Access Control Policy Generation From User Stories Using Machine Learning

John Heaps[1], Ram Krishnan[2], Yufei Huang[3], Jianwei Niu[1], and Ravi Sandhu[1]

[1] Institute for Cyber Security (ICS), NSF Center for Security and Privacy Enhanced Cloud Computing (C-SPECC), and Department of Computer Science
[2] ICS, C-SPECC, and Department of Electrical and Computer Engineering
[3] ICS and Department of Electrical and Computer Engineering
The University of Texas at San Antonio
{john.heaps,ram.krishnan,yufei.huang,jianwei.niu,ravi.sandhu}@utsa.edu

Abstract. Agile software development methodology involves developing code incrementally and iteratively from a set of evolving user stories. Since software developers use user stories to write code, these user stories are better representations of the actual code than that of the high-level product documentation. In this paper, we develop an automated approach using machine learning to generate access control information from a set of user stories that describe the behavior of the software product in question. This is an initial step to automatically produce access control specifications and perform automated security review of a system with minimal human involvement. Our approach takes a set of user stories as input to a transformers-based deep learning model, which classifies if each user story contains access control information. It then identifies the actors, data objects, and operations the user story contains in a named entity recognition task. Finally, it determines the type of access between the identified actors, data objects, and operations through a classification prediction. This information can then be used to construct access control documentation and information useful to stakeholders for assistance during access control engineering, development, and review.

Keywords: Access Control · Software Engineering · Agile Development · User Stories · Machine Learning · Deep Learning

1 Introduction

Agile development has gained great popularity in recent years among software development teams. It is able to rapidly produce and update software, and easily react to changes in requirements. However, it has been observed [2, 4, 14, 20] that agile development practices often facilitate the introduction and propagation of access control and other security vulnerabilities. Some such practices include the constant changes in code and requirements which drastically limits the ability to perform security assurance review; frequent code refactoring, changes in functional requirements, and modifications to system design which have a tendency to break security constraints of previously implemented functionality; and the

necessity of continuously delivering development iterations on time as well as a push for developing software as quickly as possible often take precedence over time-consuming security assurance activities.

One of the most notable reasons for the proliferation of access control and other security vulnerabilities is that agile development discourages producing comprehensive documentation about the software to be developed [9]. That is, it is normal that no security or access control policy is defined for the software before development begins. A primary reason for this is that agile development allows for changing requirements during development, unlike traditional development models. Since documentation is primarily derived from software requirements, a change in requirements leads to a review and update of all documentation to reflect those changes. Since requirements can change frequently throughout the agile development process, the act of constantly updating development documentation becomes time consuming and burdensome. Agile development elects to forego this work in the interest of time, and instead relies on the expertise of developers to make on-the-fly decisions during development. By pushing this responsibility on the developers and maintaining little to no documentation on decisions made throughout the development process, many opportunities for security vulnerabilities to arise are created.

There have been attempts [3, 5, 15, 17] to address how agile development may be modified to mitigate security issues and bugs by suggesting more documentation be produced by stakeholders. This documentation often focuses on creating and maintaining additional information for user stories, relieving developers from making on-the-fly security decisions. However, the creation and maintenance of the additional user story documentation still requires more time and labor than most development teams are willing to spend.

To help stakeholders mitigate the propagation of access control security vulnerabilities under the agile development model, we propose an automated approach to produce additional documentation so stakeholders have a more holistic view and common understanding of the access control of the software to be developed. Further, it will identify user stories with high ambiguity and allow stakeholders to refine user stories throughout the software development process. In this initial work, we will automatically identify and extract access control information from user stories and then visualize the access control information to stakeholders. This will also give product owners an overview of the access control so they may confirm or indicate changes to it. Ultimately, this approach will relieve developers from making most on-the-fly decisions, help reduce bugs and security vulnerabilities that may be overlooked, save time and money, and better protect the product owner and end users' information.

This initial work will focus specifically on the extraction and presentation of access control information from user stories. This only relates access control information to actors, or users, of the software. Further, due to the ambiguity and limited context of user stories and natural language, in order to determine the exact access control of the system some human involvement is necessary.

Ongoing work will incorporate active learning and human interactivity to best refine the access control model.

The contributions we present in the paper are as follows:

- A dataset of over 1600 user stories, labeled for three separate learning tasks related to the extraction of access control information.
- A transformer-based learning model that categorizes user stories into "contains access control information", "does not contain access control information", and "too ambiguous to determine if access control is present".
- A transformer-based learning model that performs a named entity recognition task that predicts if a word in a user story is an "actor" (or end user), a "data object", or an "operation" of the system to be built.
- A transformer-based learning model that predicts the type of access an actor has for a data object present in a user story.

The rest of the paper is organized as: Section 2 presents background information on user stories and related work; Section 3 describes our data and model on extracting access control information from user stories; Section 4 presents the results of our work; and Section 5 discusses the conclusion and future work.

2 Background

2.1 User Stories

The agile software development model was conceived in 2001 through a set of tenets and principles [9]. It broke from traditional software models in many ways, but primarily it allowed for changing software requirements throughout the software development process, discouraged spending time on comprehensive documentation, and focused on producing software at regular intervals and as fast as possible. Agile development quickly became popular as it was able to support the ever-faster changing environment of industry.

The only documentation that agile development requires are user stories. User stories define the requirements of the software to be built, and are often written by product owners (rather than software developers) usually making them more ambiguous and abstract than traditional software requirements. They help define software requirements by describing how actors, or end users, will interact with the system. They relate an actor of the system to system data objects and operations. A simple example of a user story would be, "As a system administrator, I want to create user accounts, so that new employees can use the system." The actor of this user story is the "system administrator" and is being related to the "user account" data object. From an access control perspective, this relation would be "create".

While not necessary, user stories are often written in some specified format, as shown in Figure 1. It is expected that the set of user stories changes during development based on the changing needs of the product owner causing different functionality to be integrated into the system with user stories added, deleted, or modified accordingly. Such changes to user stories cause a change to the requirements and access control of the system.

```
"As an <ACTOR>, I want to <PERFORM SOME ACTION>, so that <A PURPOSE IS FULFILLED>"

"In order to <FULFILL A PURPOSE> as an <ACTOR>, I want to <PERFORM AN ACTION>."

"As an <ACTOR> <AT SOME TIME> <IN SOME PLACE>, I <PERFORM AN ACTION> for <A PURPOSE>."
```

Fig. 1. Different formats, or templates, for defining user stories.

2.2 Related Work

Extracting Information From User Stories In all previous literature over user stories we reviewed [11, 12, 19], heuristics and rule-based approaches were used to parse and extract information from user stories as they are written in a similar format. In many datasets that these related works used, it was reported that most of the user stories conformed to defined templates. However, those datasets were not made available as they were proprietary data from industry partners, and in the only public dataset we found almost half the user stories did not conform to defined templates. Heuristics or rule-based approaches would fail for such user stories. We further observed that such approaches were limited in how much information they could extract from user stories. In all cases, such approaches was not able to identify specific data objects or operations.

In the work by Sobieski and Zielinski [19], the authors create a new constrained natural language user story format called "mixfit" that is designed to mark, or qualify, the important words or phrases of a user story (e.g., the actor, operation, data object, etc.). With the important items marked, they can be more easily extracted and utilized when analyzing user stories. However, this requires that stakeholders learn and strictly adhere to the mixfit format, which requires time and training that certain stakeholders (e.g., product owners) cannot invest in.

In the works by Lucassen et al. [12], the authors propose a quality user story framework, which lists criteria on what makes a good user story and what stakeholders should strive for when writing user stories. They also propose a heuristic approach for extracting conceptual models from user stories. However, the model was only tested on user stories that strictly followed user story templates, which is not common in real world datasets. Further, only general requirements information was identified, and was not fine-grained enough to identify unique data objects.

Extracting Access Control Information From Natural Language These works [1, 10, 13, 18, 22] are most closely related to our current research, and focuses on the extraction of access control (or other security information) from a natural language security policy. This work assumes that a natural language policy describing the system's security requirements exists, is complete, and is mostly unambiguous. The general state-of-the-art approach is to perform preprocessing on the natural language text and then use machine learning techniques to extract or analyze security requirements detailed by the policy. However, in the context of agile development, we are not able to make the assumption

that such a security policy exists, as the only documentation required by the agile development model is user stories. Further, user stories are often not complete (changing often throughout the development cycle) and are usually purposefully ambiguous as very few design choices are defined before implementation begins. Since agile development has become one of the most popular development models we believe that in many real-world software development scenarios it is not reasonable to assume a security policy exists and instead that only user stories are available as that is what developers more often actually work with.

Because of the ambiguity of user stories, most of the access control information in them is not explicitly stated, having to infer or predict how the actors, data objects, and operations in user stories imply different kinds of access. Further, all user stories are written from the system actors perspective. While actors are related to access control roles or subjects, they do not necessarily maintain a one-to-one translation from the end user of the system to the different types of user roles or subjects within the system. This is different from a natural language policy which is assumed to contain most, if not all, role and subject information.

Further, most of the current work pre-processes the policy text into other formats (such as dependency graphs) and learns patterns or rules based on these alternative formats. However, our approach uses and learns the text directly without the need to convert it. As far as we know, we are also the first to apply transformers to the access control information extraction process.

In the work by Slankas et al. [18], the authors extract access control rules from access control statements. This is achieved through parsing a natural language sentence into a dependency graph and then using machine learning techniques to learn patterns in the graph to identify access control rules. However, this approach has a primary basis in learning grammatical patterns associated with a dependency graph. This causes the approach to identify all instances of the learned patterns as access control rules, but there are many cases where a learned pattern occurs that is not related to access control. The paper attempts to statistically separate which identified patterns are more or less likely to be access control rules using a threshold value. Our approach does not learn such patterns and instead directly predicts on the sentences themselves.

In the work by Alohaly et al. [1], the authors propose to extract subject and object attributes for attribute-base access control from natural language policy. This is achieved by parsing a sentence into a dependency tree based on its grammatical structure and then using convolutional neural networks between grammar relations to predict if a word is an attribute. However, due to a lack of available data, they create a synthetic corpus by artificially injecting attribute information into role-based access control policies; meaning both the corpus injection and machine learning is based on the grammar of sentences.

3 Approach

We have chosen to use a deep learning approach to be more robust and generalizable in our ability to handle many different formats of user stories. The model

```
As a camp administrator, I want to be able to add
parents, so that they can enroll their kids at camp.

As an authenticated user, I want to see specific
details on summits, so that I can learn more about
the summit I'm interested in to see if it matches
my interests, register for the event, and get day-
of knowledge to help me get to the location.
```

Fig. 2. A simple and complex example of a user story from the dataset.

identifies general access control information between users (or actors) and the data objects or operations of the system. It is most easily applied to Role-Based or Attribute-Based Access Control. While the access control information can be used in constructing other access control models, it would be difficult to do so without additional context since user stories are written from the actor perspective. The following sections describe the data, labeling process, and model for our approach, which we have made available to the public[4].

3.1 Data

While searching for a dataset of user stories, we found that most papers in the literature use proprietary datasets and were not allowed to publish their dataset along with their work. This unfortunately has revealed a great need for robust datasets of user stories for research involving agile development. We were able to find one dataset published by Dalpiaz et. al. [6, 7].

The dataset consists of 21 web apps, each with 50–130 user stories for a total of over 1600 user stories. The apps are from various domains, including financial management, medical care, administrative management, and more. The most common types of app were data management platforms, accounting for 12 of the apps in the dataset.

As shown in Figure 2 user stories can range from the very simple, in the first sentence, to the more complex, in the second sentence. Since many of the user stories were complex statements with more than one actor, data object, or operation, it was difficult to determine a single model that could take a user story as input and produce all necessary access control information as output. For example, the output to the last example would be "(authenticated user, summits, view); (authenticated user, register for the event, access); (authenticated user, event, view); (authenticated user, location, view)", where "authenticated user" is an actor, "view" and "access" are the kind of access to the data object or operation, "register for the event" is an operation, and "summits", "event", and "location" are data objects. The best single model solution we identified was to utilize a neural translation approach. However, a neural translation model would likely have great difficulty for a number of reasons, such as: listing multiple access control tuples where the order of the tuples is not important and there is

[4] https://github.com/jheaps/AccessControlPolicyGeneration

```
As a camp administrator, I want to be able to remove campers if they don't
attend the camp anymore, so that I can keep the records organized.

As an OlderPerson, I want to use only well-visible buttons.

As an Archivist, I want to see Dates and Extents displayed in both the read
and edit views for Accessions and/or Resources before the list of Subjects.
```

Fig. 3. User stories containing access control, not containing access control, and ambiguous

constant recurrence of words; analyzing many user stories which do not contain any access control information all of which would be translated to the same word or symbol; and the existence of multiple different user stories that produce the exact same tuples. We therefore constructed a model containing three subcomponents to perform the prediction: access control classification - deciding if a user story contains access control information; named entity recognition - identifying the actors and data objects contained in the user story; and access type classification - determining the relation between the actors and the data objects. We used transformers for the model prediction, from the Transformers library [21][5] with PyTorch[6], and also implemented the components using CNN and SVM as baseline comparisons.

Labeling Each of the three component models to our deep learning approach required a separate labeling of the dataset.

The first component determines if the user story contains access control information with labels: "contains access control" or "does not contain access control". In many cases this was obvious. For example, the first user story in Figure 3 contains access control information where "camp administrator" is the actor and "campers" is a data object that camp administrator should be able to delete. In contrast, the second user story of Figure 3 does not contain any access control information. However, there were many user stories where it was difficult to determine if they contained access control information or not. For example, in the third user story in Figure 3, it is unclear if this is only describing a change to the user interface of the software, in which case it would not contain access control information, or if it is describing a requirement for what data objects an Archivist can view. Further, it is not clear if the data objects are protected or simply available to everyone. During the normal development of a system, a developer would often have to make such a determination themselves. Since we do not know the final decision made about this particular user story, we have decided to include a third label, "unknown", in our labeling. The primary reason for this is to indicate to stakeholders that this is ambiguous or a decision about the access control policy needs to be made here. The earlier the ambiguity about the access control policy can be identified, the more decisions about the access control policy can be removed from developers and placed on product owners who should be better able to make such decisions or provide additional context. An "unknown" is treated the same as a "contains access control" from

[5] https://huggingface.co/transformers/
[6] https://pytorch.org/

```
As              Other
a               Other
camp            B-Actor
administrator   I-Actor
,               Other
I               Other
want            Other
to              Other
schedule        Other
events          B-DataObject
.               Other
```

Fig. 4. Named Entity Recognition Example

the model's perspective, and is primarily used for presentation and visualization purposes that are described more thoroughly in Section 3.3. Further, it will be a great asset during further research when incorporating human aid and interactivity, as described in Section 5.

The second component identifies the actors, data objects, and operations in the user story. We have chosen to use a named entity recognition approach to tag words (or sequence of words) that represent actors or data objects in the user story, if they occur. In the example in Figure 4 the words "camp administrator" are tagged as an actor and "events" as a data object. For labeling, each user story is broken into individual word tokens and labeled with one of seven different labels: "B-Actor" to denote the beginning token of an actor name, "I-Actor" to denote the continuation of actor name, "B-DataObject" to denote the beginning token of a data object name, "I-DataObject" to denote the continuation of a data object name, "B-Operation" to denote the beginning token of an operation, "I-Operation" to denote the continuation of an operation, and "Other" to denote other or a token we do not want to tag.

The third component, determines what type of access exists between the tagged entities in the user story. For the relationship between actors and data objects, the relationship may be: "view", "edit", "create", "delete", or "none". This is a multi-label classification and the user story may imply none or more of the labels between entities. For a simple user story, such as the first user story in Figure 2, where there is only one actor ("camp administrator") and one data object ("camper") or operation ("remove camper"), it is obvious what actor the access type the label is referring to for the data object and operation. However, there are many user stories in the dataset that contain multiple actors and data objects. In this case, it is difficult to determine which of the actors or data objects would be indicated by a label, or in what order they would be referenced if multiple labels were added to it. To circumvent this problem, we copy the user story for each possible actor and data object present and label then separately with the appropriate label for each couple. Simple user stories with only one actor and data object pair are labeled in the same way. How the presence of the same user story with different labels are input to the model is discussed further in Section 3.2. For the operations, we found almost no actor-operation pair where the actor was not allowed access. This seems to be due to the formatting of the user stories in our dataset. User stories are almost always stated in a positive

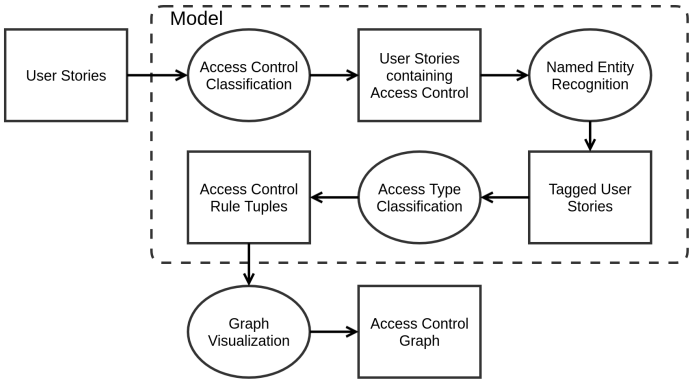

Fig. 5. User Story Access Control Extraction and Prediction Model

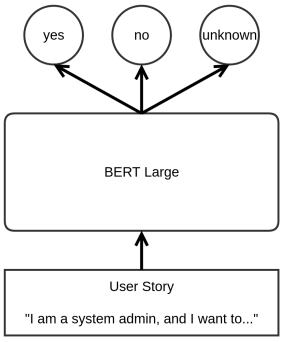

Fig. 6. Access Control Classification Model

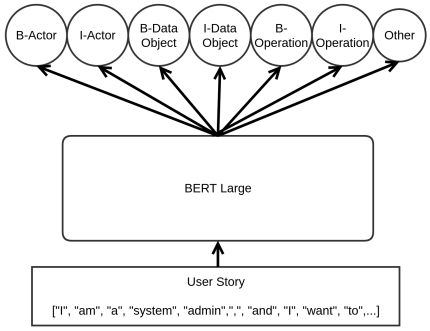

Fig. 7. Named Entity Recognition Model

form (i.e., stating that an actor can do something) and are very rarely found in a negative form (i.e., stating that an actor cannot do something). Because of this, there was no way to learn or predict if a actor had "access" or "no access" to an operation as almost all user stories were formatted in a way that implied the actor had "access".

3.2 Model

We utilized the Transformers library to implement our models. The input, output, and flow of the component models are shown in Figure 5, with rectangles representing input/output and ovals representing operations or learning tasks.

The first component, shown in Figure 6, is the Access Control Classification task which takes the initial set of all user stories and predicts whether each individual user story contains access control information or not. We performed a sequence classification task using the BERT Large [8] transformer model. All

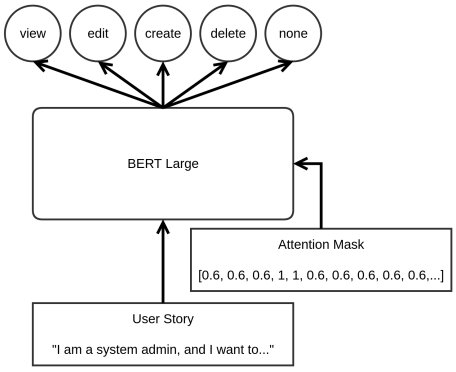

Fig. 8. Access Type Classification Model

hyperparameters were set to default except the dropout rate which was set to 0.4. The model was trained for 15 epochs with a batch size of 30. Those user stories that were predicted as "does not contain access control" were removed from the set of user stories. As discussed earlier, the "unknown" label is treated the exact same as the "contains access control" label during modeling and prediction, and is only utilized during the visualization of the access control information.

The second component, shown in Figure 7, is the Named Entity Recognition task and is used to tag the words in each user story that contains access control information. The Named Entity Recognition task identifies the actors, data objects, and operations in each user story. Each user story is tokenized into individual words and then a classification task is performed on each word to determine how it should be tagged. We use the BERT Large transformer model with all hyperparameters set to default, except the dropout rate which was set to 0.5. The model was trained for 15 epochs with a batch size of 30.

The final component, shown in Figure 8, is the Access Type Classification and is used to identify the type of access ("view", "edit", "create", "delete", or "none") a given actor has for a given data object within each user story. We performed a multi-label sequence classification task using the BERT Large transformer model with all hyperparameters set to default, with the exception of the dropout rate which was set to 0.5. The model was trained for 15 epochs with a batch size of 30. This gives the final information to produce a list of tuples, "(actor, data object, access type)", that define the access control of the system. As mentioned in Section 3.1, if a user story has multiple actor and data object couples, the same user story is repeated and uniquely labeled for each unique couple. This poses a problem in that the same input implies multiple different outputs, but given the same input a model will always predict the same output (after training is completed). While the most important tokens in each user story is the unique actor and data object couple, we do not want to lose the context of the rest of the tokens in the user story that are vital to determining the type of access between the actor and data object. That is, we wish to emphasize the actor and data object tokens and de-emphasize (but not ignore) the other tokens

in the user story. To achieve this we utilize the attention, or padding, mask of the transformer model. We see this as a logical and natural extension to the functionality and purpose of the attention mask. Normally utilized to negate the values of padding tokens so they do not affect the computations and results of a transformer's prediction, the attention mask can use values between $[0, 1]$. The different values in the mask will reduce the values of certain tokens toward the model's prediction while placing greater value on others. As can be seen in Figure 8, the partial user story has the actor "system admin". If we assume the user story has multiple data objects (user accounts, permission files, etc.), then we will need to predict the type of access between each system admin and data object couple. The normal attention mask for an input would be a 1 for each token and 0 for any padding, if present. So if no padding existed it would be a vector of all ones the same size as the number of tokens in the user story. To emphasize the related actor and data object, the corresponding indices of "system admin" and a data object will be left as a 1 in the attention mask, but the rest of the tokens will have their value changed to less than 1 but greater than 0. In Figure 8 this is shown in the attention mask where "system admin" is represented by ones and the rest of the tokens (that are shown) are 0.6. In this way, the same input can be used to predict multiple different outputs as those tokens deemed more important to the current prediction will be given greater value. As a final note, the value chosen for this example to represent de-emphasized tokens was 0.6. This is not necessarily the optimal value to achieve the best results with the model and likely multiple different values will need to be trained to determine the optimum performance.

3.3 Access Control Presentation

The final step in our approach is to present the access control information to stakeholders. We first perform some logical reduction of the set of access control tuples. In some cases, the same access control information is predicted from different user stories and so duplicates are removed. Also different user stories may imply different access control relationships between the same actor and data object and are combined into a single tuple.

We believe that one common and useful presentation of the access control information in software documentation is in graphical form. We have written a script to transfer the set of tuples to graphviz[7] format where actors, data objects, and operations are vectors and the types of access between them are edges.

Any access control tuples that were predicted as "unknown" by the first model component are colored in red, and those predicted as "contains access control" are colored in black. This is to stress the ambiguity of the access control information in the related user stories. This should prompt discussion and refinement of those user stories with stakeholders early on in production so that decisions about access control can be made in advance. This will save stakehold-

[7] https://graphviz.org/

App Name	Metric	ACC Score	NER Score
Frictionless	Precision	92.3% ± 1.8	88.2% ± 2.9
	Recall	89.7% ± 2.1	86.4% ± 4.4
	F1 Score	91.0% ± 2.0	87.3% ± 4.7
Alfred	Precision	79.1% ± 3.4	80.8% ± 4.7
	Recall	86.6% ± 2.7	80.1% ± 6.1
	F1 Score	82.7% ± 3.0	83.8% ± 5.3
CamperPlus	Precision	80.2% ± 2.5	84.4% ± 5.3
	Recall	88.3% ± 3.2	76.0% ± 4.1
	F1 Score	84.1% ± 2.8	80.0% ± 4.6

Table 1. Precision, Recall, and F1 Score for the Access Control Classification and Named Entity Recognition tasks across three testing apps.

ers time and money and relieve developers from the responsibility of making on-the-fly decisions about access control policy.

The presentation of access control information is not limited to graphical form: question and answer systems, rule lists, and (with limited human involvement) generating an access control natural language policy and policy specification can be achieved. This is further discussed as ongoing and future work in Section 5.

4 Evaluation

For our evaluation, we performed training, validation, and testing using, roughly, a 80%, 10%, 10% split, respectively. The testing set was created by removing all the user stories of one app from the rest of the dataset so that we can check the performance of the model on a set of user stories from an app that the model has never seen before. For validation, 10% of the remaining user stories were taken after being shuffled. The training set consisted of all other apps' user stories not in the testing or validation sets. We performed three separate training, validation, and testing runs for each component. Each run used a different app for the testing phase: Frictionless for data management platform and data archiving, Alfred for medical and elderly care, and CamperPlus for administrative management. In general, we observed that Frictionless performed better than the other apps during testing. This is likely due to the fact that the data management platform category represents over half of the apps in the training set. In contrast, there are very few apps in the training set that belong to the same category as Alfred and CamperPlus.

4.1 Access Control Classification

As seen in Table 4, Frictionless outperformed Alfred and CamperPlus by a margin of 7 and 9 percentage points, respectively. We analyzed those user stories that were predicted incorrectly and found some commonalities. In general, it seemed that non-functional user stories (i.e., user stories that describe attributes of the

App Name	Metric	F1 Score
Frictionless	View	86.4% ± 4.2
	Edit	84.6% ± 5.5
	Create	81.1% ± 4.6
	Delete	81.7% ± 3.7
	None	82.2% ± 4.2
Alfred	View	80.6% ± 3.8
	Edit	79.8% ± 4.3
	Create	75.6% ± 5.7
	Delete	75.3% ± 6.0
	None	80.5% ± 5.3
CamperPlus	View	83.2% ± 5.1
	Edit	79.3% ± 3.6
	Create	76.5% ± 4.9
	Delete	75.6% ± 3.9
	None	79.9% ± 4.6

Table 2. Precision, Recall, and F1 Score for the access type classification task across three testing apps.

system, but not a functional usage of the system, for example "As an Older Person, I want to use only well-visible buttons") were more difficult for the model to categorize. This is likely due to much fewer non-functional user stories being present in the dataset. Other user stories that were predicted poorly contained acronyms or uncommon names of file types or 3rd party software. Again, since the model has not seen such words before, it was likely difficult to predict over.

4.2 Named Entity Recognition

As seen in Table 4, Frictionless again outperformed Alfred and CamperPlus of about 3 and 7 percentage points, respectively. The named entity recognition task suffered from the same main setback as the access control classification task, with most unique or uncommon words leading to a decrease in performance and were the most common missed words. Interestingly, there seemed to be some words common to multiple apps, but with different labels. For example, "team" was an actor name in some apps but a data object in others. It seems it was difficult for the model to reconcile this.

4.3 Access Type Classification

As seen in Table 4.3, Frictionless also outperformed Alfred and CamperPlus in the access type classification. The problem of a lack of data can definitely be seen here, as "create" and "delete" had much fewer occurrences than "view" and "edit", and did much poorer amongst all three apps.

We tested how well the average F1 score of all access categories performed at different attention mask values of de-emphasis (as described in Section 3.2). We tested mask values of 0.25, 0.5, 0.6, 0.75, and 0.9. For Fictionless, the best

Model	Component	F1 Score
Transformers	Access Control Classification	91.9% ± 2.0
	Named Entity Recognition	87.3% ± 3.4
	Access Type Classification	83.2% ± 4.4
CNN	Access Control Classification	84.3% ± 4.1
	Named Entity Recognition	86.7% ± 3.6
	Access Type Classification	79.1% ± 5.4
SVM	Access Control Classification	84.4% ± 1.3
	Named Entity Recognition	69.8% ± 3.9
	Access Type Classification	73.2% ± 4.3

Table 3. Transformers, CNN, and SVM models and their average F1 scores for the three model components

mask value was 0.6, Alfred performed best at 0.75, while CamperPlus did best at 0.6 as well.

4.4 SVM and CNN Comparison

In Figure 4.4, the average F1 score for all components for each the Transformers, CNN, and SVM models are shown. Transformers performed the best in all categories, with CNN as a close second. Interestingly, CNN performed almost just as well as Transformers in the named entity recognition task, where SVM performed the worst in that category. We see that the Transformers model performed better than CNN and SVM as baseline measures, but not as significantly as we would have hoped.

We believe the primary approach to increasing the Transformer model's (or any model's) accuracy is a much larger and more robust dataset. It is clear that with 1600 example user stories the models were able to learn in all three components but the learning was quite limited, both in the size of the dataset as well as the variety of types of apps.

4.5 Graph Visualization

Finally, we have performed an end-to-end run of one of the testing apps, CamperPlus, through the full model and have visualized a sample of the graph in Figure 9. We have chosen only a sample as the full graph was too large to insert into the paper, and these showed the more interesting results. In the graph, the different types of shapes denote the type of entity it is: diamonds are actors, rectangles are data objects, and ovals are operations. Only data objects have a type of access associated with them, as the operations were all permitted access, as described in Section 3.1. The red lines are those user stories that were predicted as "unknown", and would prompt stakeholders to take a closer look at those user stories related to them. In future work, human interactivity would allow a stakeholder to confirm or invalidate the "unknown" predictions, as described in Section 5. In some cases the model worked well. We can see that the

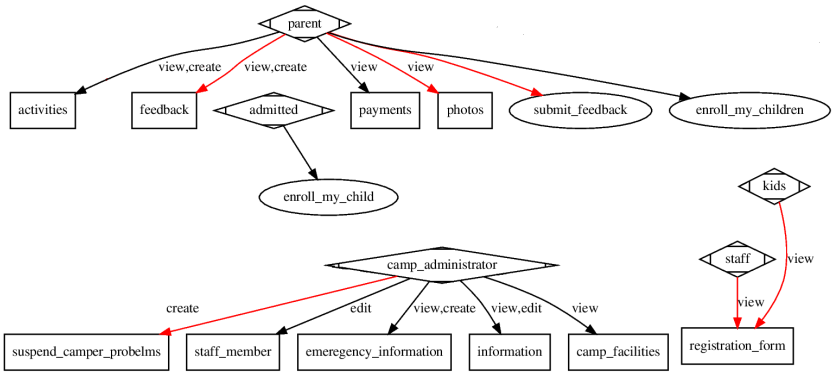

Fig. 9. Graph Visualization Sample of CamperPlus App

"parent", "camp administrator", "staff", and "kids" actors were all predicted correctly, and many of the data objects and operations were correctly predicted as such. However, "admitted" was not an actor (or a data object or operation). We believe it was predicted as an actor because it may be close enough to "administrator" which was labeled as an actor. The "suspend camper problems" should just be "suspend camper" and be an operation, not a data object. The "information" data object was likely supposed to be "medical information" as that was a data object missing from the full graph. While we tried to remove most plural words after the final output of the model, we can see that some were missed, such as "enroll my child" and "enroll my children".

We can see that the graph is not perfect, which was expected given the F1 scores from testing. However, there is a significant portion of the access control graph that is correct, and with the help of human aid shows great potential for assisting stakeholders during the agile development software process.

5 Conclusion

This paper describes an automated approach to extracting access control information from user stories through the labeling of a dataset of 1600 user stories and a learning model based on transformers with three components: access control classification, named entity recognition, and access type classification. The resulting list of tuples was visualized to help stakeholders better refine user stories, maintain an overview of the system's access control information, and ultimately help reduce the propagation of security vulnerabilities in code.

Automated Analysis of Access Control Tuples In future work we plan to perform automated analysis of the extracted access control information. This analysis would include: contradiction detection of access control rules across a set of user stories, inferring additional access control information from groups of user stories that may not be identified by examining user stories individually,

and detecting duplicate user stories. Evidence for the existence and need for each of these scenarios was present in our dataset, however they occurred in too small a number for experiments to be conducted with less than 10 occurrences of each across the entire dataset. This further demonstrates the need for larger, more robust user story datasets.

Integration of Human Interactivity We recognize that there is little additional information about access control that can be extracted or determined directly from user stories in a fully automated approach. That is, it is difficult, or impossible, to determine a software's exact access control from user stories without human involvement. We plan to extend our approach to make it interactive with stakeholders so that they can help refine the access control information by providing additional context, such as specific roles and attributes of the system. Humans will be able to validate (or invalidate) ambiguous relations that were marked as "unknown" or were incorrectly predicted. This will also allow for active learning [16] with the models to better refine prediction results. While this interactivity will require some continual review from humans, the vast majority of the maintenance of access control information and documentation will still be performed automatically by our approach. This should keep human involvement at a minimum, increasing development time as little as possible.

Tracking the Agile Development Cycle A primary aspect of agile development is changing requirements through the modification of the set user stories, which occurs throughout the agile development cycle. We plan to show how our approach can handle changing user stories throughout the development process.

Additional Security Policy Generation Finally, while this work describes an automated approach to the construction of access control documentation, other security and system documentation could be inferred or extracted from user stories. For example, privacy requirements, interaction and use of external technologies and third party applications, and generally ensuring security best practices based on security decisions and implementations can all be handled in a mostly automated approach to ensure that stakeholders have necessary documentation when developing the system.

6 Acknowledgments

We would like to thank the CREST Center For Security And Privacy Enhanced Cloud Computing (C-SPECC) through the National Science Foundation (NSF) (Grant Award #1736209), the NSF Division of Computer and Network Systems (CNS) (Grant Award #1553696), the and NSF Division of Computing and Communication Foundations (Grant Award #2007718) for their support and contributions to this research.

References

1. Alohaly, M., Takabi, H., Blanco, E.: A deep learning approach for extracting attributes of abac policies. In: Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies. pp. 137–148 (2018)
2. Bartsch, S.: Practitioners' perspectives on security in agile development. In: 2011 Sixth International Conference on Availability, Reliability and Security. pp. 479–484. IEEE (2011)
3. Ben Othmane, L., Angin, P., Weffers, H., Bhargava, B.: Extending the agile development process to develop acceptably secure software. IEEE Transactions on dependable and secure computing **11**(6), 497–509 (2014)
4. Beznosov, K., Kruchten, P.: Towards agile security assurance. In: Proceedings of the 2004 workshop on New security paradigms. pp. 47–54 (2004)
5. Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., Kruchten, P.: Extending xp practices to support security requirements engineering. In: Proceedings of the 2006 international workshop on Software engineering for secure systems. pp. 11–18 (2006)
6. Dalpiaz, F.: Requirements data sets (user stories). Mendeley Data. **doi: 10.17632/7zbk8zsd8y.1** (2018)
7. Dalpiaz, F., Van der Schalk, I., Lucassen, G.: Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and nlp. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 119–135. Springer (2018)
8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
9. Fowler, M., Highsmith, J., et al.: The agile manifesto. Software Development **9**(8), 28–35 (2001)
10. Karimi, L., Aldairi, M., Joshi, J., Abdelhakim, M.: An automatic attribute based access control policy extraction from access logs. IEEE Transactions on Dependable and Secure Computing (2021)
11. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E., Brinkkemper, S.: Improving agile requirements: the quality user story framework and tool. Requirements Engineering **21**(3), 383–403 (2016)
12. Lucassen, G., Robeer, M., Dalpiaz, F., Van Der Werf, J.M.E., Brinkkemper, S.: Extracting conceptual models from user stories with visual narrator. Requirements Engineering **22**(3), 339–358 (2017)
13. Narouei, M., Takabi, H., Nielsen, R.D.: Automatic extraction of access control policies from natural language documents. IEEE Transactions on Dependable and Secure Computing **17**, 506–517 (2020)
14. Oueslati, H., Rahman, M.M., ben Othmane, L.: Literature review of the challenges of developing secure software using the agile approach. In: 2015 10th International Conference on Availability, Reliability and Security. pp. 540–547. IEEE (2015)
15. Pohl, C., Hof, H.J.: Secure scrum: Development of secure software with scrum. arXiv preprint arXiv:1507.02992 (2015)
16. Settles, B.: Active learning. Synthesis lectures on artificial intelligence and machine learning **6**(1), 1–114 (2012)
17. Siponen, M., Baskerville, R., Kuivalainen, T.: Integrating security into agile development methods. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences. pp. 185a–185a. IEEE (2005)

18. Slankas, J., Xiao, X., Williams, L., Xie, T.: Relation extraction for inferring access control rules from natural language artifacts. In: Proceedings of the 30th Annual Computer Security Applications Conference. pp. 366–375 (2014)
19. Sobieski, Ś., Zieliński, B.: User stories and parameterized role based access control. In: Model and Data Engineering, pp. 311–319. Springer (2015)
20. Wäyrynen, J., Bodén, M., Boström, G.: Security engineering and extreme programming: An impossible marriage? In: Conference on Extreme Programming and Agile Methods. pp. 117–128. Springer (2004)
21. Wolf, T., Chaumond, J., Debut, L., Sanh, V., Delangue, C., Moi, A., Cistac, P., Funtowicz, M., Davison, J., Shleifer, S., et al.: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 38–45 (2020)
22. Xiao, X., Paradkar, A., Thummalapenta, S., Xie, T.: Automated extraction of security policies from natural-language software documents. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. pp. 1–11 (2012)