Mining Threat Intelligence about Open-Source Projects and Libraries from Code Repository Issues and Bug Reports

Lorenzo Neil, Sudip Mittal, and Anupam Joshi
University of Maryland, Baltimore County, Baltimore, MD 21250, USA
Email: {lneil1, smittal1, joshi}@umbc.edu

***Abstract*— Open-Source Projects and Libraries are being used in software development while also bearing multiple security vulnerabilities. This use of third party ecosystem creates a new kind of attack surface for a product in development. An intelligent attacker can attack a product by exploiting one of the vulnerabilities present in linked projects and libraries.**

In this paper, we mine threat intelligence about open source projects and libraries from bugs and issues reported on public code repositories. We also track library and project dependencies for installed software on a client machine. We represent and store this threat intelligence, along with the software dependencies in a security knowledge graph. Security analysts and developers can then query and receive alerts from the knowledge graph if any threat intelligence is found about linked libraries and projects, utilized in their products.

***Index Terms*— Cybersecurity, Artificial Intelligence, Threat Intelligence, Open Source Projects, Intelligence Acquisition**

I. INTRODUCTION

In the normal course of software development, developers and coders rely on various open source projects and libraries. Open source projects and libraries comprise of source code that is open for anyone to inspect, modify, update or enhance [17]. This type of programming embraces innovative ideas of collaboration, open exchange, transparency, and community-oriented development. Software developers and programmers are then able to access this source code easily and add their input of code or fix parts that may not function as intended. This is much different from "proprietary" or "closed source" software which only allows access and modifications from the person, group, or organization that own it. Also, closed source software requires users to sign a license and accept the terms placed under it by the originators which differs drastically from legal terms of open source software [25]. The design of open source projects and their licenses encourages all computer programmers to access public projects and collectively make edits however they find fit.

This use of third party ecosystem creates a new kind of attack surface for the product in development. The developers link these products and code libraries with little consideration of threats, vulnerabilities, and exposures present in them. It is estimated that about 70-80% of source code implemented in current day projects are from open source communities [25]. The abundance of open source projects also increases the need to track security vulnerabilities and bugs that are present within the libraries they are linked to. Such security vulnerabilities are then inherited by the product in development. An intelligent attacker can then attack the product by exploiting one of these vulnerabilities. The problem gets more complex with the number of recorded open source vulnerabilities each year [17]. Another problem that even compounds the issue is the fact that the open source libraries linked by a developer can themselves link to other vulnerable libraries and so on. A vulnerability present in any of these can cause a domino effect, resulting in the product being developed inheriting a vulnerability.

A recent case would be the exploitation by hackers on an open source security vulnerability within Equifax's, a U.S. based credit rating bureau, database during July - August 2017 [17]. In this attack, it was reported that hackers attacked Equifax by exploiting a web applications vulnerability to access private files of customers. The reported breach stemmed from the popular open source programming framework that Equifax utilized for their web applications known as Apache Struts [17]. The Apache Struts framework included a critical vulnerability that allowed easy access for hackers to Equifax's database. Names, social security numbers, birth dates, addresses, and drivers license numbers from an estimated 143 million customers were compromised [17]. Equifax is not alone in the sense that other companies also have implemented known vulnerable open source components into their organization and products. This implementation of open source software with persistent vulnerabilities is a growing concern for software developers.

In order to better protect the product in development, it is necessary to create a repository of known vulnerabilities in these open source libraries and projects. Threat intelligence about some of these projects can be mined using *traditional* sources like NIST's National Vulnerability Database (NVD)[1], United States Computer Emergency Readiness Team (US-CERT)[2], etc. Other sources which are more *non-traditional* are, Twitter, Reddit, blogs, and news. Non-traditional sources are faster than the traditional ones. There is a significant gap between initial vulnerability announcement and NVD release [19]. Vulnerability threat intelligence appears first on non-traditional sources [18]. Mining non-traditional sources is becoming really important. In our previous work, we have developed *CyberTwitter* [13] and *Cyber-All-Intel* [14] systems that mines threat intelligence

[1] `https://nvd.nist.gov/`

[2] `https://www.us-cert.gov/`

from various OSINT sources. The systems then represent cybersecurity intelligence in knowledge graphs and vector spaces so it can be used by artificial intelligence based cyber-defense systems.

In this paper, we propose a 'shift-left' [11] form of security. We create a system that will inform a developer about potential threats and vulnerabilities that the product in development might inherit as a result of linking to a vulnerable open source project or library. We mine threat intelligence from issues and bugs raised on web-based hosting service for version control like, GitHub [3], GitLab [4], bitbucket [1], etc. These platforms have been used by developers to host and collaborate on source code development [5]. We extract vulnerabilities raised on these platforms and represent them in a security knowledge graph. The knowledge graph then becomes a store for various vulnerabilities and exposures present in various open source projects, products and libraries (see Figure 1). This knowledge can then be queried by various developers helping them create products that are secure from the ground up. Pervasive software security entails cyber supply chain risk management, where the developers are made aware of various threats present in libraries they link with the development of their products.

We also create another application that can track installed software on a client machine, and then use the above mentioned knowledge graph to *reason* alerts for the security analyst. These alerts warn the analyst if an installed software is linked to a vulnerable open source library or project. We built a proof of concept for this application that runs on a linux installation.

The rest of the paper is organized as follows - Section II discusses some of our related work. Our methodology is presented in Section III. We discuss our experimental setup and evaluation in Section IV. We conclude in Section V.

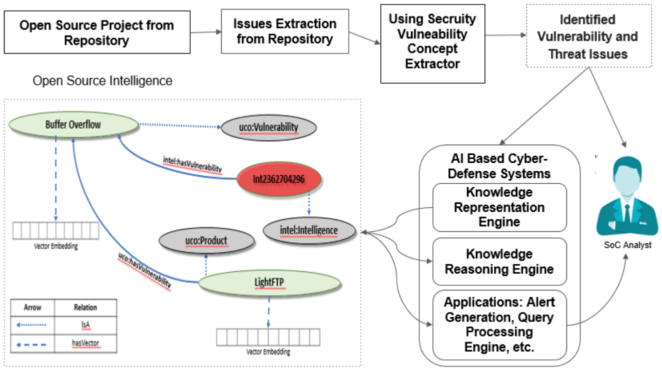

Open Source Project from Repository
Issues Extraction from Repository
Using Secruity Vulneability Concept Extractor
Identified Vulnerability and Threat Issues
Open Source Intelligence
Buffer Overflow
uco:Vulnerability
intel:hasVulnerability
Int2362704296
intel:Intelligence
uco:Product
uco:hasVulnerability
LightFTP
Vector Embedding
Vector Embedding
Arrow
Relation
IsA
hasVector
AI Based Cyber-Defense Systems
Knowledge Representation Engine
Knowledge Reasoning Engine
Applications: Alert Generation, Query Processing Engine, etc.
SoC Analyst

Fig. 1. Mining threat intelligence from bug and issue reports. Intelligence is stored in a Security Knowledge Graph. The graph represents the threat intelligence: "*I've noticed a buffer overflow in the Unix version of LightFTP v1.1*"

II. Related Work

A. Open Source Software Security

Open source development has created a variety of new software security challenges. Closed source proponents claim that the availability of open source software's code allows hackers to easily find a way to compromise the security. They believe that "hackers finding the source code and placing back doors for unauthorized access to their systems is one of the biggest limiting factors for open source software" [10]. Closed source systems based on the principle of security through obscurity, may have theoretical or actual security vulnerabilities, but its owners or designers believe that they are more secure if the flaws are not known.

Security of open source systems stems from the Kerckhoffs's Law (sometimes refereed to as Shannon's maxim), which states that "A cryptosystem should be designed to be secure if everything is known about it except the key information." [21]. The strength behind open source software stems from it's wide audience looking at the code and collectively finding problems. In our work, we are interested in gathering threat intelligence about issue and vulnerability reports in open source repositories from project contributors.

B. Mining Threat Intelligence

The production of new computer systems and software has always attracted the likes of cyber criminals who have compromised such technology through hacking or infecting with viruses and/or malware. Their objectives can range from shutdown of a website, data breaches, acts of fraud, or distribution of viruses. However, understanding the volume of these cyber attacks and all the broad spectrum of vectors in which they occur greatly aids cyber security professionals effort to create cyber-defense strategies. The exponential growth of hacking communities among the web also means that security professionals also gain an exponential amount of threat intelligence from those communities and are tasked with a challenge to monitor their behavior. The threat intelligence that comes from these communities can be forums consisting of numerous members, posts, or threads that require constant filtering and mining for security vulnerabilities [11].

The use of semantic knowledge graphs in cybersecurity has gained traction in the past few years. Considerable attention has been dedicated to develop techniques for extracting concepts related to security vulnerabilities, affected software, hardware, and organizations and generating its semantic representation [16][8][22][13][14]. While previous research focused on sources such as NVD, social media, and security blogs, our work is applied to bug and issue reports taken from public code repositories, where the content is different from other sources.

C. Knowledge Graph systems for Threat Intelligence

Effective strategies to precisely counter cyber attacks relies heavily on the detection of future threats, attacker's behavior, and accessibility of threat intelligence. Being able to attain cyber threat intelligence will eliminate the possibility of vulnerabilities being exploited and improves the ability to react to future attacks [7].

Knowledge graphs for cybersecurity have been used by [9] to create association and combination of data and information from multiple sources. Takahashi et al. [24], [23] built an ontology for cybersecurity operational information based on actual cybersecurity operations mainly focused on cloud computing-based services. Rutkowski et al. [20] created a cybersecurity information exchange framework, known as CYBEX. Another insightful work by Xie et al. [26] discusses uncertainty modeling for cyber security centered around near real-time security analysis such as intrusion response.

Intrusion detection and prevention systems (IDPS) help by examining signature markings of cyber infrastructure for malicious activity and generating alerts. These systems however cannot detect malicious activity if the systems do not already have that signature in their databases [13], [15]. Also, these systems are point source solutions and cannot organize threat intelligence coming from heterogeneous sources. Cyber research has focused on combining traditional signature based intrusion detection with ontological reasoning to further improve knowledge graphs. This allows for the interpretation of means and consequences for links between cyber threats and vulnerabilities whose signatures are not yet in the databases.

III. Methodology

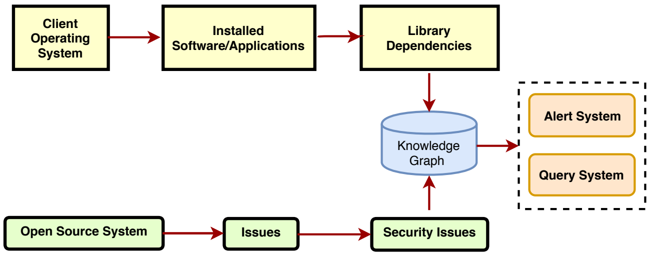


Fig. 2. Knowledge Graph with ontology of projects and threat intelligence.

In this section, we will describe our system architecture (see Figure 2). Our system has a two subparts. The first part tracks repositories of various open source projects and libraries. It monitors various issues and bugs reported on these repositories and filters out all security related issues that mention vulnerabilities, threats, etc. It then converts this intelligence into a machine readable format. The second part tracks software running on a machine and identifies all libraries and dependencies. Both parts feed the collected information to a common security knowledge graph.

Once the security knowledge graph gets populated, a system administrator or a developer can use a query engine and an alert generation system to find vulnerable projects and libraries. We now describe each of the different subparts and applications in detail:

A. Initializing a Security Knowledge Graph

Our knowledge graph consists of vulnerabilities, threats found on open source projects, library repositories, and installed software with their dependencies. For our knowledge graph we used the Unified Cybersecurity Ontology (UCO) [22] to provide cybersecurity domain knowledge. An Intelligence ontology [13] was used to represent threat intelligence. We also created a *software dependency* ontology that helps represent installed software and its dependencies (see Section III-B). In our system, we also matched entities of various softwares and libraries that we encounter to their DBpedia [6] equivalent entities. For example the installed software *vlc* is matched to *dbr:VLC_media_player*, this helps us retrieve more global information like, developer, genre, operating system, etc. about a particular software.

B. Tracking Installed Software & Mining Library Linking

To track and generate alerts for all software that inherits vulnerabilities from open source dependencies, we compiled the knowledge of all installed software on a machine and its dependencies. We obtained all of the software programs already installed on a machine and then for each program we listed all of the library dependencies and environment variables. The environment variables helped us identify open source projects that are utilized by a particular installed software. In order to create a proof of concept, we focused on a linux installation. We utilized the *objdump*[3] tool to disassemble installed software and trace dependencies. Once we obtained the information we asserted it in our security knowledge graph mentioned in Section III-A, using the software dependency ontology.

An abstract view of the software dependency ontology is show in Figure 3. The software class is divided into three sub classes which are the project, product, and library. The product class represents installed software on a machine. A product *utilizes* a project, and *is linked to* a library.

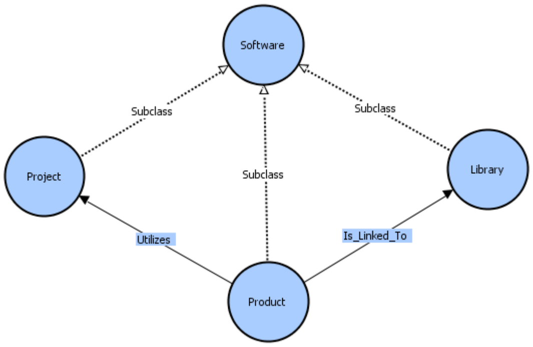


Fig. 3. Software Dependency Ontology Schema.

C. Bug & Issue Tracking

So as to mine active threat intelligence for various open source projects and libraries, we collected bug reports and issues posted by developers on code repositories. We then asserted the information in our security knowledge graph (see Section III-D). Figure 5, shows a vulnerability in a popular FTP client for Unix. For our system, as a proof of concept, we have developed a crawler and tracker to collect threat intelligence from repositories hosted on GitHub [3]. We utilized GitHub's Rest API to collect and track open issues, closed issues, and pull requests from a project repository.

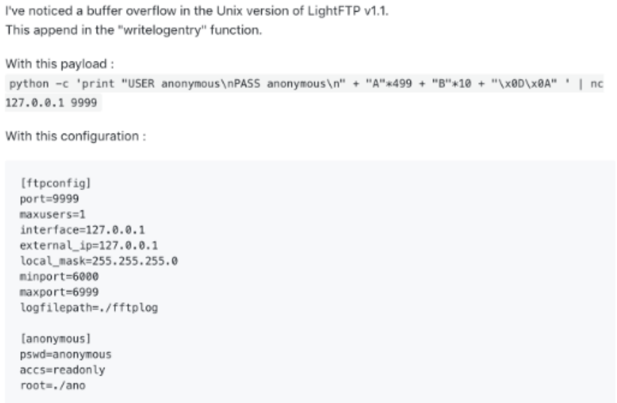
[3] https://linux.die.net/man/1/objdump

```
@prefix uco: <http://accl.umbc.edu/ns/ontology/uco#> .
@prefix soft: <http://accl.umbc.edu/ns/ontology/software#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dbp: <http://dbpedia.org/resource#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

</usr/bin/python3.6> a soft:Product ;
soft:Is_Linked_To <libutil.so.1> ;
soft:Is_Linked_To <libpython3.6m.so.1.0> ;
soft:Is_Linked_To <libm.so.6> ;
soft:Is_Linked_To <libdl.so.2> ;
soft:Is_Linked_To <libpthread.so.0> ;
soft:Is_Linked_To <libc.so.6> .

<libutil.so.1> a soft:Library .
<libpython3.6m.so.1.0> a soft:Library .
<libm.so.6> a soft:Library .
<libdl.so.2> a soft:Library .
<libpthread.so.0> a soft:Library .
<libc.so.6> a soft:Library .
```

Fig. 4. RDF for libraries linked by the program Python 3.6.



I've noticed a buffer overflow in the Unix version of LightFTP v1.1.
This append in the "writelogentry" function.

With this payload :

```
python -c 'print "USER anonymous\nPASS anonymous\n" + "A"*499 + "B"*10 + "\x0D\x0A" ' | nc
127.0.0.1 9999
```

With this configuration :

```
[ftpconfig]
port=9999
maxusers=1
interface=127.0.0.1
external_ip=127.0.0.1
local_mask=255.255.255.0
minport=6000
maxport=6999
logfilepath=./fftplog

[anonymous]
pswd=anonymous
accs=readonly
root=./ano
```

Fig. 5. Sample issue showing security buffer overflow in a popular Unix FTP client.

We retrieved all the issues present in multiple project repositories in JSON format and stored them in data frames. The resulting data frames for each project were then filtered to remove any non-security related issues. We extracted issues that consisted of terms related to security vulnerabilities using a *Security Vulnerability Concept Extractor* (SVCE) [13]. The SVCE is able to tag every sentence with the following concepts: Means of an attack, Consequence of an attack, affected software, hardware and operating system, version numbers, network related terms, file names and other technical terms.

SVCE discards all issues for which it fails to identify at-least two security concept, thus further removing non-security related data. The extracted concepts are also used to generate an RDF Linked Data representation for every issue that may be queried by security systems to protect against potential attacks.

D. Asserting Threat Intelligence

Once the SVCE identifies security concepts and entities. We then associated them with Uniform Resource Identifiers (URIs). These URIs are then converted to nodes in our security knowledge graph. We used the Unified Cybersecurity Ontology [22] which integrates heterogeneous data and knowledge schemas from different cybersecurity systems and standards.

We used DBpedia to link various knowledge graph nodes to real world concepts. Entity matching process is performed by using DBpedia [6] and DBpedia spotlight [12]. For example we can use DBpedia to map the string "Adobe Flash" to *dbr:Adobe_Flash*. This external knowledge graph help us map our entities to real world conceptual instances.

After entity linking, we stored the linked data as RDF triples in our security knowledge graph. In our system we need information of cybersecurity intelligence. Threat intelligence is temporal in nature and may contain other meta-data like, origin, credibility, provenance, etc. UCO though gives us a domain overview of cybersecurity it cannot handle temporal nature of events. So as to handle time in events we use the intelligence ontology [13].

To give an example, Figure 6 shows the RDF statements created for the intelligence "*I've noticed a buffer overflow in the Unix version of LightFTP v1.1*" (Figure 5). A graphical representation of the same intelligence, Int2362704296 has been shown in Figure 1. Once we obtain the intelligence in the RDF format, we use it to create the alert and the query system.

```
@prefix uco: <http://accl.umbc.edu/ns/ontology/uco#> .
@prefix intel: <http://accl.umbc.edu/ns/ontology/intelligence#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dbp: <http://dbpedia.org/resource#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

<Int2362704296> a intel:Intelligence ;
intel:hasVulnerability <buffer_overflow> .

<LightFTP> a uco:Product ;
uco:hasVulnerability <buffer_overflow> ;
owl:sameAs dbp:FTP-server .

<buffer_overflow> a uco:Vulnerability ;
uco:affectsProduct <LightFTP> ;
owl:sameAs dbp:buffer_overflow .
```

Fig. 6. RDF for textual input "I've noticed a buffer overflow in the Unix version of LightFTP v1.1". Also, $owl : sameAs$ property has been used to augment the data using an external source 'DBpedia' [6].

E. Applications

Once we populated our security knowledge graph with the information about installed software and linked dependencies (see Section III-B) and also add, threat intelligence mined from issues and bug reports (Section III-D), we utilized it to create an alert generation system and a query system.

1) Query System: The developer before linking to an open source library or using a project should be able to query the security knowledge graph to check for known vulnerabilities. We have created a SPARQL[4] endpoint that can accept queries which run on our knowledge graph. An example query to list all vulnerabilities in *LightFTP*:

```
SELECT ?y WHERE {
?LightFTP <hasVulnerability> ?y .
}
```

An example query to look up vulnerabilities in linked libraries to the installed application *firefox*:

```
SELECT ?x WHERE {
?firefox <Is_Linked_to> ?z
?z <hasVulnerability> ?x.
}
```

2) Alert Generation System: The system will reason on our security knowledge graph and generate alerts. In our system we include SWRL rules[5] so as to generate alerts. SWRL rules contain two parts, antecedent part (body), and a consequent (head). The body and head consist of conjunctions of a set of atoms. Informally, a rule may be read as meaning that if the antecedent holds (is true), then the consequent must also hold. For our system we see two potential alert scenarios:

1) A developer is linking to a library or a project with known vulnerabilities and threats: The system will take in all the libraries that a developer wants to use and then trigger an alert if it finds a vulnerability or threat in any one of these libraries. We see this as a developer initiated scenario.

   Our alert system also checks other linked libraries that link to the ones mentioned by the developer. Some example rules included in our system:

   ```
   Rule for vulnerable project utilization:

   Product(?x)^ Utilizes(?x, ?y)^
       hasVulnerability(?y, ?z)
   ==> RaiseAlert(?x,"Yes")

   Rule for linked library vulnerability
       check:

   Product(?x)^ Is_Linked_To(?x, ?y)^
       hasVulnerability(?y, ?z)
   ==> RaiseAlert(?x,"Yes")

   Rule for secondary linked library
       vulnerability check:

   Product(?x)^ Is_Linked_To(?x, ?y)^
       Is_Linked_To(?y, ?z)^
       hasVulnerability(?z, ?u)
   ==> RaiseAlert(?x,"Yes")
   ```

   The rules raise an alert if any vulnerability or threat is found in linked libraries and projects. For the first rule above 'Rule for vulnerable project utilization', given a product node $?x$, the rules check for edge: $Utilizes(?x, ?y)$, to hop to the graph node $?y$. Once at $?y$ it checks for the edge: $hasVulnerability(?y, ?z)$ to hop to node $?z$. If the above node exists an alert is generated. A similar technique is used to evaluate other rules mentioned above.

2) An installed application on a client machine is linked to compromised dependencies: This is an information triggered alert, where influx of new threat intelligence warrants a lookup for vulnerable installed software. The system should automatically inform a security analyst that an installed application on a client machine is vulnerable. An example rule for this alert:

   ```
   Rule for vulnerable libraries:

   Library(?x)^ hasVulnerability(?x, ?y)^
       Is_Linked_To(?z, ?x)
   ==> RaiseAlert(?z,"Yes")

   Rule for vulnerable projects:

   Project(?x)^ hasVulnerability(?x, ?y)^
       Utilizes(?z, ?x)
   ==> RaiseAlert(?z,"Yes")
   ```

IV. Experimental Setup & Evaluation

In order to evaluate the system and collect empirical data we ran our system under experimental conditions. The system was run on a Ubuntu[6] Linux installation with 81 installed programs, some of these were pre-installed. We extracted the library and project dependencies for these 81 installed programs and represent this information in our security knowledge graph (see Section III-B). Figure 4, shows the triples generated for a popular installed software.

We collected 110,800 issues posted on GitHub [3], using the GitHub Rest API. For our experiments and to create a valid proof of concept, we limit issue collection to the GitHub repositories for the 81 installed projects. We also use only the issues posted after January 2018 in our analysis. Out of the 110,800 issues collected our SVCE (see Section III-C) filtered 9,194 security issues. We then assert these security vulnerabilities in our knowledge graph (see Section III-D). Figure 6, lists triples generated for a popular FTP client.

We performed an initial evaluation of our prototype system using the bug-reports collected. We evaluate the quality of the tags generated by the SVCE module, and how often our system missed intelligence because it discarded relevant details. We did not evaluate our entity matching process as it was done through DBpedia APIs. Human assessments and annotation was done by students familiar with the cybersecurity domain.

For our first evaluation measure we check the quality of tags generated by our SVCE module. We tagged 150

[4] https://www.w3.org/TR/rdf-sparql-query/

[5] https://www.w3.org/Submission/SWRL/

[6] https://www.ubuntu.com/

randomly selected security issues and then manually checked the tags. The annotators had to evaluate if the SVCE output was correct, partially correct or wrong. Our annotators agreed on the fact that 98 issues were marked correctly by the SVCE module and out of the remaining 52 issues, 18 were tagged completely wrong and the remaining were tagged partially correct. The annotators were then asked to look into the alerts generated for the 98 issues correctly identified, the system raised appropriate alerts for each.

We evaluated the loss of intelligence because of discarded issues, i.e., those not included in the dataset of 9,194 security issues. A random sample of 200 issues was generated from the discarded issues. In these, our annotators found 9 issues with actionable security related information. We believe that these were wrongfully tagged by our SVCE module because of spelling mistakes, unidentifiable characters, informal slang expressions, non-English words, etc.

V. Conclusion & Future Work

In this paper, we described our method for mining threat intelligence about open-source projects and libraries from host repository issues and bug reports. We collected bug reports consisting of all reported issues from multiple open source repositories on Github with Github's Rest API. Then, we extracted all the issues that had information related to cybersecurity. These security issues were then stored into our knowledge graph as RDF triples. We also track library and project dependencies for installed software on a client machine. The library dependencies with their corresponding applications are stored as RDF triples in the security knowledge graph. The knowledge graph was then used to create an alert and query system that can warn users of security vulnerabilities. Security analysts and developers can then query and receive alerts from the knowledge graph if any threat intelligence is found about linked libraries and projects, utilized in their products. The system can also issue alerts, about installed libraries, if new vulnerabilities are disclosed on these code repositories.

For future work, we will like to mine threat intelligence from other web services, such as GitLab [4], Code Triage [2], etc, as they will also contain information about open-source projects with security vulnerabilities. These different code repositories host different projects which can serve as vital sources of threat intelligence. This variety would further improve our system and add more information to our knowledge graph. We would also like to investigate other knowledge representation techniques that can be used to store threat intelligence. Using multiple representations like vector space embeddings, can further improve the quality of applications built to utilize them.

Acknowledgement

The work was partially supported by a gift from IBM Research, USA and the UMBC Louis Stokes Alliance for Minority Participation - LSAMP.

References

[1] Bitbucket. https://bitbucket.org.

[2] Codetriage. https://www.codetriage.com/.

[3] Github. https://github.com.

[4] Gitlab. https://about.gitlab.com/.

[5] Github predicts hottest 2018 open source trends, Feb 2018.

[6] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. *DBpedia: A Nucleus for a Web of Open Data*. Springer, 2007.

[7] Michael Iannacone. Developing an ontology for cyber security knowledge graphs, 2015.

[8] Arnav Joshi, Ravendar Lal, Tim Finin, and Anupam Joshi. Extracting cybersecurity related linked data from text. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, pages 252–259, Sept 2013.

[9] Michael Kandefer, S Shapiro, Adam Stotz, and Moises Sudit. Symbolic reasoning in the cyber security domain. 2007.

[10] G. Lawton. Open source security: opportunity or oxymoron? *Computer*, 35(3):18–21, Mar 2002.

[11] Ryan Littlefield. Cyber threat intelligence: Applying machine learning, data mining and text feature extraction to..., Feb 2018.

[12] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia spotlight: shedding light on the web of documents. In *7th Int. Conf. on Semantic Systems*, pages 1–8. ACM, 2011.

[13] Sudip Mittal, Prajit Kumar Das, Varish Mulwad, Anupam Joshi, and Tim Finin. Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*, pages 860–867. IEEE, 2016.

[14] Sudip Mittal, Anupam Joshi, and Tim Finin. Thinking, fast and slow: Combining vector spaces and knowledge graphs. *corpus*, 2:3, 2017.

[15] Sumit More, Mary Matthews, Anupam Joshi, and Tim Finin. Ieee xplore full-text pdf:, 2012.

[16] Varish Mulwad, Wenjia Li, Anupam Joshi, Tim Finin, and Krishnamurthy Viswanathan. Extracting information about security vulnerabilities from web text. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, volume 3, pages 257–260. IEEE, 2011.

[17] opensource.com. What is open source? `https://opensource.com/resources/what-open-source`, 2018.

[18] The Register. Most vulnerabilities first blabbed about online or on the dark web. `https://www.theregister.co.uk/2017/06/08/vuln_disclosure_lag/`, Jun 2017.

[19] The Register. Make america late again: Us 'lags' china in it security bug reporting. `https://www.theregister.co.uk/2017/10/20/us_china_vuln_reporting/`, Oct 2017.

[20] Anthony Rutkowski, Youki Kadobayashi, Inette Furey, Damir Rajnovic, Robert Martin, Takeshi Takahashi, Craig Schultz, Gavin Reid, Gregg Schudel, Mike Hird, et al. Cybex: the cybersecurity information exchange framework (x. 1500). *ACM SIGCOMM Computer Communication Review*, 40(5):59–64, 2010.

[21] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[22] Zareen Syed, Ankur Padia, M. Lisa Mathews, Tim Finin, and Anupam Joshi. UCO: A unified cybersecurity ontology. In *AAAI Workshop on Artificial Intelligence for Cyber Security*, pages 14–21. AAAI Press, 2015.

[23] Takeshi Takahashi, Hiroyuki Fujiwara, and Youki Kadobayashi. Building ontology of cybersecurity operational information. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, page 79. ACM, 2010.

[24] Takeshi Takahashi, Youki Kadobayashi, and Hiroyuki Fujiwara. Ontological approach toward cybersecurity in cloud computing. In *Proceedings of the 3rd international conference on Security of information and networks*, pages 100–109. ACM, 2010.

[25] Steven J. Vaughan-Nichols. It's an open-source world: 78 percent of companies run open-source software, Dec 2015.

[26] Peng Xie, Jason H Li, Xinming Ou, Peng Liu, and Renato Levy. Using bayesian networks for cyber security analysis. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 211–220. IEEE, 2010.