Cluster Crash: Learning from Recent Vulnerabilities in Communication Stacks

Anne Borcherding[1,3][a], Philipp Takacs[1] and Jürgen Beyerer[1,2,3]
[1]*Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB, Karlsruhe, Germany*
[2]*Vision and Fusion Laboratory (IES), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*
[3]*KASTEL Security Research Labs, Karlsruhe, Germany*

Keywords: Industrial Control Systems, Anti-Patterns, Vulnerability Testing, Ripple20, Amnesia:33, Urgent/11.

Abstract: To ensure functionality and security of network stacks in industrial device, thorough testing is necessary. This includes blackbox network fuzzing, where fields in network packets are filled with unexpected values to test the device's behavior in edge cases. Due to resource constraints, the tests need to be efficient and such the input values need to be chosen intelligently. Previous solutions use heuristics based on vague knowledge from previous projects to make these decisions. We aim to structure existing knowledge by defining Vulnerability Anti-Patterns for network communication stacks based on an analysis of the recent vulnerability groups Ripple20, Amnesia:33, and Urgent/11. For our evaluation, we implement fuzzing test scripts based on the Vulnerability Anti-Patterns and run them against 8 industrial device from 5 different device classes. We show (I) that similar vulnerabilities occur in implementations of the same protocol as well as in different protocols, (II) that similar vulnerabilities also spread over different device classes, and (III) that test scripts based on the Vulnerability Anti-Patterns help to identify these vulnerabilities.

1 INTRODUCTION

Industrial Control Systems (ICSs) are elementary to provide a fast, reliable and flexible production. The term ICS comprises different types of control systems, including the necessary industrial device, systems and networks. Due to high requirements regarding digitalisation and complexity, modern industrial device consist of a plethora of components. In order to provide reliable and secure services, thorough tests of the components need to be conducted. One important part of a thorough test of an industrial device is to conduct blackbox tests, which do not consider any internal details of the Device under Test (DUT), but test the DUT from the outside. In the domain of industrial device, blackbox tests are usually performed via the Ethernet interface. Amongst other, these tests can reveal vulnerabilities in the communication with other devices and especially vulnerabilities regarding the handling of malformed communication packets. Errors in communication stacks can lead to a crash of the affected device and such to a potential outage of the whole industrial facility (Pfrang et al., 2018).

Many of these vulnerabilities are caused by software errors that have been created during the design or development of the industrial device. One way to describe the design or development decisions that lead to these errors are Anti-Patterns or Vulnerability Anti-Patterns (VAPs) (Nafees et al., 2018). VAPs help to communicate design and coding practices that potentially lead to vulnerable code. Although VAPs are intended to prevent errors and vulnerabilities during design and development, we propose to use them to build targeted blackbox tests.

One possible technique for blackbox tests of industrial device is blackbox fuzzing. Even though classic fuzzing has been introduced in the 1980s, it is still relevant and helps to find vulnerabilities (Miller et al., 2020). For blackbox network fuzzing, one usually tests several expected or unexpected values for the different fields of a network packet. Due to time and resource restrictions, it is usually not possible to test all fields with all possible values. We propose to use the information derived from previous vulnerability groups represented as VAPs to guide the prioritization of fields and values.

Our aim is to understand which VAPs form the basis of recently published vulnerabilities in communi-

[a] https://orcid.org/0000-0002-8144-2382

Borcherding, A., Takacs, P. and Beyerer, J.
Cluster Crash: Learning from Recent Vulnerabilities in Communication Stacks.
DOI: 10.5220/0010806300003120
In *Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP 2022)*, pages 334-344
ISBN: 978-989-758-553-1; ISSN: 2184-4356

cation stacks of industrial device, and how this information can be used to test industrial device more efficiently. In order to achieve this, we analyze three vulnerability groups that have been published recently: *Ripple20* (Kohl and Oberman, 2020; Kohl et al., 2020), *Amnesia:33* (dos Santos et al., 2021), and *Urgent/11* (Seri et al., 2019). All of these vulnerabilities concern communication stacks for standard Internet protocols such as TCP, DNS, and IPv4, which are also used in industrial device. Based on our analysis, we develop VAPs for the communication stacks. Subsequently, we use these VAPs to develop blackbox fuzzing test scripts that aim to test a given blackbox industrial device for the implementation errors resulting from these VAPs.

For the evaluation of our test scripts, we use 8 industrial device from 5 device classes. We run the test scripts against the devices and observe their behavior. As a result, we observed three crashes and 9 anomalies in the behavior of the industrial device. Our evaluation shows that similar vulnerabilities occur in implementations of the same protocol as well as in different protocols. In addition, we show that similar vulnerabilities also spread over different device classes and that the test scripts we developed based on the VAPs help to identify these vulnerabilities.

In summary, our main contributions are:

- We analyse the vulnerabilities of Ripple20, Amnesia:33, and Urgent/11, and identify VAPs for the development of communication stacks.
- We implement fuzzing test scripts to search for implementations of the VAPs in a blackbox setting, and evaluate the test scripts using 8 industrial devices.

The rest of this paper is structured as follows. Section 2 presents related work in the domains of blackbox fuzzing for industrial device, vulnerability scanning, and Anti-Patterns. Our analysis of the recent vulnerability groups Ripple20, Urgent/11, and Amnesia:33 is described in Section 3. The test scripts we implemented based on these VAPs are described in Section 4. In Section 5, we evaluate the VAPs and test scripts. This section also describes our methodology, presents the results and discusses them. Section 6 concludes our work.

2 RELATED WORK

Our work is located in the domain of blackbox fuzzing for industrial device and also touches the domains of vulnerability scanning, and Anti-Patterns.

Blackbox Fuzzing for Industrial Devices. The framework for blackbox network fuzzing for industrial device *ISuTest* is presented by Pfrang et al. (Pfrang et al., 2017). The included fuzzer generates new test cases based on heuristics which are used to determine which values should be used for a packet field with a certain data type. As a basis for these heuristics, the authors use knowledge from previous projects. Our work chooses a more general approach by identifying VAPs and then deriving promising field types and values based on these VAPs.

Vulnerability Scanning. In contrast to fuzzing, vulnerability scanning generally is less intrusive. The aim of vulnerability scanning is to check whether the DUT contains vulnerable software by identifying the software running on the DUT. For the vulnerability groups considered in this work, vulnerability scanners have been developed. These scanners aim to detect whether the considered DUT includes one of the vulnerable communication stacks. For Ripple20, a scanner was developed by JSOF which can be received upon request. Forescout published their scanner for Amnesia:33 on GitHub[1]. Likewise, ArmisSecurity published a scanner for Urgent/11[2]. In general, all three scanners use active fingerprinting to determine whether the DUT includes one of the vulnerable stacks. Due to the high impact that one single small corrupted component can have on a wide range of domains, it is especially challenging to find all vulnerable communication stacks. In addition, fingerprinting for ICSs has some special challenges (Caselli et al., 2013). In contrast, our work aims to actively test for the implementations of VAPs. With this, it has potentially a higher impact on the availability of the DUT since it might crash during the testing. However, with the active tests, one can be sure that a DUT is indeed vulnerable if the tests report a crash.

Anti-Patterns. Anti-Patterns are usually used to describe common errors during the design or development of software (Tuma et al., 2019; Hecht et al., 2015), for management (Julisch, 2013), and for vulnerabilities in design or development (Nafees et al., 2018). Examples for such Anti-Patterns are the use of deprecated software or missing authentication. In addition, performance Anti-Patterns have been presented for communication (Wert et al., 2014; Trubiani et al., 2018), for cyber physical systems (Smith, 2020), and for simulated models (Arrieta et al., 2018).

[1] https://github.com/Forescout/project-memoria-detector

[2] https://github.com/armissecurity/urgent11-detector

Table 1: Stacks affected by the vulnerabilities published by Ripple20, Amnesia:33, and Urgent/11.

Stack	**Language**	**Availability**
Ripple20		
Treck TCP/IP	C	closed source
Amnesia:33		
lwIP	C	open source
uIP	C	open source
Nut/Net	C	open source
FNET	C	open source
picoTCP	C	open source
CycloneTCP	C	open source
uC/TCP-IP	C	open source
Urgent/11		
Interpeak IPNet	C	closed source

None of the Anti-Patterns in literature concern the security or vulnerabilities of network protocols, industrial device, or cyber physical systems. Nevertheless, the various applications of Anti-Patterns show that they are an accepted method to approach the prevention of errors. In our work, we aim to find implementations of errors that have not been prevented during design or development.

We base the representation and description of our Anti-Patterns on the VAPs Template presented by Nafees et al. (Nafees et al., 2018).

3 ANTI-PATTERN ANALYSIS

The aim of our analysis is to understand which similarities occur in the recently published vulnerability groups and to develop corresponding VAPs. The authors of Amnesia:33 state that their analysis shows that “implementing the same protocols under similar constraints tends to produce similar bugs in similar places” (dos Santos et al., 2021, page 22). With our analysis, we want to additionally analyze whether similar issues and VAPs also occur in implementations of different protocols.

In order to do so, we analyze the vulnerabilities published under the names of Ripple20, Urgent/11, and Amnesia:33. The nine stacks in which the vulnerabilities have been found are presented in Table 1. Amnesia:33 analyzed seven open source stacks whereas Ripple20 and Urgent/11 analyzed one closed source stack respectively. All in all, the three vulnerability groups consist of 63 vulnerabilities. One of the vulnerabilities (CVE-2020-11904) is not concerned with a specific protocol but is a protocol-independent issue of the Treck stack. This is why we exclude the vulnerability from our analysis. Another vulnerability (CVE-2020-13987) is concerned with TCP as well as with UDP which we consider as two different vulnerabilities for our analysis.

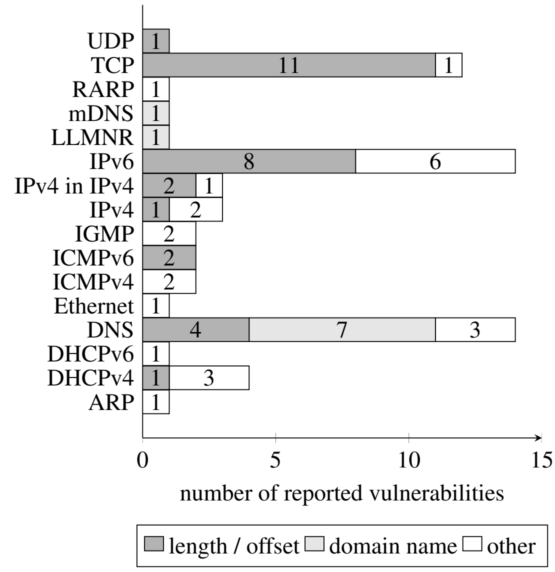

Figure 1: Number and type of vulnerabilities present in Ripple20, Urgent/11, and Amnesia:33, sorted by protocol.

First, we classify the vulnerabilities based on the packet field the vulnerability occurs in. Second, we cluster the fields and vulnerabilities of the different protocols in order to find similarities. Third, we choose the most common fields and vulnerabilities to develop corresponding VAPs and test scripts.

Our analysis shows that more than half of the vulnerabilities are caused by a length / offset field, or the domain name. For both fields, the checks and the parsing is complex and error prone (see Sections 3.1 and 3.2 for more details). Figure 1 shows how the vulnerabilities are spread over the different protocols. The vulnerabilities that have been reported in the vulnerability groups are represented by one bar for each protocol. Within one bar, the vulnerabilities are again classified by the field the vulnerability occurs in. The two most common fields (length / offset fields and domain name) are highlighted respectively. We categorize length and offset fields together, since the underlying handling and resulting errors are similar. Out of the 63 vulnerabilities, 29 (46.03%) are concerned with a length / offset field and 9 (14.29%) are concerned with the domain name. The next most occurring field, `id`, only occurred 2 times (3.17%).

Figure 1 shows that the vulnerabilities concerning a length / offset field spread over almost all protocols. This already points towards a spreading of similar issues over different protocols. In contrast, vulnerabilities concerning the domain name only occurred in DNS and LLMNR even though domain names are also used in DHCP, DHCPv6 and ICMPv6.

Since the length / offset and domain name are responsible for more than 50% of the reported vulnerabilities, we will use these field types for the formulation of VAPs. We will then use these Anti-Patterns to design and develop test scripts in order to find implementations of these VAPs.

3.1 Lengths and Offsets

Our analysis shows that 46.03% of the reported vulnerabilities are concerned with invalid or unexpected values in length / offset fields. From our point of view, there are two main reasons for the high number of these vulnerabilities. The first reason is that all the investigated TCP/IP stacks are implemented in C. In C, many robustness issues are caused by missing boundary checks (Fetzer and Xiao, 2002) which often are triggered by unexpected or misfit lengths and offsets. The second reason is that there are many lengths and offsets in the investigated protocols. For example, TCP includes six length or offset fields. Each of these six fields poses a parser the challenge to verify and parse them correctly.

In addition, protocol standards usually focus on the parsing of packets that are compliant to the standard and not all requirements for the lengths and offsets are defined explicitly in the standard (dos Santos et al., 2021). Necessary checks for validity are even harder because some lengths interact with each other and depend on the current data. This is why parsing the lengths and offsets is error prone, especially regarding possible edge cases.

As an example, we want to have a closer look to the `option length` field of TCP. A necessary insight is that a valid `option length` value cannot be higher than the length of the remaining data of the header. The `option` that is described by the `option length` is itself contained in the header and such a value longer than the remainder of the header would surely be invalid. In order to check whether the `option length` of a given packet is compliant with this requirement, several calculations need to be conducted. For the necessary comparison, one needs to calculate how much data is left in the header. This includes the steps to (I) multiply the `data offset` by four to receive the full length of the data in Bytes, and to (II) subtract the length of the already parsed data. Then, the validity check of the `option length` can be conducted by comparing the calculated possible length and the given `option length`. Additional necessary checks are for example that (I) the already parsed data is at least 20 bytes long, since this is the length of a header without options, and (II) that the `data offset` multiplied by four is not greater than the available data from the IP layer. This small example makes apparent that different checks are necessary that partly depend on different fields of the current packet.

3.2 Domain Names

Of the vulnerabilities reported by Ripple20, Amnesia:33, and Urgent/11, 14.29% are concerned with domain names. Technically, the issues with domain names are also concerned with length, offsets and termination. Nevertheless, we handle these issues separately. The first reason for this is that DNS is a widely used protocol and in many cases, DUTs will just parse any crafted DNS packet. Second, domain names combine many ways to use length and offsets such that it poses even more challenges to a parser.

The specification of DNS can be found in RFC 1034 and RFC 1035 (Mockapetris, 1987a; Mockapetris, 1987b). Domain names are usually represented as a list of labels separated by dots (e.g. `my.domain.xy`). To encode them, the length of each label is prepended to the labels respectively. The example domain would be encoded as follows:

chars	2	m	y	6	d	o	m	a	i	n	2	x	y	0
bytes	02	6D	79	06	64	6F	6D	61	69	6E	02	78	79	00

A valid domain name encoding is always terminated with a length byte of zero, which is the length of the root domain.

In order to reduce redundancy, DNS supports a feature called *compression*. This feature acts like a pointer and allows to reference a prior list of labels. With this, it is not necessary to encode the same list of labels twice.

A full DNS packet contains different sections, where each has its own lengths, offsets and other specialities, making parsing a DNS packet error prone. These sections include the header, the question section, and three resource records sections. The question section defines the question that is being asked by the specific request. Each resource record section includes a variable number of resource records. A resource record specifies the domain name, the type, the class, and the time to live for a resource. This complexity is reflected by the vulnerabilities found, which concern out-of-bounds read and write, remote code execution and denial of service amongst others.

3.3 Analysis Conclusions

Our analysis indicates that the vulnerabilities reported by Ripple20, Amnesia:33, and Urgent/11, show similarities over different implementations but also over different protocols. For example, CVE-2020-11912 allows an out-of-bounds read caused by improper input validation based on a missing sanity check regarding the TCP `option length` and the actual length of the TCP `options`. This vulnerability was reported as part of Ripple20. Similarly, CVE-2020-17441 also leads to an out-of-bounds read caused by improper input validation and is based on a missing sanity check regarding the IPv6 `payload length` and the actual length of the payload. It was reported as part of Amnesia:33. Both vulnerabilities are caused by a missing sanity check regarding the given length and the actual length of the data. They occur in different protocols and at different places.

The six VAPs resulting from our analysis are published on GitHub[3]. We use these VAPs and the detailed results of our analysis to construct test scripts which test for the VAPs and vulnerability types.

4 TEST SCRIPTS

Based on our analysis, we implement 15 fuzzing test scripts. As has been stated, we implement our test scripts from a blackbox point of view. That means on the one hand that we do not have any insight into the source code during the development and on the other hand that the test scripts interact with the DUT only via the Ethernet interface. In general, the aim of the test scripts is to provoke anomalies or crashes in the tested devices.

In order to make our test scripts usable for continuous testing, we designed the test scripts in a way that they can easily be integrated into the security testing framework ISuTest (Pfrang et al., 2017). To show the integratability, we developed proof-of-concept implementations of our test scripts for ISuTest. In the standalone implementations of the test scripts, a security analyst needs to analyze the behavior of the industrial device during the tests. With the integration of the test scripts into the security testing framework ISuTest, an automated monitoring of the industrial device would be possible.

Since the scripts can provoke crashes of industrial device via routable protocols, but we still want to support scientific reproducibility, the standalone implementations of the test scripts are provided upon request.

For the implementation of the test scripts, we consider DHCPv4, DNS, IPv4, TCP, and UDP. We chose these protocols based on the protocols that showed the most vulnerabilities during our analysis, and based on practical considerations. As a fist selection criterion, we select the protocols that show most vulnerabilities. This includes DNS and IPv6 (14 vulnerabilities each), TCP (12 vulnerabilities), DHCPv4 (4 vulnerabilities), IPv4 and IPv4 in IPv4 (3 vulnerabilities each), and ICMPv4, ICMPv6 and IGMP (2 vulnerabilities each). Since the UDP protocol has shown a vulnerability based on a length field and is widespread, we additionally consider it. From this set of protocols, we select the protocols that are present in current industrial device, since our research is located in this domain. This leads to a selection of DHCPv4, DNS, IPv4, TCP, ICMP, and UDP. Each of these protocols except ICMP includes testable length / offset fields and domain names. This is why our final set of protocols to be considered for our test scripts contains DHCPv4, DNS, IPv4, TCP, and UDP.

Table 2 shows the packet fields we consider for our test scripts based on our analysis of the affected fields in Ripple20, Urgent/11, and Amnesia:33. For DHCP, we include two additional test scripts which also concern lengths of packet fields. First, we implement a test script to test for string termination issues in the DHCP options payload. In C, a string is usually terminated by a byte with a value of zero. For our test, we include a string that includes the zero byte in its middle (e.g. `example\0string`). With this, we can check if the DUT handles this unexpected string termination well. Second, we implement a test script to test how the DUT reacts to a DHCP options payload with a length of zero. Again, we want to test if the DUT can handle this unexpected payload length. Another speciality is the use of DHCP with the domain search option (option 119) (Aboba and Cheshire, 2002). With this option set, one can configure the domain search list. Caused by this direct connection to DNS, it includes similar fields, and we can implement similar test scripts.

The test scripts can be categorized in three groups: *stateless*, *request response* and *stateful*. In the following subsections, we present details on our implementation, for which we use Scapy[4] as a basis. Since Scapy is designed to be a high level API for packet generation, some packet fields are not intended to be set programmatically. This is why we need to use workarounds at some places, which are also described in the following.

[3] https://github.com/anneborcherding/vulnerability-anti-patterns

[4] https://scapy.net

Table 2: Packet fields considered in the test scripts, and their types (length/offset (l/o) or domain name (dn)).

Protocol	Type	Field
IPv4	l/o	length
	l/o	internal header length (ihl)
	l/o	option length
TCP	l/o	data offset
	l/o	option length
	l/o	urgent pointer
UDP	l/o	length
DHCP	l/o	option length
	dn	search option
	l/o	option payload termination
	l/o	zero length option payload
DNS	dn	compression pointer
	dn	label length
	l/o	qdcount
	l/o	rdlength

4.1 Stateless Protocols

Stateless protocols do not contain a state and such are easier to test since there is no need to, for example, establish a connection before sending a packet. For our evaluation, this includes IPv4 and UDP. The basic approach of the stateless test scripts is to create a packet, send it to the DUT, and check the response.

It is important to add a meaningful payload to the packet. With this, the probability that the DUT drops the packet at an early stage is decreased. For IPv4, we are using ICMPv4 as payload. Regarding UDP, the used payload depends on the DUTs. Some industrial device accept LLDP or DNS resolver payloads, or it might be the case that the DHCP implementation accepts packets. If there is no service on the DUT that accepts UDP-based packets, a request-response protocol needs to be used on top of the UDP packet.

Most of the test scripts regarding IPv4 and UDP can be implemented with Scapy in a straight forward way. We use Scapy to generate a packet, change the corresponding field and send the packet to the DUT. Since most of the fields only contain a few bytes, we are able to test the full input space for these fields.

However, for the implementation of test scripts regarding the IP options, some additional work needs to be done since Scapy's interface doesn't allow to set the length of an option. In general, this is a helpful interface, but for our test scripts, we intentionally want to set the length in a wrong way. As a workaround, we generate the options manually and append them as a payload to the IP layer. In addition, the IP header length and the checksum must be set properly.

4.2 Request-response Protocols

From the protocols considered for the test scripts, DHCP and DNS can be categorized as request-response protocols. That means that no complex state machine is required to test these protocols. Nevertheless, implementing the test scripts for DHCP and DNS is more challenging than the test scripts for stateless protocols. Since most industrial device implement the client of the protocol, the test script needs to represent the server. An additional challenge is that the first paket of the communication has to be sent by the DUT, and the communication cannot be initiated by the test script. For each field and each value that should be tested, the DUT needs to send one request. The answer to this request is then crafted according to the test script and sent to the DUT.

For the implementation, we encapsulate the mutation of the packet defined by the test script from the implementation of the protocol server. As a basis for the implementation of the protocol server, we use Scapy's `AnsweringMachine`. This part implements the general protocol behavior and is the same for each test script. The other part, the mutation of the packet, depends on the test script and interacts with the server implementation via an interface. For DHCP, we implement an interface to control the DHCP options which are handled by Scapy similar to the IP options, and such are not fully controllable via the Scapy interface. Our interface allows to set the options and the option lengths independent of the actual content. For DNS, we implement an interface to build the DNS packets. With the interface, we are able to control each part of the DNS packet. For example, it is possible to change the length of a label or to set a compression pointer to an arbitrary value. With this design and implementation, we can build the test scripts for the different fields (see Table 2).

Depending on the protocol, there are several ways to trigger the DUT to generate a request. Regarding DHCP, one can choose between two reasonable ways. On the one hand, the DUT usually will generate a DHCP request after booting. On the other hand, a DUT will send a new DHCP request after a timeout which can be set by the DHCP server (Droms, 1993). For our test scripts, we choose the option based on the timeout since a reboot of a industrial device can take even longer than the time needed to wait for the timeout. We set the `renewal timeout` to 10 seconds, and the `lease time` to 20 seconds. As a result, the DUT will send a new request after a time between 10 or 20 seconds, depending on the implementation. This still is a long time and such the tests need some time to finish (see Section 5.3.2 for an evaluation).

Regarding DNS, the possibilities to trigger a request highly depend on the concrete industrial device, its device type and implemented features. For example, one of the DUTs used in our evaluation provides a dDNS implementation which can be used to trigger a DNS request on a configurable frequency. A different DUT can be configured to send mails based on different events. This will trigger a DNS request as well. Note that these configurations do not break the blackbox assumption since they can be conducted by using the Ethernet interface. In general, the DNS test scripts require a individual configuration of the DUT for it so send DNS request as often as possible.

4.3 Stateful Protocols

We also consider TCP, which is a highly stateful protocol. That means that out implementation needs to generate, keep and cleanup the current state. Fuzzing stateful protocols has been identified as a challenge by Böhme et al. (Böhme et al., 2021). In order to reach a vulnerability in a deeper state of the protocol, one first needs to execute the steps that are necessary to reach said state. One specific challenge is to maintain the state during the tests since the tests may interact with the state. Each test script needs to establish a specific state, run the tests and reset the state afterwards. Some test scripts additionally need to interact with the general state machine. Depending on the payload of the TCP packet, the test script also needs to be aware of the payload's protocol. For example, it might be necessary to make a valid HTTP request on top of the tested TCP packet. Only with this valid payload the packet will be accepted und further processed by the DUT. Similar to the implementations of the stateless and request-response protocols, we implement an interface to set the options, and we need to recalculate the checksum.

5 EVALUATION

The aim of our work is to understand whether similar vulnerabilities occur in implementations of different protocols, and whether blackbox test scripts can be used to identify these vulnerabilities. Additionally, this can be used to guide blackbox fuzzers, making them more efficient. For the evaluation of these questions and of the implemented test scripts, we choose 8 industrial device from 5 device classes as a basis. We set up and configure the industrial device, run our scripts and analyse the resulting behavior of the industrial device. The following section describes our methodology, presents the results and discusses them.

Since our aim is to generally explore the spreading of vulnerabilities and not to explicitly point out single vendors, the used industrial device are identified by pseudonyms. The vulnerabilities we find during our work are disclosed responsibly and in collaboration with the vendors.

5.1 Methodology

Before the presentation of the results, we present our methodology. This includes the formulation of our hypotheses and the presentation of our evaluation strategy.

5.1.1 Hypotheses

Our evaluation is driven by the following hypotheses:

H.1 Similar vulnerabilities do not only occur in implementations of the same protocol but also in different protocols.

H.2 Black-box test scripts derived from previously defined VAPs help to identify vulnerabilities.

H.3 Implementations of the VAPs are present in different device classes.

5.1.2 Strategy

Depending on the protocols each industrial device supports, we use our implemented test scripts to test the devices for vulnerabilities. During the whole process, the industrial device are blackboxes. This represents the view of an attacker who aims to find vulnerabilities in industrial device, but also the view of a test late in the development life cycle. With this, we are able to evaluate realistically how our approaches and test scripts work in a blackbox setting. The devices we choose are not known to be vulnerable with regard to the analysed vulnerability groups Ripple20, Urgent/11, and Amnesia:33. This supports our aim to evaluate whether the identified VAPs can be transferred to other stacks and devices.

5.2 Devices under Test

We include 8 industrial device from 5 different device classes in our evaluation. The two leftmost columns of Table 3 list the DUTs as well as their device classes. It can be seen that the industrial device cover a wide range of device classes. A fingerprinting showed that each of the DUTs uses a unique stack. This increases the variance, and the coverage of the DUTs even more. In the following, we give a short overview on the functions and specialities of the device classes considered in this evaluation (firewall, controller, gateway, I/O device, and sensor). In order to follow a responsible disclosure strategy, we will not include details on the manufacturers of the devices. We reported the vulnerabilities to the concerned manufacturers, and we will publish the vulnerabilities after the disclosure process is finished.

Firewall. A firewall connects two or more network segments and restricts the traffic between the network segments. *FW1* is a firewall that provides an HTTPS Webserver for configuration purposes.

Controller. Controllers have the task to control and monitor the ICS. Additionally, they can collect and analyse data from the industrial device. For our evaluation, we choose two controllers: *Ctl1* and *Ctl2*. *Ctl1* is a safety controller which means that it is especially suited for applications with high safety requirements. This includes higher requirements regarding redundancy and reliability.

Gateway. The two gateways, *GW1* and *GW2*, are OPC UA Gateways. That means that they connect different communication protocols to OPC UA, which is a machine-to-machine communication protocol for industrial automation. With this, data from different sources can be combined and analyzed.

I/O Device. In general, industrial I/O devices connect analog and digital actuators and sensors to the controllers. *I/O2* is an I/O device that couples PROFINET communication from the controllers to digital or analog signals. *I/O1* translates the communication between PROFINET and I/O Link, and digital and analog signals.

Sensor. *Sens1* is a temperature sensor that provides its measurements using different communication protocols such as FTP, SNMP, and MQTT. In addition, *Sens1* provides a web application which also shows the current measurements but can additionally be used for configuration purposes.

5.3 Results

We present the results of our evaluation in this section, whereas Section 5.4 discusses the presented results. The test scripts lead to crashes of the DUTs as well as to anomalies in their behavior. We chose the term *finding* as an umbrella term for crashes and anomalies.

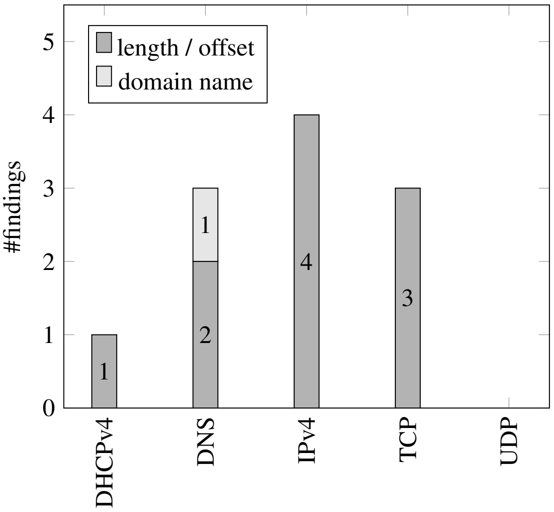

Figure 2: Findings (crashes and anomalies) identified by the test scripts, sorted by the protocol they concern.

5.3.1 Findings

During our evaluation, we observed 11 findings. Figure 2 presents the findings sorted by the affected protocol, and the type of the finding (length/offset or domain name). The test scripts provoked findings in each of the considered protocols except UDP. Most findings are concerned with IPv4.

The more detailed overview of the results presented in Table 3 shows that we observed anomalies in IPv4 and TCP, and crashes in DHCPv4 and DNS. In the table, anomalies are represented by **A** and crashes by **C**. Checkmarks (✓) represent runs that did not lead to a finding, and dashes (-) represent combinations which were not run since the DUT did not support the protocol. Three DUTs did not show any finding, for the remaining five DUTs, findings can be reported.

Two crashes we observed regarding DNS (see Table 3) are resulting from an `rdlength` value bigger than the available data. In our cases, the value of `rdlength` was expected to be 4. For *Sens1* as well as for *GW2*, an unexpected value for `rdlength` leads to the behavior that the DUT no longer sends DNS requests. We conclude that the DNS resolver crashes because of this unexpected value. Interestingly, the concrete value differs. The resolver of *Sens1* crashes if `rdlength` is set to `0x084a`, and the resolver of *GW2* crashes for the value `0xfd8c`.

The third crash regarding DNS is concerned with the compression pointer. *GW2* stops sending DNS requests after two responses with compression pointers (`0x05` and `0x06`). However, the DUT is still running, and with the webinterface a DNS request can still be triggered. Based on our observations, we assume that the NTPd process, which generates regular DNS re-

Table 3: Crashes (**C**) and anomalies (**A**) of the DUTs provoked by the test scripts. Checkmarks (✓) represent runs that did not provoke an anomaly or a crash, and dashes (-) represent combinations where the DUT does not support the corresponding protocol.

ID	**Device Class**	**IPv4**			**TCP**			**UDP**	**DHCPv4**				**DNS**			
		len	ihl	optlen	len	optlen	urgent	len	optlen	search	term	zero	compr_ptr	labellen	qdcount	rdlen
Sens1	Sensor	✓	✓	**A**	✓	✓	✓	✓	**C**	-	✓	✓	-	-	-	**C**
FW1	Firewall	✓	✓	**A**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Ctl1	Controller	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-
Ctl2	Controller	✓	✓	**A**	✓	✓	**A**	✓	✓	✓	✓	✓	✓	✓	✓	✓
I/O1	I/O Device	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-
I/O2	I/O Device	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-
GW1	Gateway	✓	✓	✓	✓	✓	**A**	✓	✓	✓	✓	✓	-	-	-	-
GW2	Gateway	✓	✓	**A**	✓	✓	**A**	✓	✓	✓	✓	✓	**C**	-	✓	**C**

quests, has crashed. After a reboot, the NTPd works again as expected.

The test script concerning the `option length` of DHCPv4 provoked a full crash of *Sens1*. If the DHCP ACK packet does not contain the values expected by the DUT, the DHCP client as well as the other services crash. After a reboot, the DUT and all of its services are up and running again.

The test script regarding the `urgent pointer` leads to anomalies in three industrial device. If the `urgent pointer` is set to zero, the industrial device return different HTTP response codes. This is an unexpected behavior since the `urgent data` requires at least one byte (Seri et al., 2019). Expected would be the same response as for requests without the `urgent flag` set, closing the connection or ignoring the packet. With the `urgent pointer` pointing inside the available data, *GW2* and *GW1* response sometimes with a `400 Bad Requests` and sometimes with a `200 Ok`. However, *GW2* also sometimes responses with `401 Unauthorized` which is the expected behavior since *GW2* uses HTTP Authentication. From a blackbox perspective, we interpret this behavior as an anomalie in the HTTP implementation since the `urget pointer` influences the HTTP response code.

Regarding IP, the test script regarding the `option length` lead to similar anomalies in four industrial device. Even in cases in which the option length is too short or too long, the DUTs reflects the content of the options to the sender. This also includes cases in which the content of the options is not parseable. Our analyses suggest that this anomaly can not be exploited. Still, the finding is communicated to the vendors of the effected DUTs.

5.3.2 Duration

Our evaluation of the timing performance shows high differences between the protocols, as expected. The results of our evaluation are shown in Figure 3, where each bar represents the mean duration of a run of all the test scripts concerned with the corresponding protocol. With this, we want to give a sense of how long the tests for one protocol need. As can be seen in the figure, the test scripts for DNS need much longer than the other test scripts. The reason for this is that the DUT needs to send the first packet of the connection and this needs its time (see also Section 4.2).

5.4 Discussion

Our evaluation gives valuable results regarding the hypotheses formulated in Section 5.1.1. In the following, we discuss the implications of our results in

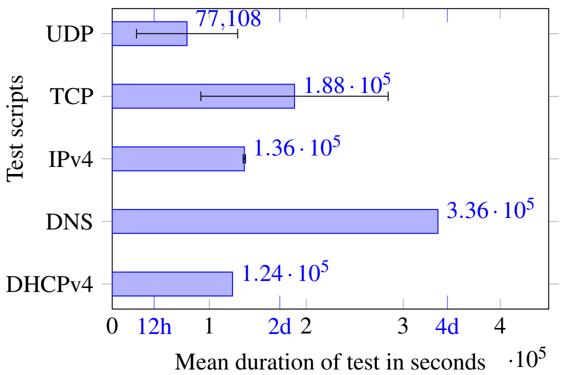

Figure 3: Mean duration of tests including 95%-quantile.

regard to the hypotheses and further discuss explanations for interesting observations as well as limitations.

First, our evaluation supports H1 since the results show that similar vulnerabilities do not only occur in the same protocol but also in different protocols. For example, this is the case for length values which are greater than the available data. On the one hand, in DNS we observed a crash caused by a `rdlength` value bigger then the available data. On the other hand, the anomalies found in IP are based on an `option length` value bigger then the available data. Additionally, our results also support the hypotheses that similar vulnerabilities can be found in different implementations of the same stack, which has been formulated by dos Santos el al. (dos Santos et al., 2021). This is for example shown by the similar behavior of *Sens1*, *FW1*, *Ctl2*, and *GW2* regarding the IPv4 option length.

Second, our implementation and evaluation support H2. The blackbox test scripts we developed based on the VAPs are indeed able to identify anomalies and crashes related to the VAPs. For example, this is shown by the findings regarding the `urgent pointer` which represent an implementation of *VAP1: Assume validity of length / offset field*. A detailed description of the VAPs has been published on GitHub.

Third, out evaluation supports H3 since the results include findings of the same VAPs in different device classes. For example, this is the case for the finding regarding the IPv4 `option length`. In this case, four devices from four different device classes (sensor, firewall, controller and gateway) show the same anomaly in their behavior. The controller and the gateway additionally share the anomaly in behavior regarding the TCP urgent pointer.

Our experiments show no findings regarding UDP. The reason for this could be that UDP is the least complex protocol we tested and such vulnerabilities are less likely. This is also supported by the previous reported vulnerabilities where only one vulnerability was reported in UDP.

One of our assumptions is the realistic setting that we see the DUTs as true blackboxes. On the one hand, this allows us to truly understand how the tests and evaluations need to be conducted to be realistic for blackbox testers. On the other hand, this setting allows to take the attacker's point of view which helps to understand possible approaches and attacks. However, this assumption also restricts our possibilities to evaluate the vulnerabilities in some sense. The first restriction is that we are not able to tell which stacks are used by the DUTs. As has been stated earlier, we compensate for this missing information by using fingerprinting techniques. By using nmap, we fingerprint the different stacks and were able to tell that all used DUTs include different stacks. With this, we are able to show that similar vulnerabilities occur in different stacks. The second restriction concerns the analysis of the found vulnerabilities. To be sure which finding correlates to which VAP, one would need to see the source code and analyse the reason for the anomaly or crash. Nevertheless, the behavior that is visible from the outside already gives hints to which VAP an anomaly or crash is related. For a deep analysis of the findings in relation to the VAPs, one needs to break the blackbox assumption. We are working in this direction by disclosing the vulnerabilities to the vendors and by cooperating with the vendors. This helps to understand the vulnerabilities better and also helps to make the industrial device more secure.

6 CONCLUSIONS

In this work, we analyzed the vulnerability groups Ripple20, Amnesia:33, and Urgent/11, developed VAPs, and implemented test scripts based on these VAPs. Our evaluation shows that these test scripts help to identify vulnerabilities based on the VAPs. In addition, it shows that similar vulnerabilities occur in implementations of the same protocol as well as in different protocols, and that these similar vulnerabilities also spread over different device classes. With this, we build a base for more efficient fuzzing strategies based on a structured way to organize knowledge regarding usual vulnerabilities.

The test scripts we developed include specific information that can be used to crash industrial device. Nevertheless, in order to support scientific reproducibility, our standalone test scripts can be received upon request.

Future work comprises the extension of our approach to other domains and vulnerability groups. It would be interesting to analyze whether our Vulnerability Anti-Patterns and test scripts are also applicable to IoT and IIoT protocols and devices. From our point of view, our Vulnerability Anti-Patterns and test scripts are written in a generic way so that this should be possible. In addition, one could include other vulnerability groups (e.g. INFRA:HALT) into the analysis and develop new test scripts if necessary. To advance the automation of our test scripts, the stateful answering machines that we developed for our test scripts could be integrated to the security testing framework ISuTest as well.

ACKNOWLEDGEMENTS

This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs.

REFERENCES

Aboba, B. and Cheshire, S. (2002). Dynamic host configuration protocol (dhcp) domain search option. Technical report, Apple Computer, Inc.

Arrieta, A., Wang, S., Arruabarrena, A., Markiegi, U., Sagardui, G., and Etxeberria, L. (2018). Multi-objective black-box test case selection for cost-effectively testing simulation models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1411–1418.

Böhme, M., Cadar, C., and Roychoudhury, A. (2021). Fuzzing: Challenges and reflections. *IEEE Softw.*, 38(3):79–86.

Caselli, M., Hadžiosmanović, D., Zambon, E., and Kargl, F. (2013). On the feasibility of device fingerprinting in industrial control systems. In *International Workshop on Critical Information Infrastructures Security*, pages 155–166. Springer.

dos Santos, D., Dashevskyi, S., Wetzels, J., and Amri, A. (2021). Amnesia:33 - how tcp/ip stacks breed critical vulnerabilities in iot, ot and it devices. Technical report, Forescout Research Labs.

Droms, R. (1993). Dynamic host configuration protocol. RFC 1541, RFC Editor. http://www.rfc-editor.org/rfc/rfc1541.txt.

Fetzer, C. and Xiao, Z. (2002). An automated approach to increasing the robustness of c libraries. In *Proceedings International Conference on Dependable Systems and Networks*, pages 155–164. IEEE.

Hecht, G., Rouvoy, R., Moha, N., and Duchien, L. (2015). Detecting antipatterns in android apps. In *2015 2nd ACM international conference on mobile software engineering and systems*, pages 148–149. IEEE.

Julisch, K. (2013). Understanding and overcoming cyber security anti-patterns. *Computer Networks*, 57(10):2206–2211.

Kohl, M. and Oberman, S. (2020). Ripple 20 - CVE-2020-11896 RCE CVE-2020-11898 Info Leak. Technical report, JSOF Research Lab.

Kohl, M., Schön, A., and Oberman, S. (2020). Ripple 20 - CVE-2020-11901. Technical report, JSOF Research Lab.

Miller, B., Zhang, M., and Heymann, E. (2020). The relevance of classic fuzz testing: Have we solved this one? *IEEE Transactions on Software Engineering*.

Mockapetris, P. (1987a). Domain names - concepts and facilities. STD 13, RFC Editor. http://www.rfc-editor.org/rfc/rfc1034.txt.

Mockapetris, P. (1987b). Domain names - implementation and specification. STD 13, RFC Editor. http://www.rfc-editor.org/rfc/rfc1035.txt.

Nafees, T., Coull, N., Ferguson, I., and Sampson, A. (2018). Vulnerability anti-patterns: a timeless way to capture poor software practices (vulnerabilities). In *24th Conference on Pattern Languages of Programs*, page 23. The Hillside Group.

Pfrang, S., Meier, D., Friedrich, M., and Beyerer, J. (2018). Advancing protocol fuzzing for industrial automation and control systems. In *ICISSP*, pages 570–580.

Pfrang, S., Meier, D., and Kautz, V. (2017). Towards a modular security testing framework for industrial automation and control systems: Isutest. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–5. IEEE.

Seri, B., Vishnepolsky, G., and Zusman, D. (2019). Urgent/11 - critical vulnerabilities to remotely compromise vxworks, the most popular rtos. Technical report, ARMIS.

Smith, C. U. (2020). Software performance antipatterns in cyber-physical systems. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, pages 173–180.

Trubiani, C., Bran, A., van Hoorn, A., Avritzer, A., and Knoche, H. (2018). Exploiting load testing and profiling for performance antipattern detection. *Information and Software Technology*, 95:329–345.

Tuma, K., Hosseini, D., Malamas, K., and Scandariato, R. (2019). Inspection guidelines to identify security design flaws. In *Proceedings of the 13th European Conference on Software Architecture-Volume 2*, pages 116–122.

Wert, A., Oehler, M., Heger, C., and Farahbod, R. (2014). Automatic detection of performance anti-patterns in inter-component communications. In *Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures*, pages 3–12.