to comprehend the vulnerability reported.

Different Scanners Necessary. The need to conduct security tests with many different WAS is addressed by ISuTest. Once integrated in HelpMeICS, many scans with different WAS can be scheduled one after another.

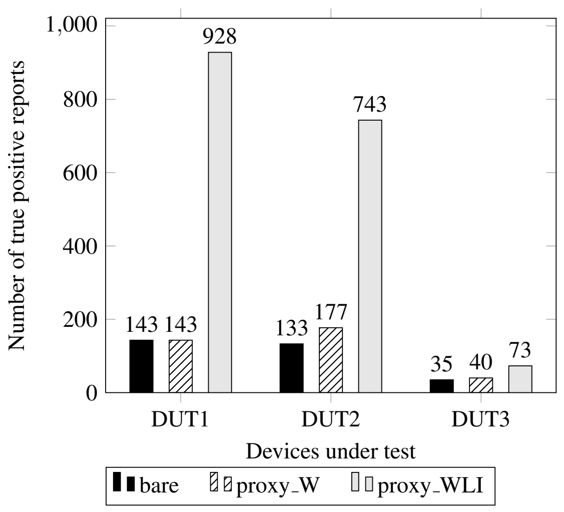

Figure 3: Number of true positive reports summarized over the WAS for the different configurations, excluding false positives but including duplicates. Using more add-ons of the proxy leads to more true positive reports.

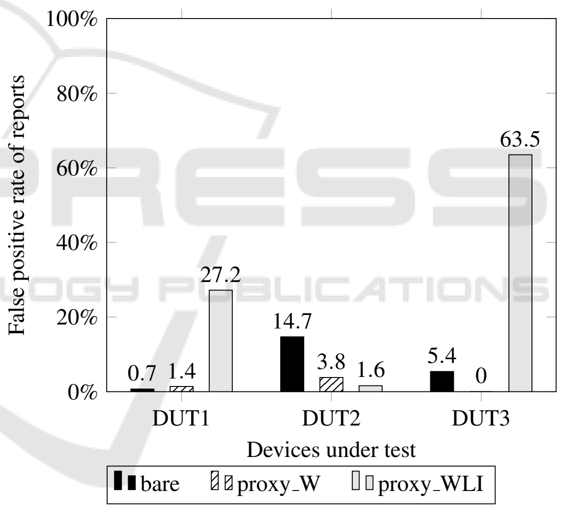

Figure 4: False positive rate of reports summarized over the WAS. Depending on the DUT, using more add-ons of the proxy decreases or increased the rate.

5.3 Quantitative Evaluation

The quantitative evaluation includes an analysis of the true positive reports given by the WAS, the URL coverage, the performance of the watchdog, as well as the impact on the runtime of the tests. For the evaluation, we executed each of the presented WAS against each ICS using each of the configurations (see Section 5.1).

Reported Vulnerabilities. The first evaluation analyses the number of true positive reports given by the WAS. Note that this number excludes false positives but includes duplicates. Since our work is

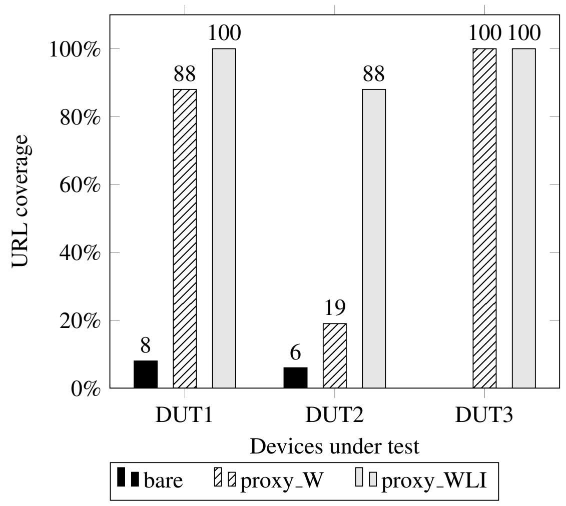

Figure 5: Percentage of existing URLs on a DUT that have been visited by at least one WAS. Using more add-ons of the proxy leads to a higher percentage of vistited URLs.

situated in a black-box setting, it is not possible to identify the number of concrete vulnerabilities the reports correspond to. This would be necessary in order to eliminate duplicates. In Figure 3, the number of true positive reports given by all WAS together is depicted for each DUT. The configurations shown are *bare* (execution of the WAS without HelpMeICS), *proxy_W* (usage of HelpMeICS including the logging and watchdog add-on), and *proxy_WLI* (additional usage of the login and the URL injection add-on). For DUT2 and DUT3, the number of reports rises steadily with the number of add-ons. For DUT1, the number of reports stays the same for the configurations bare and proxy_W but rises as well for proxy_WLI. Even tough the number of true positive reports does not directly correspond to the number of vulnerabilities, it gives an evidence for the effectiveness of HelpMeICS.

Complementary, the false positive rate of the reports is analysed. As shown by Figure 4, the false positive rate of the given reports is decreased by the usage of HelpMeICS in the configuration proxy_W for DUT2 and DUT3. In contrast, using the configuration proxy_WLI, the false positive rate increases for DUT1 and DUT3. Regarding DUT2, the false positive rate decreases the more add-ons are used.

The analysis of the number of reports shows that more true positive reports are given by the WAS the more add-ons of the proxy are used. In some cases, the trade-off for more true positives is an increased false positive rate. Our findings coincide with the results of Esposito et al. (Esposito et al., 2018). The authors state as well that adding URL injection and user login leads to an increased number of reports.

URL Coverage. One indicator of an improved security test is the number of called URLs. If a WAS does not call a URL, it is surely not able to find vulnerabilities on this URL. It follows that visiting more URLs generally means a higher probability to find more vulnerabilities. In the following, the percentage of URLs of a DUT visited by at least one WAS will be analysed. Since we are conducting black box security tests, we are not able to monitor the internal coverage of the web application of the DUT. This is why we compare the URLs existing on a DUT with the URLs visited by the WAS. We define the URLs existing on a DUT as the URL index generated by the logging add-on of the proxy (see Section 4). For the configuration proxy_WLI it follows that we check which percentage of injected URLs are actually tested.
The results of this evaluation are presented in Figure 5. Note that for DUT3 it shows no value for the bare configuration since DUT3 uses HTTPS. This results in encrypted traffic that can not be analysed without using the proxy. Since our proxy supports HTTPS, we are able to analyse the packets sent when using the proxy. For DUT1 and DUT2, the percentage of URLs visited increases with the number of add-ons used. For DUT3, the percentage stays constant at 100%. The results of DUT2 for proxy_WLI show the challenge of automatically creating URL indices using non-reproducible WAS (see Section 4.2). The URL index of DUT2 contains some URLs the WAS visited during the generation of the URL index but not during the evaluation. Nevertheless, the analysis of the percentage of URLs visited by the WAS shows that it is increased with the number of add-ons used.

Runtime. Since HelpMeICS adds functionality and computations to the WAS, its impact on the runtime needs to be analysed. For this analysis we chose Nikto as WAS and DUT2 as target. As has already been stated in Section 5.1, Nikto creates its tests from a static database of URLs and checks. With this, the runtime of Nikto is relatively stable. This allows us to analyse the impact of HelpMeICS. To make the results even more reliable, we executed Nikto ten times with each configuration of HelpMeICS. The medium value of these ten runs as well as the minimum and maximum runtime is shown in Figure 6.

A first insight from this analysis is that the virtualisation has no great impact on the runtime of Nikto. A second insight is that the proxy increases the runtime of Nikto by the factor of 9. A third insight is that the addition of add-ons has again no great impact.

However, for some cases using other WAS and DUTs, HelpMeICS was able to reduce the runtime. For example, this has been the case for Arachni

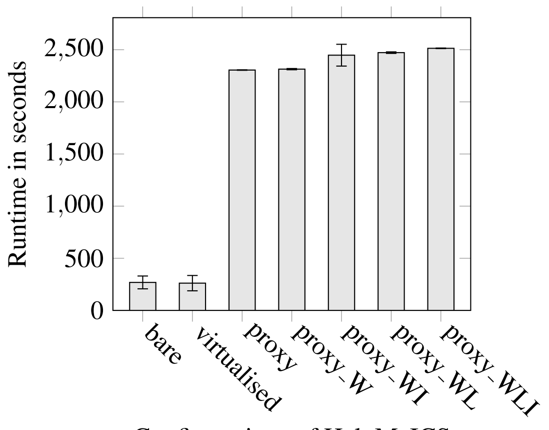

Figure 6: Runtime of Nikto against DUT2 using different configurations of HelpMeICS. Presented are medium values of ten runs as well as minimum and maximum. The proxy itself has a high impact on the runtime, the additional add-ons of the proxy do have a smaller impact.

against DUT4. Arachni opens many TCP connections and such reaches the maximum number of TCP connections accepted by DUT4 after roughly 31 seconds. During the remainder of the test, Arachni tries to establish new connections to DUT4 unavailingly until the maximum time of the test is reached. If HelpMeICS is used, the watchdog detects this problem and alerts ISuTest. ISuTest then restarts DUT4. As a result, DUT4 is able to accept new TCP connections again and Arachni can finish its tests successfully. In this case, the runtime has been reduced by a factor of 68 comparing the bare configuration to proxy_W. In summary, the analysis of the runtime shows an increase of runtime in most cases but also a decrease of runtime in some cases. The proxy itself has the most impact on the runtime, which means the runtime might be improved by improving the used proxy.

Watchdog Performance. Depending on the behavior of the DUT, the watchdog can have a high impact on the runtime of a test. If the watchdog produces many false positive alerts, the runtime increases unnecessarily since ISuTest will conduct a monitoring cycle after every alert. In the case of a true positive alert from the watchdog, the runtime is increased even more since the DUT is restarted. Note that this increase of runtime is necessary to improve the quality of the test, but the increase of runtime caused by false positives is unnecessary. Because of this, the false positive rate of the watchdog is analysed.

Our analysis shows high variance in the false positive rate of the watchdog depending on the WAS and the DUT. For example, the combination of Nikto and DUT1 shows a false positive rate of 8%. In contrast, the combination of Skipfish and DUT4 results in a false positive rate of 96%. Even though the watchdog shows a good performance in some cases, it still might be improved further. Note that we are currently using the error function given by mitmproxy as trigger for alerts. One cause of false positive alerts are malformed HTTP packets sent by the WAS. These could be excluded in a post processing step of the error trigger given by mitmproxy.

In summary, the watchdog possibly increases the runtime unnecessarily but helps to improve the tests by detecting crashes of the DUT. For DUT1, a full test without the watchdog is not even possible since DUT1 crashes before the tests might be completed.

6 CONCLUSIONS

We presented HelpMeICS, a *Helper-in-the-Middle for ICS*, which improves different limitations of existing WAS. HelpMeICS consists of a proxy, which is placed between the WAS and the DUT, containers the WAS run in, and ISuTest as an industrial security testing framework that orchestrates all components.

Particularly, the applicability and the methodology have been enhanced notably by HelpMeICS. Advanced crawling features, authentication and the possibility to test with many different WAS improve the methodology. The usage of ISuTest and containers enhance the applicability of WAS. It enhances the reproducibility of tests and the scanner behavior in case of DUT crashes. As significant result, more true positive reports have been generated by the WAS and even one severe vulnerability which crashed one DUT completely has been found.

The evaluation showed that not every limitation of the WAS could be solved by HelpMeICS. Script-based authentication systems turned out to be very DUT-specific. With this, it was not possible to create an universal solution for every web application. The reproducibility of security tests posed another problem after the introduction of snapshots: Some WAS produced different test sequences even when being started from the same snapshot. This problem cannot be solved by HelpMeICS, it needs to be solved by the particular WAS. Nevertheless, for most of the WAS the snapshot feature proved to be very valuable. One last limitation consists of a trade-off: The additional features and improvements of the WAS increase the runtime of the tests in most cases.

Concluding, HelpMeICS allows for conducting automated web security tests using different existing WAS. It improves the performance of WAS in contrast to their single use. Discovered vulnerabilities help manufacturers and integrators of ICS to reduce the attack vectors of their web applications and improves their security in general.

Future work comprises the improvement of URL indices in order to reduce false positives. The authentication add-on for selenium needs some work to extract session data from the used browser engine to pass them to the WAS. More WAS can be evaluated and integrated in HelpMeICS to improve its performance. Finally, more DUTs can be tested to detect and analyze even more vulnerabilities.

ACKNOWLEDGEMENTS

This work was supported by the German Federal Ministry of Education and Research within the framework of the project KASTEL SKI in the Competence Center for Applied Security Technology (KASTEL) and by the Ministry of Economy, Labor and Housing, Baden-Württemberg, within the research project CyberProtect.

REFERENCES

Alassmi, S., Zavarsky, P., Lindskog, D., Ruhl, R., Alasiri, A., and Alzaidi, M. (2012). An analysis of the effectiveness of black-box web application scanners in detection of stored xssi vulnerabilities. *International Journal of Information Technology and Computer Science*, 4(1).

Bau, J., Bursztein, E., Gupta, D., and Mitchell, J. (2010). State of the art: Automated black-box web application vulnerability testing. In *2010 IEEE Symposium on Security and Privacy*, pages 332–345. IEEE.

BSI (2019). Industrial control system security. Technical report, German Federal Office for Information Security, Publications on Cyber-Security.

Deepa, G., Thilagam, P. S., Praseed, A., and Pais, A. R. (2018). Detlogic: A black-box approach for detecting logic vulnerabilities in web applications. *Journal of Network and Computer Applications*, 109:89–109.

Doupé, A., Cavedon, L., Kruegel, C., and Vigna, G. (2012). Enemy of the state: A state-aware black-box web vulnerability scanner. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 523–538.

Doupé, A., Cova, M., and Vigna, G. (2010). Why johnny can't pentest: An analysis of black-box web vulnerability scanners. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 111–131. Springer.

Esposito, D., Rennhard, M., Ruf, L., and Wagner, A. (2018). Exploiting the potential of web application vulnerability scanning. In *ICIMP 2018, Spain, July 22-26, 2018*, pages 22–29. IARIA.

Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R., and Pretschner, A. (2016). Security testing: A survey. In *Advances in Computers*, volume 101, pages 1–51. Elsevier.

Ferreira, A. M. and Kleppe, H. (2011). Effectiveness of automated application penetration testing tools.

Fonseca, J., Vieira, M., and Madeira, H. (2007). Testing and comparing web vulnerability scanning tools for sql injection and xss attacks. In *13th Pacific Rim international symposium on dependable computing (PRDC 2007)*, pages 365–372. IEEE.

Galloway, B. and Hancke, G. P. (2012). Introduction to industrial control networks. *IEEE Communications surveys & tutorials*, 15(2):860–880.

Idrissi, S., Berbiche, N., Guerouate, F., and Shibi, M. (2017). Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *International Journal of Applied Engineering Research*, 12(21):11068–11076.

IEC62443 (2019). IEC-62443: Security for industrial automation and control systems. Standard, International Electrotechnical Commission.

Khoury, N., Zavarsky, P., Lindskog, D., and Ruhl, R. (2011). An analysis of black-box web application security scanners against stored sql injection. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 1095–1101. IEEE.

Langner, R. (2011). Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51.

Makino, Y. and Klyuev, V. (2015). Evaluation of web vulnerability scanners. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 399–402. IEEE.

McAllister, S., Kirda, E., and Kruegel, C. (2008). Leveraging user interactions for in-depth testing of web applications. In *International Workshop on Recent Advances in Intrusion Detection*, pages 191–210. Springer.

Pfrang, S., Borcherding, A., Meier, D., and Beyerer, J. (2019). Automated security testing for web applications on industrial automation and control systems. *at-Automatisierungstechnik*, 67(5):383–401.

Pfrang, S., Meier, D., and Kautz, V. (2017). Towards a modular security testing framework for industrial automation and control systems: Isutest. *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*.

Suteva, N., Zlatkovski, D., and Mileva, A. (2013). Evaluation and testing of several free/open source web vulnerability scanners.

Vega, E. A. A., Orozco, A. L. S., and Villalba, L. J. G. (2017). Benchmarking of pentesting tools. *International Journal of Computer and Information Engineering*, 11(5):602–605.