THE ANALYSIS OF BINARY FILE SECURITY USING A HIERARCHICAL QUALITY MODEL

by

Andrew Lucas Johnson

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

December 2021

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

ABSTRACT

Software security is commanding significant attention from practitioners. In many organizations, security assessment has been integrated into the software development lifecycle, which allows for continuous monitoring of software weaknesses and vulnerabilities throughout the development process. One often overlooked aspect of the software development lifecycle is the end of the lifecycle. Prior to delivering software to customers, many vendors digitally sign and compile source code into a binary. In binary form, analysis may be done to reveal security flaws that were not present in the original code or that were injected at some point between the code being written and the code being compiled.

Our research goal is to improve our ability to assess the security quality of a binary from different stakeholders' perspectives. While many analysis tools exist that identify security flaws, there is little work done to enable the use of multiple tools, which is necessary to identify different types of security flaws. To accomplish our goal, we approach the problem from the perspective of quality modeling. We have designed and developed a software quality model for assessing security quality in binaries (PIQUE-Bin) and operationalized the model by using PIQUE, the Platform for Investigative software Quality Understanding and Evaluation. The design of our model is based on the Microsoft STRIDE model and the software development view of the Common Weakness Enumeration (CWE). The model produces a relative and subjective security score for a binary file.

An informal literature review reveals a lack of model-based security metrics targeting binary files, which helped motivate this research. To enhance the validity of this work, a sensitivity analysis assessment based on a benchmark repository of 700 binary files was performed. Model output is validated by measuring tool output sensitivity and calibrated against the presence of injected vulnerabilities. We find that our model is able to measure the security quality of binaries relative to the benchmark repository.

1

INTRODUCTION

It is no secret that cyber threats have been on the rise in recent years. One recent large scale compromise occurred in 2020 with the compromise of SolarWinds software in the attack referred to as SUNBURST[1]. Attackers were able to compromise the SolarWinds build process such that any system with the SolarWinds software would have an exploitable vulnerability. Attacks such as these show the need for security at multiple layers, or defense in depth. Software may be compromised at any point in its lifecycle which means that security assessment must be done throughout the lifecycle, including when it has reached its final destination in the form of a binary.

There are a plethora of different analysis tools available for identifying vulnerabilities, weaknesses, and malware within a binary (for examples, see [44, 13, 11, 36, 45]). There is also evidence to suggest that multiple methods of automated analysis are necessary to cover many different types of security findings [45, 43, 28]. With the use of multiple tools comes the drawback of a massive amount of varied findings. Because these tools often find false positives or a large amount of low priority true positives, there can be too many findings to feasibly check them all manually. This motivates the scoring, management, and aggregation of security findings. One way to accomplish this is through the lens of software quality modeling.

Quality modeling enables developers to track the quality of their project over time. This assists in the regulation of code quality, allowing developers to set and maintain a quality goal [25]. The ISO 25010 software quality model defines how software quality may be broken up into characteristics and sub-characteristics. Though security is a part of the ISO 25010 software quality model, the definition is abstract, broad, and does not include availability as a sub-characteristic, which can be a very important security aspect in some settings such

[1] https://www.solarwinds.com/sa-overview/securityadvisory

as critical infrastructure. We will be expanding on the aspects of security quality presented in the ISO 25010 standard and utilizing a software quality modeling approach to assess security quality in binaries. The Platform for Investigative software Quality Understanding and Evaluation (PIQUE) [40] is used to build an operational quality model.

In this thesis we propose, operationalize, and validate a security quality model for binaries, PIQUE-Bin. First, background will be given on security findings, quality modeling, vulnerability scoring, management, prioritization, and operation technology in chapter 2. Support quality modeling frameworks are described in chapter 3, and an informal review of similar models is presented. We define our research goals in chapter 4, and then explain the methods and reasoning surrounding the model design, development, and calibration in chapter 5. Finally, several case studies are performed to improve the model in chapter 6 and validate the model in chapter 7.

3

BACKGROUND

Binary Analysis

Automated binary analysis is done to identify vulnerable or malicious code within a binary. We first review malware analysis of binaries, then move on to vulnerability analysis.

Malware analysis is the analysis of software to identify malicious behavior. Malware analysis has been around for as long as malware, when the latter was classified into trojan horses, viruses, and worms. Early malware analysis was primarily focused on viruses. Cohen defines the term 'virus' and begins feasibility analysis with respect to virus detection in his 1987 work [12]. Cohen finds that a virus can be made for which the identification process is undecidable [12]. A study in 2003 then found that reliable identification of a mutating computer virus is NP-complete [47]. Clearly, virus detection, and therefore malware detection, is a difficult problem. This led to the need for malware analysis methods that would use either some proxy for virus detection or an approximation algorithm.

Malware analysis methods can be classified as *static* or *dynamic*, [10] otherwise referred to as *signature-based* or *behavior-based*[5]. Many methods blur the lines between these two classifications resulting in a hybrid method.

Early forms of malware detection were focused on static methods such as pattern matching and string identification. One of the simplest of these methods is taking the hash of a binary to compare it with the hashes of known malware binaries. This method is used by popular malware intelligence databases such as VirusTotal[1] and is a good first step to identify malware. This is a fast method to determine if a binary is a malware binary that had already been identified; however, it suffers from several faults. Any changes in the malware will render this detection method null as the hash would completely change and it

[1] https://www.virustotal.com/

is only effective for known malware.

Static methods tend to share these faults: they excel in identifying known malware, but tend to have difficulty identifying new or obfuscated malware. On the other hand, dynamic analysis of malware is able to identify new malware, but can suffer from a lower accuracy rate and a higher false positive rate [10] [5], as well as greater complexity in the analysis due to the need to execute the binary [6]. When dealing with potentially malicious binaries, execution must be handled with care - this means sandboxing and isolating, as well as avoiding any anti-analysis capabilities malware might have. Both techniques have their strengths and weaknesses, so the use of multiple methods may yield the greatest results.

There are many recent studies that propose methods for analyzing malware through novel feature extraction, new analysis methods, or new machine learning techniques (for example, [17, 37, 30, 38, 51]). Clearly, malware detection techniques are abundant and improving. Similarly, vulnerability analysis is an evolving landscape in which new techniques are being actively developed and improved. The National Vulnerability Database (NVD) defines a vulnerability as "A weakness in the computational logic (e.g., code) found in software and hardware components that, when exploited, results in a negative impact to confidentiality, integrity, or availability"[2]. Weaknesses are more generic forms of security flaws in software, meaning that vulnerabilities are the result of an instance of a weakness occurring in software.

Automated vulnerability analysis, much like malware analysis, is often separated into static and dynamic analysis. Static analysis techniques analyze a binary without executing the file. This often takes the form of pattern identification, such as in the tool cwe_checker. This tool checks for patterns that often indicate the presence of elements of the Common Weakness Enumeration (CWE), an enumerated catalogue of software weaknesses. To check for integer overflow or wraparound (CWE-190) in a binary, cwe_checker checks for the

[2]https://nvd.nist.gov/vuln

following [3]:

> For each call to a function from the CWE190 symbol list we check whether the basic block directly before the call contains a multiplication instruction. If one is found, the call gets flagged as a CWE hit, as there is no overflow check corresponding to the multiplication before the call. The default CWE190 symbol list contains the memory allocation functions *malloc*, *xmalloc*, *calloc* and *realloc*.

This type of analysis typically suffers from a high false positive rate but is fast and does not require as much overhead as dynamic analysis. These methods also excel in identifying known vulnerabilities in binaries, such as those incorporated through third party libraries.

Dynamic analysis involves executing a binary to look for behavior that is indicative of a vulnerability. This type of analysis offers much more semantic information about identified vulnerabilities and typically identifies less false positives. However, the requirement of executing the binary can often complicate the analysis process and generally takes longer than static analysis [6].

It has been found that while static and dynamic analysis techniques have their strengths and weaknesses, a mixture of techniques is required to find all kinds of vulnerabilities [6], [23]. It has also been found that static analysis is an effective method for initial analysis to identify vulnerabilities [43]. This motivates the use of multiple analysis tools to identify a wider variety of vulnerabilities.

Quality Modeling

The model proposed in this thesis is built upon quality modeling, so we first discuss the benefits and reasoning behind the use of a quality model. Software quality modeling provides

[3] https://github.com/fkie-cad/cwe_checker/blob/master/src/cwe_checker_lib/src/checkers/cwe_190.rs

"a systematic approach for modeling quality requirements, analyzing and monitoring quality, and directing quality improvement measures. They thus allow ensuring quality early in the development process" [52]. Therefore, the modeling process enables tracking of quality for purposes of improvement or assessment. Typically, software quality modeling would be employed by an organization seeking to increase their own software quality, but can also be used by an outside entity seeking to ensure a product is being delivered with a certain level of quality. This would be the most likely use case for PIQUE-Bin, where we are looking at applying a quality model at the latest point in the development lifecycle, without necessarily measuring the product in early stages.

At the lowest level, software is assessed through metrics and findings. **Metrics** are mathematical formulas backed by empirical evidence to describe the state of a product. The evaluation of a metric provides a measure. One example of a metric is *coupling*, which attempts to provide a numeric representation of the level of dependence between software modules using data parameters, control parameters, and number of modules [48]. **Findings** represent a section of code with some phenomenon in it and are evaluated as a count of findings. This could be something that breaks a program or introduces a vulnerability through something such as buffer overflow. An example of a weakness finding, building from the prior section where we defined the cwe_checker rule, is CWE-190. Findings can take many forms, but for this work findings will primarily take the form of weaknesses, vulnerabilities, and malicious indicators.

One prominent software quality model is the ISO 25010 standard [25]. This standard breaks quality into characteristics and sub-characteristics, giving a high level overview of software quality without detailing how to specifically measure characteristics of quality or how they compose overall quality in an operationalized model. The ISO 25010 model is shown in Figure 2.1. At the highest level is software quality. Below quality are the characteristics and even sub-characteristics of quality.

Further definition of how characteristics should compose overall quality and how characteristics may be operationalized has led to the creation of quality meta-models such as Quamoco [54], which defines a hierarchical structure through which tool findings aggregate to score quality characteristics, which then contribute to the overall quality score. This meta-model was further refined in the Quality Assessment Tool Chain (QATCH) quality meta-model [46], which limits the model to four layers through which tool findings are aggregated. PIQUE, the platform we use to create our model, enables the creation of flexible tree-based models such as Quamoco and QATCH based models. Building our model using PIQUE enables us to easily adapt and improve the model when weaknesses of the model are identified [40]. To provide additional context and history, the Quamoco and QATCH quality modelling approaches are briefly summarized in chapter 3. The PIQUE platform is detailed as well in chapter 3.

Figure 2.1: The ISO 25010 Quality Model, from [25]

Security Metrics and Vulnerability Management

As discussed in previous sections, multiple analysis types and tools are necessary to discover all types of vulnerabilities. This leads to the need to manage the findings

from multiple tools. Quite often, these findings may be assigned a score to quickly and automatically give some level of priority. However, tools may differ in how they score findings as well as what takes priority in their scores. Additionally, the scale of severity and number of findings may vary greatly. This makes the comparison of different tools' output difficult, and creates the need to manage these findings and compare findings amongst tools. We may accomplish this through providing a security metric, managing vulnerabilities, or prioritizing findings.

One difficulty of validating security metrics is that there is no way to know the true overall security quality of a binary, creating an oracle problem for any models seeking to evaluate some security metric [7, 41]. The oracle problem occurs in scenarios in which there is no known output to test output against, thus making validation difficult. Additionally, this means that most security metrics are subjective in some sense.

The Common Weakness Enumeration[4] (CWE) and Common Vulnerabilities and Exposures[5] (CVE) are two catalogs that capture information pertinent to weaknesses and vulnerabilities. The CVE catalog is an enumeration of specific vulnerabilities in certain products and platforms, while the CWE catalog is an enumeration of general software and hardware weaknesses. This means that individual CVEs are realizations of a CWE. For example, CVE-2021-3031[6] is an instance of CWE-200[7]. CVEs may be scored using the Common Vulnerability Scoring System (CVSS[8]), which gives a numeric score that represents the threat that a particular CVE poses.

Similar to the CVSS which scores CVEs, the Common Weakness Scoring System (CWSS) is a method to score CWE instances. The main problem with the CWSS for automated scoring is that the information required to use it typically requires manual work

[4]https://cwe.mitre.org/
[5]https://cve.mitre.org/
[6]https://nvd.nist.gov/vuln/detail/CVE-2021-3031
[7]https://cwe.mitre.org/data/definitions/200.html
[8]https://nvd.nist.gov/vuln-metrics/cvss

to find. While CVSS also requires this, the work has been done for many CVEs already and so may be leveraged automatically. Theoretically, there are many ways to score a vulnerability as the CVSS does, but CVSS in particular has been found to be consistent with expert opinion [21] and credible through Bayesian analysis [27]. These findings indicate it may be a good catalog to consider when building a security quality model.

The CVSS gives a score for a single vulnerability, but does not provide insight into how multiple CVEs in a system impact the overall security. Similarly, there are many methods to manage or prioritize vulnerabilities. Methods such as the ones found in [4, 18, 24, 15] shed light on how vulnerabilities may be effectively prioritized, but do not give an overall idea of how serious the set of security findings may be within a given binary.

Security metrics based solely on CVSS scores have not been found to accurately reflect the security of a system in terms of time-to-compromise [22], though the reason for this is not clear. Additionally, simplistic models such as the weakest link appear to be less promising than more complex models that take into account multiple vulnerabilities [22]. These results should be taken with a grain of salt, as the study that makes these statements had a small sample size.

Surveys have been conducted on security indicators [41] and network-based security models [39]. Mellado et al. [35] compare software design security metrics. Rudolph and Schwarz [41] define the following terms:

- **Security Indicator.** A security indicator is any observable characteristic that correlates (or is assumed to correlate) with a desired security property. The set of feasible indicator values is assumed to form (at least) a nominal scale.

- **Security Measure.** A security measure assigns to an object a security indicator value from an ordinal scale according to a well-defined measurement protocol.

- **Security Metric.** A security metric is a security measure with an associated set of rules for the interpretation of the measured data values.

Let us consider an example for each of these definitions to provide some clarity. A security indicator could be something such as a potentially dangerous system call. A security measure based on this indicator is the count of these indicators for some specific software. A metric, then, would be the comparison of this number of dangerous system calls compared to other software systems, allowing a relative score to be given. Interpretation of this metric is as simple as understanding that it is relative to other software, so a score of 0.5 means that the severity of dangerous system calls in this binary is very similar to other software, whereas greater than 0.5 is better (fewer dangerous system calls).

Under these definitions, PIQUE-Bin will be considered a security metric, as it is may be considered a set of rules for interpreting the measures which come from the indicators provided by the tools used in our model. Indicators are synonymous with diagnostics in the model and security measures to measures in the models. It is important to note that we should consider security metrics rather than indicators or measures when looking to holistically assess security in a binary. While indicators and measures provide valuable information pertaining to security, a holistic security perspective must consider multiple analysis types and synthesize information available, and therefore must be composed of multiple measures and have guidelines for interpretation. Additionally, we should compare our model to other metrics rather than measures or indicators.

Rudolph et al. [41] conducted a survey of security indicators. Security indicators, as previously defined, are observable characteristics that are assumed to correlate with security. These are more 'raw data' than metrics which should be acted upon. They conclude that indicators (as of 2008) are lacking in several ways. They list several requirements for reliable and meaningful security indicators, measurements, and metrics. It must have a well-defined goal, provide a rationale, be based on objective parameters, be reproducible, be quantitative

(with a baseline and goal), provide guidance for interpretation, be constructive, and be applicable to large/complex targets. They also state that a more balanced coverage of lifecycle is desirable. This includes coverage of early stage designs and of final products, which for software would be a working project in the form of an application or binary.

Ramos et al. [39] perform a survey of model-based quantitative security metrics at the network level. The authors define a method of classification for security models, as seen in Figure 2.2. This classification method will be used in section 3 to categorize models for review. While the authors fail to define their search strategy for identifying these model-based metrics, they present many model-based security metrics at the network level. The authors conclude that the state of model-based network security metrics is still in development and needs much more progress. No current approaches are totally satisfactory, but the value of the models is in assisting the decision-making process rather than perfectly assessing security.

One such model presented in [39] is VEA-bility [50], which uses multiple input types to assess network security in a holistic fashion. They consider the vulnerability, exploitability, and attackability, input as network topology, attack trees, and CVSS scores of vulnerabilities. The model is successfully able to compare the security of network topologies. This model is similar to the type of metric we are looking for in chapter 3 section 3; however, VEA-bility aims to assess network topology rather than binary files.

The Operational Technology Environment

The Operation Technology (OT) is a computing environment where the purpose of the system is to monitor and control a physical process, typically industrial processes such as those found in manufacturing plants. The OT environment differs from traditional

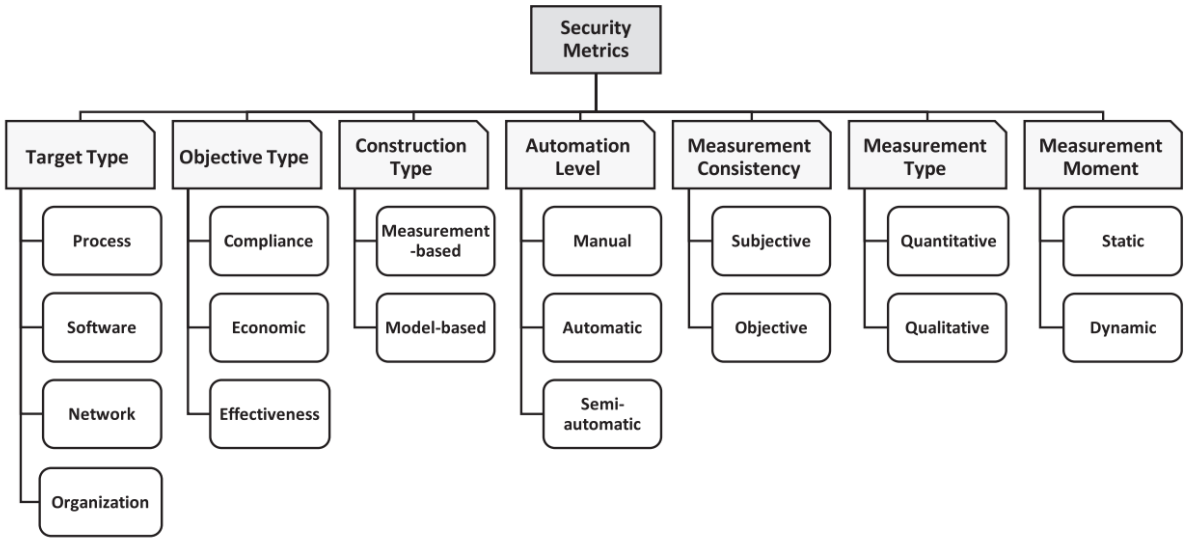

Figure 2.2: Security Metric Classification Method as presented in [39]

Information Technology (IT) environments in several important ways when considering security. The primary difference between OT and IT environmental is the purpose. While the IT environment is focused on information, OT environments are focused on maintaining physical processes. This difference alone puts much more emphasis on securing OT environments because security compromises can have potentially life-threatening physical consequences. Industrial Control Systems (ICS) are an OT environment of focus in recent years. ICSs include Supervisory Control and Data Acquisition (SCADA) systems, Distributed Control Systems (DCS), and Programmable Logic Controllers (PLC) [49].

McLaughlin et al. [34] enumerate the primary differences between IT and OT systems:

- OT systems maintain the integrity of an industrial process
- OT systems require high availability due to to their continuous nature
- OT systems often focus heavily on physical processes
- OT systems have limited resources for security

- OT systems require timely response to human reactions and/or physical sensors
- OT systems use proprietary communication protocols to communicate with field devices
- OT systems rarely replace components
- OT systems are composed of distributed and isolated components

In addition to this, the authors paint a picture of the modern ICS cybersecurity landscape in [34]. In recent years, there has been a trend of moving OT networks to be in contact with the internet [34]. This enables remote access and control of those systems which may allow for higher process efficiency and response times. However, this has the unintended consequence of opening these systems to a huge new attack vector. These systems are often legacy and cannot be upgraded [49, 34], meaning that they have many known exploitable vulnerabilities.

Attacks on ICSs have been increasing in frequency and severity in recent years. In 2015, a SANS Instute survey found a 7% increase in the number of ICSs that had been attacked, and 5% increase in ICSs that were potentially attacked [33]. In part, this increase in attacks is likely attributable to a rise in nation-state sponsored or organized cyber attacks/cyber warfare, which rose to represent over 50% of malicious attacks from outsiders in 2019 from 0.0% in 2017 according to another SANS Institute survey [16]. Although it seems unlikely that none of the cyber attacks in 2017 were attributed to organized crime or nation-states, it is possible that the attacks simply could not be traced to a specific source. Additionally, because these are self reported sources of attacks, they are to some extent subjective. Another troubling trend in ICS security is that, while in 2017 25% of survey responders were unable to answer these questions, 43% were unable to answer in 2019. This indicates that there may be an even higher number of undisclosed attacks that are not considered in these numbers.

The National Institute of Standards and Technology (NIST) 800-82 publication [49] gives a comprehensive guide to security for ICS. This publication provides methods for securing ICSs while addressing their unique requirements in comparison to IT systems. NIST 800-82 also provides an overview of common system topologies, typical threats and vulnerabilities, and effective countermeasures to mitigate these risks. Several common topologies covered in the 800-82 may be seen in 2.3.

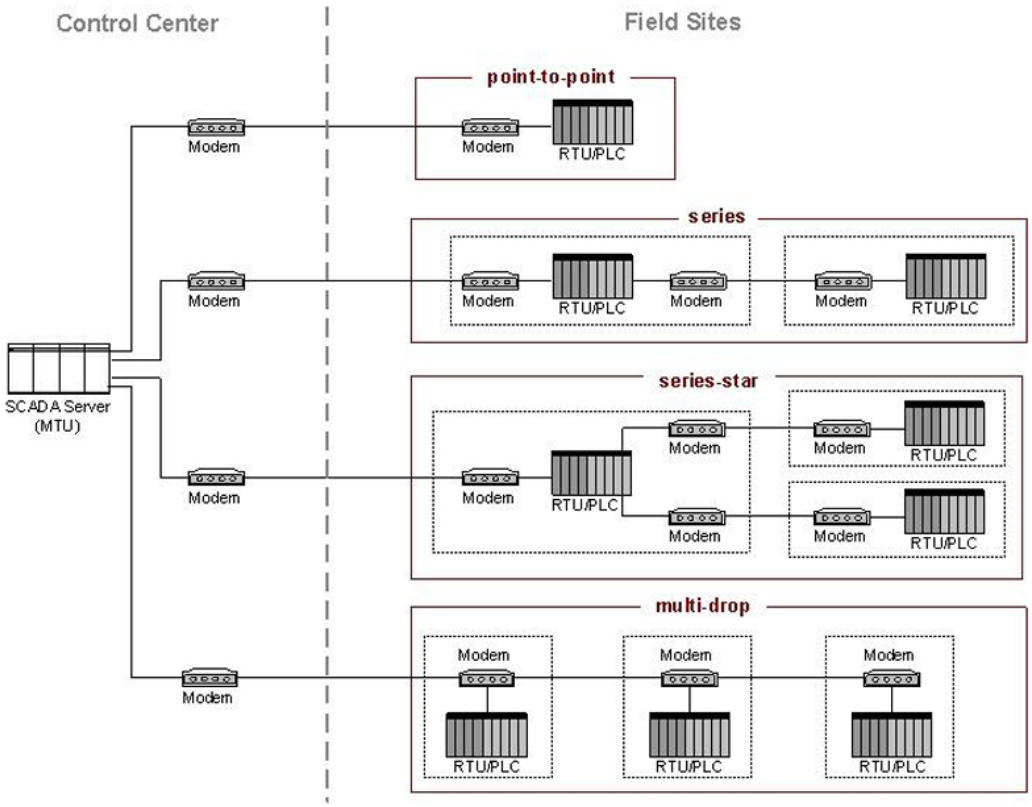

Figure 2.3: Basic SCADA Communication Topologies as presented in [49]

The Purdue model was developed in the early 90s and has developed as a method of identifying best practices for communications of components in ICS and IT networks [55, 32].

The model separates ICS systems into five different layers of components:

- level 5: enterprise networks
- level 4: business networks
- level 3: site-wide supervisory
- level 2: local supervisors
- level 1: local controllers
- level 0: field devices

Levels four and five are often combined to "IT networks", and Levels one and zero are often combined in modern systems where field devices are often sophisticated enough to handle local automation of a process [32]. Between levels three and four we see a boundary (demilitarized zone) between IT and OT systems. This model helps clearly define boundaries and components of systems, allowing for air-gaps to clearly segregate systems properly.

In a similar vein to the NIST 800-82, Li et al. [31] give additional information, highlighting microgrids as an instance of a DCS. They cover the specific consequences of vulnerabilities being exploited, and common cyber vulnerabilities in microgrids. They separate the most common vulnerabilities into vulnerabilities that occur in application software, through communication networks, and on field devices. Primarily, Li et al. highlight the importance of proactivity to mitigate cyber risk and to ensure resiliency and reliability in critical systems.

As an example of the consequences of a successful OT attack, we describe Stuxnet. Stuxnet may be one of the earliest, most successful and widely known attacks on OT systems [29]. Stuxnet targeted Iranian uranium facilities to systematically destroy the facilities while feeding false information to the monitoring systems. This worm was able to precisely target

the specific Siemens controller in use by the Iranian facility to ensure that the only target that would be destroyed would be the "correct" one. Stuxnet did this by exploiting the vendor's driver DLL file in use by the SCADA software and the programming software. When it found the correct target, it dropped malicious code onto the controller alongside the code running the controller correctly. When the time came, the malicious code executed and destroyed the target systems. In the end, one of the actual vulnerabilities that Stuxnet exploited is technically a feature of the controllers: not allowing for code signing. This cannot be patched aside from changing the controllers in use by the system. The way to mitigate this attack is to avoid the compromised DLL that enabled the malicious code to be dropped onto the controller [29].

17

SUPPORTING WORK

PIQUE-Bin is built upon the PIQUE framework [40]. PIQUE is a platform that builds upon two earlier quality modeling approaches, Quamoco and QATCH. In this chapter, we briefly summarize Quamoco, QATCH, and PIQUE as the predecessors that enable the building of PIQUE-Bin.

We introduce each model's terminology as they define and use it, but for other sections of this paper it is safe to assume it is the definition from PIQUE, as this is the latest and most relevant of the models for PIQUE-Bin.

Quamoco

The Quamoco Project aimed to develop and validate operationalized quality models for software, as well as to "provide the missing connections between generic descriptions of software quality characteristics and specific software analysis and measurement approaches" [52]. Quamoco also gives a meta model which defines valid Quamoco quality models [54].

Quamoco Model Terms

Quamoco defines a model as depicted in Figure 3.1.

Factors are high-level terms which define some property of an entity. Factors are not directly measurable, but can be approximated through the aggregation of measures.

Quality Aspects are the highest level of factor in a quality meta model, and directly compose quality. In the ISO 25010 mode, factors would be high-level characteristics such as maintainability.

Product Factors are factors that are one level below quality aspects which compose quality aspects. In the ISO 25010 model, these would be sub-characteristics of quality such as modularity or reusability.

Measures are concretely defined concepts that are quantifiable by values directly obtained from instruments.

Instruments are tools which obtain values or findings from the software product.

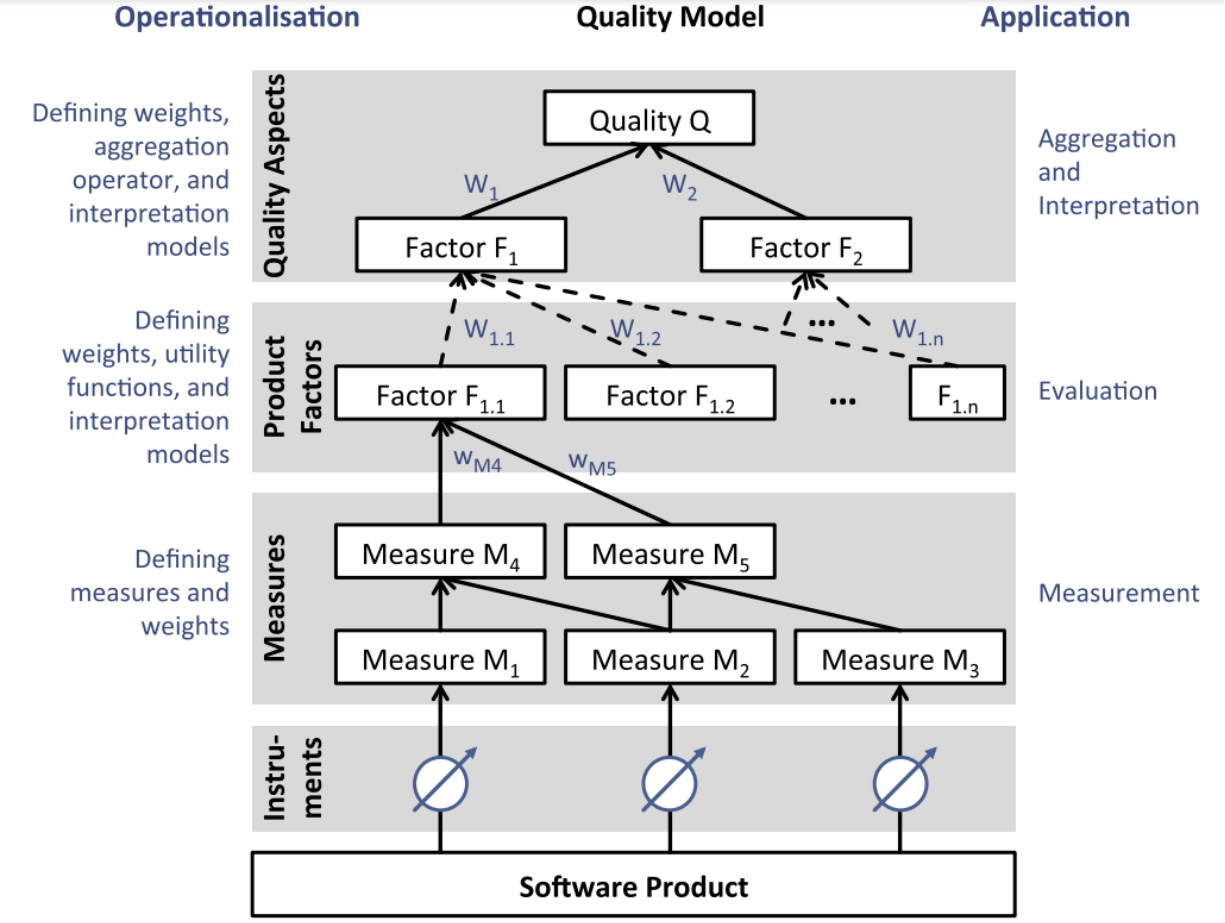

Figure 3.1: The Quamoco Quality Modeling Approach [52]

Main Concepts

The Quamoco quality assessment approach aims to bridge the gap between the ISO 25010 standard, which gives a high-level overview of quality, and metrics output by tools.

This is done through defining an 'impact' relationship and a quality meta model. The impact relationship allows quantitative values from instruments to aggregate through measures and factors which are impacted by these values. The quality meta model defines a structure which a Quamoco quality model should adhere to. Defining and adhering to this meta model enables a common framework of understanding and confidence in these models.

Additionally, Quamoco emphasizes the modularity of their quality assessment approach. This enables the high-level factors to remain the same, regardless of how lower level instruments gather their values. As such, one may use the same model structure for analysis of multiple languages, as the only necessary changes are the instruments themselves.

Mechanisms

Beyond the definition of a tree structure and aggregation of findings, several additional mechanisms are involved with using an operationalized Quamoco model to assess quality. These mechanisms are normalization of tool output, the use of a utility function, and factor edge weighting.

Normalization occurs in the measure nodes in Quamoco models. A measure for normalization is found, such as *lines of code*, which is then used to calculate the value of additional measures which have their values divided by the normalization value.

Utility Functions are applied in the measure nodes in Quamoco models. A benchmark data set is used to derive the utility function for each measure, which is then applied when evaluating measures.

Factor Edge Weighting is applied to all product factor and quality aspects as they are aggregated into the next layer of the model. This is done using their relative importance, calculated by using human-gathered importance orderings for the rank-order centroid method [8].

Assessment using a Quamoco quality model involves several steps. The measurement

step involves running tools against the product under assessment. Then, evaluation occurs in which measure nodes take on a value according to the tool findings and previously defined utility function. Aggregation of all factors is the next step, in which factors take on values according to their incoming weighted edges. Finally, interpretation must occur. Quamoco suggests mapping values to a grading scale from A to F [52].

Results

Overall, the Quamoco project contributes greatly to taking quality modeling from an abstract concept as in the ISO standards to an operational model. By bridging the gap, they enable quality modeling and quality modeling research to move forward with the idea of using a model to extend the definitions provided by the standard.

To ensure validity of the Quamoco approach, researchers compared how their model ranked five projects against expert judgements of those projects. They found high correlation and significance of the correlation between these rankings, indicating that a Quamoco quality model is able to judge quality in a way that is consistent with expert opinion.

Although the Quamoco model is able to assess software effectively, the authors of Quamoco state that in its final state, the operational Quamoco quality model became very large and complicated. It became slow and unresponsive. The authors indicate that they are unsure of the extent to which it may be utilized in industry due to these problems. This motivates the development of QATCH, an extension and simplification of Quamoco quality models.

QATCH

QATCH, the Quality Assessment Tool Chain [46], is an extension to work done in the Quamoco project. QATCH focuses on automating the derivation of quality models that are sensitive and responsive to the subjectivity of stakeholders. Additionally, they approach

modeling with simplicity and transparency as a bigger focus, likely due to the results reported in the Quamoco project.

QATCH defines its own model structure, terms, and approach; however, we focus primarily on the differences between Quamoco and QATCH, as largely, they follow the same approach.

QATCH Model Terms

QATCH defines a quality model in several layers. An example of a generic QATCH model instance is given in Figure 3.2.

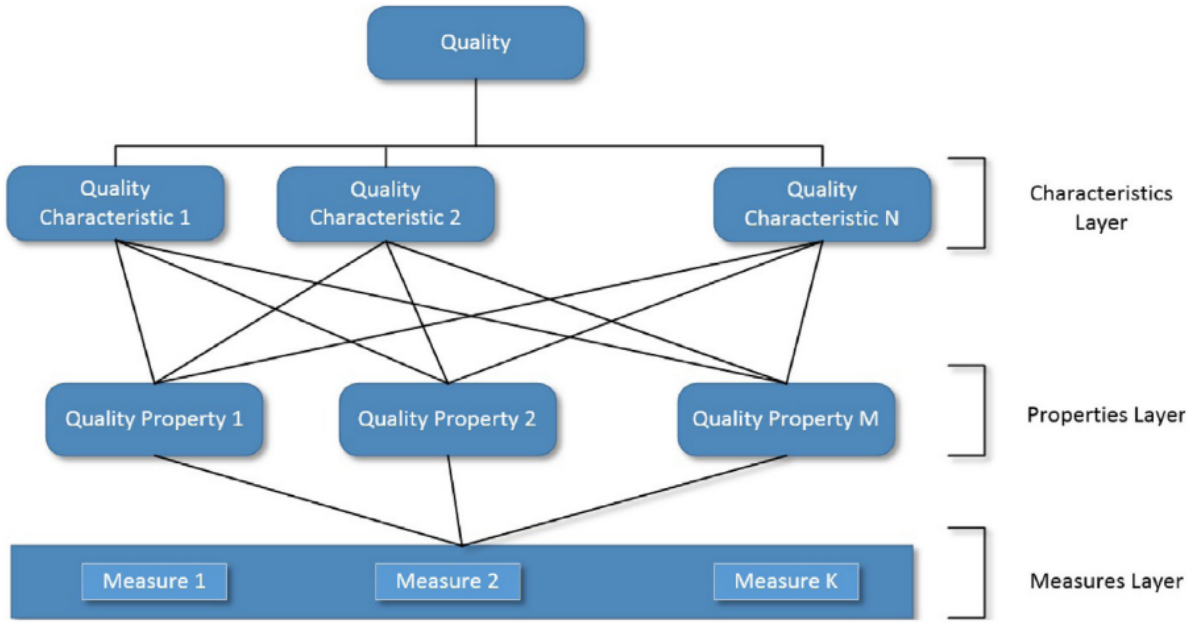

Figure 3.2: A Generic QATCH Model Instance [46]

Characteristics are defined as the abstract components of a system. This closely matches the quality aspects in Quamoco models.

Properties are attributes of an object that can be directly measured. This relates to the product factors in Quamoco models.

Measures are concrete, quantified representations of tool results. This definition is very similar to a Quamoco measure.

Unique Mechanisms

QATCH Models aim to restrict the complexity of quality models to enable simpler models that are more easily understood. To achieve this aim, they restrict QATCH models with the following rules:

- The model must not be derived from black box methods, such as machine learning techniques.
- The model must have only three layers.
- A property node is quantified by a single measure only.

These design decisions are supported by issues brought on by the complexity of the Quamoco quality model used in [52], as well as [19], which argues that comprehensibility and ease of extension outweigh loss of granularity brought on by complex models such as the model in [52].

Quality is and always will be subjective to some extent. One prominent concern of quality modeling then, is the time requirement to tune a quality model to a stakeholder's subjectivity. QATCH approaches this by allowing stakeholders to express judgments of quality concepts using intuitive terms that do not require technical knowledge of quality modeling. They then integrate these judgements into the model derivation process using the Analytical Hierarchy Process (AHP).

The AHP provides the ability for stakeholders to input their priorities and values to a quality model. The AHP is a method typically used to help decision makers in

scenarios where there are several objectives [42]. AHP allows stakeholders to make pairwise comparisons between criteria to derive an order of importance for decision. In the context of quality modeling, our overall goal is quality, and our criteria are quality aspects. In this way, stakeholders are able to decide what quality aspects are important for overall quality in the context of a quality model, without the need of extensive knowledge of quality modeling or model structure.

The pairwise comparisons necessary for AHP may be expressed linguistically, meaning that the comparisons may be very intuitive for non-technical stakeholders to express their values and judgements. QATCH also expands upon the use of AHP, presenting a fuzzy AHP that also allows the stakeholders to express uncertainty about specific comparisons.

Results

The QATCH model underwent the same experiment as the Quamoco validity experiment in [52]. The QATCH model reports perfect correlation with expert opinion, indicating that QATCH is able to perform comparably (possibly even better) than Quamoco in how it judges quality compared to expert opinion. Additionally, this indicates that the loss of complexity in the model did not result in a loss of correlation with expert opinion.

PIQUE

The Platform for Investigative software Quality Understanding and Evaluation (PIQUE) [40] was created to provide an environment for quality assessment research. The platform focuses on reducing the resources required to build a hierarchical quality model, from conceptualizing to operationalizing a quality model. PIQUE was built with nine design goals:

1. Benchmarking, utility functions, and adaptive edge weighting

2. Default model mechanisms
3. Extension or modification of model mechanisms
4. Models are easy to derive
5. Derived models are easy to operationalize
6. Adding, removing, or modifying tool support is simple
7. Input and output is easy to interact with
8. Facilitate automation and continuous integration
9. Facilitate trustworthy models

Altogether, PIQUE is a platform that enables small scale teams to build quality models without the resources required for models such as the Quamoco model in [52]. This enables models to be created by single developers, as is the case with the model in this thesis. The models built by PIQUE are flexible, meaning they may adhere to Quamoco or QATCH meta models.

PIQUE Model Terms

Most of the definitions for PIQUE follow closely from Quamoco definitions.

Factors are abstract terms for high-level nodes at the top layers of the quality model. Their definition follows from Quamoco. Factors are not directly measurable, but can be approximated through the aggregation of measures. Factors include the Total Quality Index (TQI), quality aspects, and product factors.

TQI is the root node of a PIQUE model. It is a factor, and the value of the TQI can be thought of as the output of the model. The TQI value is the result of the final aggregation of values in the model. Rice [40] states that "in the context of ISO/IEC 25010 [25], TQI

represents Software Product Quality, but it can represent whatever concept the researcher desires, such as Security."

Quality Aspects (QA) are the highest level of factor in a quality meta model, and directly compose the TQI. In the ISO 25010 mode, factors would be high-level characteristics such as maintainability.

Product Factors (PF) are factors that are one level below quality aspects. In the ISO 25010 model, these would be sub-characteristics of quality such as modularity or reusability. Product factors are decomposed into directly measurable concepts represented by measures.

Measures are concretely defined concepts that are quantifiable by values directly obtained from instruments. Measures may be 'negative' or 'positive', where a negative measure is one in which findings negatively impact the TQI.

Diagnostics are a new term introduced by PIQUE. A diagnostic is "a representation of the parts needed for a measure to evaluate" [40]. A diagnostic is evaluated directly from tool output and is tool specific. Including diagnostics allows quality model designers to configure a measure in terms of its parts.

Diagnostics also add the ability to configure evaluation methods at another level, adding to the flexibility that is part of the design of PIQUE. Whereas traditional quality modeling only allows for the measure to be some function of a tool output, this enables multiple finding types to be considered at the measure level.

Findings are the data object representation of a "hit" from a tool. A finding is instantiated when the its associated tool is run on the system under evaluation and identifies some portion of the system which fits a rule defined by the tool.

Exemplary Model

An example of a software quality model built using PIQUE and evaluated on a project

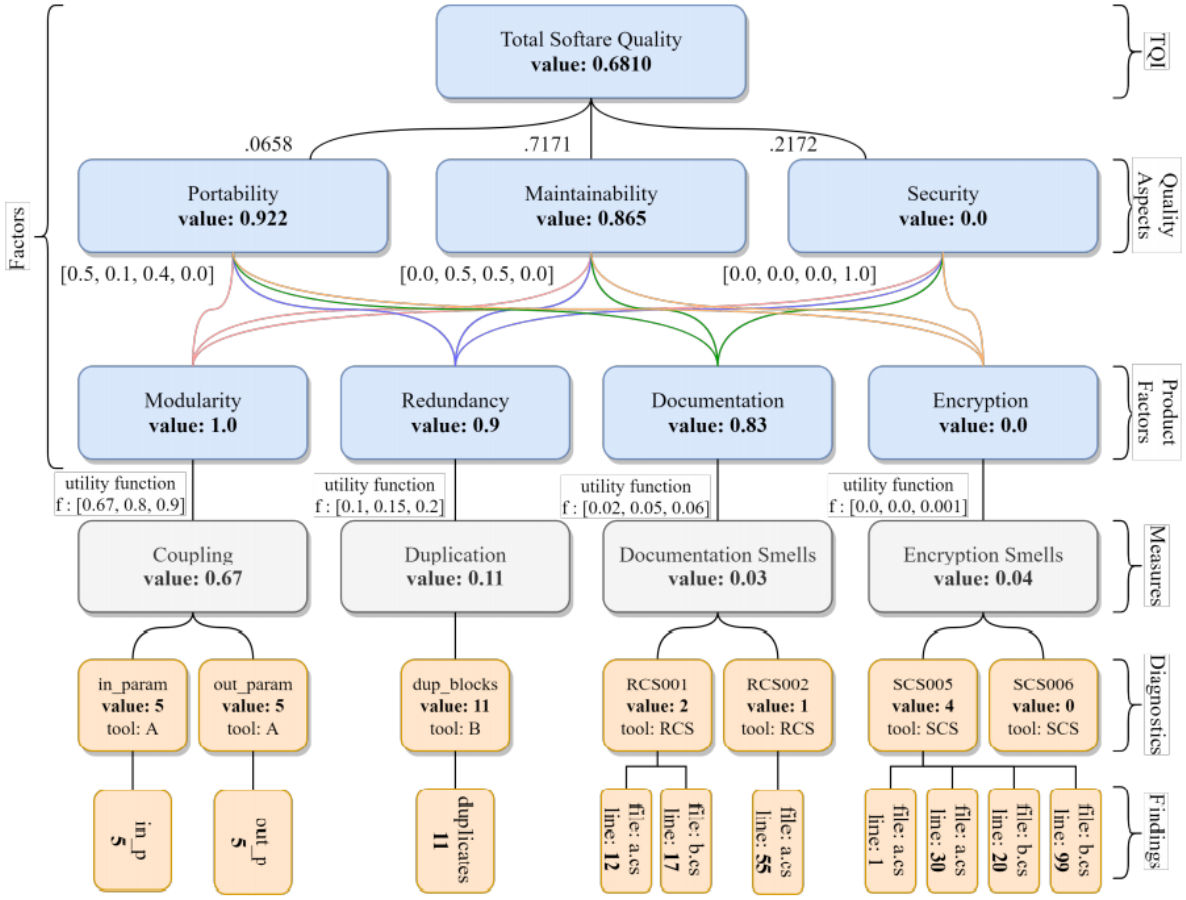

Figure 3.3: An exemplary PIQUE model, from [40]

is shown in Figure 3.3. Below the TQI are the quality aspects, factors that define the characteristics that compose overall quality.

Below the quality aspects are the product factors, which are factors that decompose directly into measurable concepts. In the ISO 25010 standard, these are concepts such as capacity or re-usability. Below the product factor layer in PIQUE, the ISO 25010 standard does not give any guidance.

Measures are concrete definitions that compose product factors. Measures contain a method for normalizing diagnostic values as well as a utility function, derived from a benchmark repository, which is used when evaluating. Figure 3.4 shows the structure of

a measure node in PIQUE. Figure 3.5 shows the sequence diagram of a call to evaluate a measure node. First, a call is made to evaluate the measure. This leads to the Evaluator's Evaluate() function being called. The Evaluator first finds the node value through a specified method - by default, this is the average value of the children nodes. This value is then normalized and used as input for the utility function. The output of the utility function is then returned as the value of the measure node.

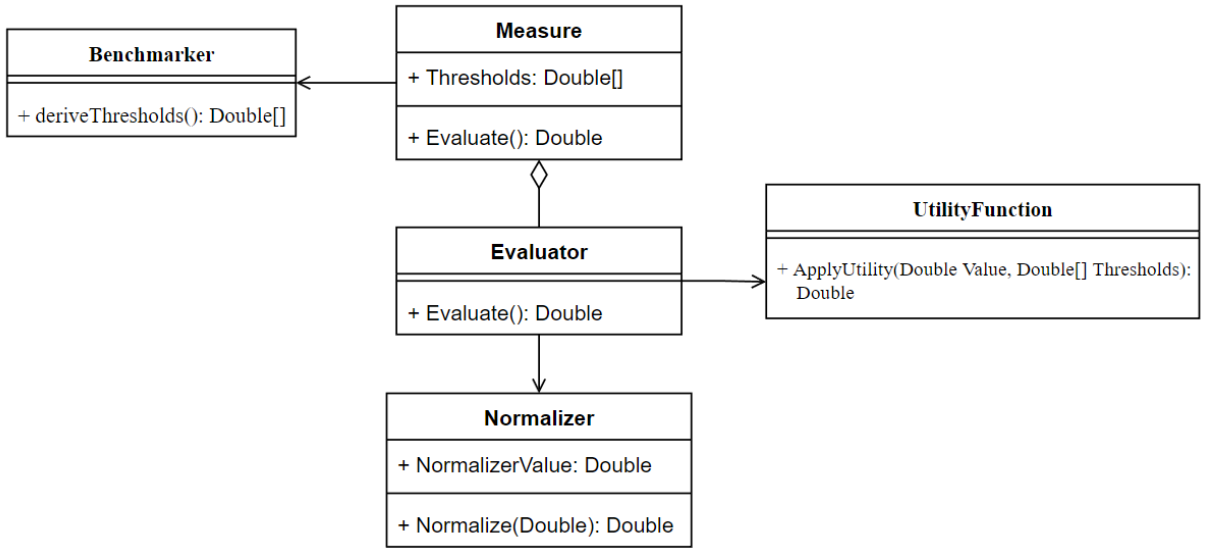

Figure 3.4: The UML Class Diagram Of A Measure Node

PIQUE Mechanisms

Utility Functions are utilized by PIQUE models at the measures layer. Utility functions are created through a benchmarking process that begins with the creation of a repository of similar projects to the one which the model will be applied. The tools of the model are

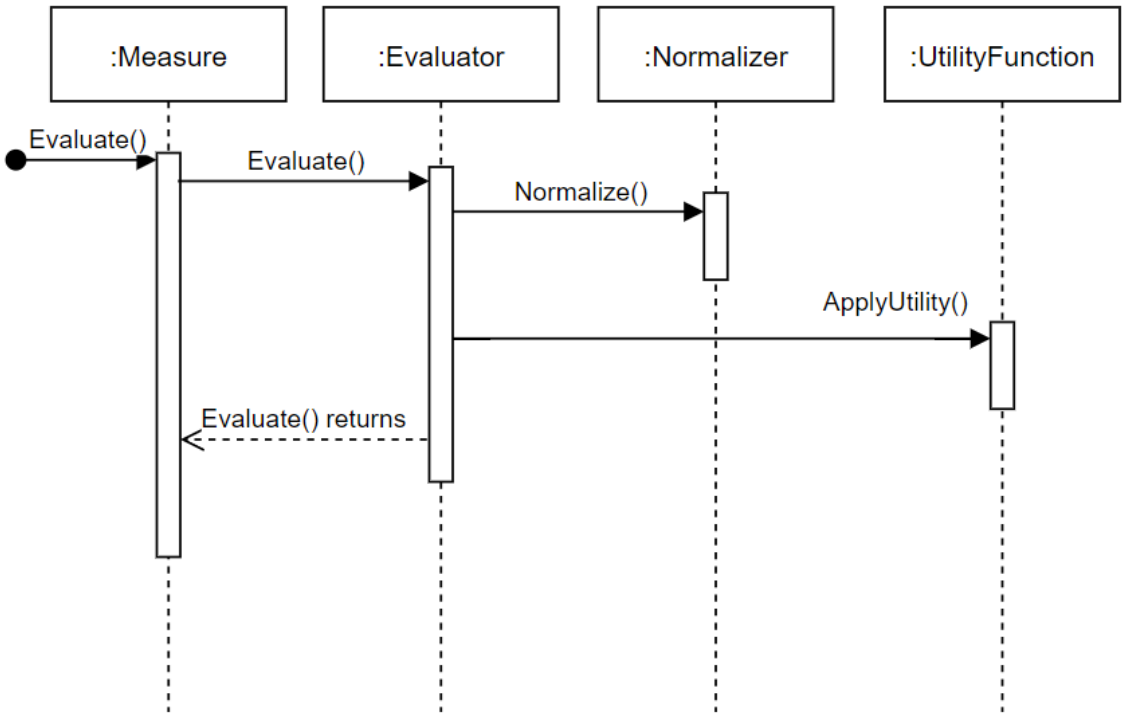

Figure 3.5: The UML Sequence Diagram Of Evaluating A Measure Node

then applied to all the projects in the repository to identify metrics such as the mean and standard deviation for each measure. This provides an estimate of the number of findings that can be expected in the average project, allowing the creation of utility functions.

The utility function for a measure takes the measurement of some diagnostic for a specific software project and outputs a value based on how it compares to other software projects in the benchmark repository. In the default case of PIQUE, the utility function is linear interpolation between the minimum and maximum value found within the benchmark repository for each measure.

Utility functions also account for differences in scale of findings between tools. That is, if one tool typically reports thousands of findings while another reports tens of findings, the

output of the utility function for both measures will be between 0 and 1.

The use of utility functions implies that a final quality score will be relative to the benchmark projects. The composition of the benchmark repository has a large impact on the output of the quality model, so selection of systems for the repository is a potential threat to validity of the model's output. It also greatly impacts how the output should be interpreted - a 0.5 might be good or bad depending upon the quality of the benchmark repository projects.

As well as the makeup of the benchmark repository, stakeholders must consider the weighting of edges when interpreting the results of applying a PIQUE model.

AHP is used to configure a model for stakeholders' needs and priorities [42]. This is also done in QATCH models. AHP allows for stakeholders to identify how important quality aspects and product factors are in relation to each other, which dictates how they will aggregate to the overall score. This process dictates the weights of edges in the higher levels of PIQUE models.

The AHP enables stakeholders to input their priorities and values to a quality model through pairwise comparisons between criteria to derive an order of importance for a decision. In this way, stakeholders are able to decide what quality aspects are important for overall quality in the context of a quality model, without the need of extensive knowledge of quality modeling or model structure. This technique allows non-technical stakeholders to impart their values on a model.

While utility functions, benchmark composition, and normalization are important concepts in PIQUE, these concepts are much more oriented towards developers. The AHP is a feature that is aimed primarily towards stakeholders. The combination of these features results in a quality model that is guided by the subjectivity of developers who created the benchmark repository, utility functions, and normalization methods, and the stakeholders

who determined the weighting of nodes in the aggregation of the model.

Default PIQUE Model Behavior

There are several default behaviors that PIQUE models use when not configured. For the most part, these are standard practices - such as the weighted average aggregation of quality aspect nodes to produce the TQI score. We define the default behaviors at each level to be clear. We briefly note where PIQUE-Bin deviates from these defaults, and in chapter 5 we discuss reasoning for the deviations. We start at the lowest level of the model, at the level of findings.

A finding is specific to a diagnostic. They have a severity of 1, unless specified otherwise. Diagnostics are evaluated as sums of the severities of their children findings. Measures are evaluated using a utility function with the input being the sum of their children's values. By default, PIQUE implements the utility function as using linear interpolation between two values, bounded to $[0, 1]$ characterized by piece-wise Equation 3.1:

$$f(x) = \begin{cases} 0 & x < a \\ 1 & x > b \\ \frac{x-a}{b-a} & a < x < b, \end{cases} \tag{3.1}$$

where a, b are referred to as thresholds, and evaluated as the *minimum* and *maximum* of the sums of children in the benchmark repository. Negative measures are evaluated as $1 - f(x)$. As an example, let us assume we have some measure M. Assume we evaluate M on five benchmark projects and receive a set of values $S = [1, 2, 3, 3, 4]$. Then we set $a = min(S) = 1$ and $b = max(S) = 4$. If we evaluate M on the binary under assessment and find a value of 2, we evaluate this as $\frac{x-a}{b-a} = \frac{2-1}{4-1} = \frac{1}{3}$, which is the value the measure M takes in our assessment. In PIQUE-Bin, we re-define the utility function and remove the piece-wise component of the utility function.

All factors (TQI, QAs, PFs) are evaluated as the weighted sum of their children. For PFs, this is the average of their children, as the structure defines what children feed into PFs. However, The layers between PFs and QAs, and QAs and the TQI are fully connected. These layers are weighted through the AHP as defined above. In PIQUE-Bin, we use AHP to weight the QAs to the TQI and rely on manual assignment of weights between the PFs and QAs.

Literature Review of Model-Based Binary Security Metrics

Search Strategy

To review the current state of binary assessment, we performed a systematic review of literature with regard to binary security assessment metrics. We first defined our research questions, search strategy, acceptance criteria, quality control strategy, and data extraction strategy.

We define what type of security metric we are searching for through the classification method presented in [39]. This classification scheme is shown in Figure 2.2. We look for security metrics that target binaries (software). We do not restrict the object of the metric, so it may be a metric of compliance, economics, or effectiveness. We look for model-based construction of the metric, as that will be similar to our proposed method. A model-based construction will be more holistic as it will not be utilizing a single type of measurement for a metric. We look for automatic or semi-automatic automation level because we seek to use these metrics for easily and effectively assessing binaries without the need for an expert to do analysis. We do not restrict the measurement consistency, so they may be subjective or objective. We do look for any measurement type, quantitative or qualitative, and any measurement moment, static or dynamic.

It is important to note that in this review we look specifically for holistic security metrics and not tools that identify specific weaknesses or vulnerabilities. Papers such as [14] and [13]

present methods for finding weaknesses in binaries, but do not attempt to holistically assess a binary based on the vulnerabilities found. In this review, we are searching for methods of assessment for binaries that take into consideration multiple weaknesses or vulnerabilities as is necessary when assessing security holistically.

With this literature review we look to answer the following question: How well do current metrics assess the security of binaries? This answer will give insight to the gaps present in the binary security assessment area. In chapter 4 we show how this research question fits into our overall research goal.

We examine how models are validated, how they perform, how comprehensive they are, and flexible the model is. While model validation is obviously important, it is also important to have a model that is flexible to allow for new security threats to be incorporated, and comprehensive such that all types of security threats can be accounted for.

Our search strategy was to use IEEE Xplore to search for papers from journals and conferences within the past 10 years (2011-2021). Our search terms are papers that have ("Binary" OR "Executable") AND "Security" AND ("Model" OR "Metric") within the abstract of the paper. We also examine the references of any papers we identify for further investigation. The goal is to identify any papers that could be relevant while limiting the search to more recent papers which may reference any older relevant metrics. One consideration is that some metrics may be missed if they are focused on a type of domain specific binary and the abstract does not specify that the domain specific entity is a binary or executable file.

To accept a paper, we ensure that the paper is focused on a method for analyzing a binary file specifically. The method in the paper must not be for generating security findings given a binary, but rather to take security findings and utilize those for a security metric. Initially, we read the papers' titles to eliminate papers that are clearly not producing some security metric. After the initial pass of reading titles to reduce the number of papers,

we read through the remaining papers' abstracts to determine if the paper fits the criteria and should be accepted. If the specific target of the metric cannot be determined from the abstract, the paper is read until it can be determined if the metric is targeting binaries. Additionally, we will search through the citations of all papers that are accepted to identify any additional papers that should be included.

Results

After performing the search as defined above, over 400 papers met the search terms on IEEE Xplore, but we have not identified any papers which fit the criteria. We intentionally used search terms that would hopefully catch any papers that would be relevant at the cost of an abundance of papers to sort through, but the lack of papers that fit the criteria using these search terms indicates that we have identified a gap in current research on model-based security metrics for binaries.

Threats to Validity

The results of the literature review indicate a gap in research. However, there are threats to this review that must be acknowledged.

First and foremost, the search criteria could have missed relevant papers. It is possible that by using some domain specific terminology, a paper could be left out of this criteria. This could be analysis of firmware update patches that take the form of a binary, or specific programs that are analyzed in binary form but not referred to as such. We attempted to use broad search terms to avoid missing any relevant papers to mitigate this threat, but we cannot entirely eliminate this threat to validity.

Additionally, any papers that would not be found by IEEE Xplore are left out of this review. This means that we miss any results that would be found in non-IEEE conferences or journals. The fact that no results were found, however, would still suggest that this topic

is largely unexplored. Additionally, no papers that would fit this criteria were identified during preliminary research which included many other conferences and journals.

Another threat to validity is the filtering of papers from those found by the search. A paper could have been dismissed when it should not have been. To mitigate this threat, any time a paper was in question of being dismissed or accepted, we would read the paper carefully until it was clear what action should be taken.

Despite these threats to validity, the conclusion remains largely the same. Even if several papers were improperly removed from the review or not found by the search, there is still a lack of model based security metrics for binaries. If any methods had been found there would be potential to improve these models; however, with no papers being identified we are confident that this subject is under-researched.

RESEARCH GOALS

Our research goal is aligned with improving the gaps identified through research. While there are many tools for conducting analysis, there are far fewer methods of interpreting the results of these tools together. We notice a distinct lack of metrics or models that assess security at the binary level, in a holistic fashion, and according to stakeholder interests. To achieve our goal, we developed a security metric based on software quality modeling.

Motivation

In chapter 2 we described to increasing attack surface and number of attacks occurring in OT environments. Due to this increase in cyber attacks, we need to increase our ability to harden our OT environments to ensure these attacks are mitigated or avoided. While many methods exist to harden these systems, attacks such as SUNBURST[1] and Stuxnet [29] have been able to circumvent these defenses. In order to better the security posture in critical infrastructure systems, we must develop additional layers of security that will identify and prevent attacks like these.

One currently under-developed area in which both IT and OT systems could be improved is security analysis at the binary level of software, which is supported by the literature review presented earlier. Aside from anti-virus (which may or may not be present in OT environments), little is done to ensure that placing and running a binary on a system will not cause additional exploitable vulnerabilities. Therefore, our goal is to improve our ability to assess the security quality of a binary from a stakeholder's perspective.

As discussed in chapter 3 section 3, we have observed a distinct lack of model-based security metrics targeting the binary level of software. Therefore, to accomplish our goal we develop and validate PIQUE-Bin for analysis of the security quality of a binary file.

[1] https://www.solarwinds.com/sa-overview/securityadvisory

We follow Basili's Goal-Question-Metric approach to our research [9].

Goal Question Metric

Following [9], we aim to achieve our goal by answering a set of questions. Furthermore, we aim to answer the questions through specific metrics. These metrics will answer our questions, which in turn will help us accomplish our goal. The specific goals, questions, and metrics of this research are presented below.

Goal: Improve our ability to assess the security quality of a binary from a stakeholder's perspective

- **Q1** How well do current metrics assess the security of binaries?
- **Q2** What are the attributes of security in binaries that are inadequately measured by current metrics?
- **Q3** To what extent does utilizing a security quality model improve the ability to identify vulnerable binaries?
- **Q4** How can we adapt current metrics for security quality to binaries in the ICS environment?
- **M1** Systematic review of model-based security metrics at the binary level (answers Q1,Q2)
- **M2** How security aspects are measured by current methods identified (answers Q2)
- **M3** Iterative case studies (answers Q3)
- **M4** Experiment to assess model (answers Q3)
- **M5** Proposal of ICS-specific model (answers Q4)

PIQUE-BIN DEVELOPMENT

To achieve the research goal of improving our ability to assess the security quality of a binary from a stakeholder's perspective, we must either improve upon existing methods or create a method which is better than existing methods. Because no existing methods for assessing security quality holistically in binaries were found in section 3, we must create our own. To this end, we utilize PIQUE to create a security-focused quality model for binaries.

In this chapter, we present the methods and reasoning used to develop the PIQUE-Bin model via the PIQUE framework. We present this in the phases of model operationalization: gather requirements, design, and develop.

Gather Requirements

As part of the process of creating PIQUE-Bin, two objectives must be achieved. First, a collection of relevant binaries for benchmarking must be gathered. These binaries should be similar in function and size to the binary under analysis. We gather these binaries from the Ubuntu 18.04 '/bin' file, Andrew-d's static-binaries repository[1], Mosajjal's binary-tools repository[2], and Kali Linux's '/bin' file.

We chose this to be our benchmark repository due to ease of gathering and similarity of function to the binary under analysis. This results in approximately 700 binaries gathered in total.

We then limit the benchmark repository to files above 50 KB and below 10 MB. We limited binaries based on size to remove trivial binaries that will have very few weaknesses and to ensure the tools will run in a reasonable amount of time. There is currently no established method to determine the best way to gather the benchmark repository or identify the ideal

[1] https://github.com/andrew-d/static-binaries
[2] https://github.com/mosajjal/binary-tools

characteristics of the benchmark repository, which are further investigated in chapter 7.

The second step in this phase is to acquire a set of relative rankings of security characteristics in terms of stakeholder concern. Security characteristic rankings are chosen using the researcher as a surrogate stakeholder. This represents a threat to the construct validity of this study because the researcher may make ranking selections that are not reflective of the stakeholders who would utilize the binary. We investigate the impact of stakeholder values in chapter 7.

Design

The model design phase is the phase in which we construct and populate the tree structure of PIQUE-Bin. This means identifying security aspects and product factors that will be included in the model, as well as populating and weighting the edges of the model from the stakeholder ranking gathered in the requirements phase.

Security aspects are chosen with stakeholder interests in mind. The concept of security aspects aligns well with the CIA triad. The classical CIA triad includes confidentiality, integrity, and availability. This paradigm is commonly critiqued for being too simplistic in its view of security aspects, primarily because it lacks the security requirements for accountability and responsibility. We chose a form of the extended CIA triad based on Microsoft's STRIDE threat model[3]. Microsoft's STRIDE model is a threat model for classifying threats. STRIDE is an acronym of Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Escalation of privileges. These are all threats that impact the target system in different ways, corresponding to different desired security properties that would be negatively affected. These desired properties are authenticity, integrity, non-repudiation, confidentiality, availability, and authorization respectively. This

[3]https://www.microsoft.com/security/blog/2007/09/11/stride-chart/

may be seen in Table 5.1. We chose these properties to be the security aspects for PIQUE-Bin to give a more sophisticated view of security aspects than the CIA triad, while still utilizing a well-established, mature paradigm.

Threat	Desired Security Property
Spoofing	Authenticity
Tampering	Integrity
Repudiation	Non-repudiation
Information disclosure	Confidentiality
Denial of service	Availability
Escalation of privileges	Authorization

Table 5.1: STRIDE Threats and Desired Security Properties

With STRIDE guiding the choice of quality aspects, we now look to the product factors. Product factors are measurable properties of the binary that can be tied to the security aspects. We chose to utilize the CWE-699 view[4] for this layer of the model. CWE-699 is a view of the CWE catalog based on software development. It breaks the catalog into 40 categories of weaknesses, which then have base level weaknesses under them. The product factors in PIQUE-Bin are the category-level CWEs found in CWE-699. As an example of a CWE category found in CWE-699, see CWE-1210 and the CWEs classified under CWE-1210 in Figure 5.1. We chose to use the CWE because it allows us to tie the findings of different tools to the CWE that they are instances of while preserving the impact of a finding to the security aspects that we are interested in. In addition, the information for each CWE provided by MITRE guides the stakeholder in how each CWE might impact the quality aspects. We choose to limit the set of CWEs because the whole CWE catalog is over 1,000

[4]https://cwe.mitre.org/data/definitions/699.html

weaknesses, which would make the model cumbersome and overly complicated, with no clear way to aggregate through these weaknesses. The amount of comparisons required for the stakeholder to weight the model using the AHP is the number of product factors squared, so reducing the number of nodes greatly helps the ease of using the model. Additionally, we add one 'unknown/other' node which encompasses CWEs not included in CWE-699 as well as findings for which the CWE is unknown. This would be the case for CVEs that have not been mapped to a CWE. Finally, we have one more product factor that is 'potential malicious indicators', which is added because CWE-699 lacks anything that would capture these findings.

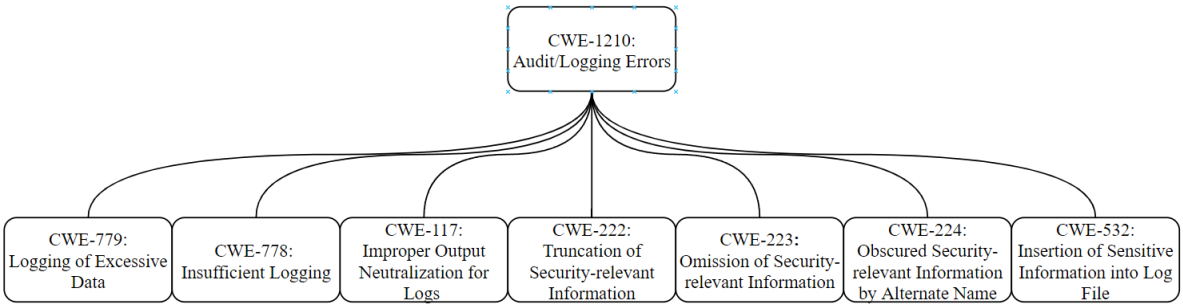

Figure 5.1: An Exemplary CWE Category From CWE-699

After defining the structure of the top two layers, we must now weight the edges from product factors to security aspects and from the security aspects to the overall security score. For the pairwise comparisons of aspects and weighting of the aspects to the overall score, Table 5.2. We approached this as if it were an ICS environment where availability is of utmost importance, followed by authorization/authenticity and integrity. For the product factor layer, we manually weighted the connections to the most likely impacts. We chose this approach due to the large number of comparisons that would be required for pairwise

comparisons between product factors for each quality aspect. We followed the information provided in the 'technical impact' section of the CWEs as well as the description of the CWE categories. When multiple security aspects were likely to be impacted, we allowed the CWE category to impact multiple security aspects. This weighting can be seen in Table 5.3. The final weights for each quality aspect are calculated as the value associated with each product factor divided by the sum of values, giving a set of weights that sum to one.

Table 5.2: Comparison Matrix And Final Weights for Quality Aspects

Criteria	Availability	Authenticity	Authorization	Confidentiality	Non-repudiation	Integrity	Final Weight
Availability	1	3	3	8	8	5	0.455
Authenticity	0.333	1	1	4	4	2	0.179
Authorization	0.333	1	1	4	4	2	0.179
Confidentiality	0.125	0.25	0.25	1	1	0.5	0.048
Non-repudiation	0.125	0.25	0.25	1	1	0.5	0.048
Integrity	0.2	0.5	0.5	2	2	1	0.092

Development

The development phase of PIQUE-Bin is the stage at which we derive the model and calibrate. Model calibration consists of applying the chosen tools to the set of benchmark binaries to create utility functions. After this stage the model is ready to be evaluated on a binary to assess security quality.

This process is guided by multiple exploration-motivated applications of the model, as detailed in chapter 6. These iterative applications of PIQUE-Bin to binaries have lead us to changes we detail in the following subsections.

Table 5.3: Weighting of Product Factors to Quality Aspects

Product Factor (CWE Number)	Availability	Authenticity	Authorization	Confidentiality	Non-repudiation	Integrity
API / Function Errors - (1228)	1	1	1	1	1	1
Audit / Logging - (1210)	0	0	0	0	1	0
Authentication Errors - (1211)	0	1	0	0	0	0
Authorization Errors - (1212)	0	0	1	0	0	0
Bad Coding Practices - (1006)	1	1	1	1	1	1
Behavioral Problems - (438)	1	0	0	0	0	0
Business Logic Errors - (840)	1	1	1	1	1	1
Communication Channel Errors - (417)	0	1	1	1	0	0
Complexity Issues - (1226)	1	1	1	1	1	1
Concurrency Issues - (557)	1	1	1	1	1	1
Credentials Management Errors - (255)	0	1	0	0	0	0
Cryptographic Issues - (310)	0	1	0	1	0	0
Key Management Errors - (320)	0	1	0	0	0	0
Data Integrity Issues - (1214)	0	0	0	0	0	1
Data Processing Errors - (19)	1	1	1	1	1	1
Data Neutralization Issues - (137)	1	1	1	1	1	1
Documentation Issues - (1225)	1	1	1	1	1	1
File Handling Issues - (1219)	1	1	1	1	1	1
Encapsulation Issues - (1227)	1	1	1	1	1	1
Error Conditions, Return Values, Status Codes - (389)	1	0	0	1	0	0
Expression Issues - (569)	1	1	1	1	1	1
Handler Errors - (429)	1	1	1	1	1	1
Information Management Errors - (199)	0	0	0	1	0	0
Initialization and Cleanup Errors - (452)	1	1	1	1	1	1
Data Validation Issues - (1215)	1	1	1	1	1	1
Lockout Mechanism Errors - (1216)	1	0	0	0	0	0
Memory Buffer Errors - (1218)	1	1	1	1	1	1
Numeric Errors - (189)	1	1	1	1	1	1
Permission Issues - (275)	0	0	1	0	0	0
Pointer Issues - (465)	1	1	1	1	1	1
Privilege Issues - (265)	0	0	1	0	0	0
Random Number Issues - (1213)	1	1	1	1	1	1
Resource Locking Problems - (411)	1	1	1	1	1	1
Resource Management Errors - (399)	1	1	1	1	1	1
Signal Errors - (387)	1	1	1	1	1	1
State Issues - (371)	1	1	1	1	1	1
String Errors - (133)	1	1	1	1	1	1
Type Errors - (136)	1	0	0	0	0	0
User Interface Security Issues - (355)	0	0	0	1	0	0
User Session Errors - (1217)	0	1	1	1	0	0
Potential Malicious Indicators	1	1	1	1	1	1
Unknown-Other	1	1	1	1	1	1

Utility Function

By default, PIQUE utilizes linear interpolation between two threshold values, a and b. It limits the value to the range $[0, 1]$ via a piece-wise function. This function is defined as

$$f(x) = \begin{cases} 0 & x < a \\ 1 & x > b \\ \frac{x-a}{b-a} & a < x < b. \end{cases} \tag{5.1}$$

This is a standard utility function in the context of quality modeling. This utility function, however, makes one critical assumption which we must re-assess now that we are primarily focusing on a security context.

In models that follow the Quamoco/QATCH/PIQUE paradigm, the output of a utility function is limited to $[0, 1]$ because there is some a minimum utility and maximum utility value, where utility is defined as a value that "quantifies the relative satisfaction of a decision maker concerning the quality of a software product characterised by specific measurable factors"[53]. However, in the context of security measures and factors, we argue that there is no true maximum utility at the measure or factor level because enough vulnerabilities of the same type (categorized under the same measure/factor) can completely compromise some aspect of security.

We argue that this is because there is never a point where an additional vulnerability should not increase the output of the utility function. To see why this would be an issue, consider a case where we have a measure whose utility function is evaluating to 1 (the maximum utility score). Then, consider injecting a new vulnerability that would be categorized under the same measure and then re-analyzing it. The value of the utility function cannot increase and therefore we see no change in the output of the model. A change is required to the utility functions of the model in order to allow additional vulnerabilities to always change the utility score.

This phenomenon is seen in the analysis done on Wireshark[5] in section 6. To remedy

[5]https://www.wireshark.org/

this issue, we move the bound to $[0, 1]$ from the utility function to the quality aspect evaluation function. In this way, we enable a single measure to cause a quality aspect to drop all the way to 0. Additionally, this change implies that the quality aspects have some maximum or minimum level of utility, which we claim to be true. Complete compromise of a security aspect such as availability is possible, for example if a trivially exploited vulnerability causes an application to hang indefinitely, availability could be considered fully compromised. However, this should not compromise the total score because availability has a set amount of weight within the TQI and we seek to preserve that weight to allow the stakeholder's values to remain.

Therefore, PIQUE-Bin makes one change to this functionality, which is to remove the piece-wise component of the utility function. The utility function in PIQUE-Bin becomes

$$f(x) = \frac{x - a}{b - a}. \tag{5.2}$$

To see what this change in the utility function looks like in a model, consider Figures 5.2 and 5.3. In Figure 5.2 we see two model evaluations utilizing the default bounded utility function. We see that in Figure 5.2a, with a severe finding, the model evaluates to a score of 0.875 and with just a few minor findings as seen in Figure 5.2b, we see a score of 0.8125 - a worse score. We would like to see the high severity finding be more influential in the model. Now consider 5.3, the same analysis done using the unbounded utility function and bounded QA evaluation. We see the desired effect in 5.3a, allowing the high severity impact to push a QA value to 0, therefore impacting the overall score greatly. Additionally, this change does not impact analysis when no measure would exceed the thresholds, which may be observed in Figures 5.2b and 5.3b not changing between the two methods.

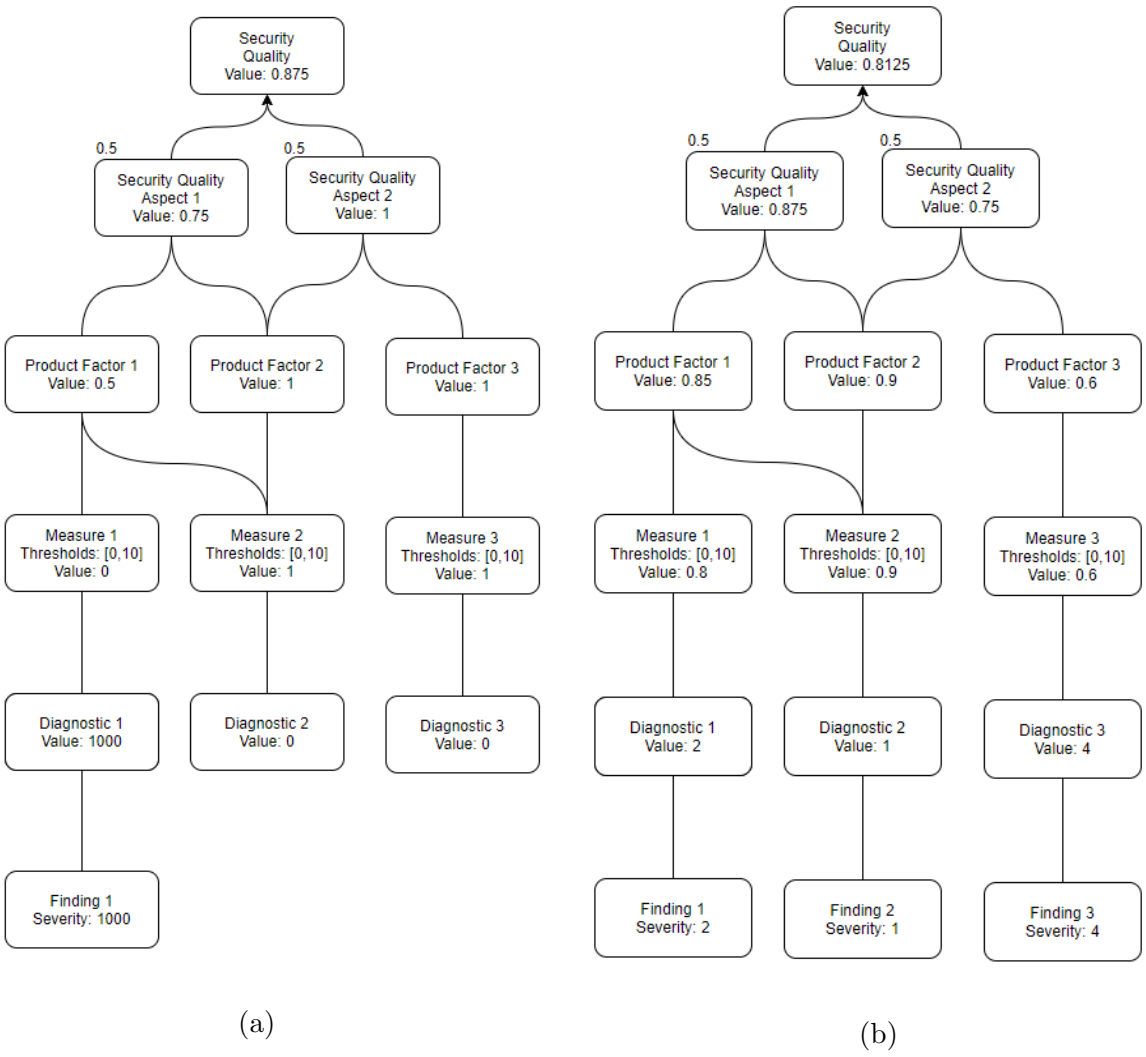

Figure 5.2: Two simple model evaluations using the default PIQUE utility function

Threshold Calculation

By default, PIQUE uses the maximum and minimum value found in the benchmark repository for the threshold values. For PIQUE-Bin, we change this calculation method for several reasons. First of all, we want to compare the binary under analysis to the benchmark repository as a whole, rather than just two of the binaries in the repository which have the worst and best values for a given measure. Additionally, we find that these thresholds are

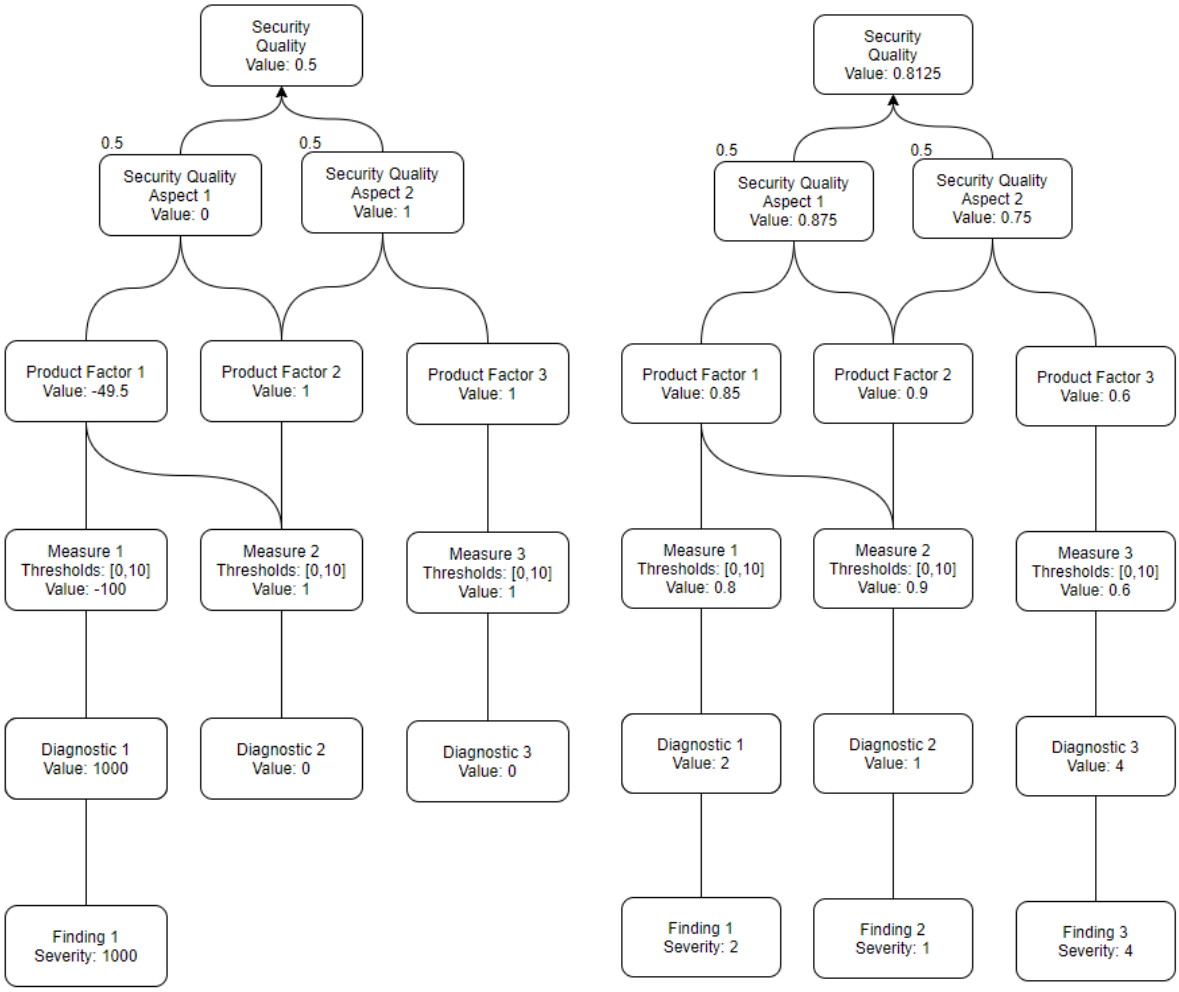

Figure 5.3: Two simple model evaluations using an unbounded utility function with bound QA values

almost never influenced by adding more projects to the repository unless it is an exceptionally poorly developed project.

We change the thresholds to be calculated as the mean plus and minus the standard deviation of the values in the benchmark repository. This is similar to using the interquartile range as seen in QATCH and Quamoco models; however, the interquartile range does not work well for benchmarking measures whose findings are rarely found. If greater than 75

percent of the measure values are 0, then we get $[0, 0]$ for our thresholds which is not ideal for obvious reasons. Therefore we use the mean and standard deviation to calculate the thresholds and limit the lower threshold to be non-negative. We limit the lower threshold to be non-negative because when analyzing a binary for vulnerabilities, a measure with no findings should evaluate to the best possible value which would not occur if the lower threshold was negative.

Tools

For PIQUE-Bin, we implement three binary static analysis tools. These tools are cwe_checker, cve-bin-tool, and Yara using the Yara-Rules Github repository. Of course, additional tools will improve the model, but the time to implement the tools and integrate the tools into the model structure is a limiting factor for the number of tools. More tools also makes the benchmarking and analysis process take longer.

These tools each serve a different purpose in the model, supplying a different source of information with each finding. The use of multiple tools highlights our ability to incorporate many different type of security tools within a model.

CVE-Bin-Tool The first tool in the model is cve-bin-tool[6]. This tool was chosen because it is under active development, popular, and it can identify many CVEs that may appear in a binary due to third party library usage.

The tool works by searching a binary for patterns known to be associated with third party libraries. The tool then cross-references the library with the NVD to identify known vulnerabilities. As an example, here are the patterns associated with Wireshark:

```
CONTAINS_PATTERNS = [
    r"'usermod -a -G wireshark _your_username_' as root.",
```

[6]https://github.com/intel/cve-bin-tool

```
    r"Are you a member of the 'wireshark' group\? Try running",
]
FILENAME_PATTERNS = [r"rawshark", r"wireshark"]
VERSION_PATTERNS = [r"Wireshark ([0-9]+\.[0-9]+\.[0-9]+)"]
VENDOR_PRODUCT = [("wireshark", "wireshark")]
```

The tool is defeated rather easily by obfuscation, so more sophisticated analysis methods may help identify additional CVEs. The CVEs findings are then classified under a CWE according to the National Vulnerability Database. The severity for a finding is taken from the CVSS2 that NVD also provides.

One final advantage of this tool is that because it is simple pattern scanning, it should work on any architecture or form of binary, making this an ideal tool for ICS settings.

A major drawback to cve-bin-tool is that the applications of it are limited. Only a certain set of third party libraries are identified and we cannot guarantee that the CVEs in a third party library will also be present within the binary that utilizes the library. We also do not know the prevalence of third party libraries with OT environments, which may further limit how useful this tool will be in such environments.

cwe_Checker cwe_checker is a suite of checks for CWE instances in a binary[7]. It currently has the capability to search for 19 different CWEs and is under active development. It can be run on multiple popular architectures, which makes it a good tool to consider in environments where multiple architectures may be in use across a system, such as in an ICS. Each finding from this tool is classified under the specific CWE it is an instance of, and each finding receives the same severity of 1.

The tool works by using Ghidra, a disassembler built by the National Security Agency (NSA), to disassemble a binary. After disassembly, the tool searches through an intermediate

[7]https://github.com/fkie-cad/cwe_checker

representation of the binary code for patterns associated with known vulnerabilities. As stated in an earlier section, cwe_checker searches for CWE190 using the following method:

> For each call to a function from the CWE190 symbol list we check whether the basic block directly before the call contains a multiplication instruction. If one is found, the call gets flagged as a CWE hit, as there is no overflow check corresponding to the multiplication before the call. The default CWE190 symbol list contains the memory allocation functions *malloc*, *xmalloc*, *calloc* and *realloc*.

cwe_checker is prone to false positives, but will find vulnerabilities that are unknown or new. As such, this tool is a valuable tool for identifying vulnerabilities inserted throughout the development process.

Additionally, Ghidra is able to disassemble many different architectures, with the number of architectures that can be disassembled actively increasing. Also, the tool is open source, so if an important architecture is not available, it may be developed and used to allow Ghidra for that architecture.

Yara-Rules The final tool is the Yara-Rules repository[8], which is a collection of rules for the YARA (Yet Another Ridiculous Acronym) tool. The tool allows the definition of patterns and logic surrounding the patterns to search for in a binary or text file. The Yara-Rules repository is a set of rules put together by a group of IT security researchers. The rules are classified under anti-debug/anti-VM, capabilities, cryptography, exploit kits, malicious documents, malware, packers, and malicious emails. All these rules have their own diagnostic and measure, with each broken rule becoming a finding with a severity of 1. These measures are aggregate into the 'Potential Malicious Indicators' product factor.

[8]https://github.com/Yara-Rules/rules

This tool will allow for identification of significant changes in the capabilities of binaries, as well as identification of malicious indicators within a binary. This tool is also able to run on any architecture or form of binary, which is ideal for utilizing it in ICS environments.

One example of a finding, we define the rule Check_DLLs, a anti-debug/anti-VM rule. This rule defines several strings, then applies a rule surrounding these strings:

```
rule Check_Dlls
{
meta:
Author = "Nick Hoffman"
Description = "Checks for common sandbox dlls"
Sample = "de1af0e97e94859d372be7fcf3a5daa5"
strings:
$dll1 = "sbiedll.dll" wide nocase ascii fullword
$dll2 = "dbghelp.dll" wide nocase ascii fullword
$dll3 = "api_log.dll" wide nocase ascii fullword
$dll4 = "dir_watch.dll" wide nocase ascii fullword
$dll5 = "pstorec.dll" wide nocase ascii fullword
$dll6 = "vmcheck.dll" wide nocase ascii fullword
$dll7 = "wpespy.dll" wide nocase ascii fullword
condition:
2 of them
}
```

If any two of the defined strings are present, then DLLs commonly associated with anti-debug/anti-VM are being used in a binary and that is cause for concern.

Changes to PIQUE

For the most part, PIQUE as a platform provides all the functionality to build a standard operationalized quality model while enabling the implementation of different behavior. In the context of security analysis, it was found that there are several changes required. We have already covered several changes and the respective reasoning: the utility function, threshold calculation, and manual weighting for the PF to QA level of the model.

One additional change we have made to PIQUE concerns the thresholds for measures that had no findings in the benchmark repository. These measures get a threshold value of $[0, 0]$. Ideally, all thresholds would have some value aside from $[0, 0]$. This is achieved when every diagnostic has at least one finding among the entire benchmark repository. However, when dealing with models that have a lot of diagnostics for which findings are rare, achieving that goal may not be realistic. It is not desirable to force every finding to be a part of the benchmark repository. In the case of $[0, 0]$ thresholds then, we really do not know how the binary under analysis compares to the benchmark repository, so it wouldn't make sense to say it is better or worse.

By default for negative measures, PIQUE assigns a value of 1 to a measure that has no finding in the benchmark repository and no finding in the binary under analysis. This is stating that although there were no findings, PIQUE claims that the binary under analysis is better than the benchmark repository with respect to that measure. Although some value must be assigned, the assignment of 1 for a measure where we have no information and no context may mislead interpretation by giving a score for a binary that indicates it is more secure than it is, simply due to the sparsity of tool findings.

Therefore, for any measure with $[0, 0]$ thresholds and no findings for the binary under analysis, we assign a value of 0.5. This essentially means that we do no have enough information for a conclusion so we assume that the binary is equivalent in quality to the benchmark repository. One interesting effect of this is that the more nodes with no findings

and $[0, 0]$ threshold values, the lower the maximum possible score of the binary will be. With this change, we can no longer achieve a score of 1 if we have any $[0, 0]$ thresholds.

There is still discussion on the impact of this decision and what default behavior is preferable. Either method has its benefits, but one choice must be made. We choose a default value of 0.5 because it tends to give a lower score for binaries, which is preferable for a security metric. In the face of uncertainty, we should not make the score higher for something that may be used in security-critical decisions.

Two other options are being actively considered and discussed. The first option is to not include the measures that receive $[0, 0]$ thresholds, and to raise some warning flag when a finding that has $[0, 0]$ thresholds occurs in a binary, but leave the score unchanged. The other option is that we could create theoretical thresholds that would simulate adding a single binary to the repository that has the finding. For example, consider a repository that has 700 binaries. If we have some measure with $[0, 0]$ thresholds but has a finding in the binary under analysis, we would assume that a single binary among the benchmark repository had that finding. Using the mean plus and minus standard deviation (with a lower limit of 0), we would get thresholds of $[0, 0.04]$. A single finding (with a severity of 1) of this measure would take on a value of -24.5, giving a high impact.

EXPLORATORY CASE STUDIES

In this section, we apply PIQUE-Bin to several exemplary binaries to discover how it may be improved. These case studies lead us to make many of the design decisions that were described in chapter 5.

Application to Wireshark Binaries

In this case study, we apply an early version of PIQUE-Bin to two Wireshark binaries. The purpose of our study is to propose design changes to create an improved model with which further evaluations will be done. This version of PIQUE-Bin uses PIQUE default behaviors. It bounds the utility function to $[0, 1]$ and uses threshold calculations of $[min, max]$. The only tool in this version of the model is cve-bin-tool.

The binaries under analysis are Wireshark 1.8.1 and Wireshark 3.0.0. Wireshark[1] is a popular network protocol analysis tool that allows users to observe what is happening on a network with a great level of detail.

Wireshark was chosen as a case study due to the high number of CVEs associated with it. Wireshark is known to be very vulnerable and can be a blacklisted application in high security environments due to enabling privilege escalation attacks on systems. Wireshark 1.8.1 has 109 CVEs contained within it[2], making it a good case study for a vulnerable binary. We analyze Wireshark 3.0.0 as well, which has 24 vulnerabilities[3], making it a good comparison. We expect to see version 1.8.1 get a low score (less than 0.2), while version 3.0.0 should get a significantly higher score.

In addition to applying the model to the binaries, we also apply the model to the binaries with manually set thresholds: rather than threshold values of $[min, max]$, we use

[1] https://www.wireshark.org/

[2] As of 06/2021, according to NVD using cpe:2.3:a:wireshark:wireshark:1.8.1:*:*:*:*:*:*:*

[3] As of 06/2021, according to NVD using cpe:2.3:a:wireshark:wireshark:3.0.0:*:*:*:*:*:*:*

$[0, 10]$ for all thresholds. The purpose of doing this is to get some idea of the impact of the thresholds (and therefore the benchmark repository composition) on the overall score. We expect that the scores of the binaries will be impacted similarly and will maintain their ordering.

Results

The score for the derived PIQUE-Bin model applied to Wireshark versions 1.8.1 and 3.0.0 with and without manually set thresholds can be seen in Table 6.1. As expected, the score of version 3.0.0 is greater than that of version 1.8.1 in the derived threshold case, but in the manual threshold case we see that 3.0.0 receives a worse overall score.

Table 6.1: Wireshark Model Application Results

Threshold Type	Derived Thresholds		Manual Thresholds	
Wireshark Version	1.8.1	3.0.0	1.8.1	3.0.0
Total Score	0.7962	0.8004	0.7690	0.6209
Availability	0.7884	0.7662	0.7550	0.6200
Authenticity	0.8072	0.8318	0.7414	0.5756
Authorization	0.8287	0.8559	0.8108	0.6270
Confidentiality	0.7443	0.8502	0.6720	0.5920
Non-repudiation	0.8079	0.8384	0.7878	0.5818
Integrity	0.7896	0.7973	0.8571	0.7183

Discussion

The application of the model has revealed several flaws that must be addressed before the model will meet expectations. The model failed to give a score in the expected range

(<0.2 for Wireshark 1.8.1), and in the manual threshold case Wireshark 3.0.0 scored lower than 1.8.1.

The issues noted above are caused by several phenomena. Primarily, the reason is the grouping of CVEs under a single measure in combination with the lack of the ability for a single measure to greatly impact the model. Wireshark 3.0.0 scores higher than 1.8.1 when using derived thresholds, but lower than 1.8.1 when using manual thresholds. The most likely reason for this is the interaction between threshold derivation and CVEs grouped under a single CWE. Wireshark 3.0.0 has fewer CVEs but the CVEs are more spread out among CWEs, causing less information loss and allowing more CVEs to impact the overall score in comparison to 1.8.1. This reversal of the ordering of scores due to thresholds shows a need for further investigation of the benchmarking process and utility function impact.

In this analysis, one specific issue is the number of CVEs that are not assigned a CWE. This causes a large amount of findings to be classified under the 'unknown-other' category of weaknesses. As a result, this node receives a very large value evaluates to 0. This leads to many of the CVEs for Wireshark 1.8.1 not impacting the score because the measure node associated with the 'unknown-other' CVEs reaches the maximum utility value with only a small set of the total vulnerabilities. The thresholds of the 'unknown-other' node are $[0, 96.33]$ and the value for Wireshark 1.8.1 is 371, meaning that it would evaluate to -2.85 if not for the minimum value of 0. Essentially, two thirds of the findings have no impact on the TQI due to this limit. This significantly reduces the potential impact of this node which contains 62 CVE findings.

Threats to Validity

We will not be addressing internal or conclusion threats, because we are not asserting any causal relationships nor are we concluding anything about our model generally in this case study.

External threats include threats to our ability to use this same model on other binaries with other vulnerabilities. The model is highly stakeholder, tool, and benchmark dependent. Our choice of binaries for benchmarking reflects the type of binary we expect to apply the model to - it is possible that this model will not be applicable to binaries dissimilar to the ones we use to benchmark. Additionally, it is certain that there are security findings that are not identified by the tool in the model, which means the model's output is not considering all security threats within a binary. These problems are remedied by adapting the model for different scenarios, as this is the advantage of using PIQUE to build the model.

The primary threat to validity of this model is construct validity. There are many threats that must be addressed as the model is improved. Many of these threats were touched on in the discussion section.

There is some risk that we lose too much relevant information through generalization - a CVE provides more information about the impact of the flaw than the CWE that the CVE is categorized under. In addition, the impact of the CVE may be different than that of the CWE it is categorized as. For instance, take some vulnerability that allows anyone to read the username and password of an admin account. Although this is clearly a threat to authorization, we could see the CVE be categorized as a CWE that is related to confidentiality because the weakness itself is an exposure of information. There also may be too many CVEs that are either categorized improperly or not at all. One potential way to mitigate these issues is through the use of machine learning to categorize CVEs under CWEs as in [1].

The benchmarking process needs to be investigated further to determine that the benchmark binaries are appropriate.

Additionally, it may be that we are not adequately measuring security quality by only observing security flaws. Perhaps to obtain a holistic picture, security measures that have been implemented properly need to be taken into consideration. This is something that

should be considered as a possibility when this model is being assessed to determine efficacy, but is outside the scope of this work.

Finally, security flaws that have not been documented as a CVE exist, and the current model does not account for these. Additional tools should be implemented in the model to identify non-CVE security findings. This will be done later by utilizing cwe_Checker and YARA.

Application to Busybox Binaries

For this case study, we investigate the use of PIQUE-Bin to analyze several versions of busybox binaries. Busybox is a set of system utility functions for Linux systems. It was chosen at the request of our research contractors, the Idaho National Laboratory. This version of PIQUE-Bin utilizes all the latest developments in PIQUE inspired by initial applications to Wireshark binaries.

Specifically, the changes made in PIQUE-Bin from the Wireshark case study are:

- Removed limit to $[0, 1]$ for utility functions
- Added limit to $[0, 1]$ for QA evaluation
- Thresholds are calculated using the mean plus/minus standard deviation
- Added two tools, cwe_checker and yara
- Changed default evaluation of measures that receive $[0, 0]$ thresholds and no findings to evaluate to 0.5 rather than 1

These changes were made due to the shortcomings of the model identified in the analysis of Wireshark.

The goal of this case study is to observe a scenario in which we are able to apply PIQUE-Bin to the same binary across multiple versions. This will give insight on how much

a binary is expected to change over time, as well as confirm that the model is working by showing a score that increases over time. We expect to see an increase in score because this binary has patched known vulnerabilities over time. This is as close as we can get to a proxy for the expected relative security - because there are fewer known vulnerabilities in more recent versions of the binary, we expect recent versions to have a better score. If this does not prove to be the case according to the output of PIQUE-Bin, we should investigate and be sure that our findings are accurate. If we do not see a score that increases over time, then we should attribute this to a specific cause/finding and confirm that the decrease in TQI is reasonable.

Results

The results of the application of PIQUE-Bin to the busybox binaries may be seen in Figure 6.1 and Table 6.2. The values are increasing as we analyze newer versions, indicating (as expected) that the binary has increased in security quality over its development history. The only few points where we do not see a change in TQI is between versions 1.29.0, 1.29.1, 1.29.2, and 1.29.3. This is likely due to the fact that these are all small, consecutive patches to busybox, fixing bugs that are not reported as vulnerabilities. Therefore the differences between the versions of 1.29.x are minimal. This is confirmed by the patch notes of busybox[4]:

```
    9 September 2018 -- BusyBox 1.29.3 (stable)
Bug fix release. 1.29.3 has a fix in libbb for xmalloc_fgets().


    31 July 2018 -- BusyBox 1.29.2 (stable)
Bug fix release. 1.29.2 has fixes for fdisk (compat fixes, allow 2TB+
sizes), gzip (FEATURE_GZIP_LEVELS was producing badly-compressed .gz),
hexedit (segfault fix).
```

[4]https://www.busybox.net/

```
    15 July 2018 -- BusyBox 1.29.1 (stable)
Bug fix release. 1.29.1 has fixes for wget (http->https redirect) and
sendmail (angle bracket parsing).
```

The patch notes do not mention any known vulnerabilities in the form of CVEs, nor any CWE categories that would be detected by our tools.

Overall, these results confirm that the busybox binaries have improved security quality throughout their patch history with respect to the benchmark repository according to PIQUE-Bin's output. From June 2013 to February 2019, we see a change of 0.25.

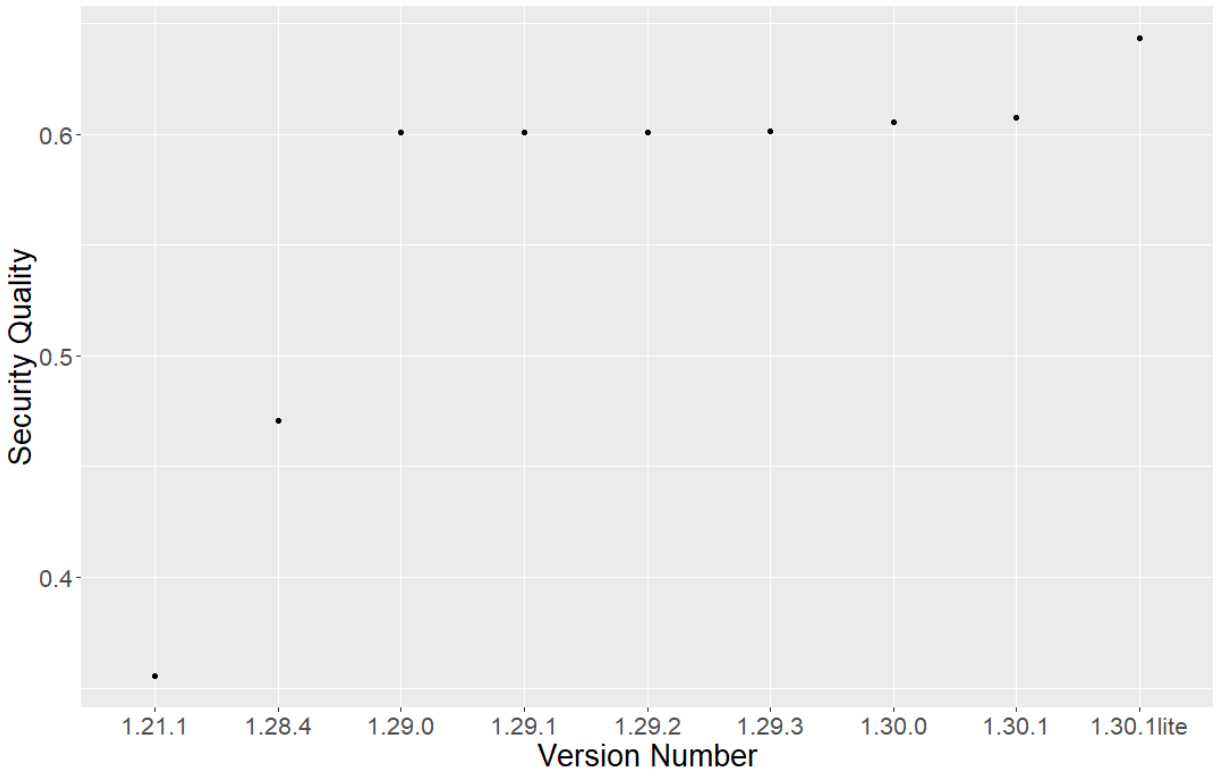

Figure 6.1: TQI Values for Busybox Versions

Table 6.2: Busybox Model Application Results

BusyBox Version	Total Score	Availability	Authenticity	Authorization	Confidentiality	Non-repudiation	Integrity
1.21.1	0.355	0.357	0.351	0.367	0.375	0.330	0.330
1.28.4	0.470	0.476	0.456	0.476	0.480	0.460	0.460
1.29.0	0.601	0.610	0.574	0.599	0.599	0.607	0.607
1.29.1	0.601	0.610	0.574	0.599	0.599	0.607	0.607
1.29.2	0.601	0.610	0.574	0.599	0.599	0.607	0.607
1.29.3	0.601	0.610	0.574	0.599	0.599	0.607	0.607
1.30.0	0.605	0.614	0.578	0.603	0.602	0.611	0.611
1.30.1	0.607	0.617	0.580	0.606	0.604	0.614	0.614
1.30.1 lite	0.643	0.653	0.613	0.639	0.637	0.654	0.654

Discussion

The changes made to the PIQUE-Bin model after the application to Wireshark binaries appear to have improved our ability to assess the security quality of binaries by both changing how we create our utility functions and allowing measure output to be unbounded, allowing for higher impact.

Additionally, we see in this case study what we hope to see: an increasing score over time. We expect this because the older versions of busybox have more known vulnerabilities, indicating that they are less secure. As we mentioned, if the PIQUE-Bin analysis deviated from this trend, we would have reason for concern, as it goes against expectations.

This case study shows that PIQUE-Bin is able to successfully analyze a binary as it is updated and patched and is able to reflect changes in security quality through the removal of vulnerabilities.

One potential cause for concern is the lack of change in score for the versions of busybox that are 1.29.x. These versions all receive the same score because they have small bug fixes that are not detected by the tools currently used by PIQUE-Bin. Therefore, this case shows that in order to improve PIQUE-Bin we should add additional tools that would identify the

minor bugs fixed in the 1.29.x versions.

Threats to Validity

The threats to validity are similar to the threats to validity noted in the application to the Wireshark binaries.

Additional threats to validity include a threat to the construct validity of the study. We assume that the true security score of the binaries we are analyzing is improving, but that is not necessarily the case. One may argue that an unknown or undisclosed vulnerability could be present in newer versions but has not been discovered, detected by tools, or disclosed. Therefore, we may be missing some significant findings.

MODEL VALIDATION

We have designed, developed, and applied the PIQUE-Bin model, but we still are unsure of what the output of PIQUE-Bin indicates and how it should be interpreted. We must build trust in the process that PIQUE-Bin applies to assess a binary and build understanding of what the output means.

To this end we first analyze relationships between the attributes of binaries and tool output. This will help build trust in the model by ensuring that we are comparing binaries to the appropriate population of binaries to create a score that may be relied upon as a security assessment metric. To better show why this analysis is important, consider the following analogy. A 20 year old person goes to his/her doctor. The doctor compares the lung capacity of the 20 year old to a population of 90 year old individuals and reaches the conclusion that the younger individual's lungs are average. Of course, a 20 year old should have much better lung capacity than a 90 year old, so this comparison leads the doctor to an incorrect conclusion. We want to avoid this scenario by ensuring that we are not comparing the equivalent of a '20 year old' binary to a population of '90 year old' binaries, or vice versa.

We also investigate the impact of individual vulnerabilities being injected into the binary under analysis. This will allow us to isolate the change in TQI for a finding classified under each diagnostic. This will give much better context for analyzing changes in scores. Currently we do not have any idea whether a change of 0.05 is a large, noteworthy change in PIQUE-Bin's output, or if it is not significant.

Tool Output Sensitivity To Binary Attributes

The benchmark repository can have a large impact on the output of any PIQUE model which makes sensitivity analysis important. We conduct sensitivity analysis by looking into what attributes of the gathered binaries correlate with the number of findings they produce.

This analysis will allow us to be confident that our overall score is due to differences in security between the benchmark repository and binary under analysis rather than due to differences in other attributes such as size. These attributes may cause the output to be artificially high or low, which may mislead stakeholders when interpreting the score produced by PIQUE-Bin.

To gather data, we use the tool Detect-It-Easy[1] to identify size, compiler, and whether the binary was statically or dynamically linked. We will also manually categorize the binaries as either 'System' or 'Network' focused binaries as a proxy for a more specific categorization of domain or purpose. We will then apply the three tools in the model, CVE-Bin-Tool, cwe_Checker, and Yara Rules, to the binaries and record the count of findings for each tool. Once we have collected this data, we fit a model to the data for each tool to identify what attributes of the binary are correlated with each tool's output.

Hypotheses

Before we begin, we define the research question and hypotheses that we are seeking to answer.

Our research question is: What attributes of a binary correlate with changes in the number of findings from a tool?

To answer this question, our null hypotheses are:

- $H1_0$: For each tool, there is no correlation between the size of the binary under analysis and number of findings
- $H2_0$: For each tool, there is no correlation between the method of linking (static vs dynamic) and number of findings

[1] https://github.com/horsicq/Detect-It-Easy

- $H3_0$: For each tool, there is no correlation between the compiler used and number of findings

- $H4_0$: For each tool, there is no correlation between the domain (system vs network) and number of findings

Finding a significant correlation ($p < 0.05$) between output from one of the tools and an attribute will lead us to reject the null hypothesis for that attribute. For any rejected null hypothesis, we must be sure to consider that attribute's presence in the benchmark repository, and the potential impact on the output of PIQUE-Bin.

Data Exploration

To begin data exploration, we explore the distributions of our factors (size in bytes (continuous), static linking (categorical), compiler (categorical), and domain (categorical)), which can be seen in Figure 7.1. There is a clear imbalance of classes for the categorical variables which could be important to consider as we begin the data analysis phase. Additionally, the compiler variable contains an 'unknown' category which contains all the binaries for which the compiler could not be determined. Because this category could contain binaries that should fall under a different category, we must consider this a threat to validity and be wary as we interpret results and make conclusions.

We also find that our compiler factor contains perfect separation with some of the tool output, causing issues with fitting a model. Perfect separation occurs when the output of 0s and 1s is perfectly separated by a certain predictor, which causes issues when performing logistic regression. In the case of our data, the 'gcc(Alpine)' compiler received cve-bin-tool findings for all of the binaries that it compiled. To alleviate this issue we combine the compiler variable in either "GCC" or "Unknown".

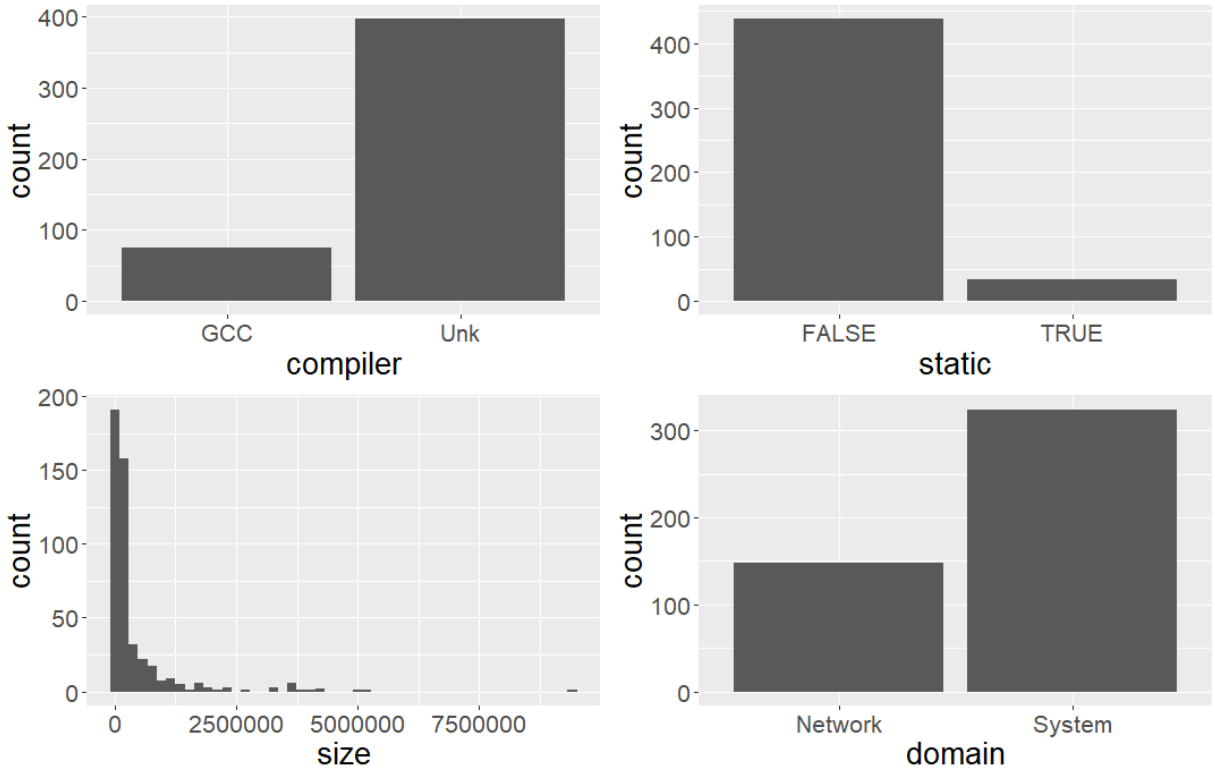

Figure 7.1: Distributions of Factors

Additionally, we should consider any collinearity within our factors. If collinearity does exist, we must consider this as we choose an appropriate model to consider the effects of each factor on the tool output. After initial investigation, size appears to be the only factor that has collinearity with the other factors. To see the comparison of size and the other factors, as well as the p-value given by Kruskal-Wallis tests, see Figure 7.2. Static compiling and the compiler used both appear to have some association with size, which is shown by the p-value being less than 0.05 for the non-parametric ANOVA alternative, the Kruskal-Wallis test.

This finding indicates that whatever model is fit in the analysis phase should account for size when investigating the effect of other factors. Now we investigate each tool's output compared to the binaries' factors.

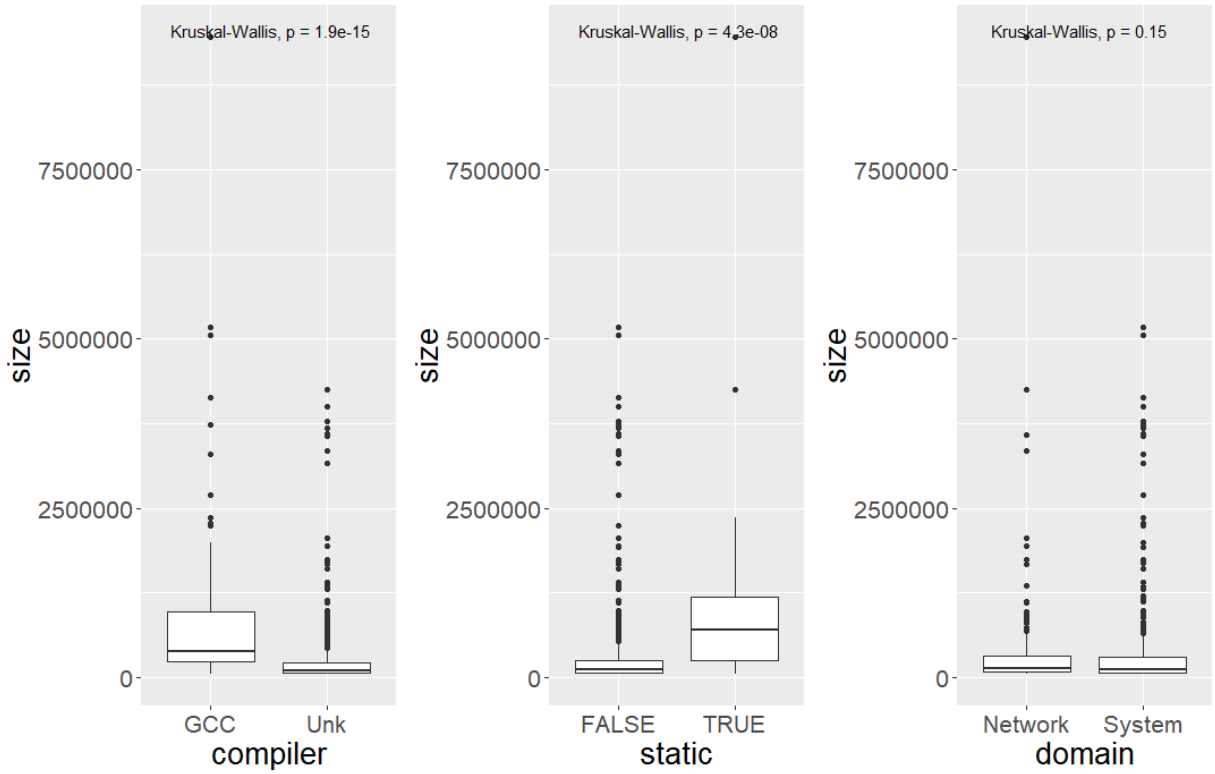

Figure 7.2: Size Compared to Other Factors

We also investigate the distribution of the tools' output. This may be seen in Figure 7.3. Apart from cwe_checker, our tools are outputting predominantly 0 findings for each binary. This is a very important consideration to move forward with - we must ensure that our model is robust to a large number of 0 values. This means that simple linear regression will not be appropriate as the assumptions of normally distributed errors and equal variance will most likely not be met.

One more consideration is that this data is a count of findings per binary and therefore Poisson regression could be appropriate. However, the mean of our data is much smaller than our variance, indicating over-dispersion is occurring. To account for this as well as the large number of 0 findings, we may investigate the use of a zero-inflated model, a quasi-poisson model, or a negative binomial regression model.

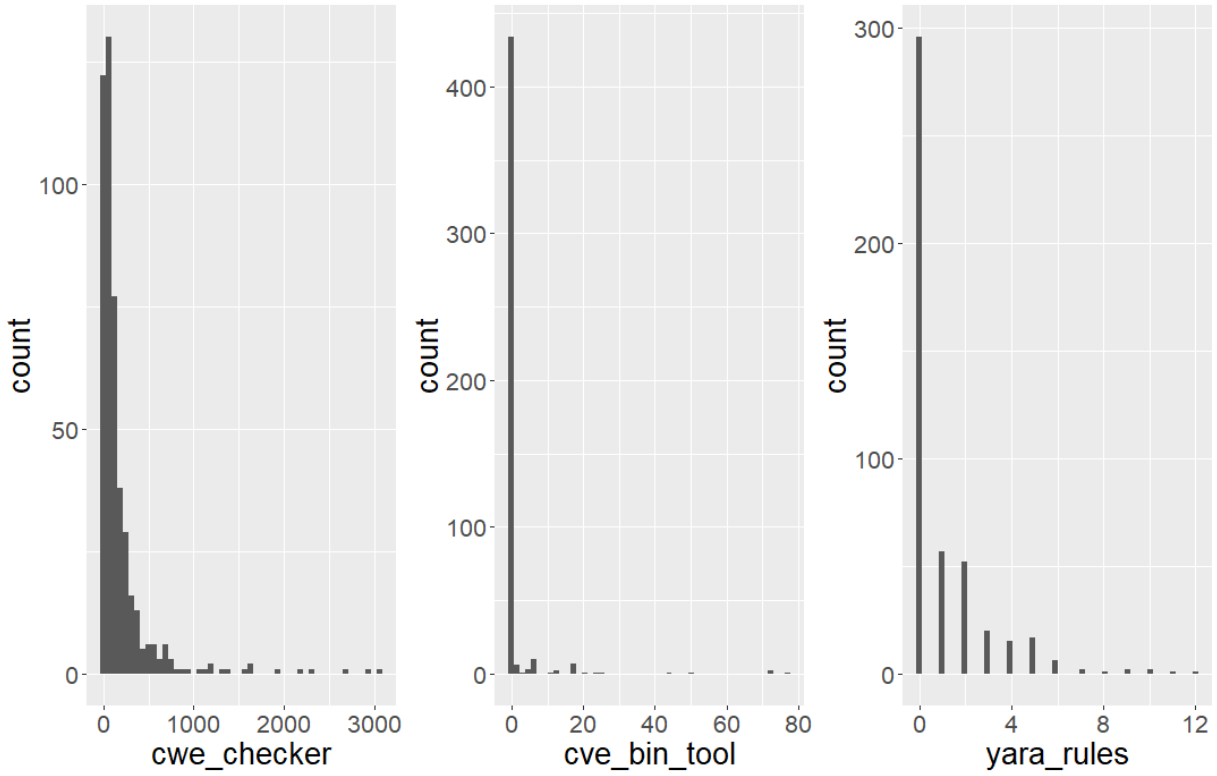

Figure 7.3: Distributions of Tool Outputs

First we investigate cwe_Checker, seen in Figure 7.4. Without considering the effect of size (which is known to be collinear with the compiler and static linking), we can see that the Kruskal-Wallis test is giving a p-value of less than 0.05 for compiler and static linking, indicating that these could be factors that are important to consider when building the benchmark repository.

CVE-Bin-Tool's output may be seen compared to binaries' factors in Figure 7.5. Clearly the extreme number of 0 findings is evident in these plots - any non-zero value is considered an outlier. Modeling this data will require a model that accounts for a large number of zeros

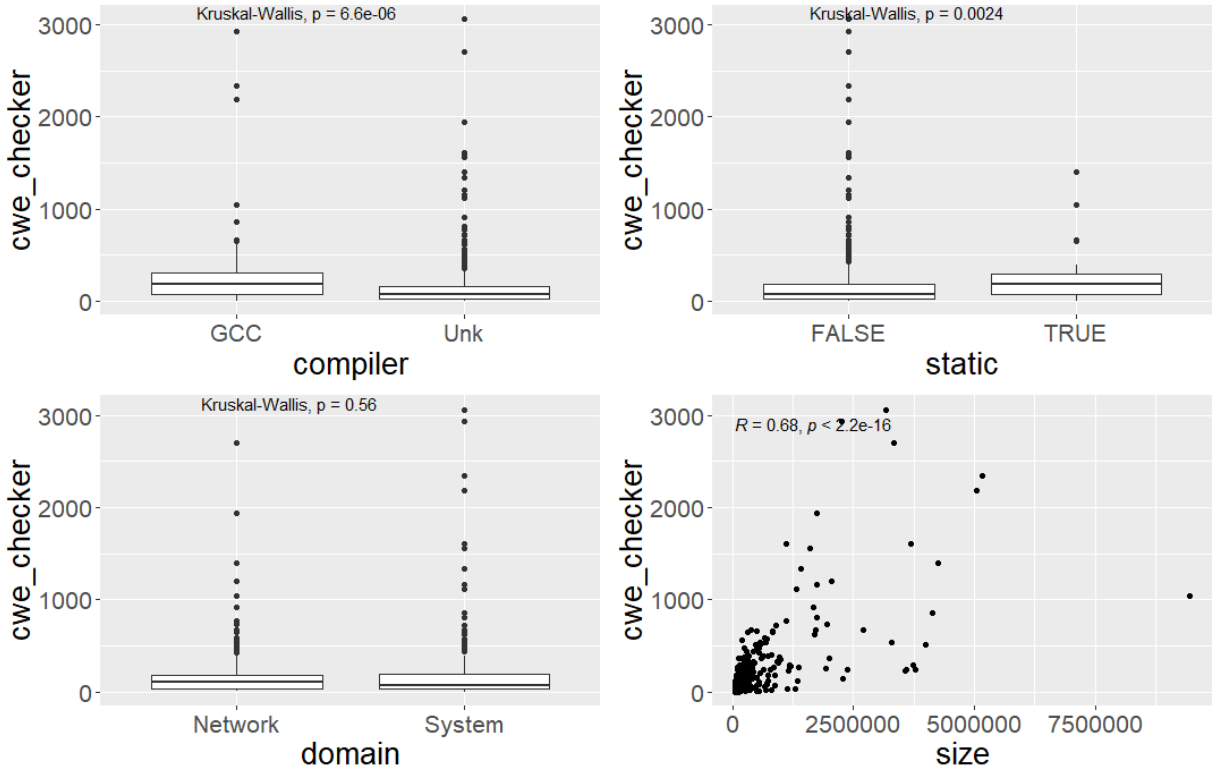

Figure 7.4: cwe_Checker Output Compared to Factors

in the dependent variable, further indicating a zero-inflated model would be appropriate to use.

Yara Rules' output may be seen compared to binaries' factors in Figures 7.6. It appears that size, compiler, and static linking all have some correlation with findings, but this analysis does not account for the effect of collinearity between size and the other factors. Network versus System domain does not appear to have a significant effect on the number of findings we expect.

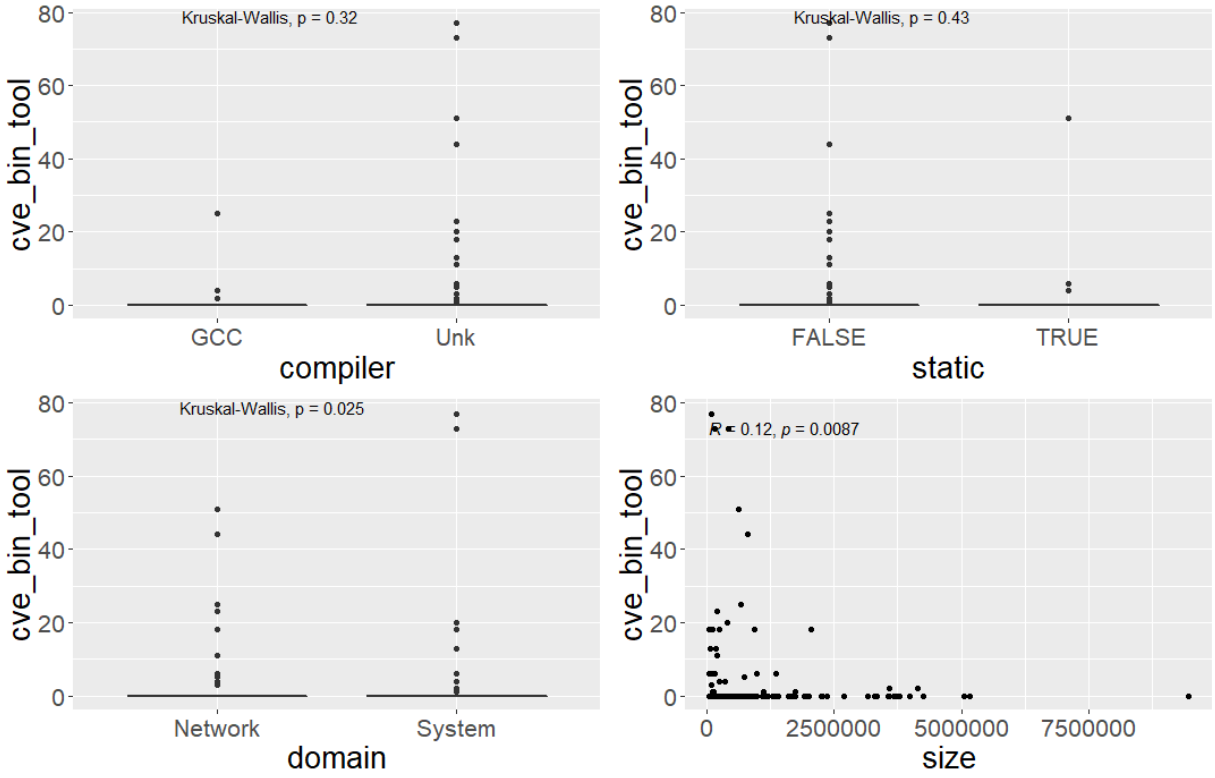

Figure 7.5: CVE-Bin-Tool Output Compared to Factors

Data Modeling

To reach a conclusion for the hypotheses stated above, we will fit a model to determine what attributes of the binaries are significant in helping to predict the tool output for a binary. This will result in three separate models, one for each tool. There are several caveats we must consider as we determine what model is appropriate for our data. Because there is some collinearity in our factors, we should consider a comprehensive model for each tool that is able to account for effects of factors together rather than separately. Otherwise, we would be able to answer our hypotheses through ANOVA tests.

One note before we begin: we do not expect to achieve accurate models. Security is largely influenced by the organizational processes (such as quality assurance) and the specific developer(s) who produce a binary. We also believe that age of a binary could play a large

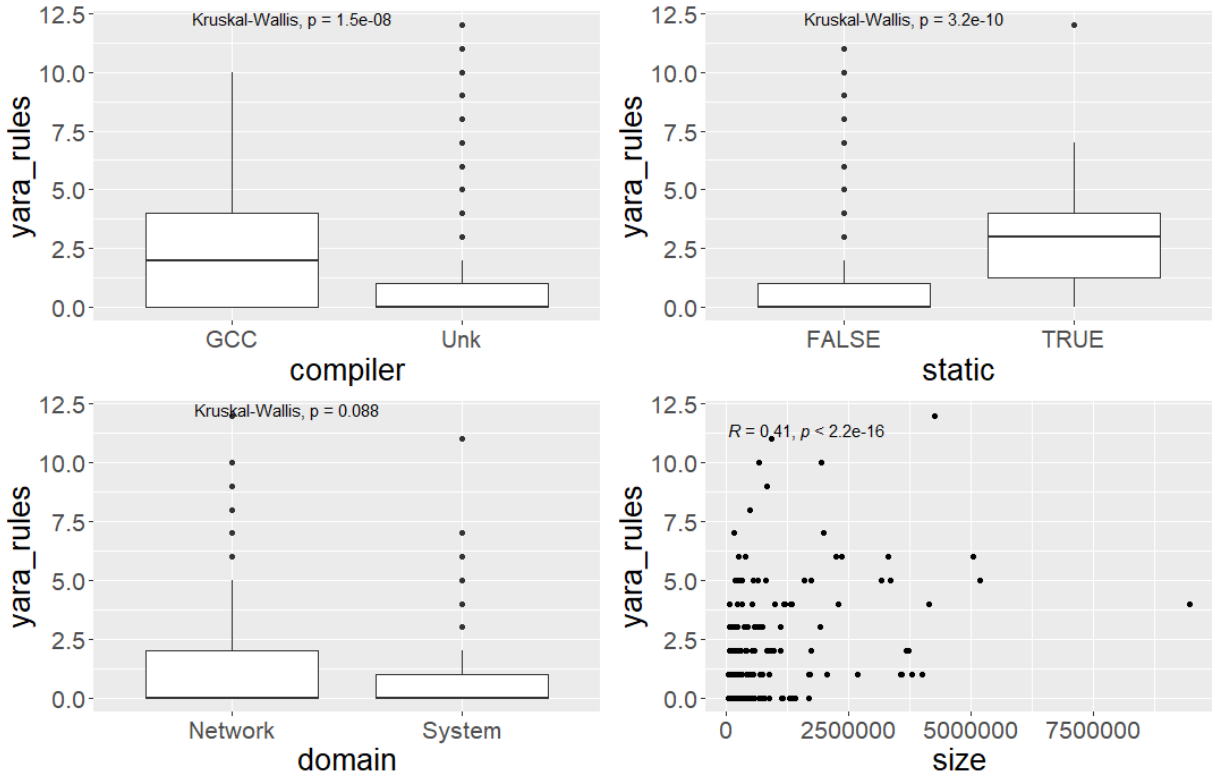

Figure 7.6: Yara Rules Output Compared to Factors

part in the security findings from tools such as CVE-Bin-Tool. This is because as a binary ages, additional vulnerabilities may be discovered that are present in older versions of a binary, so logically older binaries are going to tend to have more known vulnerabilities than newer binaries. Therefore, age should be considered as an attribute important in benchmark makeup as well. Unfortunately, we were unable to determine the age of the binaries.

cwe_checker output model The cwe_checker tool output, as seen compared to factors in Figure 7.4, is count data. Typically we would begin this analysis by looking at using a Poisson regression model; however, the cwe_checker tool output is large enough that Poisson regression may not be viable. Instead, we begin by attempting to fit a linear regression model.

After fitting a linear regression model, we find that our assumptions of equal variance

and normally distributed residuals are not met. This leads us to attempt a transformation. Using a log transform on either size or the tool output alone do not yield good results, but transforming both provides a model that meets the assumptions of linear regression. The diagnostic plots for this model may be seen in Figure 7.7.

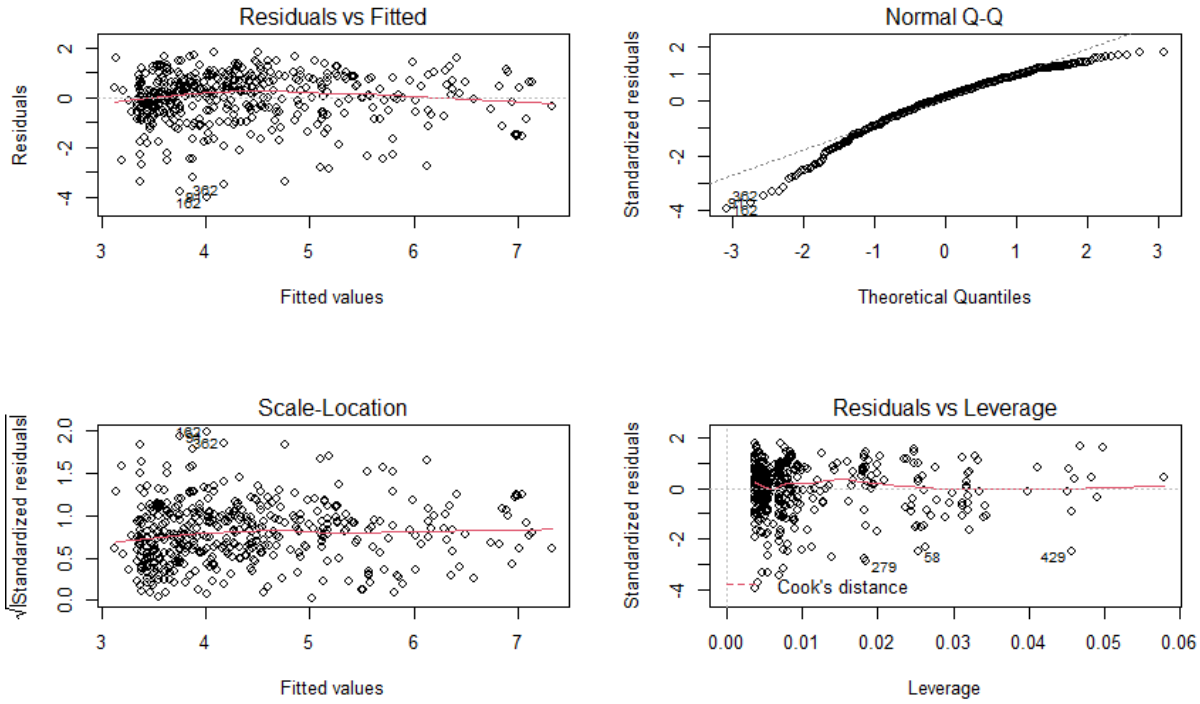

Figure 7.7: cwe_Checker Output Linear Model Diagnostic Plots

From these plots, we may conclude that there is some evidence against normality and equal variance, but the model should be robust enough to allow for this small amount of evidence against the assumptions. Additionally, the residuals appear to be independent, and there are no outliers with large leverage.

Now that we have determined that the assumptions for linear regression are met, we fit the model and begin identifying significant variables for the model. We will fit the model

initially with all factors, then remove them one by one, removing the non-significant factor with the smallest coefficient first. Each time a variable is removed, we re-fit the model and determine if another variable should be removed. AIC (Akaike's Information Criteria) is a general indicator of goodness of fit for models [3, 2]. AIC estimates the amount of error in the residuals of the model with a penalty for more complex model, so a lower AIC typically indicates a better model. We verify that the model's AIC does not increase as we removed variables as an additional measure to ensure that the model is being improved. The coefficients of the final model may be seen in Table 7.1. The formula for the model is

$$\mu\{log(cwe\_\hat{checker})\} = C_0 + C_1 log(size) + C_2 StaticallyCompiled \tag{7.1}$$

$$\mu\{log(cwe\_\hat{checker})\} = -5.71 + 0.84 log(size) - 0.38 StaticallyCompiled \tag{7.2}$$

$$median\{cwe\_\hat{checker}\} = 0.0033 size^{0.043} e^{-0.38 StaticallyCompiled}, \tag{7.3}$$

where $StaticallyCompiled$ is 1 if the binary is statically compiled and 0 if not.

To interpret these results, we must pay close attention to the log transformation of the response and one explanatory variable. Equation 7.3 showcases the back-transformed model. This model predicts the median output of cwe_checker for a binary given the size and whether it was statically compiled. A binary that uses static compilation is expected to have $e^{-0.382} = 0.682$ times the median number of findings from cwe_checker. We expect a binary with 10000 bytes to have 1.48 times the median number of findings compared to a binary with 1 byte.

Due to the p values of our coefficients being less that 0.05 for all coefficients in the model (size and static linking), we may reject the null hypotheses H_1 and H_2 for this tool. We can not reject H_3 or H_4 based on this model, as the variables associated with those hypotheses were not found to be significant.

Table 7.1: cwe_Checker Output Model Coefficients

	Estimate	Std. Error	t value	P-value
Intercept	-5.71261	0.51395	-11.115	0.000
Log(size)	0.83639	0.04276	19.559	0.000
Statically Compiled	-0.38200	0.18738	-2.039	0.042

CVE-Bin-Tool Output Model In the case of CVE-Bin-Tool, the output is largely composed of 0 values. We begin by fitting a simple Poisson regression model, and assess assumptions from that point. Diagnostic plots for this Poisson regression model can be seen in Figure 7.8.

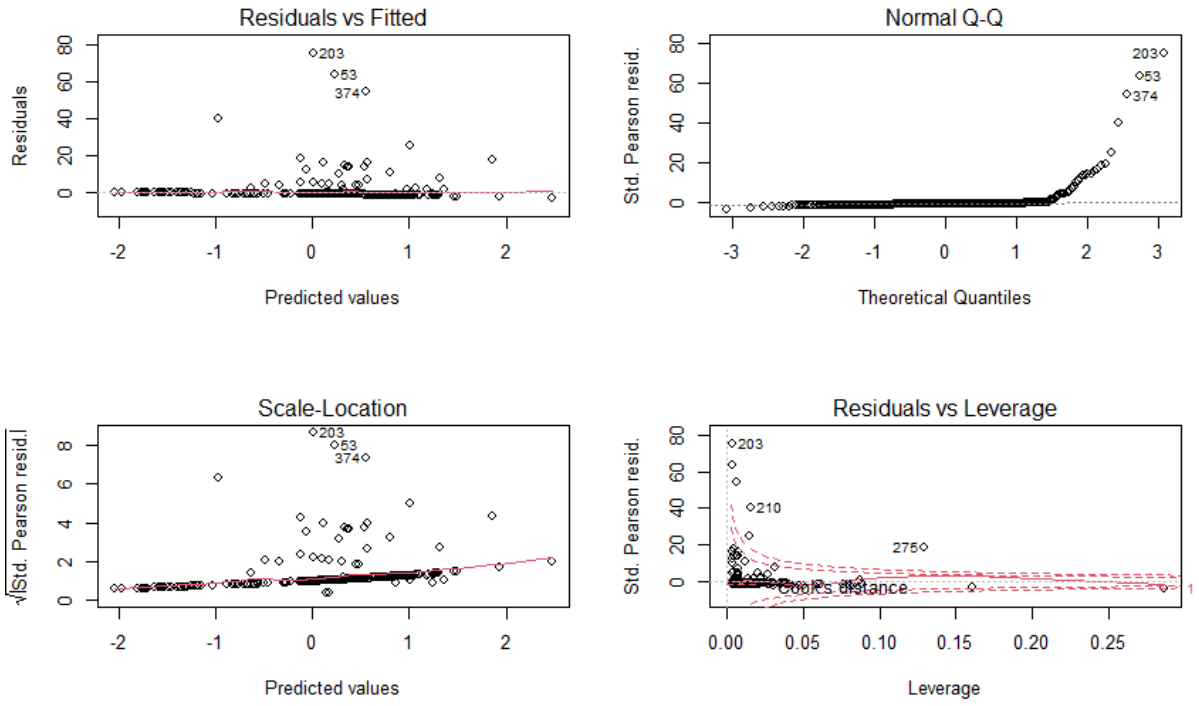

Figure 7.8: CVE-Bin-Tool Output Poisson Regression Model Diagnostic Plots

From Figure 7.8, we can see that several extreme outliers are present in the data. We observe that the residuals are on the scale of the data points themselves, indicating that the model is likely predicting every point as a 0, and we are simply seeing residuals that are equal to the value of each point. We also observe a high amount of overdispersion in the model, likely due to the large number of 0 counts. One method to relax these constraints would be to fit a quasi-Poisson model, or a Negative Binomial Model (NBM). Hoef and Boveng [26] suggest that these two models are largely similar aside from certain situations that can occur, in which they find largely different estimates of the effects of covariates. However, fitting either of these models does not get around the fact that the data is mostly 0 counts. Therefore we fit a zero-inflated Poisson regression model [20].

The zero-inflated Poisson (ZIP) regression model will separate the data into two models: one logistic model that estimates whether a binary will have a 0 count or greater than 0 count, and one Poisson regression model on the non-zero data to estimate what factors influence the size of the count [20].

The initial comparison of AIC between the Poisson regression model and the ZIP model reveals that the ZIP model performs much better - the Poisson regression model has an AIC of 3824.69, while the ZIP model has an AIC of 1087.9.

We also fit a zero-inflated negative binomial (ZINB) regression model to compare to the ZIP model. The AIC of the ZINB is found to be 554.77 compared to the ZIP's AIC of 1087.9.

Therefore we will fit a ZINB model and follow the same process as we did for the cwe_Checker output model: remove the non-significant variable with the smallest coefficient and then reassess, repeating until all variables are significant.

Following that process, we find that no variables are significant to predict the size of the count for non-zero counts of CVE-Bin-Tool findings. This indicates that the only prediction

we may do is on whether or not CVE-Bin-Tool will have findings, and not the number of vulnerabilities that we find. This makes intuitive sense - when we import a library, nothing about the binary we import into defines how many vulnerabilities will be in the library that is being utilized.

This makes interpretation much easier - we only need to look at the logistic component of the ZINB model, or to make things simple, fit a logistic model that predicts whether the count will be zero or non-zero. We fit a logistic model for simplicity, and follow the same process that we did to remove non-significant variables for the cwe_Checker model.

Assumptions for logistic regression require that we have no dependent response variables, no extreme outliers, and no collinearity in our predictors. We have no extreme outliers, which can be observed in Figure 7.9. Our response variables are independent. Collinearity does occur in our predictors, so there is some evidence against that assumption; however, the collinearity is not so severe that we should need to account for it through a different model.

The coefficients for the final logistic regression model are seen in Table 7.2, and the final equation becomes

$$p(CVEBinTool) = \frac{e^{-6.03+0.33log(size)-0.74DomainSystem}}{1+e^{-6.03+0.33log(size)-0.74DomainSystem}}, \tag{7.4}$$

and the logit function becomes

$$\frac{p(CVEBinTool)}{1-p(CVEBinTool)} = e^{-6.03+0.33log(size)-0.74DomainSystem}, \tag{7.5}$$

$$logit(p(CVEBinTool)) = -6.03 + 0.33log(size) - 0.74DomainSystem \tag{7.6}$$

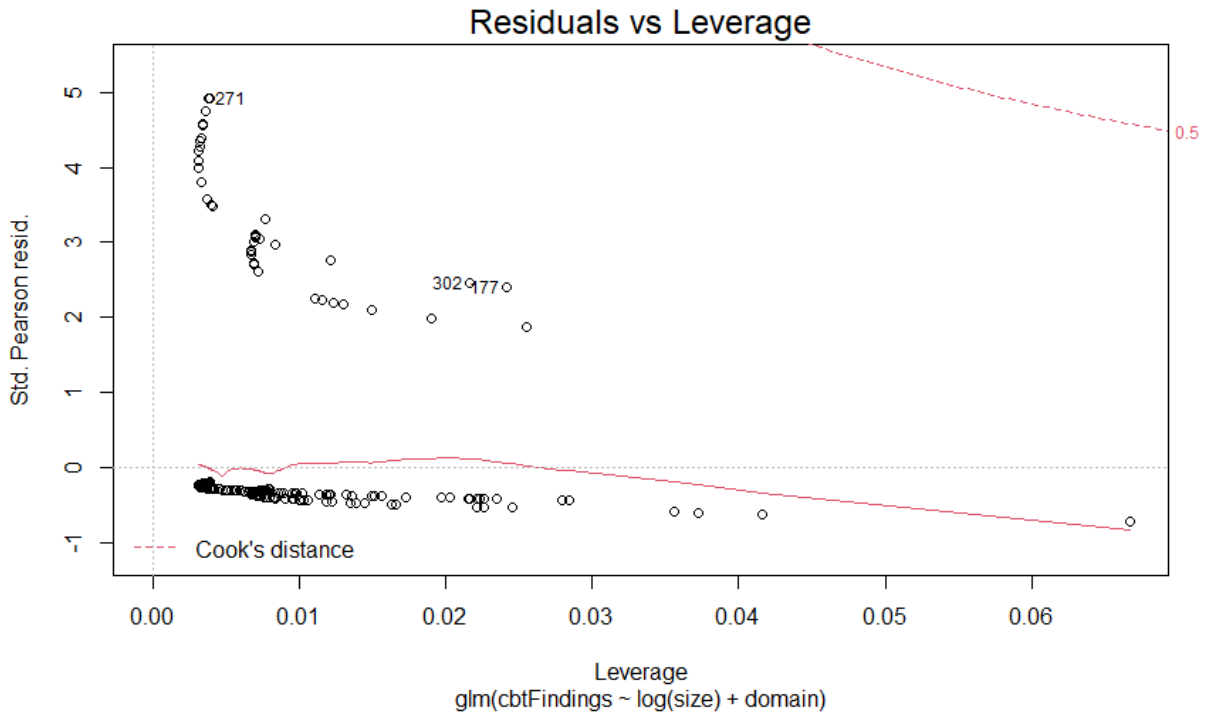

Figure 7.9: CVE-Bin-Tool Output Logistic Regression Model Residuals vs Leverage Plot

where DomainSystem is 1 if the binary's domain is system, and 0 if the binary's domain is network.

Table 7.2: CVE-Bin-Tool Output Model Coefficients

	Estimate	Std. Error	z value	P-value
Intercept	-6.0264	1.6743	-3.599	0.0003
Log(size)	0.3305	0.1334	2.477	0.0132
DomainSystem	-0.7433	0.3441	-2.160	0.0307

The coefficients for this model indicate that for a network-based binary with a $log(size)$ of 0, the odds of have findings from CVE-Bin-Tool are 0.002. For a system-based binary with a $log(size)$ of 0 the odds become 0.001. For a network-based binary with a $log(size)$

of 15 (around 5 MB), there is a 0.163 probability of having findings. This does mean that we will never predict a positive occurrence, indicating that we are missing critical variables that would fill the gaps in our model or that we may not be able to accurately predict the occurrence of findings from CVE-Bin-Tool.

While we do find $log(size)$ and binary domain to be significant in a model for predicting findings for CVE-Bin-Tool, it is difficult to say that there are any conclusions that may be drawn from this model due to the fact that everything will be predicted as having zero findings. This is almost certainly caused by the fact that most of the binaries do not have any findings from CVE-Bin-Tool. This likely requires further investigation in which we look at a wider variety of binaries, but is outside of the scope of this study.

Yara-Rules Output Model The Yara rules data has some resemblance to the CVE-Bin-Tool data with a large number of 0s. It is not as extreme in the number of 0s, however. The distribution of Yara Rules' output among the factors can be seen in Figure 7.6.

We will begin by attempting to fit a basic Poisson regression model, as well as a Poisson rate regression model using $log(size)$ as an offset variable. We find that the Poisson rate regression model does not perform as well as the basic model when comparing by AIC or residual deviance. In the basic Poisson regression model we do not find any extreme outliers, but there does appear to be some overdispersion occurring in the model. Running a dispersion test (from the "AER" R package) on our model confirms that our model is overdispersed.

Fitting a NBM allows for the overdispersion to be accounted for. Additionally, the NBM has an AIC of 1167 compared to 1318, and around half of the residual deviance. We still do not have any extreme outliers and our responses are independent, meaning that we have no assumptions violated for this model.

We also fit a ZIP model to determine if it would be a better fit. Using AIC, we do find

that this model improves on the NBM by about 10; however, this is a much more complicated model. Therefore we acknowledge that our model could potentially be improved by adding much more complexity, but will utilize the NBM as a simpler approach to modeling this data.

We take the same approach to removing non-significant variables as in the previous two models: by removing the non-significant variable with the smallest coefficient in the model.

The NBM model's variables can be found in Table 7.3. The equation for this model is

$$ln(\hat{YaraRules}) = \beta_0 + \beta_1 log(size) + \beta_2 SystemDomain + \beta_3 StaticallyCompiled \quad (7.7)$$

$$\hat{YaraRules} = e^{\beta_0 + \beta_1 log(size) + \beta_2 SystemDomain + \beta_3 StaticallyCompiled} \quad (7.8)$$

$$\hat{YaraRules} = e^{-7.28 + 0.6 log(size) - 0.45 SystemDomain + 0.68 StaticallyCompiled}. \quad (7.9)$$

Table 7.3: Yara-Rules Output Model Coefficients

	Estimate	Std. Error	z value	P-value
Intercept	-7.28110	0.79228	-9.190	0.00000
log(size)	0.59893	0.06365	9.410	0.00000
SystemDomain	-0.44560	0.16042	-2.778	0.00547
StaticallyCompiled	0.67812	0.25444	2.665	0.00769

For a dynamically compiled, network based binary with size of 1, we find that the baseline number of findings from Yara Rules is $e^{-7.28} = 0.0007$. Statically compiling the binary correlates with an expected number of findings $e^{0.678} = 1.97$ times higher. A binary of the system domain is correlated with $e^{-0.446} = 0.64$ times the number of expected findings, meaning that network based binaries tend to have more findings. Finally, a change of 1 in

the $log(size)$ of the binary is correlated with a $e^{0.599} = 1.82$ times change in the expected number of findings. As an example, consider a network-based binary, statically compiled, with a $log(size)$ of 15 (around 5 MB). We would expect 11.0 findings from Yara-rules.

We find $log(size)$, static compilation, and domain to correlate with the number of findings from Yara-Rules. We may therefore reject the null hypothesis $H1_0$, $H2_0$, and $H4_0$, but we may not reject $H3_0$.

Conclusion

From the models we have fit, we reject $H1_0$, $H2_0$, and $H4_0$. This indicates that when we are creating a benchmark repository for a PIQUE model for binaries, we must take into consideration the size, domain, and linking strategy used. If we fail to do so, we will receive a score that is inflated in some way, meaning that there is a systematic reason for a lower or higher score that is not necessarily due to security differences between binaries.

Table 7.4 contains the final variables and their p values for each model.

Table 7.4: Output Models Coefficient P-Values

	cwe_Checker	CVE-Bin-Tool	Yara-Rules
Compiler	-	-	-
log(size)	0.000	0.000	0.000
Static Compilation	0.042	-	0.008
Domain	-	0.03	0.005

Discussion

When creating a security metric, we must be sure that the metric does not mislead people to believe that a system is more secure than it is in reality. We also need to ensure that there is trust and understanding of the security metric.

Without trust in the output of PIQUE-Bin, it is likely to be ignored by stakeholders and developers alike. Without understanding, the output of PIQUE-Bin may be used to take inappropriate security actions, such as allowing a binary in a system due to a high score (without considering the benchmark makeup), or rejecting a critical patch to a binary due to a lower score from PIQUE-Bin.

These tests help to build trust and understanding by showing how the benchmark repository may influence the score. For example, we know now that if the benchmark is composed of large binaries and we analyze a small binary, we will have an artificially high score.

For all tools, size was found to be significant. This makes sense, as a larger binary is more likely to have more libraries being used, potential capabilities, and more potential lines that could fit a CWE pattern.

We found static compilation to correlate with the output of cwe_Checker and Yara-Rules. For cwe_Checker, this result is somewhat unexpected; however, it makes sense when we think about what the added code in the binary is. This is often going to be common functions, especially functions that interact with the operating system. These types of functions are a higher risk, and likely contain more high-risk code than typical binary code would, causing a higher number of CWEs to be identified. In the same way, the capabilities that are now contained in the binary would likely be found by Yara-Rules, causing additional findings.

Domain is found to correlate with the output of CVE-Bin-Tool and Yara-Rules. This is interesting, but likely stems from the common libraries imported for network or system-based calls and the differences in the vulnerabilities contained in those libraries. Additionally, the network binaries have more capabilities compared to a system-based binary.

Altogether, we find that there are several important factors to consider when composing the benchmark repository for PIQUE-Bin. These considerations help to build a reliable

security metric that we understand and are confident in.

Threats to Validity

Let's begin with threats to validity stemming from the data itself, then address any model-specific threats to validity.

The binaries we gathered were not gathered randomly as there is no possible way to collect binaries randomly. This means that any conclusions we make will be limited to apply only to the collection of binaries that we have. However, even though that is the case, most of the trends we identify in these binaries would likely be present in other binaries when we apply some intuition to why we found what we did. Confounding variables that we have not accounted for could be causing us to make conclusions about certain variables that do not truly have an effect on the tools' outputs. One example of this would be a correlation between the developers who work on system based binaries as opposed to network based binaries - perhaps many of these binaries were developed by a small set of developers, and the teams were split based on the domain of the binary. Perhaps the statically compiled binaries we collected all came from the same developer. These are unlikely situations, but not impossible.

As we noted earlier, the compiler factor has some oddities. Most binaries do not include the information about the compiler used to create them. This leads to us categorizing many binaries as an unknown compiler. We cannot be sure that the two categories ("GCC" and "Unknown") are mutually exclusive - therefore, we lose some power of the conclusion we may make about this. That being said, the logical reasons that compiler would cause a systematic difference in tool output are 1) the compiler or compilation process introduces a vulnerability or 2) certain compiler-specific optimizations cause patterns that are flagged as false positives more often, or 3) the compiler is confounded with some other factor such as development environment. Therefore, we may be fairly certain that compiler is not necessary

to consider when building a benchmark repository.

cwe_Checker Model Threats The cwe_Checker model appears to fit the data quite well, however there is some evidence against the assumptions of the linear model we fit.

The residuals do not appear to have any severe patterns found in the residuals vs fitted plot in Figure 7.7, but there may be a trend of low predictions having a large negative residual. Additionally, while the residuals are not perfectly normally distributed, we expect that the effect of the lack of normal distribution of residuals in negligible when we consider the p-values being as low as they are.

CVE-Bin-Tool Model Threats This model has many threats to validity. The model was eventually fit as a logistic regression model predicting if a binary would have findings or not. Our model predicts every point in our data set as not having findings. This means that we cannot rely on this model for prediction, and we have low confidence in any conclusions we make using this model.

For this reason, we have chosen not to make any conclusions based upon this model. However, any conclusions we would make about correlated factors for tool output is confirmed by the other models we fit.

Yara-Rules Model Threats One threat to validity is that we chose not to utilize a more complex model in favor of a simpler model that is easier to interpret. Typical statistical guidelines would say that an AIC of 5 or more is significant [20]. however, the added complexity of a second model is not accounted for by the AIC.

Sensitivity to Weighting

The model output is sensitive to the weighting from stakeholders to a large extent. The weighting done in the model is done in two layers: at the QA to TQI level which

is done through pairwise comparisons and the AHP, and at the PF to QA layer which is manually weighted. These weights for PIQUE-Bin have been previously shown in Table 5.2 and Table 5.3. These weights were done from the perspective of the researcher, but they could be changed entirely by some other stakeholder.

Changes to the weighting from the PF to QA layer can have a drastic impact on the TQI - potentially, it can take the score from close to 1 to 0. This is because the weighting may be modified such that findings are considered higher priority for particular QAs that may also be important. For example, consider applying PIQUE-Bin to some project where there is only a single finding across all diagnostics in the model. This single finding causes a measure to evaluate to 0. As it is weighted now, this would likely have a small impact (as we will see later, likely less than 0.01 change to TQI). Now consider that we re-weight the PF to QA layers such that this diagnostic is the only diagnostic that aggregates to eventually impact all QAs but non-repudiation. Then all QAs will evaluate to zero except one, and we receive a score of 0.048 (the current weight from non-repudiation to the TQI).

We choose not to investigate the weighting from the PF to QA layer with a case study or experiment because the weighting of that layer (in the context of PIQUE-Bin) should be done as objectively as possible and should not change across projects or domains. This is because regardless of the domain, PFs will impact the same QAs - for example, 'Authentication Errors (CWE-1211)' will always impact authentication and only authentication. We have shown that a change in weighting at this layer may completely change the TQI, so this weighting should be handled with care.

The weighting between the QA layer and TQI provides some interesting context as well. The TQI is calculated as a weighted sum of the QAs. Therefore, the value of the TQI is limited to be between the minimum and maximum values that the QAs take on. We may also see the weighting at this layer impact the TQI to change from 1 to 0 or vice versa if there are QAs that take on values of 1 and 0. We will investigate the potential impact

of weighting at this layer in the case study performed earlier on the Busybox binaries as detailed in chapter 6.

As seen in Table 6.2, we have a set of TQI values and QA scores. Using these values, we can determine the maximum possible change in the TQI based upon the weighting by taking the difference between the minimum and maximum QA value for each assessment. This can be seen visually in Figure 7.10, where each assessment has a point for the minimum and maximum possible value achieved by changing only the weighting for each version of Busybox.

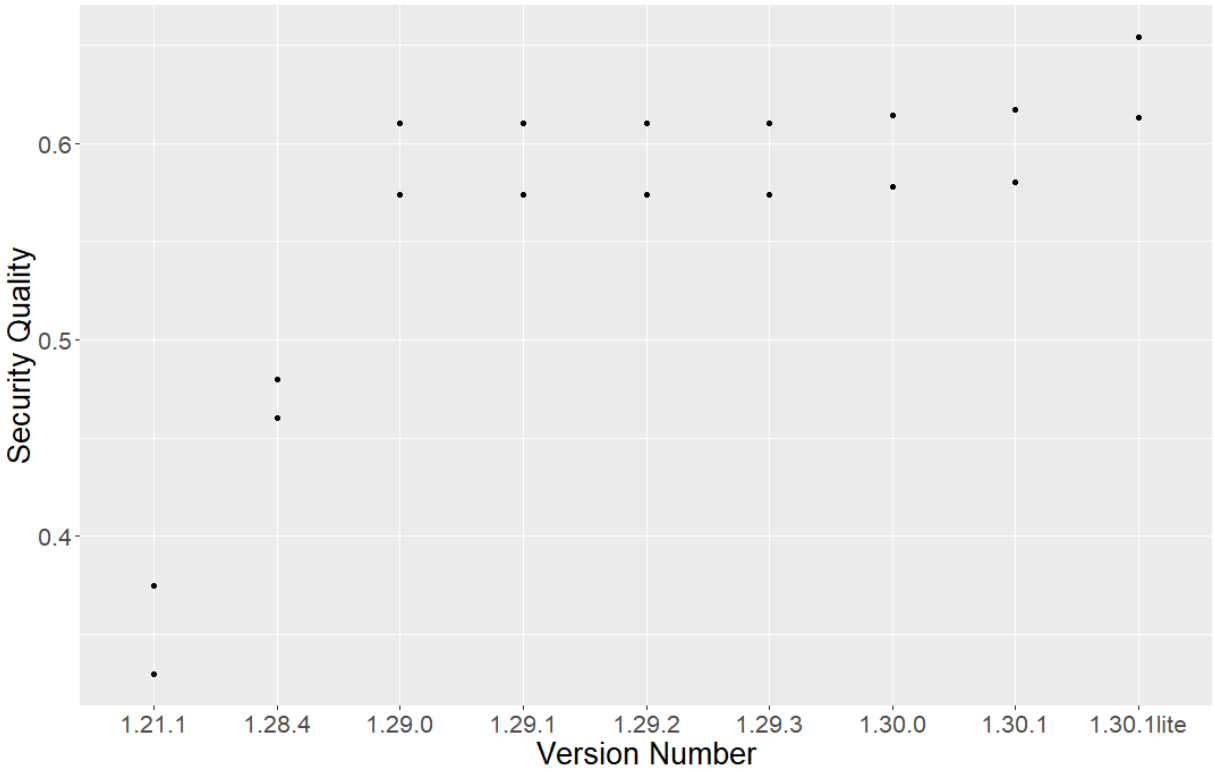

Figure 7.10: The Maximum and Minimum Possible Value for Each Assessment, Based Upon Weighting

To summarize the possible changes in TQI due to weighting, some summary statistics based

on the Busybox assessments follow. The minimum maximum possible change in TQI among these assessments is 0.02, which occurs in the assessment of Busybox version 1.28.4. The maximum maximum possible change in TQI among these assessments is 0.045, which occurs in version 1.21.1. The mean maximum possible change is 0.036.

The sensitivity is a rather small value due to the nature of the vulnerabilities that have been found in these binaries. Many findings have aggregated to impact all of the QA nodes. This is often the case when weakness categories may lead to the arbitrary execution of code. Arbitrary execution of code will compromise all QAs in PIQUE-Bin, causing the QA nodes in the model to tend towards the same value.

One way to solve this would be to add a node at the QA level that accounts for the property that is compromised by arbitrary execution of code, allowing for the composition of the QAs to be more orthogonal. This could be something such as 'control'. However, this QA would likely always be the most important and would therefore reduce the impact of the other QA nodes a significant amount.

Another cause of the small maximum possible change is due to the small number of findings in more recent versions. The reason the binaries do not receive a score close to 1 is because of the default behavior of measures with $[0, 0]$ thresholds and no findings. These nodes evaluate to 0.5 in the model currently, so the maximum score is very close to the score of busybox version 1.30.1lite.

Sensitivity to Single Findings

Another area of importance is PIQUE-Bin's sensitivity to single vulnerabilities within a binary. This will help define the expectations of stakeholders on normal fluctuations in the TQI score. Additionally, this will give greater context to what goes into a score. For instance, does a score of 0.5 indicate many findings or few findings?

To identify the impact of a single vulnerability, we perform the following: Run PIQUE-Bin on a binary and record the TQI score. Now, for each possible finding from our tools, we re-run PIQUE-Bin on the same binary with that weakness injected into the binary. All weaknesses are injected with a severity of 1. This is standard for findings from cwe_checker and yara-ruels, however cve-bin-tool reports every finding with a severity between 0 and 10, derived from the CVSS score of a vulnerability. We then take the difference from the original TQI and the new TQI found after injecting a vulnerability. This will give the impact of each finding on the TQI.

The impact of a single finding on the TQI does not change depending on the TQI - that is, a finding that changes the score by 0.01 will change a score of 0.5 and 1 equally, to get 0.49 and 0.99. The only case in which this does not occur is the case in which a QA that the finding aggregates into already evaluates to 0 because the score cannot drop any lower. This is a desirable quality to allow stakeholder priorities to properly impact the TQI.

Results

The results of following the above procedure gives us impact for each finding which may be split into three Figures, Figure 7.11, Figure 7.12, and Figure 7.13. The average impact of each product factor (CWE category) may be seen in Figure 7.14. The average impact is 0.008, the minimum is .00002 and the maximum is .085.

There is one outlying finding which has an impact of 0.52 and has not been included in this analysis. This finding is 'CWE-560 Weakness Diagnostic', which is the cwe_Checker finding for ' CWE-560: Use of umask() with chmod-style Argument'.

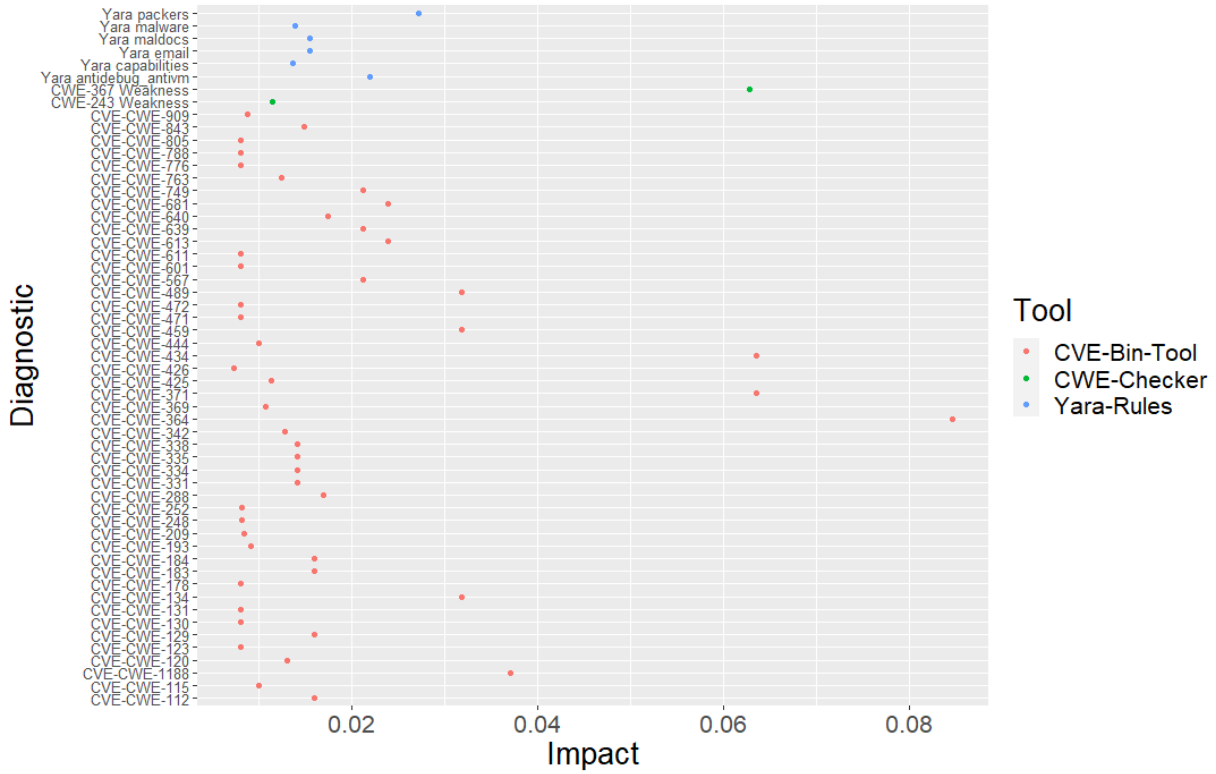

Figure 7.11: Impact On TQI For A Single Finding From Each Diagnostic, Part 1

Discussion

Overall, we see that the findings are all positive, which is a desirable outcome. Some findings have impact that is very close to 0 which is caused by model structure, benchmark thresholds, and stakeholder priorities.

The diagnostic for CWE-560 has a very high impact. There are two primary reasons for this. The first reason is that in the benchmarking process, this finding was incredibly rare, leading to threshold values of $[0.0, 0.0406]$. This means that even a single severity 1 finding will evaluate to -23.5. The second reason is that this finding aggregates into the 'CWE-1006: Bad Coding Practices' common weakness category, a category that only has two diagnostics from our tools. This causes these diagnostics to aggregate with higher impact

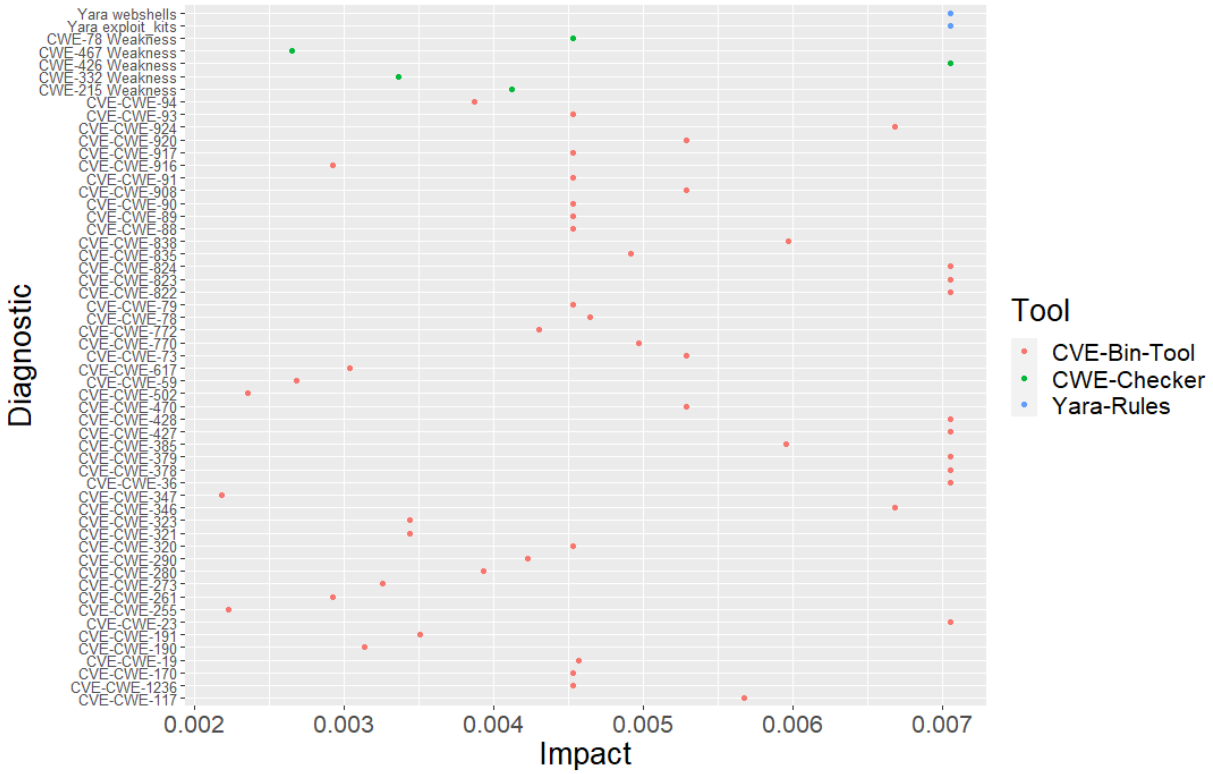

Figure 7.12: Impact On TQI For A Single Finding From Each Diagnostic, Part 2

due to getting a higher edge weighting when aggregating into the product factor layer from the measure layer.

To exemplify this second reason, consider a PF with three measures - A, B, and C. Then consider a PF with only 2 measures, X and Y. Consider a finding under measure A such that the value of A is 0 and there are no findings for B or C, giving values of 1. Then we evaluate the associated product factor to $0\frac{1}{3} + 1\frac{1}{3} + 1\frac{1}{3} = \frac{2}{3}$. Then consider the second scenario with two measures, and let X have a finding such that X evaluates to 0 and Y evaluates to 1. Then we evaluate the associated product factor to $0\frac{1}{2} + 1\frac{1}{2} = \frac{1}{2}$. The single measure that evaluates to 0 in both scenarios causes a lower score when it aggregates with fewer other nodes.

These two conditions allow the CWE-560 finding to have a huge impact on our model,

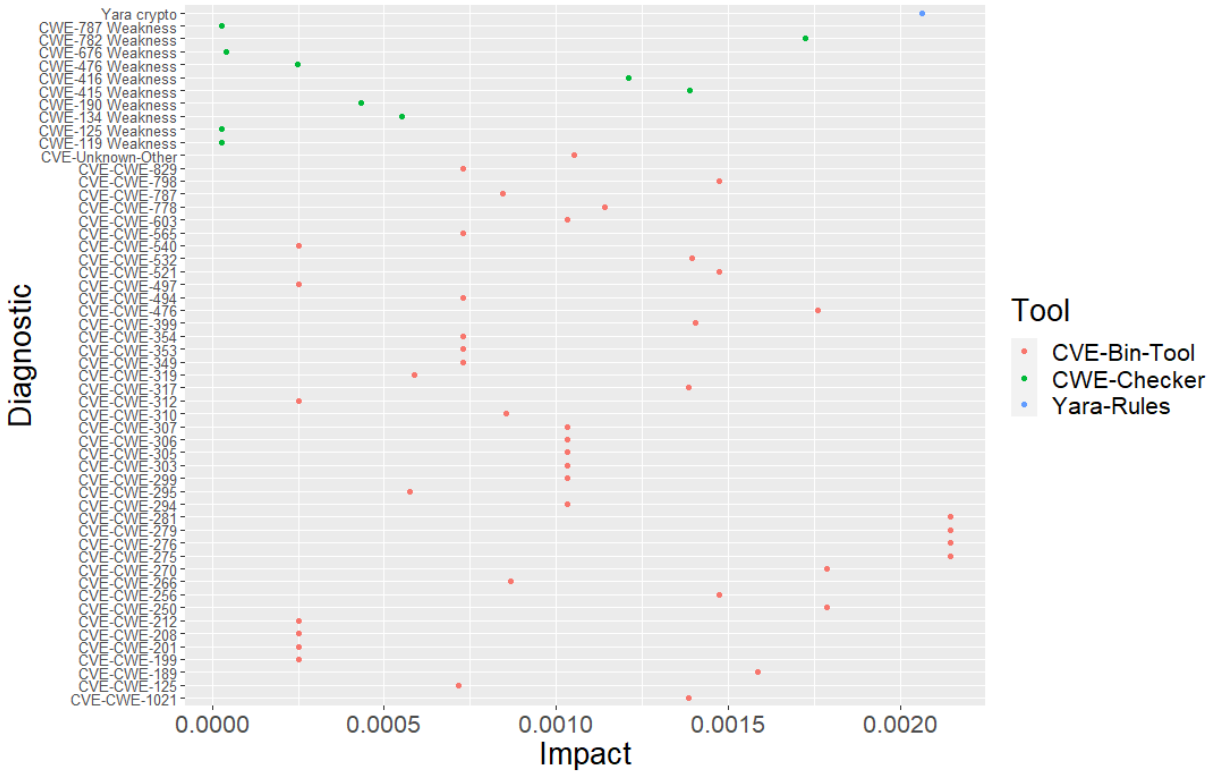

Figure 7.13: Impact On TQI For A Single Finding From Each Diagnostic, Part 3

which is important to know as a stakeholder or developer interpreting the results of any model application. Sudden and large changes in TQI may be due to severe vulnerabilities or could happen due to a high impact finding.

One other idea to keep in mind is that this analysis is done with all severity 1 findings; however, cve-bin-tool gives findings with severity between 1 and 10, meaning that a finding could potentially have 10x the impact as we see in Figure 7.11, Figure 7.12, and Figure 7.13 . Severity 1 findings are incredibly rare from cve-bin-tool, so the impacts we estimate here significantly underestimate the likely impact of a cve-bin-tool finding.

These results do hint that many nodes may have close to 0 impact, while others may have very large impacts. Whether this is a desired behavior depends upon the stakeholders who are utilizing the model. However, these impacts are a product of the entire process

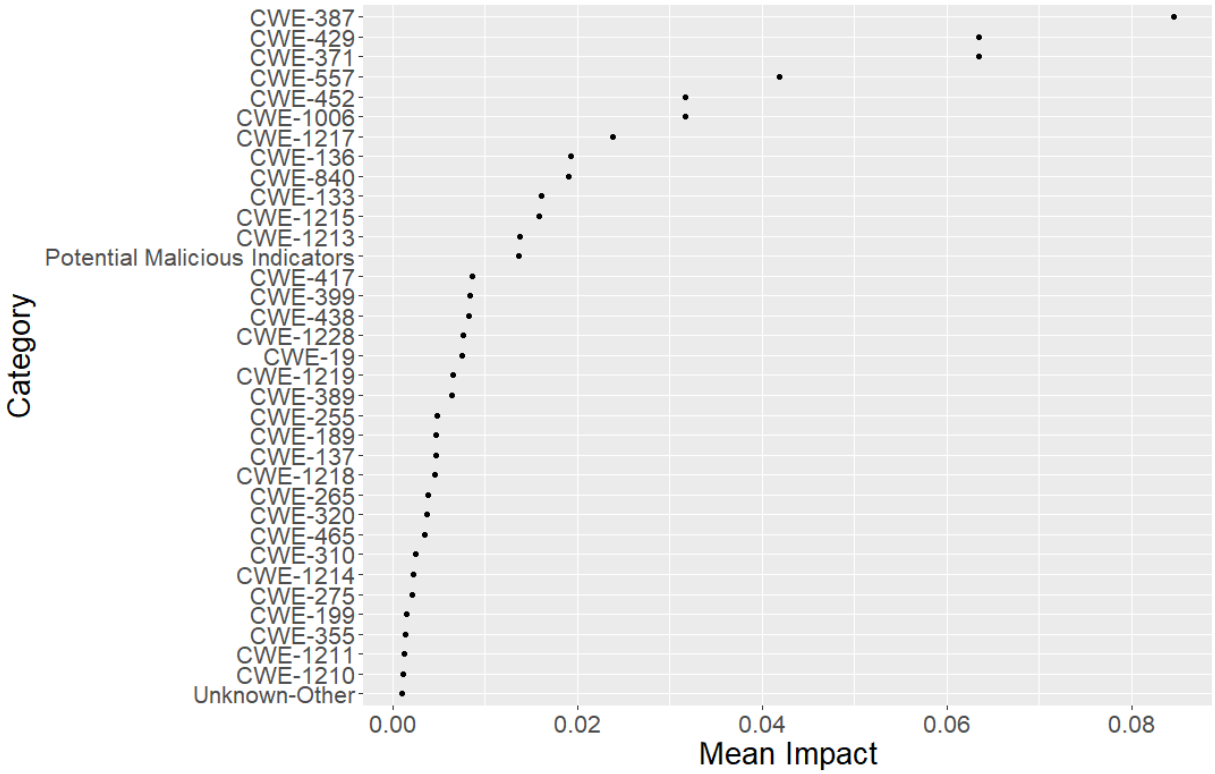

Figure 7.14: Mean Impact On TQI By Category

applied using PIQUE-bin, and as such, this is to be expected. Findings have little impact due to weighting, model structure, or high frequency withing the benchmark repository. The impact of findings may be changed through changing any one of these factors.

Threats to validity

Internal Validity Internal validity represents our ability to claim causation between our explanatory and response variables. We know that the model's change in score is directly cause by the explanatory variables, so there are no threats here.

External Validity This analysis is done for our specific model structure, with our stakeholder weighting, and our benchmark repository. A change in any one of these factors may significantly change the outcome of the experiment, and therefore, we may not extend these results to any other models/scenarios.

Despite that, we may still use these results to guide interpretation of the model in some cases, and small changes to the model are unlikely to change the results so much that they are useless for interpretation. Additionally, this impact will remain the same for this model regardless of what binary is being analyzed.

Construct Validity Threats to construct validity would indicate that we are not measuring variables in a way that reflects reality. The primary threat to construct validity for this sensitivity analysis is that we are measuring the impact of each vulnerability occurring a single time, which may not be representative of the average change in TQI for a model in practice. Perhaps software typically undergoes the addition or removal of dozens of vulnerabilities at a time, in which case we could see a large range of possible changes in TQI, and therefore this analysis does not inform the interpretation of the output.

Despite this, we still claim that this sheds light upon the expected change in TQI, as we know that a large or small change may be indicative of several or a single vulnerabilities and should be investigated further. PIQUE lends itself to easily determining what causes a change in TQI by categorizing weaknesses In addition, a visualizer for PIQUE is under active development which will make the investigation of changes in TQI easier.

Conclusion Validity We have not made any conclusions about the impact of vulnerabilities in general, so there are minimal construct threats. However, we should note that the data presented in this analysis is limited to the specific context of PIQUE-Bin, as we mentioned when addressing external validity.

THREATS TO VALIDITY

In addition to threats addressed in specific sections, we note overarching threats to validity to PIQUE-Bin. The validity in question is that of the model PIQUE-Bin. We address internal, external, construct, and conclusion validity.

Internal Validity

Internal threats are threats to the study's assertions of a causal relationship.

One inherent assumption in PIQUE-Bin is that findings have a negative impact on security. In almost all cases this is evident because vulnerabilities do not improve the security posture of a binary. However, it could be argued that certain findings may not always indicate worse security - for instance, consider the category from the yara-rules tool that represents capabilities of the binary. It is possible that a certain set of capabilities may be worse in terms of security than another set even if they are smaller in number, but we have no way of accounting for this. This would require experts to weigh in on the severity of each finding, which would reduce the autonomy of PIQUE-Bin, but allow us to have greater confidence in the model.

In a similar way, we assume that the effect of findings is linear - that is, two findings have twice the impact of one finding and four findings have twice the impact of two findings. This may not always be the case. Perhaps findings should be put on some other scale, such as an exponential curve, that assumes that as we get more findings, additional findings should have more severity. This would account for some sort of compounding effect of vulnerabilities enabling other vulnerabilities. However, different scales all come with their own assumptions, and the linear case is the simplest approach.

External Validity

External threats are threats to the study's ability to generalize to other settings and be replicated.

We have shown that PIQUE-Bin is able to assess security for busybox binaries over time. Threats to external validity represent threats to our ability to apply PIQUE-Bin to other binaries with similar success.

PIQUE-Bin will likely be valid for binaries that are similar to busybox, and we have no reason to suspect that it would fail to assess other binaries. However, as we discovered in our tool output sensitivity analysis, we must be wary of interpretation with respect to the differences between the binary under analysis and the benchmark repository. Additionally, we found that the cwe_checker tool was inconsistent at times, and was unable to analyze all binaries for unknown reasons. This discovery resulted in one bug being discovered in the cwe_checker tool and one potential bug being discovered in the NSA's Ghidra tool. One threat is that a tool may fail to run on a binary, in which case we cannot rely on the output of PIQUE-Bin for any comparison purposes.

Construct Validity

Construct threats are threats to the study's ability to represent and measure variables in a way that reflects reality.

The primary threats to validity are concerned with the process of aggregation that occurs in PIQUE-Bin. We are unsure if the sum of CVSS values of vulnerabilities represents the best way to analyze the impact of a set of vulnerabilities. This approach assumes that two severity 5 vulnerabilities are equivalent to a severity 10 vulnerability. It is not stated

in a CVSS guide that the NVD references[1] whether this approach is valid or not. However, this is one of the simplest approaches which is why it is chosen.

We also see issues in the way that aggregation occurs and the way certain vulnerabilities impact the quality aspects. That is, a vulnerability may be classified under one CWE because that is the weakness it exploits, but may have different impacts from the typical impacts of the CWE. Additionally, we may see improper impacts when aggregating from a base level CWE to a CWE category, as in the case of aggregating both CWE-353: 'Missing Support for Integrity Check' and CWE-349: 'Acceptance of Extraneous Untrusted Data With Trusted Data' into CWE-1214: 'Data Integrity Issues'. CWE-353 has impacts on integrity and non-repudiation, while CWE-349 has impacts on access control and integrity. Both of these are aggregated into the same CWE category, where they go on to impact quality aspects in the same way. This is a consequence of aggregation and has been mitigated as much as possible through proper choice of the structure, as well as attempting to account for this when determining the impact of a category on the quality aspect nodes. We rely on the expertise of the MITRE organization in the way that these CWEs have been structured, as well as reporters of CVEs to properly and accurately categorize CVEs under the correct CWE. Failure of this process means that PIQUE-Bin will improperly aggregate a finding. This is another reason that interpreters of the output of PIQUE-Bin must be wary of any changes in TQI and investigate the cause.

Other threats to construct validity include our decision to include measures which receive $[0, 0]$ thresholds. This is a threat to our ability to measure the impact of these measures, because we do not have information on how common they are among our repository. We acknowledge that this the existence of these nodes is not ideal, but leaving them out of the model may miss critical information - ignoring these findings would be detrimental because they can represent some of the most interesting findings due to their

[1] https://www.first.org/cvss/user-guide

rarity. Some findings may be very rare due to how severe they are, such as plain-text passwords.

Conclusion Validity

Conclusion threats are threats to a study's ability to draw conclusions.

We acknowledge that the conclusions we may draw from our work is very limited due to validation methods and limited ability to randomly sample. We have avoided conclusion threats by staying within scope when making conclusions. However, we are able to achieve our research goal while making minimal conclusions due to the lack of existing research in this space.

CONCLUSION

In this thesis, we present the design, development, and validation of a hierarchical quality model for the analysis of security quality in binaries by utilizing the PIQUE platform. Our research followed the goal question metric paradigm with an overall research goal of improving our ability to assess the security quality in binaries from a stakeholder's perspective.

We began by briefly covering relevant topics as well as supporting work. This includes the state of binary analysis and some examples of binary analysis, software quality modeling, vulnerability management, and the operational technology environment. The supporting work includes quality modeling approaches presented by Quamoco, QATCH, and the framework PIQUE. We also presented an informal literature review on quantitative model-based security metrics for binaries which found no papers, leading to the conclusion that this area of study is largely under-researched.

We present our research goal in Basili's Goal-Question-Metric format [9]. This process includes defining several questions, the answers to which will lead us to achieving our goal. We proceed to present and systematically answer these questions through metrics quantified in the following chapters through an informal literature review, an overview of the design and development of PIQUE-Bin, case studies, and experiments for sensitivity analysis.

The design and development of PIQUE-Bin utilizes well-established security resources that will enable the model to easily incorporate a variety of tools while drawing on the knowledge base of security research to categorize and aggregate findings. The tool utilizes the Microsoft STRIDE model to define the nodes at the highest level, and below that uses the Common Weakness Enumeration software development view (CWE-699) to define the remaining structure.

Three tools are integrated with PIQUE-Bin: CVE-Bin-Tool, cwe_Checker, and YARA.

CVE-Bin-Tool finds known vulnerabilities in third party libraries by searching for associated strings. cwe_Checker disassembles a binary using the NSA's Ghidra tool, then identifies patterns commonly associated with CWEs. YARA is a simple pattern searcher. We use this tool in combination with a repository of security-related patterns that will help to identify malicious code within a binary as well as some other concerning categories of findings such as the capabilities a binary has. These three tools cover a variety of security findings, giving a better picture of security together than any one tool individually.

We apply an early version of the model to two Wireshark binaries for initial exploration of the model's usage. We find unsatisfactory results in this version of the model, and identify how it might be improved before continuing with exemplary applications of the model.

We then apply the latest version of PIQUE-Bin to several versions of the Busybox Linux utility binary. We find satisfactory results that indicate the binary has improved in security quality over time as expected. We see that the issues identified in the application of PIQUE-Bin to Wireshark have been resolved and that the model performs satisfactorily. Finally, we note that there are several versions that do not see a change in score due to a lack of tooling that identifies the changes between those binary versions. As such, the model may be improved through the use of more tools. This comes at the cost of ease of use and the cost of integrating a tool.

Model validation is performed primarily through sensitivity analysis. We perform sensitivity analysis in three different ways. First, we analyze the sensitivity of the tools output to attributes of the binary they are run on. Second, we briefly touch on the sensitivity of the model TQI to weighting of the lower layers. Then, we perform sensitivity analysis of the TQI to individual findings.

We find that the tools' output is correlated with different attributes depending on the tool. We tested this by first gathering attributes of benchmark binaries, including size, domain (network or system), static compilation, and compiler. We then ran each tool

on the binaries and found the count of findings for each. We then fit a model for each tool where the response variable was the number of findings and the explanatory variables were the attributes of the binaries. After assessing the data and assumptions of models, we found that a different type of statistical model was required for each tool. Each tool also had different significant predictor variables, implying that different attributes must be considered for different tools. We found that all attributes aside from compiler were correlated with at least one tool's output count. This leads to the conclusion that when selecting a benchmark repository, all of these attributes should be considered aside from the compiler. We acknowledge that there are other important variables that are not considered in this study such as the age of the binary. Because we have different findings for each tool, it is clear that more study of these relationships for additional tools, binaries, and attributes would bring more clarity to this process. However, we do know that it is important to consider the makeup of the benchmark repository as it is put together and as the model is interpreted.

The sensitivity of the model's TQI to weighting is quite large. We showed that in most cases the model's output can be dropped to 0 by changing the weighting of the lower layers, meaning that the weighting must be handled with care. Additionally, we analyzed the impact of weighting at the highest layer, where the TQI score is limited to the range of the lowest valued quality aspect and the highest valued quality aspect.

The impact of individual findings is also investigated. We performed this analysis by running an initial analysis on a binary, then injecting a vulnerability and re-running the vulnerability. We do this for every diagnostic in the model. This give a resulting change in TQI for each finding, allowing us to identify high and low impact findings. This also brings context to interpretation and what changes in TQI should be concerning.

We have designed, developed, and validated a model for the assessment of security quality in binary files. Due to the lack of similar models, we have successfully improved our

ability to assess the security quality of a binary without needing to compare our model to others. There will always be additional work to be done in validating this model further, but the applications and validation undertaken to date are very promising.

REFERENCES

[1] Ehsan Aghaei, Waseem Shadid, and Ehab Al-Shaer. ThreatZoom: CVE2CWE using Hierarchical Neural Network. Technical report.

[2] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[3] Hirotogu Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected papers of hirotugu akaike*, pages 199–213. Springer, 1998.

[4] Kenneth Alperin, Allan Wollaber, Dennis Ross, Pierre Trepagnier, and Leslie Leonard. Risk prioritization by leveraging latent vulnerability features in a contested environment. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 49–57, 2019.

[5] Omer Aslan and Refik Samet. A Comprehensive Review on Malware Detection Approaches. *IEEE Access*, 8:6249–6271, 2020.

[6] Andrew Austin and Laurie Williams. One technique is not enough: A comparison of vulnerability discovery techniques. *International Symposium on Empirical Software Engineering and Measurement*, pages 97–106, 2011.

[7] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2015.

[8] F. Hutton Barron and Bruce E. Barrett. Decision Quality Using Ranked Attribute Weights. *http://dx.doi.org.proxybz.lib.montana.edu/10.1287/mnsc.42.11.1515*, 42(11):1515–1523, nov 1996.

[9] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, 2:528–532, 1994.

[10] Luca Caviglione, Michal Choras, Igino Corona, Artur Janicki, Wojciech Mazurczyk, Marek Pawlicki, and Katarzyna Wasielewska. Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection. *IEEE Access*, pages 5371–5396, 2020.

[11] Kai Cheng, Qiang Li, Lei Wang, Qian Chen, Yaowen Zheng, Limin Sun, and Zhenkai Liang. DTaint: Detecting the Taint-Style vulnerability in embedded device firmware. *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018*, pages 430–441, 2018.

[12] Fred Cohen. Computer viruses. Theory and experiments. *Computers and Security*, 6(1):22–35, feb 1987.

[13] Drew Davidson, Benjamin Moench, Somesh Jha, Thomas Ristenpart, Drew Davidson, and Thomas Ristenpart. FIE on Firmware. *Proceedings of the 22nd USENIX Security Symposium*, pages 463–478, 2013.

[14] Xiaoning Du, Bihuan Chen, Yuekang Li, Jianmin Guo, Yaqin Zhou, Yang Liu, and Yu Jiang. LEOPARD: Identifying Vulnerable Code for Vulnerability Assessment Through Program Metrics. *Proceedings - International Conference on Software Engineering*, 2019-May:60–71, 2019.

[15] Katheryn A Farris, Ankit Shah, George Cybenko, Rajesh Ganesan, and Sushil Jajodia. VULCON: A system for vulnerability prioritization, mitigation, and management. *ACM Transactions on Privacy and Security*, 21(4), 2018.

[16] Barbara Filkins, Doug Wylie, and Jason Dely. 2019 State of OT/ICS Cybersecurity Survey. *SANS Institute*, (June), 2019.

[17] William Fleshman, Edward Raff, Richard Zak, Mark McLean, and Charles Nicholas. Static malware detection and subterfuge: Quantifying the robustness of machine learning and current anti-virus. *arXiv*, pages 3–12, 2018.

[18] Christian Frühwirth and Tomi Männistö. Improving CVSS-based vulnerability prioritization and response with context information. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, pages 535–544, 2009.

[19] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A Practical Model for Measuring Maintainability. pages 30–39, 2007.

[20] Joseph M. Hilbe. *Negative Binomial Regression*. Cambridge University Press, 2 edition, 2011.

[21] Hannes Holm and Khalid Khan Afridi. An expert-based investigation of the Common Vulnerability Scoring System. *Computers and Security*, 53:18–30, 2015.

[22] Hannes Holm, Mathias Ekstedt, and Dennis Andersson. Empirical analysis of system-level vulnerability metrics through actual attacks. *IEEE Transactions on Dependable and Secure Computing*, 9(6):825–837, 2012.

[23] Hannes Holm, Teodor Sommestad, Jonas Almroth, and Mats Persson. A quantitative evaluation of vulnerability scanning. *Information Management and Computer Security*, 19(4):231–247, 2011.

[24] Chien-Cheng Huang, Feng-Yu Lin, Frank Yeong-Sung Lin, and Yeali S Sun. A novel approach to evaluate software vulnerability prioritization. *The Journal of Systems and Software*, 86:2822–2840, 2013.

[25] ISO/IEC 25010:2011 Systems and software engineering: Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models. Standard, International Organization for Standardization, Geneva.

[26] Ver Hoef JM and Boveng PL. Quasi-Poisson vs. negative binomial regression: how should we model overdispersed count data? *Ecology*, 88(11):2766–2772, nov 2007.

[27] Pontus Johnson, Robert Lagerstrom, Mathias Ekstedt, and Ulrik Franke. Can the common vulnerability scoring system be trusted? A Bayesian analysis. *IEEE Transactions on Dependable and Secure Computing*, 15(6):1002–1015, 2018.

[28] James A. Kupsch, Elisa Heymann, Barton Miller, and Vamshi Basupalli. Bad and good news about using software assurance tools. *Software - Practice and Experience*, 47(1):143–156, 2017.

[29] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security and Privacy*, 9:49–51, 2011.

[30] Quan Le, Oisín Boydell, Brian Mac Namee, and Mark Scanlon. Deep learning at the shallow end: Malware classification for non-domain experts. *Proceedings of the Digital Forensic Research Conference, DFRWS 2018 USA*, 26:S118–S126, 2018.

[31] Zhiyi Li, Mohammad Shahidehpour, and Farrokh Aminifar. Cybersecurity in Distributed Power Systems. *Proceedings of the IEEE*, 105(7):1367–1388, 2017.

[32] Stephen Mathezer. Introduction to ICS security Part 2. 2015.

[33] Luallen Matthew. Breaches on the Rise in Control Systems: A SANS Survey. *SANS Institute*, (April):31, 2014.

[34] Stephen McLaughlin, Charalambos Konstantinou, Xueyang Wang, Lucas Davi, Ahmad Reza Sadeghi, Michail Maniatakos, and Ramesh Karri. The Cybersecurity Landscape in Industrial Control Systems. *Proceedings of the IEEE*, 104(5):1039–1057, 2016.

[35] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. A comparison of software design security metrics. *ACM International Conference Proceeding Series*, (c):236–242, 2010.

[36] Thomas Panas and Daniel Quinlan. Techniques for software quality analysis of binaries: Applied to Windows and Linux. *DEFECTS 2009 - Proceedings of the 2nd International Workshop on Defects in Large Software Systems, Held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2009*, (May):6–10, 2009.

[37] Edward Raff, Jared Sylvester, and Charles Nicholas. Learning the PE header, malware detection with minimal domain knowledge. *arXiv*, pages 121–132, 2017.

[38] Edward Raff, Richard Zak, Russell Cox, Jared Sylvester, Paul Yacci, Rebecca Ward, Anna Tracy, Mark McLean, and Charles Nicholas. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques*, 14(1):1–20, 2018.

[39] Alex Ramos, Marcella Lazar, Raimir Holanda Filho, and Joel J.P.C. Rodrigues. Model-Based Quantitative Network Security Metrics: A Survey. *IEEE Communications Surveys and Tutorials*, 19(4):2704–2734, 2017.

[40] David Rice. An extensible, hierarchical architecture for analysis of software quality assurance. Master's thesis, Montana State University, 12 2020.

[41] Manuel Rudolph and Reinhard Schwarz. A critical survey of security indicator approaches. *Proceedings - 2012 7th International Conference on Availability, Reliability and Security, ARES 2012*, pages 291–300, 2012.

[42] Thomas Saaty. Decision making with the analytic hierarchy process. *Int. J. Services Sciences Int. J. Services Sciences*, 1:83–98, 01 2008.

[43] Riccardo Scandariato, James Walden, and Wouter Joosen. Static analysis versus penetration testing: A controlled experiment. *2013 IEEE 24th International Symposium on Software Reliability Engineering, ISSRE 2013*, pages 451–460, 2013.

[44] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware. *Proceedings of the Network and Distributed System Security Symposium, NDSS 2015*, (February):8–11, 2015.

[45] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. SOK: (State of) the Art of War: Offensive Techniques in Binary Analysis. *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, pages 138–157, 2016.

[46] Miltiadis G. Siavvas, Kyriakos C. Chatzidimitriou, and Andreas L. Symeonidis. QATCH - An adaptive framework for software product quality assessment. *Expert Systems with Applications*, 86:350–366, 2017.

[47] Diomidis Spinellis. Reliable identification of bounded-length viruses is NP-complete. *IEEE Transactions on Information Theory*, 49(1):280–284, jan 2003.

[48] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Syst. J.*, 13(2):115–139, June 1974.

[49] Keith Stouffer, Victoria Pillitteri, Suzanne Lightman, Marshall Abrams, and Adam Hahn. Guide to Industrial Control Systems (ICS) Security NIST Special Publication 800-82 Revision 2. *NIST Special Publication 800-82 rev 2*, pages 1–157, 2015.

[50] Melanie Tupper and A. Nur Zincir-Heywood. VEA-bility security metric: A network security analysis tool. In *ARES 2008 - 3rd International Conference on Availability, Security, and Reliability, Proceedings*, pages 950–957, 2008.

[51] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. Robust Intelligent Malware Detection Using Deep Learning. *IEEE Access*, 7:46717–46738, 2019.

[52] Stefan Wagner, Andreas Goeb, Lars Heinemann, Michael Kläs, Constanza Lampasona, Klaus Lochmann, Alois Mayr, Reinhold Plösch, Andreas Seidl, Jonathan Streit, and Adam Trendowicz. Operationalised product quality models and assessment: The Quamoco approach. *Information and Software Technology*, 62(1):101–123, 2015.

[53] Stefan Wagner, Klaus Lochmann, Lars Heinemann, Michael Kläs, Adam Trendowicz, Reinhold Plösch, Andreas Seidl, Andreas Goeb, and Jonathan Streit. The Quamoco Product Quality Modelling and Assessment Approach. Technical report.

[54] Stefan Wagner, Klaus Lochmann, Sebastian Winter, Florian Deissenboeck, Elmar Juergens, Lars Heinemann, Michael Kläs, Adam Trendowicz, Jens Heidrich, Reinhold Ploesch, Andreas Goeb, Christian Koerner, and Christian Schubert. The Quamoco Quality Meta-Model.

[55] Theodore J. Williams. The Purdue enterprise reference architecture. *Computers in Industry*, 24(2-3):141–158, sep 1994.