At-Risk System Identification via Analysis of Discussions on the Darkweb

Eric Nunes, Paulo Shakarian
Arizona State University
Tempe, AZ 85281, USA
Email: {enunes1, shak} @asu.edu

Gerardo I. Simari
Department of Computer Science and Engineering
Universidad Nacional del Sur (UNS), Argentina
Institute for C.S. and Eng. (CONICET–UNS)
Email: gis@cs.uns.edu.ar

***Abstract*—Threat assessment of systems is critical to organizations' security policy. Identifying systems likely to be at-risk by threat actors can help organizations better defend against likely cyber attacks. Currently, identifying such systems to a large extent is guided by the Common Vulnerability Scoring System (CVSS). Previous research has demonstrated poor correlation between a high CVSS score and at-risk systems. In this paper, we look at hacker discussions on darkweb marketplaces and forums to identify the platforms, vendors, and products likely to be at-risk by hackers. We propose a reasoning system that combines DeLP (Defeasible Logic Programming) and machine learning classifiers to identify systems based on hacker discussions observed on the darkweb. The resulting system is therefore a hybrid between classical knowledge representation and reasoning techniques and machine learning classifiers. We evaluate the system on hacker discussions collected from nearly 300 darkweb forums and marketplaces provided by a threat intelligence company. We improved precision by 15%–57% while maintaining recall over baseline approaches.**

I. Introduction

Adequate assessment of threats to systems is a central aspect of a mature security policy—identifying systems that are at-risk can help defend against potential cyber attacks. Currently, organizations rely on the rating system (CVSS score) provided by The National Institute of Science and Technology that maintains a comprehensive list of publicly disclosed vulnerabilities in the National Vulnerability Database (NVD [21]) to identify if their systems are at risk. Case studies have shown poor correlation between the CVSS score and the likelihood that a vulnerability on a system will be targeted by hackers [2]. Hence, organizations are constantly looking for ways to proactively identify if their vulnerable systems are of interest to hackers.

Threat intelligence from deepweb and darkweb (D2web) has been leveraged to predict whether or not a vulnerability mention on D2web will be exploited [4], [3]. This method only considers hacker discussions that have a CVE number mentioned in them—a limitation of the approach is therefore that discussions with no vulnerability identifiers (CVE) that are of interest to threat actors are not taken into account. In this paper, we propose to leverage this threat intelligence gathered from D2web markets and forums to identify the systems that might be of interest to threat actors. We identify systems based on the structured naming scheme Common Platform Enumeration (CPE [7]). We focus our efforts towards identifying the first three system components of the CPE naming scheme; Table I shows these three components, with examples for each.

TABLE I: System components and examples

Components	Explanation and Examples
Platform	Can be either hardware (h), operating system (o), or application (a) based on what the vulnerability exploits.
Vendor	The owner of the vulnerable product. Examples include Google, Microsoft, The Mozilla Foundation, and the University of Oxford.
Product	The product that is vulnerable. Examples include Internet Explorer, Java Runtime Environment, Adobe Reader, and Windows 2000.

We design a system that leverages threat intelligence (hacker discussions) and makes a decision regarding at-risk systems, at the same time providing arguments as to *why* a particular decision was made. It explores multiple competing hypotheses (in this case multiple platforms, vendors, products) based on the discussions for and against a particular at-risk component. The resulting system is a hybrid that combines DeLP with machine learning classifiers. Previously, a similar reasoning system was employed for attributing cyber-attacks to responsible threat actors [23] evaluated on a capture-the-flag dataset. Specific contributions of this paper include:

- We frame identifying at-risk systems as a multi-label classification problem, and apply several machine learning approaches to compare their performance. We find that large number of possible label choices for vendors and products with less representation in training account for the majority of the misclassified samples.
- To address misclassification, we propose a hybrid reasoning framework that combines machine learning techniques with defeasible argumentation to reduce the set of possible labels for each system component. The reasoning framework can provide arguments supporting the decisions, indicating *why* a particular system was identified over others; this is an important aspect, supporting a security analyst in better understanding the result.

978-1-5386-4922-0/18/$31.00 2018 IEEE

- We report on experiments showing that the reduced set of labels used in conjunction with the classifiers leads to significant improvement in precision (15%-57%) while maintaining comparable recall.

The rest of the paper is organized as follows. In Section II we briefly discuss some terminologies used throughout the work. A system overview is presented in Section III, and then we discuss the dataset and provide an analysis in Section IV. This is followed by the argumentation model based on [12] in Section V; then, the experimental setup and results (along with the DeLP programs for each system component) are discussed in Section VI. This is followed by a discussion on the results and related work in Section VII and Section VIII, respectively. Finally, conclusions are discussed in Section IX.

II. Background

A. Darkweb (D2web) websites

Darkweb refers to the portion of the internet that is not indexed by search engines and hence cannot be accessed by standard browsers. Specialized browsers like "The Onion Router" (Tor)[1] are required to access these websites. Widely used for underground communication, Tor is free software dedicated to protect the privacy of its users by obscuring traffic analysis [10]. The network traffic in Tor is guided through a number of volunteer-operated servers (also called "nodes"). Each node of the network encrypts the information it blindly passes on neither registering where the traffic came from nor where it is headed [10], disallowing any tracking. We retrieve information from both *marketplaces*, where users advertise to sell information regarding vulnerabilities or exploits targeting the vulnerabilities, and *forums* that provide discussions on discovered vulnerabilities among others.

Markets: Users advertise and sell their products and services (referred to as *items*) on marketplaces. D2web marketplaces provide a new avenue to gather information about the cyber threat landscape, in particular exploits targeting vulnerabilities or hacking services provided by vendors at a particular price. These marketplaces also sell goods and services relating to drugs, pornography, weapons, and software services—these need to be filtered out for our application.

Forums: These are user-oriented platforms where like-minded individuals have discussions on topics of interest, regardless of their geophysical location. Administrators set up D2web forums with communication safety for their members in mind. While structure and organization of D2web-hosted forums might be very similar to more familiar web-forums, the topics and concerns of the users vary distinctly. Forums addressing malicious hackers feature discussions on programming, hacking, and cyber-security with newly discovered vulnerabilities as well as zero-days (vulnerabilities not publicly disclosed yet). Threads are dedicated to security concerns like privacy and online-safety—such topics plug back into and determine the structures and usage of the platforms.

[1]See the Tor Project's official website (https://www.torproject.org/)

B. Vulnerability related terms

Vulnerability is a flaw in a system (software/hardware) that makes the system vulnerable to attacks compromising the confidentiality, integrity or availability of the system to cause harm [24].

CVE: Common vulnerability enumeration (CVE) is a unique identifier assigned to a system vulnerability reported to NIST [8]. NIST maintains a database of all the vulnerabilities publicly available in the National Vulnerability Database (NVD [21]). Predicting exploitability of a CVE is an important problem and recent work leveraging darkweb data has shown good performance in achieving that goal [4], [3]. But these techniques reply on direct mentions of CVE's. We Note that a very small portion of hacker discussions in the data from the commercial provider has direct CVE mentions.

CPE: Common platform enumeration (CPE) is a list of software / hardware products that are vulnerable for a given CVE. NIST makes this data available for each vulnerability in its database. Identifying at-risk systems in terms of its components (see Table I) is an important step towards predicting if those systems will be targeted by threat actors (in cases where the hacker discussion is not associated with a CVE number). For the system components under consideration, there exists a hierarchy starting from the platform to vendor to product. For instance, if we are considering operating systems, then there are limited number of vendors that provide it: Microsoft, Apple, Google, etc. If we identify Microsoft as our vendor, then the products are related to the Windows operating system. This hierarchy helps us to narrow down possible choices as we go down the hierarchy.

III. System Overview

Fig. 1 gives an overview of the reasoning system; it consists of the following three main modules:

- **Knowledge Base:** Our knowledge base consists of hacker discussions from darkweb (D2web) forums and marketplaces. This data is maintained and made available through APIs by a commercial darkweb threat intelligence provider[2]. The database is collected from 302 websites[3]. We use the hacker discussions in terms of posted content (from forums) and item descriptions (from markets), the website it is posted on, and the user posting the discussion as inputs to both the argumentation and machine learning models. We also input the CPE hierarchy from NVD to the argumentation model. We discuss and provide further analysis of the data in Section IV. For the experiment, we sort the dataset by time (depending on when the discussion was posted); the first 80% is reserved for training (knowledge base) and the remaining 20% for testing. We follow similar time split to compute the CPE heirarchy as well.
- **Argumentation Model:** This component constructs arguments for a given query (at-risk system component)

[2]Cyber Reconnaissance, Inc. (CRY3CON), https://www.cyr3con.com.
[3]At the time of writing this paper

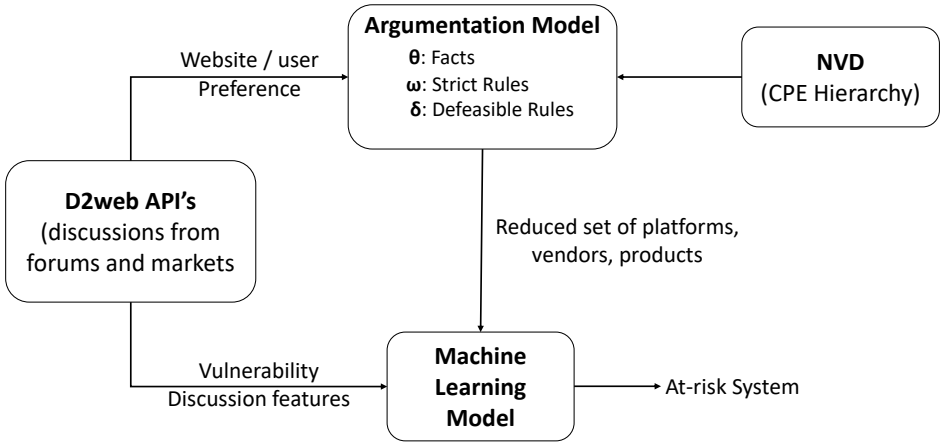

Fig. 1: Reasoning System

using elements in the knowledge base. We use a formalism called DeLP that combines logic programming with defeasible argumentation. It is made up of three constructs: *facts:* observations from the knowledge base that cannot be contradicted; *strict rules:* logical combinations of facts that are always true; and *defeasible rules:* can be thought of as strict rules but are only true if no contradictory evidence is present. We discuss the argumentation framework with examples for each of the constructs in Section V. Arguments help reduce the set of possible choices for platforms, vendors and products; this reduced set of possible system components acts as one of the inputs to the machine learning model. The argumentation model thus constrains the machine learning model to identify the system from the reduced set of possible platforms, vendors, and products.

- **Machine Learning Model:** The machine learning model takes the knowledge base and query as input, along with the reduced set of possible system components from the argumentation model, and provides a result identifying the system. It is constrained by the argumentation model to select the components from the reduced platform, vendor and product set, which aids the machine learning model (improving precision) as demonstrated in the results section of the paper. We use text-based features extracted from the discussions (TF-IDF/Doc2Vec) for the machine learning model. Any standard machine learning model can be used in this module. We provide a comparison of different machine learning models to select the best one.

IV. Dataset

A. D2web data

We use D2web data supplied by a threat intelligence company. The data is accessed via APIs. The data is comprised of forum discussions and marketplace items offered for sale in D2web. Exploration of D2web discussions in terms of their structure, content and behavior of users who post these discussions is reported in [30]. The data is collected periodically to obtain time-based information indicating changes in the forums and marketplaces. To ensure collection of cybersecurity relevant data, machine learning models are employed that filter the data related to drugs, weapons, and other irrelevant discussions. Table II shows the characteristics for the websites, posts/items, and users. The data is comprised from websites with different languages. A single website might have discussions in different languages. Fig. 2 shows the percentage of total websites from the D2web for the top ten languages used to post discussions Majority of the websites have discussions in English (73%), with other languages having an even distribution. The commercial data collection platform automatically identifies the language and translates it to English using the Google Translate API [14].

Ground Truth. In order to evaluate the performance of the reasoning framework, we need ground truth associated with the hacker discussions. To obtain ground truth we consider discussions from forums and marketplaces that mention a CVE number. From the CVE number we can look up the vulnerable systems using the NVD; we note that for both training and testing we remove the CVE number while computing features. Table II shows the characteristics for the websites, posts/items, and users that mention a CVE number. The hacker discussion with CVE mentions belong to 135 websites posted by 3361 users. On analyzing the CVE mentions most of the older

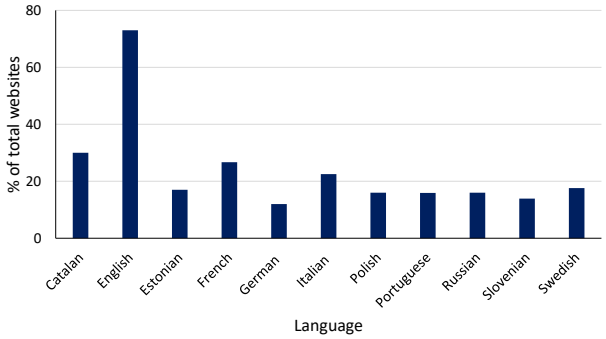

Fig. 2: Percentage of total websites belonging to the top ten languages in the D2web data.

TABLE II: Characteristics of D2web data

Number of D2web websites	302
Number of unique users	635,163
Number of unique posts / items	6,277,638
Number of D2web websites (CVE mentions)	135
Number of unique users (CVE mentions)	3,361
Number of unique posts / items (CVE mentions)	25,145

vulnerabilities target products that are no longer in use. For that reason in our experiments we consider CVE discussions posted after 2013 (starting 01/01/2014). These discussion make up around 70% of the total CVE discussions.

CPE Hierarchy. We compute the hierarchy for all the vulnerable systems from all the vulnerabilities disclosed in NVD [21], and maintain it as a dictionary to build arguments on top of it. Fig. 3 shows a subset of the built hierarchy with the three system components (platform, vendor and product).

Website/User preference. We compute and maintain a list of system components discussed for each website and user. This lets us know if a particular website is preferred by hackers to discuss specific at-risk systems. The user list gives us the preference of the user regarding what at-risk systems are of interest to him/her.

Overall in our dataset, for platforms most discussions pose a threat to operating systems (57%), following by applications (43%) and hardware makes up a small fraction of the discussions (3%). There are discussions that pose a risk to multiple platforms i.e. operating systems and application or in few instances all three. For vendors, the top five at-risk based on CVE mentions in the hacker discussions: Microsoft (24%), Linux (9%), Apple (6%), Oracle (5%), Adobe (5%). Similar to platforms discussions can pose a risk to multiple vendors. For products the distribution is more even since a single vendor can have multiple products. Even though Microsoft dominates the vendor discussion, it also has the most number of products that are at risk. The top five at-risk products based on CVE mentions in the hacker discussions: Windows server (5%),

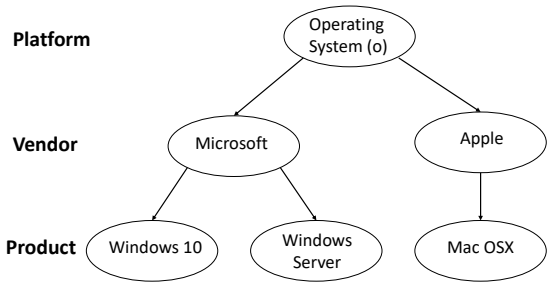

Fig. 3: Subset of CPE Hierarchy

Windows 8.1 (4%), Linux kernel (3.8%), Mac OSX (2.3%), Flash player (1.9%).

V. Argumentation Model

Our approach relies on a model of the world where we can analyze competing hypotheses. Such a model allows for contradictory information so it can handle inconsistency in the data similar to the one employed for attributing cyber-attacks to responsible threat actors [31], [23].

Before describing the argumentation model in detail, we introduce some necessary notation. Variables and constant symbols represent items such as the platform/vendor/product at-risk by the discussion and post/webID/userID represent the hacker discussion, where it was posted and who posted it respectively (we note that for privacy concerns the webID/userID is represented as an integer in the data provided by the APIs—the names are not disclosed). We denote the set of all variable symbols with $\mathbf{V}$ and the set of all constants with $\mathbf{C}$. For our model we require six subsets of $\mathbf{C}$:

- $\mathbf{C}_{post}$ denoting the hacker discussion,
- $\mathbf{C}_{web}$, denoting the websites (both forums and marketplaces) where the hacker discussion was posted,
- $\mathbf{C}_{user}$, denoting the users who posts hacker discussions, and
- $\mathbf{C}_{platform}$, $\mathbf{C}_{vendor}$, $\mathbf{C}_{product}$ denoting the three components at-risk by the discussion (see Table I).

We use symbols in all capital letters to denote variables. In the running example, we use a subset of the D2web dataset collected by the threat intelligence company.

Example 1. *The following system and post/web/user information will be used in the running example:*

$\mathbf{C}_{post} = \{post_1, post_2, ..., post_n\}$

$\mathbf{C}_{web} = \{webID_1, webID_2, ..., webID_n\}$

$\mathbf{C}_{user} = \{userID_1, userID_2, ..., userID_n\}$

$\mathbf{C}_{platform} = \{h,\ o,\ a\}$

$\mathbf{C}_{vendor} = \{microsoft,\ google,\ the\_mozilla\_foundation\}$

$\mathbf{C}_{product} = \{internet\_explorer,\ windows\_10,\ adobe\_reader\}$ ■

TABLE III: Example predicates and explanation

Predicate	Explanation
posted($post_1$, $webID_1$)	$post_1$ was posted on the website $webID_1$.
at_risk($\mathcal{D}$, V)	Post $\mathcal{D}$ discussed vendor V being at-risk.
user_preference ($userID_1$, $microsoft$)	$userID_1$ prefers to post discussions regarding Microsoft systems at-risk.
previously_seen ($webID_1$, $adobe\_flash$)	At-risk discussions regarding Adobe Flash are discussed in $webID_1$.
parent($microsoft$, $safari$)	Vendor Microsoft is a parent of product Safari.

The language also contains a set of predicate symbols that have constants or variables as arguments, and denote events that can be either *true* or *false*. We denote the set of predicates with **P**; examples of predicates are shown in Table III. For instance, $user\_preference(userID_1, microsoft)$ will either be true or false, and denotes the event where $userID_1$ prefers to post discussions regarding *microsoft* systems at-risk.

A *ground atom* is composed by a predicate symbol and a tuple of constants, one for each argument—hence, ground atoms have no variables. The set of all ground atoms is denoted with **G**. A *ground literal* L is either a ground atom or a negated ground atom. An example of a ground atom for our running example is $posted(post_1, webID_1)$. In the following, we will use $\mathbf{G}'$ to denote a subset of **G**.

In order to be able to deal with conflicting information and offer explainable results, we choose a structured argumentation framework [26] for our model; our approach works by creating *arguments* (in the form of a set of rules and facts) that compete with each other to identify at-risk system given a hacker discussion on D2web. In this case, arguments are *defeated* based on the evaluation of contradicting information in other arguments. This procedure is commonly known as a *dialectical process* since it follows the same structure as dialogues between humans—as such, arguments that are *undefeated* (or *warranted*, in DeLP) prevail. Structuring the analysis in this manner also allows us to leverage the resulting structure, since the set of all prevailing arguments give a clear map of how the conclusion is supported by the available data.

The clear benefit of the transparency afforded by such a process is that it lets a (human) security analyst not only add new arguments based on new evidence, but also eliminate information identified as incorrect (perhaps because it is out of date, or because it comes from a source newly identified as untrustworthy) and fine-tune the model for better performance. Since the argumentation model can deal with inconsistent information, it draws a natural analogy to the way humans settle disputes when there is disagreement. Having a clear explanation of why one argument is chosen over others is a desirable characteristic for both the analyst and for organizations to make decisions and policy changes. We now briefly discuss some preliminaries on DeLP.

Defeasible Logic Programming: DeLP is a formalism that combines logic programming with defeasible argumentation; we refer the interested reader to [12] for a fully detailed presentation of the system.

In summary, the formalism is made up of several constructs, namely *facts*, *strict rules*, and *defeasible rules*. Facts represent statements obtained from evidence, and are therefore always considered to be true; similarly, strict rules are logical combinations of elements (facts or other inferences) that can always be performed. On the contrary, defeasible rules can be thought of as strict rules that *may* be true in some situations, but *could* be false if certain contradictory evidence is presented. These three constructs are used to build *arguments*, and DeLP programs are simply sets of facts, strict rules and defeasible rules. We adopt the usual notation for DeLP programs, denoting the program (or knowledge base) with $\Pi = (\Theta, \Omega, \Delta)$, where Θ is the set of facts, Ω is the set of strict rules, and Δ is the set of defeasible rules. Examples of the three constructs are provided with respect to the dataset in Fig. 4. We now describe the notation used to denote these constructs.

Facts (Θ) are ground literals that represent atomic information or its (strong) negation ($\neg$).

Strict Rules (Ω) represent cause and effect information; they are of the form $L_0 \leftarrow L_1, ...L_n$, where L_0 is a literal and $\{L_i\}_{i>0}$ is a set of literals.

Defeasible Rules (Δ) are weaker versions of strict rules, and are of the form $L_0 \prec L_1, ..., L_n$, where L_0, is the literal and $\{L_i\}_{i>0}$ is a set of literals.

When a hacker discussion happens on D2web, the model can be used to derive arguments to determine the at-risk system (in terms of platform, vendor, and product). Derivation follows the same mechanism as classical logic programming [16]; the main difference is that DeLP incorporates defeasible argumentation, which decides which arguments are warranted, which arguments are defeated, and which arguments should be considered to be *blocked*—the latter are arguments that are involved in a conflict for which a winner cannot be determined.

Fig. 4 shows a ground argumentation framework demonstrating constructs derived from our D2web data. For instance, θ_1 indicates the fact that a hacker discussion $post_1$ was posted on the D2web website $webID_1$, and θ_5 indicates that user $userID_1$ prefers to post discussions regarding $apple$ products. For the strict rules, ω_1 says that for a given post $post_1$ posing a threat to operating system (o), the vendor *sandisk* cannot be at risk if the parent of *sandisk* is not operating system (o)[4]. Defeasible rules can be read similarly; δ_2 indicates that if $post_1$ poses a threat to the vendor *apple*, the product *safari* can be at-risk if *apple* is the parent of *safari*. By replacing the constants with variables in the predicates we can derive a non-ground argumentation framework that can be applied in general.

[4]This encodes the CPE hierarchical structure.

Θ :
$\theta_1 =$ posted($post_1, webID_1$)
$\theta_2 =$ posted($post_1, userID_1$)
$\theta_3 =$ parent($o, micorsoft$)
$\theta_4 =$ parent($apple, safari$)
$\theta_5 =$ user_preference($userID_1, apple$)
$\theta_6 =$ previously_seen($webID_1, o$)

Ω :
$\omega_1 =$ $\neg$ at_risk($post_1, sandisk$) $\leftarrow$
at_risk($post_1, o$),
$\neg$parent($o, sandisk$)
$\omega_2 =$ $\neg$ at_risk($post_1, internet\_explorer$) $\leftarrow$
at_risk($post_1, apple$),
$\neg$parent($apple, internet\_explorer$)

Δ :
$\delta_1 =$ at_risk($post_1, microsoft$) $\prec$
at_risk($post_1, o$),
parent($o, microsoft$)
$\delta_2 =$ at_risk($post_1, safari$) $\prec$
at_risk($post_1, apple$),
parent($apple, safari$)
$\delta_3 =$ at_risk($post_1, apple$) $\prec$
user_preference($userID_1, apple$)
$\delta_4 =$ at_risk($post_1, o$) $\prec$
previously_seen($webID_1, o$)

Fig. 4: A ground argumentation framework.

$\langle \mathcal{A}_1$, at_risk($post_1, microsoft$) $\rangle$	$\mathcal{A}_1 = \{\delta_1, \delta_4, \theta_3\}$
$\langle \mathcal{A}_2$,at_risk($post_1, safari$) $\rangle$	$\mathcal{A}_2 = \{\delta_2, \delta_3, \theta_4\}$
$\langle \mathcal{A}_3$, at_risk($post_1, apple$)$\rangle$	$\mathcal{A}_3 = \{\delta_3, \theta_5\}$
$\langle \mathcal{A}_4$, at_risk($post_1, o$)$\rangle$	$\mathcal{A}_4 = \{\delta_4, \theta_6\}$

Fig. 5: Example ground arguments from Figure 4.

Definition 1. ***(Argument)*** *An argument for a literal L is a pair $\langle \mathcal{A}, L\rangle$, where $\mathcal{A} \subseteq \Pi$ provides a minimal proof for L meeting the requirements: (1) L is defeasibly derived from $\mathcal{A}$[5], (2) $\Theta \cup \Omega \cup \Delta$ is not contradictory, and (3) $\mathcal{A}$ is a minimal subset of Δ satisfying 1 and 2, denoted $\langle \mathcal{A}, L\rangle$.*

Literal L is called the *conclusion* supported by the argument, and $\mathcal{A}$ is the *support*. An argument $\langle \mathcal{B}, L\rangle$ is a *subargument* of $\langle \mathcal{A}, L'\rangle$ iff $\mathcal{B} \subseteq \mathcal{A}$. The following examples discuss arguments for our scenario.

Example 2. *Fig. 5 shows example arguments based on the KB from Fig. 4; here, $\langle \mathcal{A}_3$, at_risk($post_1, apple$)$\rangle$ is a subargument of $\langle \mathcal{A}_2$, at_risk($post_1, safari$)$\rangle$.* ∎

For a given argument there may be counter-arguments that contradict it. A *proper defeater* of an argument $\langle A, L\rangle$ is a counter-argument that—by some criterion—is considered to be better than $\langle \mathcal{A}, L\rangle$; if the two are incomparable according to this criterion, the counterargument is said to be a *blocking* defeater. The default criterion used in DeLP for argument comparison is *generalized specificity* [32], but any domain-specific criterion (or set of criteria) can be devised and deployed.

[5]This means that there exists a derivation consisting of a sequence of rules that ends in L—that possibly includes defeasible rules.

A sequence of arguments is called an *argumentation line*. There can be more than one defeater argument, which leads to a tree structure that is built from the set of all argumentation lines rooted in the initial argument. In this *dialectical tree*, every child can defeat its parent (except for the root), and the leaves represent unchallenged arguments; this creates a map of all possible argumentation lines that can be used to decide whether or not an argument is defeated. Arguments that either have no attackers or all attackers have been defeated are said to be *warranted*.

Given a literal L and an argument $\langle \mathcal{A}, L\rangle$, in order to decide whether or not a literal L is warranted, every node in the dialectical tree $\mathcal{T}(\langle \mathcal{A}, L\rangle)$ is recursively marked as "D" (*defeated*) or "U" (*undefeated*), obtaining a marked *dialectical tree* $\mathcal{T}^*(\langle \mathcal{A}, L\rangle)$ where:

- All leaves in $\mathcal{T}^*(\langle \mathcal{A}, L\rangle)$ are marked as "U"s, and
- Let $\langle \mathcal{B}, q\rangle$ be an inner node of $\mathcal{T}^*(\langle \mathcal{A}, L\rangle)$. Then, $\langle \mathcal{B}, q\rangle$ will be marked as "U" iff every child of $\langle \mathcal{B}, q\rangle$ is marked as "D". Node $\langle \mathcal{B}, q\rangle$ will be marked as "D" iff it has at least one child marked as "U".

Given argument $\langle \mathcal{A}, L\rangle$ over Π, if the root of $\mathcal{T}^*(\langle \mathcal{A}, L\rangle)$ is marked "U", then $\mathcal{T}^*(\langle \mathcal{A}, h\rangle)$ *warrants* L and that L is *warranted* from Π. It is interesting to note that warranted arguments correspond to those in the grounded extension of a Dung abstract argumentation system [11].

An implemented DeLP system therefore takes as inputs a set of facts, strict rules, and defeasible rules, as well as a *query literal*. Note that while the set of facts and strict rules must be consistent (non-contradictory), the set of defeasible rules can be inconsistent—the presence of such inconsistency is the root of "interesting" cases. We engineer our at-risk system framework as a set of defeasible and strict rules whose structure was created manually, but are dependent on values learned from a historical corpus of D2web data. Then, for a given post discussing a vulnerability, we instantiate a set of facts for that situation; this information is then provided as input into the DeLP system, which uses heuristics to generate all arguments for and against every possible components of the system (platforms, vendors, products) for the post discussion. Dialectical trees based on these arguments are analyzed, and a decision is made regarding which components are *warranted*. This results in a *reduced set* of potential choices, which we then use as input into a classifier to obtain the at-risk system. The following section discusses these steps in full detail.

VI. Experiments

We frame the identification of at-risk systems as a multi-label classification problem for each of the system component (platform, vendor, and product)—the basic step involves extracting textual features from the discussions to be used as input to the machine learning models. We now describe the data pre-processing steps and the standard machine learning approaches, along with the metrics used for evaluating the models.

A. Data Representation

As mentioned above, we use text-based features to represent the hacker discussions on the D2web, which are then used as input to the machine learning models. Some of the discussions are in foreign languages (cf. Fig. 2). The commercial data collection platform automatically identifies the language and translates it to English using the Google Translate API [14]. The following pre-processing steps are taken to address different challenges. We employ two feature engineering techniques namely TF-IDF and Doc2Vec.

Text Cleaning. We remove all non-alphanumeric characters from hacker discussions. This removes any *special characters* that do not contribute towards making the decision.

Misspellings and Word Variations. Misspellings and word variations are frequently observed in the discussions on the D2web, leading to separate features in the feature vector if a standard bag-of-words (BOW) approach is used. In BOW, we create a dictionary of all the word occurrences in the training set; then, for a particular discussion, the feature vector is created by looking up which words have occurred and their count in the discussion. Misspellings and word variations will thus be represented as different words; to address this, we use character n-gram features. As an example, consider the word "execute"—if we were using tri-gram character features, the word "execute" would yield the set of features:

$$\{\textit{"exe"}, \textit{"xec"}, \textit{"ecu"}, \textit{"cut"}, \textit{"ute"}\}.$$

The benefit of this technique is that the variations or misspellings of the word, such as "execution", "executable", or ""exxecute", will all have common features. We found that using character n-grams in the range 3–7 worked best in our experiments.

TF-IDF Features. We vectorize the n-gram features using the term frequency-inverse document frequency (TF-IDF) model, which creates a vocabulary of all the n-grams in the discussion. In TF-IDF, the importance of an n-gram feature increases with the number of times it occurs, but is normalized by the total number of n-grams in the description. This eliminates common words from being important features. We consider the top 1,000 most frequent features (using more than 1,000 features did not improve the performance, but rather only added to the training and testing time).

Doc2Vec Features. Doc2Vec is a feature engineering technique to generate document vector (in our case document refers to a discussion), which acts as input to the classifier to identify at-risk systems. In Doc2Vec, first, a vector representation of each word in the document in computed by taking into account the words around it (to maintain context) and then these word vectors are averaged to get a representation of the document. We implement Doc2Vec using the *gensim* library in Python[6]. It was been previously used to classify tweets [33] as well as product descriptions [15].

[6]https://radimrehurek.com/gensim/models/doc2vec.html

B. Supervised Learning Approaches

We conducted our experiments using the following standard machine learning approaches implemented using a Python machine learning library[7].

Support Vector Machine (SVM). Support vector machines (SVM) work by finding a separating margin that maximizes the geometric distance between classes (in our case, different platforms, vendors, and products). Given the geometric interpretation of the data, the separating margin is referred to as a *hyperplane*.

Random Forest (RF). Ensemble methods are popular classification tools. They are based on the idea of generating multiple predictors used in combination to classify new unseen samples. We use a random forest that combines bagging for each tree with random feature selection at each node to split the data, thus generating multiple decision tree classifiers. Each decision tree gives its own opinion on test sample classification, which are then merged to make a final decision.

Naive Bayes Classifier (NB). NB is a probabilistic classifier that uses Bayes' theorem under the assumption of independent features. During training, we compute the conditional probabilities of a sample of a given class having a certain feature. We also compute the prior probabilities for each class, i.e., the fraction of the training data belonging to each class. Since Naive Bayes assumes that the features are statistically independent, the likelihood for a sample S represented with a set of features a associated with a class c is given by:

$$\mathit{Pr}(c|S) = \mathit{Pr}(c) \times \prod_{i=1}^{d} \mathit{Pr}(a_i|c).$$

Decision Tree (DT). This is a hierarchical recursive partitioning algorithm. We build the decision tree by finding the *best split feature*, i.e., the feature that maximizes the information gain at each split of a node.

Logistic Regression (LOG-REG). Logistic regression classifies samples by computing the *odds ratio*, which gives the strength of association between the features and the class.

C. Evaluation Metrics

In our experiments, we evaluate performance based on three metrics: *precision*, *recall*, and *F1 measure*. For a given hacker discussion, precision is the fraction of labels (platforms, vendors, or products) that the model associated with the discussion that were *actual labels* in the ground truth. Recall, on the other hand, is the fraction of ground truth labels *identified* by the model. The F1 measure is the harmonic mean of precision and recall. In our results, we report the average precision, recall, and F1 for all the test discussions.

[7]http://scikit-learn.org/stable/

TABLE IV: Average Precision, Recall, and F1 measure for NB, LOG-REG, DT, RF and SVM to identify at-risk systems.

Component	Model	Precision	Recall	F1 measure
Platform	NB	0.68	0.65	0.66
	LOG-REG	**0.72**	0.76	0.74
	DT	0.66	0.70	0.68
	RF	0.70	0.75	0.72
	SVM	**0.72**	**0.78**	**0.76**
Vendor	NB	0.37	0.34	0.36
	LOG-REG	0.28	0.25	0.27
	DT	0.39	0.43	0.41
	RF	**0.40**	0.43	0.41
	SVM	**0.40**	**0.48**	**0.44**
Product	NB	0.19	0.14	0.16
	LOG-REG	0.20	0.13	0.16
	DT	0.22	0.15	0.18
	RF	0.22	**0.25**	0.24
	SVM	**0.26**	0.24	**0.25**

D. Baseline Model (BM)

For the baseline model, we only leverage the machine learning technique to identify the at-risk systems. We create training and testing sets by sorting the discussions by posted time on the website (to avoid temporal intermixing). We reserve the first 80% of the samples for training and the rest (20%) for testing. We employed both TF-IDF and Doc2Vec as feature engineering techniques. On conducting the experiments, it was observed that TF-IDF performed better than Doc2Vec in all the experiments. Hence we only report the results using TF-IDF features.

Results. Table IV shows the average performance of the machine learning technique for each component of the at-risk system. For *platform* identification, SVM performs the best with the following averages:

- *precision*: 0.72,
- *recall*: 0.78, and
- *F1 measure*: 0.76.

LOG-REG had similar precision, but lower recall. Similarly, for vendor identification, SVM performs the best with averages:

- *precision*: 0.40,
- *recall*: 0.48, and
- *F1 measure*: 0.44,

with RF having similar precision. For *platform* identification, SVM had the best performance:

- *precision*: 0.28,
- *recall*: 0.24 (comparable to RF), and
- *F1 measure*: 0.25.

$$\Theta: \quad \theta_1 = \mathsf{posted}(\mathcal{D}, \mathcal{W}) \qquad \theta_2 = \mathsf{posted}(\mathcal{D}, \mathcal{U})$$

Fig. 6: Facts defined for each test discussion.

$$\Delta: \quad \text{For } s \in \mathcal{S}_w: \quad \delta_1 = \mathsf{at\_risk}(\mathcal{D}, s) \prec \mathsf{previously\_seen}(\mathcal{W}, s). \quad \text{For } s \in \mathcal{S}_u: \quad \delta_2 = \mathsf{at\_risk}(\mathcal{D}, s) \prec \mathsf{user\_preference}(\mathcal{U}, s).$$

Fig. 7: Defeasible rules for platform identification.

Since SVM performs consistently better for all three classification problems, moving forward we use SVM as our machine learning component in the reasoning framework (cf. Fig. 1).

E. Reasoning Framework (RFrame)

As we go down the CPE hierarchy, the number of possible labels for vendors and products increases largely as the number of discussions representing each label decreases, thus making learning difficult and decreasing performance. We address this issue by proposing a set of strict and defeasible rules for platform, vendor, and product identification. We note that these rules arise from the discussion that is being evaluated and do not require parameter learning.

We use the notation described in Table V for defining our constructs (facts, strict rules, and defeasible rules). We note that facts cannot have variables, only constants (however, to compress the program for presentation purposes, we use *meta-variables* in facts). To begin, we define the facts (see Fig. 6): θ_1 states that a hacker discussion $\mathcal{D}$ was posted on the D2web website $\mathcal{W}$ (can be either forum or marketplace), and θ_2 states that the user $\mathcal{U}$ posted the discussion. For each level in the CPE hierarchy, we define additional rules discussed as follows.

Platform Model. The first level of system identification is identifying the platform that the hacker discussion is a threat to. We compute previously discussed platforms on D2web websites under consideration. Similarly, which platform the user under consideration prefers (based on their previous postings) is also computed. This shows preferred platform discussions on websites and by users, which can aid the machine learning model in reducing the number of platforms it can identify from. The DeLP components that model platform identification are shown in Fig. 7. For the defeasible rules, δ_1 indicates that all the platforms $\mathcal{S}_w$ previously seen in the D2web website $\mathcal{W}$ where the current discussion $\mathcal{D}$ is observed are likely at-risk, δ_2 indicates that all the platforms $\mathcal{S}_u$ from user $\mathcal{U}$'s previous postings are also likely at-risk.

Vendor Model. The second level is identifying the at-risk vendor. For this case, we use the platform result from the previous model, taking that as a DeLP *fact*. The DeLP components that model vendor identification are shown in Fig. 8. Here, the fact θ_1 indicates the platform identified for the discussion—note that multiple platforms may be identified based on the

TABLE V: Notation and Explanations

Notation	Explanation
$\mathcal{D}$	The hacker discussion (posted on the website) under consideration.
$\mathcal{W}$	Website (marketplace or forum) where the hacker discussion was posted.
$\mathcal{S}_w$, $\mathcal{V}_w$ and $\mathcal{P}_w$	The set of platforms, vendors and products at-risk by the hacker discussions previously seen in $\mathcal{W}$ under consideration respectively.
$\mathcal{U}$	User posting the hacker discussion.
$\mathcal{S}_u$, $\mathcal{V}_u$ and $\mathcal{P}_u$	The set of platforms, vendors and products at-risk by the hacker discussions previously posted by user $\mathcal{U}$ under consideration respectively.
$\mathcal{S}_p$, $\mathcal{V}_p$ and $\mathcal{P}_p$	The set of platforms, vendors and products identified by the machine learning model at each level in the hierarchy for hacker discussions under consideration respectively.
s_i, v_i and p_i	Each element of the set $\mathcal{S}_p$, $\mathcal{V}_p$ and $\mathcal{P}_p$ representing a single platform, vendor or product respectively.

Θ : For $s \in \mathcal{S}_p$:
$\theta_1 =$ $\mathsf{at\_risk}(\mathcal{D}, s)$

Ω : For $s \in \mathcal{S}_p$:
$\omega_1 =$ $\neg\, \mathsf{at\_risk}(\mathcal{D}, \mathsf{v}_i) \leftarrow \mathsf{at\_risk}(\mathcal{D}, s), \neg\mathsf{parent}(s, \mathsf{v}_i)$

Δ : For $v \in V_w$:
$\delta_1 =$ $\mathsf{at\_risk}(\mathcal{D}, v) \prec \mathsf{previously\_seen}(\mathcal{W}, v)$.
For $v \in V_u$:
$\delta_2 =$ $\mathsf{at\_risk}(\mathcal{D}, v) \prec \mathsf{user\_preference}(\mathcal{U}, v)$.
For $s \in \mathcal{S}_p$:
$\delta_3 =$ $\mathsf{at\_risk}(\mathcal{D}, \mathsf{v}_i) \leftarrow \mathsf{at\_risk}(\mathcal{D}, s), \mathsf{parent}(s, \mathsf{v}_i)$

Fig. 8: Defeasible rules for vendor identification.

Θ : For $v \in V_p$:
$\theta_1 =$ $\mathsf{at\_risk}(\mathcal{D}, v)$

Ω : For $v \in V_p$:
$\omega_1 =$ $\neg\, \mathsf{at\_risk}(\mathcal{D}, \mathsf{p}_i) \leftarrow \mathsf{at\_risk}(\mathcal{D}, v), \neg\mathsf{parent}(v, \mathsf{p}_i)$

Δ : For $p \in \mathcal{P}_w$:
$\delta_1 =$ $\mathsf{at\_risk}(\mathcal{D}, p) \prec \mathsf{previously\_seen}(\mathcal{W}, p)$.
For $p \in \mathcal{P}_u$:
$\delta_2 =$ $\mathsf{at\_risk}(\mathcal{D}, p) \prec \mathsf{user\_preference}(\mathcal{U}, p)$.
For $v \in V_p$:
$\delta_3 =$ $\mathsf{at\_risk}(\mathcal{D}, \mathsf{p}_i) \leftarrow \mathsf{at\_risk}(\mathcal{D}, v), \mathsf{parent}(v, \mathsf{p}_i)$

Fig. 9: Defeasible rules for product identification.

discussion. The strict rule ω_1 states that for a given post $\mathcal{D}$ posing a threat to platform s, the vendor v_i cannot be at-risk if the parent of v_i is not the identified platform s. This rule is based on the CPE hierarchy obtained from NVD. For the defeasible rules, δ_1 indicates that all the vendors V_w previously seen in the D2web website $\mathcal{W}$ where the current hacker discussion $\mathcal{D}$ is observed are likely at-risk, δ_2 indicates that all the vendors V_u from user $\mathcal{U}$'s previous postings are also likely at-risk, and δ_3 states that for a given post $\mathcal{D}$ posing a threat to platform s, all the vendors whose parent is the identified platform are likely at-risk. This rule is also based on the CPE hierarchy from NVD.

Product Model. The third level is identifying the at-risk product. For this case, we use the vendor result from the previous model; as before, we use that as a DeLP *fact*. The DeLP components that model product identification are shown in Fig. 9. Here, the fact θ_1 indicates the vendor identified for the discussion—again, multiple vendors may be identified based on the discussion. The strict rule ω_1 states that for a given post $\mathcal{D}$ posing a threat to vendor v, the product p_i cannot be at-risk if the parent of p_i is not the identified vendor v (again, based on the CPE hierarchy). For the defeasible rules, δ_1 indicates that all the products $\mathcal{P}_w$ previously seen in the D2web website $\mathcal{W}$ where the current hacker discussion $\mathcal{D}$ is observed are likely at-risk, δ_2 indicates that all the products $\mathcal{P}_u$ from user $\mathcal{U}$'s previous postings are also likely at-risk, and δ_3 states that for a given post $\mathcal{D}$ posing a threat to vendor v, all the products whose parent (in the CPE hierarchy) is the identified vendor are likely at-risk.

Results. We evaluate the reasoning framework using an experimental setup similar to the one discussed in the baseline model. We report the precision, recall, and F1 measure for each of the system components and compare them with the best performing baseline model (BM). Table VI shows the comparison between the two models.

For *platform* identification, RFrame outperforms BM in terms of precision: 0.83 vs. 0.72 (a 15.27% improvement), while maintaining the same recall. Similarly, for *vendor* and *product* identification there was significant improvement in precision: 0.56 vs. 0.40 (a 40% improvement) and 0.41 vs. 0.26 (a 57.69% improvement), respectively, with comparable recall with respect to the baseline model. The major reason for the jump in precision is the reduction of possible labels based on the arguments introduced that aids the machine learning model to make the correct decision.

TABLE VI: Average Precision, Recall, and F1 measure comparison between the baseline model (BM) and reasoning framework (RFrame).

Component	Model	Precision	Recall	F1 measure
Platform	BM	0.72	**0.78**	0.76
	RFrame	**0.83**	**0.78**	**0.80**
Vendor	BM	0.40	**0.48**	0.44
	RFrame	**0.56**	0.44	**0.50**
Product	BM	0.26	**0.24**	0.25
	RFrame	**0.41**	0.21	**0.30**

VII. Discussion

The performance of the reasoning system highlights that our hybrid framework identifies at-risk systems with higher precision with respect to the approach using only machine learning classifiers. In our application, we desire a high precision—while maintaining at least comparable recall—in order to provide high value risk assessment of systems; low precision is often equated to a less reliable framework. The majority of misclassifications are a result of less data representing those systems in the training set; for some system components, the instances can be as low as having only one discussion in the training set. This issue becomes more relevant as we go down the hierarchy with large numbers of vendors and products. In some test instances, for the same platform and vendor, a new product not previously known to be at-risk becomes vulnerable due to a newly disclosed vulnerability. In this case, the reasoning framework is not able to identify the product since it was not previously observed, and this can contribute to a misclassification.

From a security analyst's perspective, the reasoning framework not only provides a list of possible at-risk systems but also provides arguments indicating *why* a particular system was identified as being at-risk. This lets the analyst evaluate the decisions made by the framework and fine-tune it if necessary. For cases where a new product (not previously discussed in training) is at-risk, even a partial identification of the system (in terms of platform and vendor) is of value to the analyst. Based on the alert provided by the framework, the analyst can manually evaluate the arguments and the discussions to identify possible products, depending on the platform and vendor identified by the framework.

VIII. Related Work

Threat assessment of systems is critical to organizations' security policy. Over the years, CVSS [9] has become a standard metric that organizations use to determine if their systems are at risk of being targeted by hackers. Unfortunately, case studies have shown poor correlation between the CVSS score and which system are at-risk [2].

Identifying targeted systems through open source intelligence. Open source intelligence has been used previously to identify and predict vulnerabilities that are likely to be exploited to determine which systems are at risk. [35] has looked to predict the likelihood that a software has a vulnerability not yet discovered using the national vulnerability database (NVD). They show that NVD has a poor prediction capability in doing so due to limited amount of information available. On the other hand, [28] looks to predict if a real world exploit is available based on vulnerabilities disclosed from Twitter data. The authors report high accuracies of 90% using a resampled, balanced, and temporal mixed dataset, not reflective of real world scenarios [6]. Identifying threats to critical infrastructure by analyzing interactions on hacker forums was studied in [17]. Here the authors reply on keyword based queries to identify such threats from hacker interactions. Tools to automatically identify products offered in cyber criminal markets was proposed in [25]. This technique looks to extract products mentioned in the description of the item that is being offered, a problem different than what we address – identifying targeted systems not explicitly stated in the forum discussions.

More recently, researchers have shown increased interest on gathering threat intelligence from D2web to pro-actively identify digital threats and study hacker communities to gather insights. Researchers have focused on building infrastructure to gather threat information from markets (regarding goods and services sold) and forums (discussions regarding exploits and) [22], [27], studying the different product categories offered in darkweb markets – creating a labeled dataset [18], analyzing hacker forums and carding shops to identify potential threats [5], identify expert hackers to determine their specialties [1], identify key hackers based on posted content, their network and since when they are active in the forum [19]. For vulnerability research, studies look to leverage vulnerability mentions in the D2web to predict the likelihood of exploitation using a combination of machine learning and social network techniques [4], [3]. These techniques rely on the mentions of CVE numbers to identify likely targeted systems (which is a small fraction of vulnerabilities [4]), not taking into account discussions where a CVE number is not mentioned. On the other hand, we look to identify the at-risk systems *without having a CVE number*, which is a different problem from those tackled in previous work.

Identifying targeted systems through software analysis. Another way of identifying targeted softwares with vulnerabilities deals with analyzing the *software itself* in order to determine which component of the software is most likely to contain a vulnerability. Mapping past vulnerabilities to vulnerable software components was proposed in [20], where the authors found that components with function calls and import statements are more likely to have a vulnerability. A similar method was employed by [29], [34], where text mining was used to forecast whether a particular software component contains vulnerabilities. Similar text mining techniques for vulnerability discovery are listed in [13]. The text mining methods create a count dictionary of terms used in the software, which are

used as features to identify vulnerabilities. These methods suffer from the issue of not knowing which vulnerabilities might be of interest to hackers. On the other hand, we work with hacker discussions posing a threat to systems that are of clearly of interest to hackers since they are discussing them on the D2web websites—vulnerabilities mentioned on D2web regarding systems are more likely to be exploited [4].

IX. Conclusion

In this paper, we demonstrated how a reasoning framework based on the DeLP structured argumentation system can be leveraged to improve the performance of identifying at-risk systems based on hacker discussions on the D2web. DeLP programs built on discussions found on forums and marketplaces afford a reduction on the set of possible platforms, vendors, and products that are likely to be at-risk by the hackers participating in the discussion. This reduction of potential labels leads to better precision while almost maintaining comparable recall as compared to the baseline model that only leverages machine learning techniques. Knowing discussed systems by threat actors as possible targets helps organizations achieve better threat assessment for their systems.

Acknowledgment

Authors of this work were supported by the U.S. Department of the Navy, Office of Naval Research, grant N00014-15-1-2742 as well as the Arizona State University Global Security Initiative (GSI), by CONICET and Universidad Nacional del Sur (UNS), Argentina and by the EU H2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement 690974 for the project "MIREL". The authors would like to thank Cyber Reconnaissance, Inc. for providing the data.

References

[1] A. Abbasi, W. Li, V. Benjamin, S. Hu, and H. Chen. Descriptive analytics: Examining expert hackers in web forums. In *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*, pages 56–63. IEEE, 2014.

[2] L. Allodi and F. Massacci. Comparing vulnerability severity and exploits using case-control studies. *ACM Transactions on Information and System Security (TISSEC)*, 17(1):1, 2014.

[3] M. Almukaynizi, A. Grimm, E. Nunes, J. Shakarian, and P. Shakarian. Predicting cyber threats through the dynamics of user connectivity in darkweb and deepweb forums. In *ACM Computational Social Science*. ACM, 2017.

[4] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian, and P. Shakarian. Proactive identification of exploits in the wild through vulnerability mentions online. In *2017 International Conference on Cyber Conflict (CyCon U.S.)*, pages 82–88, Nov 2017.

[5] V. Benjamin, W. Li, T. Holt, and H. Chen. Exploring threats and vulnerabilities in hacker web: Forums, irc and carding shops. In *Intelligence and Security Informatics (ISI), 2015 IEEE International Conference on*, pages 85–90. IEEE, 2015.

[6] B. L. Bullough, A. K. Yanchenko, C. L. Smith, and J. R. Zipkin. Predicting exploitation of disclosed software vulnerabilities using open-source data. In *Proceedings of the 2015 ACM International Workshop on International Workshop on Security and Privacy Analytics*. ACM, 2017.

[7] CPE. Official common platform enumeration dictionary. https://nvd.nist.gov/cpe.cfm, Last Accessed: Feb 2018.

[8] CVE. Common vulnerabilities and exposures: The standard for information security vulnerability names. http://cve.mitre.org/, Last Accessed: Feb 2018.

[9] CVSS. Common vulnerability scoring system. https://www.first.org/cvss, Last Accessed: Feb 2018.

[10] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, 2004.

[11] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

[12] A. J. García and G. R. Simari. Defeasible logic programming: An argumentative approach. *Theory and practice of logic programming*, 4(1+ 2):95–138, 2004.

[13] S. M. Ghaffarian and H. R. Shahriari. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM Computing Surveys (CSUR)*, 50(4):56, 2017.

[14] Google. Google cloud translation api documentation. https://cloud.google.com/translate/docs/, Last Accessed: Feb 2018.

[15] H. Lee and Y. Yoon. Engineering doc2vec for automatic classification of product descriptions on o2o applications. *Electronic Commerce Research*, pages 1–24, 2017.

[16] J. W. Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.

[17] M. Macdonald, R. Frank, J. Mei, and B. Monk. Identifying digital threats in a hacker web forum. In *Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference on*, pages 926–933. IEEE, 2015.

[18] E. Marin, A. Diab, and P. Shakarian. Product offerings in malicious hacker markets. In *Intelligence and Security Informatics (ISI), 2016 IEEE Conference on*, pages 187–189. IEEE, 2016.

[19] E. Marin, J. Shakarian, and P. Shakarian. Mining key-hackers on darkweb forums. In *International Conference on Data Intelligence and Security (ICDIS), 2018*. IEEE, 2018.

[20] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller. Predicting vulnerable software components. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 529–540. ACM, 2007.

[21] NIST. National vulnerability database. https://nvd.nist.gov/, Last Accessed: Feb 2018.

[22] E. Nunes, A. Diab, A. Gunn, E. Marin, V. Mishra, V. Paliath, J. Robertson, J. Shakarian, A. Thart, and P. Shakarian. Darknet and deepnet mining for proactive cybersecurity threat intelligence. In *Proceeding of ISI 2016*, pages 7–12. IEEE, 2016.

[23] E. Nunes, P. Shakarian, G. I. Simari, and A. Ruef. Argumentation models for cyber attribution. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*, pages 837–844. IEEE, 2016.

[24] C. P. Pfleeger and S. L. Pfleeger. *Security in computing*. Prentice Hall Professional Technical Reference, 2002.

[25] R. S. Portnoff, S. Afroz, G. Durrett, J. K. Kummerfeld, T. Berg-Kirkpatrick, D. McCoy, K. Levchenko, and V. Paxson. Tools for automated analysis of cybercriminal markets. In *Proceedings of the 26th International Conference on World Wide Web*, pages 657–666. International World Wide Web Conferences Steering Committee, 2017.

[26] I. Rahwan, G. R. Simari, and J. van Benthem. *Argumentation in artificial intelligence*, volume 47. Springer, 2009.

[27] J. Robertson, A. Diab, E. Marin, E. Nunes, V. Paliath, J. Shakarian, and P. Shakarian. *Darkweb Cyber Threat Intelligence Mining*. Cambridge University Press, 2017.

[28] C. Sabottke, O. Suciu, and T. Dumitra. Vulnerability disclosure in the age of social media: exploiting twitter for predicting real-world exploits. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 1041–1056, 2015.

[29] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen. Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering*, 40(10):993–1006, 2014.

[30] J. Shakarian, A. T. Gunn, and P. Shakarian. Exploring malicious hacker forums. In *Cyber Deception*, pages 259–282. Springer, 2016.

[31] P. Shakarian, G. I. Simari, G. Moores, and S. Parsons. Cyber attribution: An argumentation-based approach. In *Cyber Warfare*, pages 151–171. Springer, 2015.

[32] F. Stolzenburg, A. J. García, C. I. Chesnevar, and G. R. Simari. Computing generalized specificity. *Journal of Applied Non-Classical Logics*, 13(1):87–113, 2003.

[33] L. Q. Trieu, H. Q. Tran, and M.-T. Tran. News classification from social media using twitter-based doc2vec model and automatic query expansion. In *Proceedings of the Eighth International Symposium on Information and Communication Technology*, pages 460–467. ACM, 2017.

[34] J. Walden, J. Stuckman, and R. Scandariato. Predicting vulnerable components: Software metrics vs text mining. In *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*, pages 23–33. IEEE, 2014.

[35] S. Zhang, D. Caragea, and X. Ou. An empirical study on using the national vulnerability database to predict software vulnerabilities. In *International Conference on Database and Expert Systems Applications*, pages 217–231. Springer, 2011.