REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
DECEMBER 2020	TECHNICAL PAPER	JUN 2020 – AUG 2020

4. TITLE AND SUBTITLE

AUTOMATED ANALYSIS OF SECURITY-RELATED SYSTEM REQUIREMENTS SPECIFICATIONS

5a. CONTRACT NUMBER
FA8750-20-3-1004

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Viktoria Koscinski
Mark Zappavigna
Jennifer Cassetti

5d. PROJECT NUMBER
ORTA

5e. TASK NUMBER
SI

5f. WORK UNIT NUMBER
TK

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Rochester Institute of Technology
1 Lomb Memorial Dr,
Rochester NY 14623

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/Information Directorate
Rome Research Site/RIGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TP-2020-001

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA Case Number: 88ABW-2020-2792
DATE CLEARED: 09/11/2020

13. SUPPLEMENTARY NOTES

14. ABSTRACT

It is crucial to protect mission-critical software systems by building in security from the ground up. Despite this, it is both expensive and error-prone to create adequate security-focused system requirements specifications (SysRS). Analyzing the requirements brings an additional challenge since requirements engineers are not necessarily security experts. Therefore, an automated security-related SysRS analysis framework will reduce the cost, time, and other resources related to incorporating security into software, while allowing security to be built-in early in the development cycle. This report describes the previous, current, and future work related to the creation of an automated natural language processing-based SysRS analysis technique. It explores the elements required to build such a framework and discusses both preliminary results and future efforts associated with building this analysis framework.

15. SUBJECT TERMS
Security-Related System Requirements Specifications (SysRS), automated analysis, automated SysRS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**JENNIFER CASSETTI**
U	U	U	UU	12	19b. TELEPHONE NUMBER (*Include area code*) N/A

Automated Analysis of Security-Related System Requirements Specifications

Viktoria Koscinski
vk2635@rit.edu

Mark Zappavigna
mark.zappavigna@us.af.mil

Jennifer Cassetti
jennifer.cassetti.1@us.af.mil

August 7th, 2020

Abstract

It is crucial to protect mission-critical software systems by building in security from the ground up. Despite this, it is both expensive and error-prone to create adequate security-focused system requirements specifications (SysRS). Analyzing the requirements brings an additional challenge since requirements engineers are not necessarily security experts. Therefore, an automated security-related SysRS analysis framework will reduce the cost, time, and other resources related to incorporating security into software, while allowing security to be built-in early in the development cycle. This report describes the previous, current, and future work related to the creation of an automated natural language processing-based SysRS analysis technique. It explores the elements required to build such a framework and discusses both preliminary results and future efforts associated with building this analysis framework.

1 Introduction

Mission-critical software systems are essential to the operations of their organizations. Not only do these software systems tend to be complex and extensive, but it is crucial for them to have built-in security. Validating, verifying, and adding security-focused system requirements specifications (SysRS) after systems are built is error-prone and expensive due to their scale and complexity. Therefore, it is important for security to be built into these systems from the ground up. This is often resource-heavy as well, and requires the support of security experts and specialized tools. The costs associated with incorporating security may discourage organizations from sufficiently prioritizing the security of critical software.

Existing SysRS analysis techniques are either manual in nature or focus on functional SysRS not related to security. Additionally, requirements engineers are not necessarily security experts, who have the most up to date information on new vulnerabilities and weaknesses. As a result, security SysRS are prone to being under-specified. *Under-specified* SysRS do not adequately protect against possible attacks because they remain too abstract and not measurable [1]. They lack mitigation techniques or details, which results in vulnerabilities [2]. Furthermore, organizations such as the Air Force deal with a large volume of SysRS. Therefore, it is necessary to develop automated SysRS analysis techniques that can increase the productivity and efficiency of requirements engineers.

The goal of this project is to create an automated technique that can analyze security SysRS, detect specification defects, identify those SysRS that are under-specified, and inform requirements engineers about attacks that are possible based on SysRS. An effective way to inform requirements engineers about possible attacks early in the development cycle is by using attack scenarios. Attack scenarios provide detailed descriptions of security attacks, including information such as who may exploit a security vulnerability, why an attacker would exploit the vulnerability (what resources they would gain), as well as what vulnerabilities an attacker would exploit and how. Through their integration with the development process's notation and concepts, attack scenarios lead to SysRS analysis and attack prevention.

1.1 Research Objectives

This project aims to develop an automated approach that can enable requirements analysts to generate attack scenarios and bring security analysis to very early stages of software development. Particularly, this technique relies on natural language processing (NLP) techniques, SysRS formalization, and formal reasoning. The approach, shown in Figure 1, is able to take SysRS in natural language (NL) and analyze them based on a library of attack scenarios to detect under-specified security SysRS and inform requirements engineers about potential attacks. These techniques will create a tool that will not only detect specification deficiencies in existing SysRS, but will also be able to identify missing requirements and under-specified SysRS.

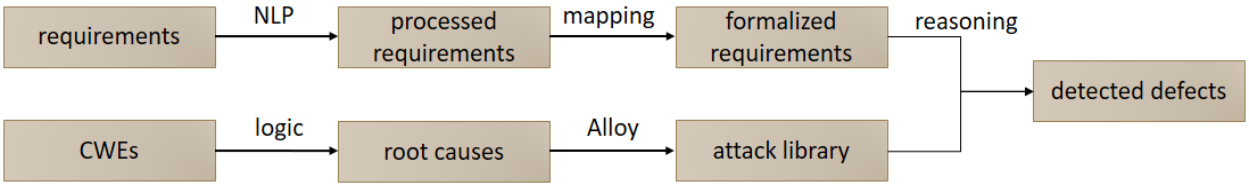

Figure 1: The process for conducting automatic analysis on SysRS is shown.

The creation of this technique is centered around following major tasks:

1. **Obtaining security-related SysRS for testing:** It is important to choose security SysRS related to multiple domains to create a generalized security SysRS analysis tool.
2. **Development of an NLP-based SysRS analysis approach:** This involves processing SysRS using NLP techniques to obtain all the relevant information entities in the specification. NLP techniques will allow the identification of under-specified SysRS.
3. **Formalization of SysRS:** Information obtained from the SysRS must be formalized in order to detect SysRS deficiencies based on the attack scenario library. This task involves mapping processed SysRS to possible elements from the attack library.
4. **Creation of a generic attack library:** A library of attack scenarios is used to detect deficiencies in security SysRS. This comprehensive attack library is expressed using a formal specification language.
5. **Automatic detection of specification defects:** The analysis is based on detecting under-specified requirements and generating attack scenarios based on the attack library. A first-order logic-based model checking tool is used to identify potential weaknesses in the model.

2 Preliminary Work

Because this report focuses on the most recent work related to the project (namely, the testing and comparison of NLP techniques and the creation of a formalized attack library), this section includes a summary of work that has been accomplished prior. Particularly, it describes the purpose and functionality of the created custom NLP-based SysRS analysis approach.

2.1 Development of an NLP-Based SysRS Analysis Approach

In order to conduct automated reasoning on NL requirements, it is necessary to process these requirements using NLP techniques, such as information extraction (IE). IE extracts structured information from unstructured or semi-structured data. Typically, IE tools categorize data into a relation tuple to reveal information about the data. For example, the data may be represented as a relation triple, featuring the subject, relation, and object of a sentence. This tuple attempts to express the meaning of a sentence in a structured manner.

Two popular IE techniques that were explored were Stanford University's *Open IE* [3] and the University of Washington's *Ollie* [4]. Both Open IE and Ollie extract information in the form of (subject; relation; object)

triples. These tools were tested using a set of 183 security-related SysRS. An example of some SysRS and extractions is shown in the first column of Figure 4. Testing Open IE on these SysRS quickly proved that Open IE contains limitations preventing it from accurately extracting information from SysRS. Open IE has trouble processing statements with negations, the phrase "shall be," or "if" clauses.

Ollie performed better than Open IE, since it allows for negations and has a fourth (optional) *enabler* field that handles clauses denoting conditions, such as "if" and "when." Rating Ollie's performance using the criteria described in section 3.1.2 shows that Ollie incorrectly extracted information from 50.8% of the dataset, and at least partially correctly extracted information from 49.2% of the dataset. Although Ollie performed better than Open IE, Ollie is a learning-based information extraction tool, and would therefore need to be trained on large amounts of SysRS data in order to further improve. Additionally, it only structures information into up to four categories: (subject; relation; object)[enabler].

In order to create a technique that is more specific to the domain of security-related SysRS, we have developed a custom IE approach. This approach is built as a Stanford CoreNLP [5] pipeline and extracts information from SysRS based on rules applied to their dependency parses. A dependency parse provides information about how words of a sentence are related, including both part-of-speech tags and grammatical dependencies. As shown in figure 2, *expire* is tagged as a verb (VB) and *passwords* is tagged as a plural noun (NNS). Additionally, when it comes to grammatical dependencies, *passwords* is the nominal subject (nsubj) of *expire*, and *user* and *account* form the compound phrase *user account*, which modifies the word *passwords*.

```
Example: dependency parse
-> expire/VB (root)
  -> passwords/NNS (nsubj)
    -> User/NN (compound)
    -> account/NN (compound)
  -> shall/MD (aux)
  -> months/NNS (nmod:after)
    -> after/IN (case)
    -> 12/CD (nummod)
  -> ./. (punct)
```

Figure 2: This is the dependency parse of the sentence "User account passwords shall expire after 12 months." It shows the part-of-speech tag and dependency relation of each word.

Using the dependency parse of a given NL sentence, information was extracted from that sentence into six categories, rather than three, in order to provide a more detailed extraction. As shown in Figure 3, these categories are the *subject descriptor*, *subject*, *relation*, *case* (such as prepositions and postpositions), *object descriptor*, and *object*. Because a future component of the project is to formalize the extracted information so that it can be analyzed for potential deficiencies, providing more categories reduces potential ambiguity in choosing what parts of the subject, relation, or object of the sentence are relevant to the information that is needed for analysis.

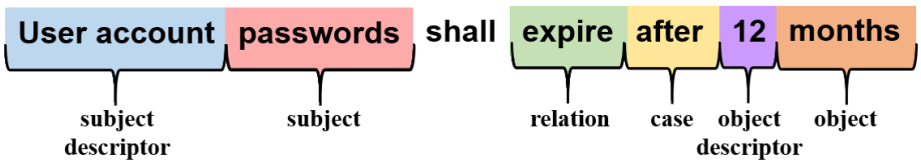

Figure 3: Information extraction on the sentence "User account passwords shall expire after 12 months" is conducted by categorizing the sentence into six different categories based on patterns found within the dependency parse in Figure 2.

Other than the six different information categories, the extraction of some clauses (such as *if*, *to*, and *so*) was implemented, meaning that that these clauses and their function in the sentence are recognized. Furthermore, the detection of verb negation was implemented, which is another essential component given the type of

data being analyzed. Finally, we have implemented some aspects of ambiguity detection. Particularly, the technique is able to identify whether a sentence has two verbs at the same "level" of priority. For example, "The system shall do X and Y" contains two main actions (*do X* and *do Y*). Results of this custom technique's performance are provided in section 3.1.2 and Figure 5.

3 Scope of Current Work

This section describes the most recent work related to the project. The focus of this section is on the testing of the NLP techniques and the creation of a formalized attack library, along with their respective sub-tasks.

3.1 Testing and Comparison of Natural Language Processing Techniques

Having previously created a custom NLP-based approach to conduct IE on NL SysRS, the original testing dataset of SysRS has been improved. Testing on both the custom IE approach and similar well-known off-the-shelf tools helps gauge and compare their performance.

3.1.1 Improving the SysRS Dataset for Testing

Before testing the IE techniques, the original dataset of 183 requirements has been improved in order to provide a more accurate representation of the SysRS that will be input to the IE tool. Although SysRS will not necessarily be well-written upon being analyzed, the first step (which is not addressed in this report) of the automatic analysis aims to inform requirements engineers about badly-written SysRS before they are analyzed by the IE tool so that the requirements engineers can improve those SysRS for accurate analysis. Once all SysRS are of an acceptable structure, input to the IE tool will be processed. In order to simulate this first step of analysis, a set of criteria has been created to filter the original dataset and provide a new dataset containing better-written input. The criteria consist of:

- **No change needed:** The requirement is grammatically correct, and does not need to be changed, with the exception of minor changes such as adding/removing punctuation.
- **Change needed:** The requirement needs significant grammatical or structural changes. For example, "The system shall do X and Y" should be modified to form two requirements: "The system shall do X" and "The system shall do Y."
- **Not usable:** The requirement is either irrelevant to security testing or it is written in a way that the proper way to express the idea is unknown.

The original security-related SysRS dataset was labeled and modified. The (subjectively) well-written or changed (now well-written) requirements were identified and compiled into a new list, which formed an improved dataset of 187 security-related SysRS.

3.1.2 Testing and Comparing Information Extraction Techniques

Once the improved dataset was created, this dataset was tested using three different IE approaches. The first tool tested was (the previously-mentioned) Ollie [4]. The dataset was tested using Ollie due to the fact that the original version of the dataset was also tested on this tool, as described in section 2.1. Testing was also conducted on the previously-created custom IE approach. This is due to the fact that the custom approach was designed to address certain weaknesses in both Ollie and Open IE to provide a more detailed output and to be easily adjustable for further improvements. The third tool the dataset was tested on was ReVerb [6], a comparable predecessor tool for Ollie. Due to the limitations also described in section 2.1, this dataset has not been tested on Stanford's Open IE tool.

Once results were obtained for each of the three methods, the performance of each of the methods needed to be analyzed. Because of the subjective nature of NL (demonstrated by the fact that different people may have different interpretations of the same sentence), the outputs of each requirement were manually analyzed. In order to provide a more objective analysis of the results, multiple members of the research

group were involved in the analysis. A set of criteria has been created in order to aid with the analysis of the output of each of the tools. These criteria are as follows:

- **Good:** All of the information in the requirement is captured by the extraction and is correctly extracted (categorized into the IE categories of the tool).
- **Ok:** Some of the information in the requirement is captured. Any information that is extracted, is correctly extracted.
- **Bad:** The information is incorrectly extracted. The meaning of the extracted output is different than the original intended meaning of the requirement, or does not make sense at all.
- **Error:** No extraction was found. Tools designed to always find an extraction will not contain this category, but will have a greater number of *bad* extractions.

An example of requirements with each of the four extraction outputs and possible labels based on the criteria above is shown in Figure 4. This example represents the Ollie tool, due to the fact the the output of Ollie is simple and intuitive to understand.

Natural langauge requirement **(subject, relation, object) extraction**	**Rating**
User account passwords shall include a minimum of two different types of characters. (User account passwords; shall include; a minimum of two different types of characters)	good
The system shall support role-based access for security controls. (The system; shall support; role-based access)	ok
The system shall allow adjusters with a supervisor role to update preferred repair facility ratings. (The system; shall allow; adjusters)	bad
Authorized personnel only shall be authorized to access sales information. No extractions found.	error

Figure 4: An example of each of four information extraction results and their corresponding ratings on requirements analyzed using Ollie is shown.

The results of each of the tools on all of the requirements are shown in Figure 5. These results provide three primary insights on the dataset and IE methodology. First, they show that the updated requirements dataset is indeed better-written than the original. Rating Ollie's performance on the original dataset resulted in close to 50% *bad* or *error* ratings, and 50% *good* or *ok* ratings (see section 2.1), with 25 of the requirements resulting in an *error* output. Currently, the Ollie results have close to 40% *bad* or *error* results, and 60% *good* or *ok* results. Testing the datasets on the same tool shows close to a 10% improvement in results for the modified dataset. Second, these results show that all three tools have a similar performance to each other. This means that for future tasks, whichever tool provides the most appropriate output and/or can be modified to accomplish the task best can be chosen for that particular task. Third, we have noticed that the tools tend to have similarities in terms of what types of phrases they interpret incorrectly. For example, each tool tends to have trouble with the phrase "shall be able to." As a result, these trends can be further investigated in order to improve their performance on phrases containing these "problem" phrases.

3.2 Creating and Evaluating a Formalized Attack Library

In order for security SysRS to be analyzed, it is necessary to create a comprehensive library of potential attacks. By formally specifying these attacks in terms of their root causes, it is possible to conduct reasoning on these attacks based on the processed NL SysRS and inform requirements engineers about potential attacks that may occur on the system.

		#	# (%)			#	# (%)
Ollie	error	20	75 (40.1%)	Ollie	error	20	78 (41.7%)
	bad	55			bad	58	
	ok	49	112 (59.9%)		ok	41	109 (58.3%)
	good	63			good	68	
Custom Approach	error	0	79 (42.2%)	Custom Approach	error	0	66 (35.3%)
	bad	79			bad	66	
	ok	68	108 (57.8%)		ok	69	121 (64.7%)
	good	40			good	52	
ReVerb	error	27	72 (38.5%)	ReVerb	error	27	61 (32.6%)
	bad	45			bad	34	
	ok	69	115 (61.5%)		ok	67	126 (67.4%)
	good	46			good	59	

Figure 5: These are the results of rating all three tested information extraction methods: Ollie, the custom approach, and ReVerb. The two columns of results represent ratings by two different members of the research group.

3.2.1 Formalizing a Set of Potential Weaknesses

In order to create a model of potential attacks, we use the *Common Weakness Enumeration* (CWE), which is a government- and community-supported list of common software and hardware weaknesses[1]. This list of common weaknesses (referred to as CWEs) contains descriptions and examples of how weaknesses can occur, potential risks and consequences, mitigation techniques, as well as other details.

In order to construct the attack library, a list of 34 CWEs was constructed to cover a wide range of potential weaknesses. The list contains the CWE ID number, name, preconditions, environment, root causes, and consequences. The new dataset of SysRS that has been created, as mentioned in section 3.1.1, has been labeled with related CWEs from this list. A requirement is related to a CWE if the requirement indicates that the CWE must be checked. For example, because the first requirement in Figure 4 implies that the system uses passwords, any CWE relating to passwords would be related to this requirement. To prepare the CWEs for formal modeling, the root cause of each of the CWEs was converted into a logic form. This was done by examining the root causes of the CWEs based on the current list, as well as reviewing the CWE documentation, to create a set of relations representing the possible causes for each CWE. An example of the logic representation for some CWEs is shown in Figure 6.

ID	Name	Root Causes
262	Not Using Password Aging	!has(password, expirationDate) OR !has(password, reuseRestriction)
324	Use of a Key Past its Expiration Date	!has(key, expirationDate)
521	Weak Password Requirements	!has(password, lengthRestriction) OR !has(password, reuseRestriction) OR !has(password, commonRestriction) OR !has(password, contextualStringRestriction)
523	Unprotected Transport of Credentials	!(transportOf(credentials) -> useOf(SSL))
603	Use of Client-Side Authentication	performs(client, authentication AND !performs(server, authentication)

Figure 6: An example of CWE root causes in their logic form is shown. Logic operators, such as *and*, *or*, *not*, and *implication* are used.

[1] https://cwe.mitre.org/

3.2.2 Modeling Formalized Attacks

With the CWEs in logic form, 15 of these CWEs were modeled using the *Alloy* specification language[2]. The modeled CWEs contain the following ID numbers: *256, 258, 262, 287, 291, 293, 307, 308, 317, 324, 521, 523, 602, 603*, and *836*. Alloy is a declarative specification language that is capable of describing the structural constraints and behavior of a software system. Alloy's modeling tool is based on first-order logic which allows models to be automatically checked for correctness. The Alloy Analyzer is an automated model checker built atop a boolean SAT solver. The analyzer provides a few solvers to choose from when running a model, the default being the *SAT4J*[3] solver. We chose to use Alloy to model the CWEs because of its modeling ability combined with its ability to conduct automatic reasoning on a model. For a better understanding, some of Alloy's syntax and structure is introduced below:

- *sig* {} represents what is known in Alloy as a signature. Signatures are similar to classes in the object oriented paradigm. Each signature can contain relations describing the behavior or structure of the object. Signatures can inherit relations from each other. An instance of a signature is visually represented as a node in an Alloy graph.
- *abstract sig* {} represents an abstract class, which can not be instantiated. Typically, regular signatures inherit relations from abstract signatures.
- Multiplicity describes how many instances of a object there may be or how many relations one object may have to another. Common multiplicities are *one*, *lone* (zero or one), *set* (zero or more), and *some* (one or more).
- *fact* {} describes a statement that always holds true.
- The *run* {} command conducts automatic reasoning on the model and finds possible instances based on any criteria the user (optionally) provides within the *run* command.

An example of the Alloy code for a simple model is shown in Listing 1. This model represents CWE 521: *Weak Password Requirements*. The model contains objects representing the CWE, password, as well as possible restrictions a password may have. Within the *run* statement, criteria is provided to ensure that the *Password* object has the relation *has_restriction* with both the *Length* and *Reuse* objects. While this model is rather simple, the complexity of the model increases rapidly as more CWEs are added.

```
1  abstract sig Authentication_Method {}
2
3  one sig Password extends Authentication_Method {
4          has_restriction: set Restriction,
5  }
6
7  abstract sig Restriction {}
8  one sig Length, Reuse, Common, Contextual_String extends Restriction {}
9
10 abstract sig CWE {}
11 one sig CWE521 extends CWE {
12         affected_by: lone Password
13 }
14
15 // CWE 521 - Weak Password Requirements
16 fact CWE521{not(Length in Password.has_restriction and Reuse in Password.has_restriction and Common
       ↪ in Password.has_restriction and Contextual_String in Password.has_restriction)
17 iff (Password in CWE521.affected_by)}
18
19 run {Length in Password.has_restriction and Reuse in Password.has_restriction}
```

Listing 1: This Alloy code describes a model which contains a password and possible restrictions on that password. The *fact* describes conditions under which CWE 521 occurs. The *run* statement describes relations that express the *Password* object having restrictions on *Length* and *Reuse*.

A visual representation of the model described in Listing 1 is shown in Figure 7. Because the *run* command includes criteria that ensure that both *Length* and *Reuse* are restrictions of the *Password* object (related by the *has_restriction* relation), all of the models in the figure contain those relations. Given the relatively small

[2]https://alloytools.org/
[3]http://www.sat4j.org/

size of the model, there are only four possible iterations that can occur with these criteria in place. These are as follows: in iteration (a) the password contains solely the restrictions specified in the *run* statement, in iterations (b) and (c) the password contains those restrictions and either one of the other two (unspecified) restrictions, and in iteration (d) the password contains all four of the possible restrictions. If our criteria in the *run* statement has not been provided, we would also see the iteration in which the *Password* object has no restrictions, all possible iterations in which the *Password* object has only one restriction, as well as all possible iterations in which the *Password* object has any two or three restrictions.

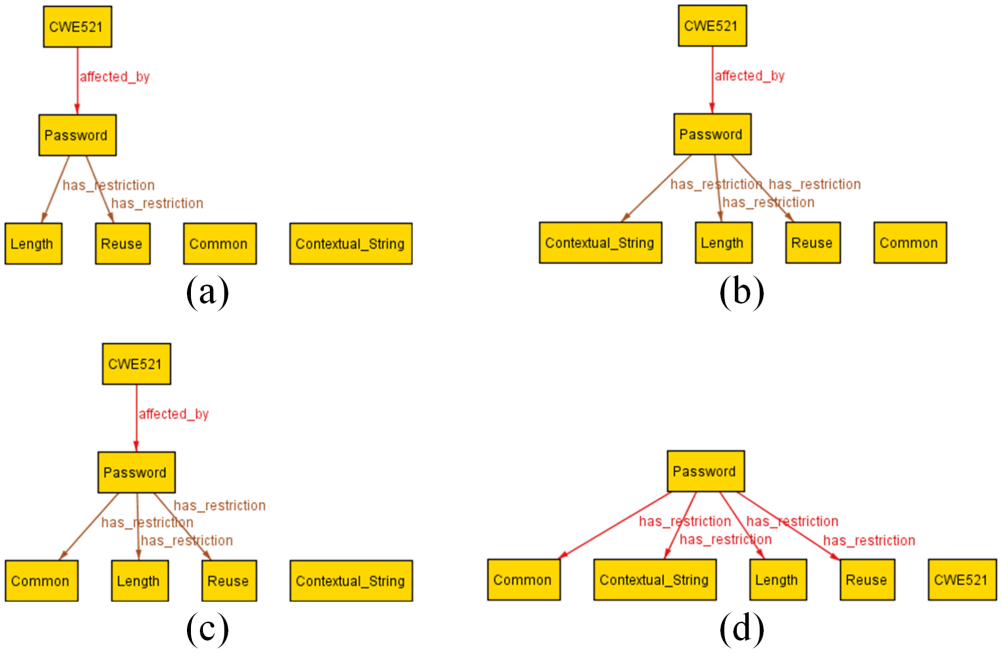

Figure 7: This image shows all iterations of the model described in Listing 1. The nodes of the graph (yellow boxes) represent the objects (instances of signatures) in the model, while the edges represent the relationships between the objects. Note that the only iteration in which the CWE does not occur (does not have a relation to the password) is iteration (d), in which the password has all four restrictions preventing the weakness from occurring.

The model created in which all 15 CWEs are represented is shown in Figure 8. This model is much more complex than the simple model shown in Figure 7. There are two main aspects contributing to the complexity of a given model. The first is the number of objects itself. Figure 8 only shows the 21 objects containing relations for the sake of preserving space. However, there are an additional 15 objects (not visualized) in this model that do not have any relations with each other or any of the other objects. Clearly, the more objects and relations the model can contain, the more iterations of that model there will be. For example, the image of the model from Figure 8 shows only one of the many iterations, which is at least 600 at which time we terminated the solver.

```
1  run{Security_Check in Server.performs and
2          Plaintext in Sensitive_Data.stored_as and
3          Credentials in SSL.use_for_transport_of and
4          GUI in Sensitive_Data.stored_as}
```

Listing 2: This run statement introduces four criteria (constraints) into an Alloy model.

The second, less obvious aspect relating to model complexity is the number of criteria included in the *run* statement itself. In Listing 2, an example *run* statement containing only four criteria is provided to the 15-CWE model (which contains 36 individual objects). Any criteria not specified are left to the analyzer's interpretation. Because Alloy generates every possible (non-equivalent) interpretation of a model within its scope, running a model with a small number of criteria (compared to the number of objects and relations) results in added complexity. As seen in Figure 8, the model shows objects that may not be relevant to its *run* statement in Listing 2. We greyed out the irrelevant objects for visualization purposes; all objects look

the same when running a model and this level of detail is not automatically provided to the user. Therefore, the requirements engineer must look through all of the CWEs and other objects in the model and manually decide whether or not each one is related to what the engineer is looking for.

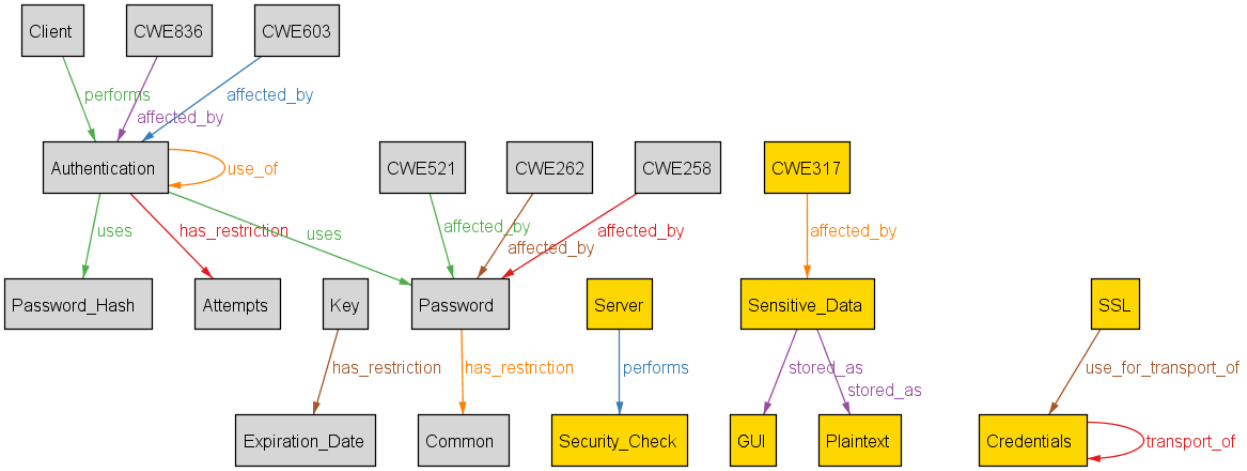

Figure 8: This image shows one iteration of a model containing 15 CWEs. Only objects containing relations are shown. Only the yellow nodes are relevant to the criteria in Listing 2. The irrelevant nodes have been greyed out manually for visualization purposes (they also appear as yellow in the original model).

In order for the model to be efficiently used by requirements engineers, the problems of too many iterations and of too many irrelevant objects both need to be addressed. The problem of too many iterations is automatically addressed when more criteria is included in the *run* statement. Because a SysRS document will contain many specifications, there will naturally be more criteria included in the *run* statement than what was provided in the example shown in Listing 2. To further reduce the number of iterations and to reduce the number of objects shown to only the relevant ones, immediate future work includes modifying the attack model to contain signatures of the *lone* multiplicity rather than the *one* multiplicity. At a first glance, this would actually increase the number of iterations of the model due to the fact that it allows all combinations of each object either appearing or not appearing. However, the goal is to enforce a rule so that only relevant objects appear and irrelevant ones do not, therefore reducing both the number of iterations and the number of irrelevant objects that can occur. To create this more relevant model, a mapping function will need to automatically recognize relevant entities in SysRS. More details about this are described in section 4.3.

4 Discussion and Future Work

Each of the five main tasks related to the project (described in section 1.1) is quite extensive in terms of what must be achieved in order to accomplish the task. For instance, obtaining a dataset of security-related SysRS for testing involves not only filtering out and labeling security-related SysRS, but also ensuring that these are of the format that will be input into the NLP tool. The NLP-based analysis approach must take into account the information that will be needed when conducting reasoning on this data with the attack library. Formalizing the SysRS must also take into account the format of the weaknesses modeled in the attack library. Creating the attack library itself involves not only investigating possible attacks to model based on the CWEs, but also the modeling tool itself (Alloy). Finally, automatically detecting specification defects requires both the potential attacks and the processed SysRS to be formalized and expressed in a language that the reasoner understands. The following subsections describe future work that will aid in accomplishing these important tasks of the project.

4.1 Analyzing Security SysRS Using Natural Language Processing

As mentioned in section 3.1.2, the IE tools show certain trends regarding which phrases in the SysRS they tend to have trouble with. For example, each rater noticed that every tool has trouble correctly extracting information from sentences containing the phrase "shall be able to." There were also instances in which only one or two of the tools had trouble with a particular phrase. Because each tool's output differs, we will choose the tool that provides output in a way which allows the output to be formalized according to the attack library and reasoning tool. Depending on which tool(s) will be used, "problem words/phrases" for the tool(s) will be investigated in order to improve the IE performance on those phrases.

4.2 Modeling Potential Weaknesses and Attacks Using a Formal Attack Library

As described in section 3.2, creating a formalized attack library involves first investigating possible attacks and vulnerabilities. Currently, 34 CWEs are expressed in formal logic. However, in the future, more CWEs will be formalized in order to extend the possible weaknesses covered. Additional government- and community-supported security resources, such as CAPEC[4] and Pre-ATT&CK[5], will be explored systematically in order to create a comprehensive attack library. Although currently only root causes are represented in the attack model, more detailed descriptions and consequences of attacks will be included in the future in order to output complete attack scenarios.

The Alloy model of the CWEs (described in section 3.2.2) will be modified to allow for objects irrelevant to the *run* statement to not be shown or analyzed. As more potential attacks will be added to the model, this will significantly reduce both the number of iterations of the model given specific criteria as well as the complexity of each iteration itself. In order to do this, specific concepts in the model will be designated so that those concepts can be checked for in the SysRS.

4.3 Mapping and Formalizing Security SysRS

In order to conduct reasoning on security-related SysRS and output potential attacks based on the SysRS and the attack library to the requirements engineer, the SysRS must first be mapped to a set of criteria which will form the *run* statement. For this mapping to occur, related concepts will be found in the SysRS through the use of keywords. These keywords will be included in a thesaurus of terms in which each related term will map to an equivalent keyword. Any concepts from the CWE model that are not found in the SysRS or implied by any of them will be set to not occur in the model. Existing concepts will be further mapped by checking their corresponding relations. These relations may also be mapped to keyword relations using a thesaurus and/or potentially *linguistic modality*, an NLP concept that shows how likely it is that an action occurs. Implications, conjunctions, and negations can be mapped directly, as the Alloy language supports these operators. Finally, these mappings will be used to construct the *run* statement which will output potential attacks after reasoning is conducted on the model.

5 Conclusion

This report describes the work that has been accomplished towards creating an automated security-related SysRS analysis technique. This technique will help reduce costs, time, and other resources associated with incorporating security into software, and will allow security to be incorporated early in the development cycle. This report described preliminary work associated with building this technique, such as the development of an NLP-based SysRS analysis approach, which provides a detailed extraction of the information found in a NL requirement. Additionally, this report describes the most recent work accomplished related to the project. This work includes improving the NL SysRS dataset, which allowed for the testing and comparison of IE techniques. The comparison shows that the custom IE technique works well compared to other well-known techniques, and provides insight into how the NLP technique(s) can be improved in the future. The creation and analysis of a formalized attack library is shown as well. This demonstrates the importance of creating

[4]https://capec.mitre.org
[5]https://attack.mitre.org

a comprehensive library using CWEs as well as information from other security resources. It also describes how the attack model is created and emphasizes the need for simplifying the model. Finally, the objectives of future work are discussed, along with how accomplishing them will improve the project. Overall, this report has demonstrated how various components will be combined in order to create an automated technique for the analysis of security-related SysRS.

References

[1] H. Villamizar, A. Anderlin Neto, M. Kalinowski, A. Garcia, and D. Méndez, "An approach for reviewing security-related aspects in agile requirements specifications of web applications," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 2019, pp. 86–97.

[2] S. S. Yau and Z. Chen, "A framework for specifying and managing security requirements in collaborative systems," in *Autonomic and Trusted Computing*, L. T. Yang, H. Jin, J. Ma, and T. Ungerer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 500–510.

[3] G. Angeli, M. J. J. Premkumar, and C. D. Manning, "Leveraging linguistic structure for open domain information extraction," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015, pp. 344–354.

[4] M. Schmitz, S. Soderland, R. Bart, O. Etzioni *et al.*, "Open language learning for information extraction," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012, pp. 523–534.

[5] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[6] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *Proceedings of the 2011 conference on empirical methods in natural language processing*, 2011, pp. 1535–1545.