Semantic Mapping of Security Events to Known Attack Patterns

Xiao Ma[1], Elnaz Davoodi[1], Leila Kosseim[1], and Nicandro Scarabeo[2]

[1] Dept. of Computer Science and Software Engineering, Concordia University
1515 Ste-Catherine Street West
Montréal, Québec, Canada H3G 2W1
maxiao0215@gmail.com, ed.davoodi@gmail.com, leila.kosseim@concordia.ca

[2] Hitachi Systems Security Inc., 955 Boulevard Michéle-Bohec Suite 244
Blainville, Québec, Canada J7C 5J6
nicandro.scarabeo@hitachi-systems-security.com

Abstract. In order to provide cyber environment security, analysts need to analyze a large number of security events on a daily basis and take proper actions to alert their clients of potential threats. The increasing cyber traffic drives a need for a system to assist security analysts to relate security events to known attack patterns. This paper describes the enhancement of an existing Intrusion Detection System (IDS) with the automatic mapping of snort alert messages to known attack patterns. The approach relies on pre-clustering snort messages before computing their similarity to known attack patterns in Common Attack Pattern Enumeration and Classification (CAPEC). The system has been deployed in our partner company and when evaluated against the recommendations of two security analysts, achieved an f-measure of 64.57%.

Keywords: Semantic Similarity · Clustering · Cyber Security

1 Introduction

With the increasing dependence on computer infrastructure, *cyber security* has become a major concern for organizations as well as individuals. *Cyber security* refers to the collection of tools, approaches and technologies which are used to prevent unauthorized behavior in cyber environments [1]. In order to detect and prevent harmful behaviour, sensors are typically installed in computer networks. Each sensor is equipped with several security systems, such as Intrusion Detection Systems (IDS) or Asset Detection Tools [2]. These systems perform network traffic analysis in real-time, detect suspicious activities and generates security events. Snort [3] is a widely used IDS system installed in many sensors [4]. By capturing and decoding suspicious TCP/IP packets, snort generates messages regarding network traffic data to facilitate the task of security analysts to recognize suspicious behaviours and act accordingly [3]. Fig 1 shows seven examples snort messages.

Semantic Mapping of Security Events to Known Attack Patterns
Xiao Ma, Elnaz Davoodi, Leila Kosseim, Nicandro Scarabeo
Pages 91-98

Conference proceedings | © 2018
Natural Language Processing and Information Systems
23rd International Conference on Applications of Natural Language to Information Systems, NLDB 2018, Paris, France, June 13-15, 2018, Proceedings

1. *APP-DETECT Apple OSX Remote Mouse usage*
2. *FILE-IDENTIFY RealPlayer skin file attachment detected*
3. *SQL generic sql exec injection attempt - GET parameter*
4. *FILE-OTHER XML exponential entity expansion attack attempt*
5. *MALWARE-OTHER Win.Exploit.Hacktool suspicious file download*
6. *BROWSER-PLUGINS MSN Setup BBS 4.71.0.10 ActiveX object access*
7. *BROWSER-IE Microsoft Internet Explorer asynchronous code execution attempt*

Fig. 1. Examples of 7 snort messages

Recognizing suspicious behaviours from snort messages is difficult as the messages are typically short. Thus, today the task is still mostly performed by human security experts. However, because of the increasing volume of network traffic, the workload of security analysts has become much heavier and the possibility of not detecting a security risk has become critical. In order to allow security analysts to better assess risks, the automatic mapping of security events to attack patterns is desired. The goal of this paper is to enhance the performance of an existing Intrusion Detection System with the automatic mapping of snort alert messages to known attack patterns.

2 Previous Work

An Intrusion Detection System (IDS) is an essential part of a typical Security Information and Event Management System (SIEM). An IDS is responsible for detecting unauthorized behaviours by matching security events to known network and host-based attack patterns stored in knowledge bases [5]. If an attack pattern does not exist in the knowledge base, the IDS cannot detect the related malicious behaviour [6]. Therefore, it is necessary to keep this knowledge base up to date as new attack patterns are discovered. Much research in this area has thus focused on the automatic curation of knowledge bases of attack patterns.

As many novel attack patterns or cyber security related concepts are discussed online in forums and chat rooms (e.g. [7]), several natural language processing techniques have been proposed to extract security concepts from these unstructured texts (e.g. [8–12]). In particular, a system to extract cyber security concepts from semi-structured and unstructured texts, such as online documents and the National Vulnerability Database (NVD) [13], was introduced in [7]. It consists of an SVM classifier to identify cyber security related texts, a Wikitology [14] and a named entity tagger [15] to extract security concepts [10, 12, 16].

In addition to work on mining known attack patterns from the Web, the research community has also curated and made publicly available several resources of vulnerability, cyber attack patterns and security threats. These include the

U.S. National Vulnerability Database (NVD) [13], Common Weakness Enumeration (CWE) [17] and Common Attack Pattern Enumeration and Classification (CAPEC) [18]. The CWE inventories software weakness types; while the NVD is built upon the CWE to facilitate data access. On the other hand, CAPEC not only provides common vulnerabilities, but also contains attack steps that hackers mostly use to explore software weaknesses as well as the suggested mitigation.

To our knowledge, not many novel techniques have been proposed to automatically match security events messages to known attack patterns based directly on their natural language descriptions. [4] proposed a system based on a KNN classifier to address this issue. However no formal evaluation of the quality of the output is provided. Our work builds upon this work by formally evaluating its performance with real cyber security data and enhancing it to improve its recall.

3 Original System

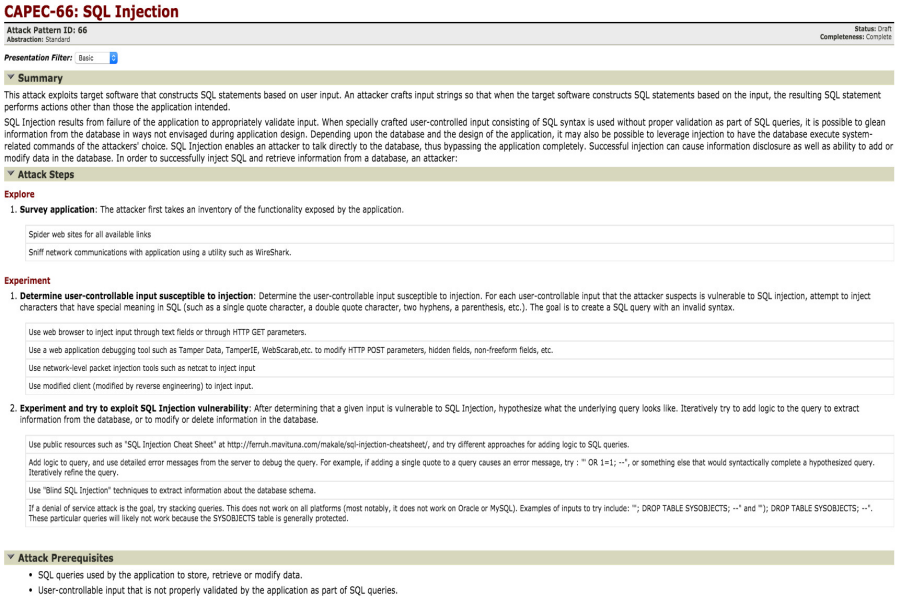
The approach of [4] was used as our baseline system. Its purpose is to map security events to known attack patterns in CAPEC [18] based solely on their natural language descriptions.

3.1 Description of the Original System

The system of [4] uses snort alert messages as input and tries to identify the n most relevant CAPEC fields to propose to cyber security analysts. CAPEC (Common Attack Pattern Enumeration and Classification [18]) is a publicly available knowledge base containing 508 known attack patterns. As shown in Fig. 2, an attack pattern in CAPEC is composed of various fields that are described in natural language. These include a *Summary* of the attack pattern, *Attack Steps Survey*, *Experiments*, *Attack Prerequisites etc.* On average, each CAPEC pattern contains 10 fields for a total of 5,096 fields.

Since the snort messages are relatively short (only 8 tokens on average), not much information can be extracted from them to be used in building automated model for detecting the attacks. To overcome this limitation, they are first expanded by replacing common domain keywords with a richer description. To do this, security experts analyzed 32,246 snort alert messages, and identified 68 important terms. For each term, they then wrote a description of about 5 words. Figure 3.2 shows the expansion of 3 such terms. By replacing domain terms in the original snort messages with their longer descriptions, the length of each snort message increased from an average of 8 words to an average of 15 words. To map these expanded security events to specific fields in CAPEC, both snort messages and CAPEC fields are pre-processed (tokenized, stop words removed and stemmed), and unigrams, bigrams and trigrams are used as features. Frequency variance (FV) is then used to filter out features that appear either

too often or too rarely. Using TF-IDF and the cosine similarity, the distance between snort messages and each CAPEC field is then computed. Finally, the 3 most similar CAPEC fields that have a distance smaller than some threshold t, are selected.



CAPEC-66: SQL Injection

Attack Pattern ID: 66
Abstraction: Standard
Status: Draft
Completeness: Complete

Presentation Filter: Basic

Summary

This attack exploits target software that constructs SQL statements based on user input. An attacker crafts input strings so that when the target software constructs SQL statements based on the input, the resulting SQL statement performs actions other than those the application intended.

SQL Injection results from failure of the application to appropriately validate input. When specially crafted user-controlled input consisting of SQL syntax is used without proper validation as part of SQL queries, it is possible to glean information from the database in ways not envisaged during application design. Depending upon the database and the design of the application, it may also be possible to leverage injection to have the database execute system-related commands of the attackers' choice. SQL Injection enables an attacker to talk directly to the database, thus bypassing the application completely. Successful injection can cause information disclosure as well as ability to add or modify data in the database. In order to successfully inject SQL and retrieve information from a database, an attacker:

Attack Steps

Explore

1. **Survey application**: The attacker first takes an inventory of the functionality exposed by the application.

Spider web sites for all available links
Sniff network communications with application using a utility such as WireShark.

Experiment

1. **Determine user-controllable input susceptible to injection**: Determine the user-controllable input susceptible to injection. For each user-controllable input that the attacker suspects is vulnerable to SQL injection, attempt to inject characters that have special meaning in SQL (such as a single quote character, a double quote character, two hyphens, a parenthesis, etc.). The goal is to create a SQL query with an invalid syntax.

Use web browser to inject input through text fields or through HTTP GET parameters.
Use a web application debugging tool such as Tamper Data, TamperIE, WebScarab,etc. to modify HTTP POST parameters, hidden fields, non-freeform fields, etc.
Use network-level packet injection tools such as netcat to inject input
Use modified client (modified by reverse engineering) to inject input.

2. **Experiment and try to exploit SQL Injection vulnerability**: After determining that a given input is vulnerable to SQL Injection, hypothesize what the underlying query looks like. Iteratively try to add logic to the query to extract information from the database, or to modify or delete information in the database.

Use public resources such as "SQL Injection Cheat Sheet" at http://ferruh.mavituna.com/makale/sql-injection-cheatsheet/, and try different approaches for adding logic to SQL queries.
Add logic to query, and use detailed error messages from the server to debug the query. For example, if adding a single quote to a query causes an error message, try : '' OR 1=1; --'', or something else that would syntactically complete a hypothesized query. Iteratively refine the query.
Use "Blind SQL Injection" techniques to extract information about the database schema.
If a denial of service attack is the goal, try stacking queries. This does not work on all platforms (most notably, it does not work on Oracle or MySQL). Examples of inputs to try include: ''; DROP TABLE SYSOBJECTS; --'' and ''); DROP TABLE SYSOBJECTS; --''. These particular queries will likely not work because the SYSOBJECTS table is generally protected.

Attack Prerequisites

- SQL queries used by the application to store, retrieve or modify data.
- User-controllable input that is not properly validated by the application as part of SQL queries.

Fig. 2. Example of a CAPEC attack pattern

3.2 Evaluation of the Original System

In [4], the system was only evaluated in terms of coverage, i.e. the number of snort messages that were matched to at least one CAPEC field. However, the quality of the recommended CAPEC fields were not evaluated. To address this issue, we created a gold-standard dataset by asking 2 cyber security experts to evaluate the mapping of 3,165 snort messages mapped to at most 5 CAPEC fields. This gave rise to 16,826 mappings. Each mapping was annotated by the 2 experts with one of 3 levels of quality:

- *a correct mapping*: the analysts could use the CAPEC field directly as a solution,
- *an acceptable mapping*: the analysts could use the CAPEC field in order to generate a solution, or
- *a wrong mapping*: the recommended CAPEC field was not useful.

Term	Expanded Description
file-identify	*file extension file magic or header found in the traffic*
server-webapp	*web based applications on servers*
exploit-kit	*exploit kit activity*

Fig. 3. Examples of Domain Term Expansion

Table 3.2 shows statistics of the gold-standard. As the table shows, 9,222 mappings were labelled as correct; 5,496 mappings were tagged as acceptable and 2,108 mapping were judged wrong.

Table 1. Statistics of the gold-standard dataset

Tag	Number of Mappings
Correct	9,222
Acceptable	5,496
Wrong	2,108
Total	**16,826**

The output generated by the original system was then evaluated against the gold-standard dataset. We used the same 3,165 snort messages and evaluated the overlapping answers; mappings provided by the system that were not included in the gold-standard data were therefore not evaluated.

Following the recommendation of our security analysts, recall was deemed more important than precision. Indeed, in this domain, it is preferable to alert clients too often with false alarms than to miss potential cyber threats. To account for this, we used lenient definitions of precision (P^L) and recall (R^L) where *acceptable* mappings are considered *correct*. In addition, two values of β were used to compute the f-measure: $\beta = 1$, and $\beta = 0.5$, where recall is more important than precision. This gave rise to two f-measures: F_1^L and $F_{0.5}^L$.

When evaluated against the gold-standard dataset, the original system[3] achieved a lenient recall R^L of 35.22%, a lenient precision of P^L of 97.96%, an $F_{0.5}^L$ of 72.23% and an F_1^L of 51.82%. Although precision was high, recall was particularly low.

[3] with the parameters $FV = 0.98$ and $t = 0$ (see Sect. 3.1).

4 Enhancing the Original System

After analyzing the result of the original system, we noticed that many snort messages share similar content, hence it would seem natural that they be mapped to similar CAPEC fields. To ensure this, we experimented with clustering the snort messages prior to mapping them. Each snort message within a cluster is then mapped to the same CAPEC field. Specifically, snort messages are first expanded (see Sect. 3.1), then clustered into n clusters using k-means clustering. All messages in the same cluster are then concatenated into a single long message, and the resulting longer message is then mapped using the same approach as in the original system.

We experimented with various numbers of clusters (n) which as a side-effect also varied the length of the resulting message to map. The trade-off is that a larger number of clusters (n) should lead to a greater number of possible CAPEC fields being mapped to each snort message, but should also lead to a shorter message and sparser representation.

4.1 Results and Analysis

Table 4.1 shows the results of the system with and without clustering as part of the pre-processing for various values of n. As shown in Table 4.1, the best configurations in terms of f-measure are when using clustering with values of n between 500 and 3000. With these values, the results are not statistically different, with $F^L_{0.5} \approx 80\%$ and $F^L_1 \approx 64\%$. Recall itself has reached $\approx 48\%$ from a low 35.22% in the original system. Recall from Sect. 3 that the average length of CAPEC fields is 214 words. Hence using $n = 2000$ allows us to bring the average size of snort messages (235 words) at par with the size of CAPEC fields.

Table 4.1 also shows the trade-off between the use of a smaller number of clusters (smaller n) which leads to a smaller number of possible output CAPEC fields and the use of a larger n which leads to a sparser snort representation. Hence leading to lower f-measures with $n \geq 2000$ and $n \leq 100$.

5 Conclusion and Future Work

In this paper, we described an enhancement to the approach proposed by [4] to maps security events to related attack fields in CAPEC. We have shown that by expanding domain terms and clustering snort messages prior to mapping them, the recall of the approach can be increased significantly without much loss in precision.

As future work, we plan to investigate the use of automatic snort expansion, by using existing knowledge bases such as the Common Weakness Enumeration [17] rather than relying on hand-written term expansion. In addition, it

Table 2. Results of the system with different cluster numbers

System	n	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	Snort Length
Original	n/a	97.96%	35.22%	72.23%	51.82%	15
Clustering	5000	85.66%	47.93%	74.01%	61.47%	94
Clustering	4000	85.83%	47.70%	73.99%	61.32%	117
Clustering	**3000**	95.86%	48.46%	80.18%	64.38%	157
Clustering	**2000**	95.40%	48.18%	79.77%	64.03%	235
Clustering	**1000**	95.71%	48.18%	79.94%	64.09%	470
Clustering	**500**	95.39%	48.80%	80.10%	64.57%	941
Clustering	100	93.80%	36.69%	71.53%	52.75%	4,707
Clustering	50	94.13%	36.23%	71.34%	52.33%	9,415
Clustering	20	94.17%	36.11%	71.25%	52.20%	23,539

would be interesting to look beyond the mapping of individual snort messages, but try to identify and match entire patterns/groups of snort messages as an indication of possible cyber attacks.

Acknowledgement

The authors would like to thank the anonymous reviewers for their feedback on the paper. This work was financially supported by an Engage Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Daniel Schatz, Rabih Bashroush, and Julie Wall. Towards a more representative definition of cyber security. *Journal of Digital Forensics, Security and Law*, 12(2):8, 2017.
2. Asmaa Shaker Ashoor and Sharad Gore. Importance of intrusion detection system (IDS). *International Journal of Scientific and Engineering Research*, 2(1):1–4, 2011.
3. Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of LISA'99: The 13th Conference on System Administration*, pages 229–238, Seattle, Washington, USA, November 1999.
4. Nicandro Scarabeo, Benjamin CM Fung, and Rashid H Khokhar. Mining known attack patterns from security-related events. *PeerJ Computer Science*, 1:e25, 2015.
5. Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.
6. Sumit More, Mary Matthews, Anupam Joshi, and Tim Finin. A knowledge-based approach to intrusion detection modeling. In *Proceedings of the IEEE Symposium on Security and Privacy Workshop (SPW)*, pages 75–81, San Francisco, California, USA, May 2012. IEEE.
7. Varish Mulwad, Wenjia Li, Anupam Joshi, Tim Finin, and Krishnamurthy Viswanathan. Extracting information about security vulnerabilities from web text.

In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 3, pages 257–260, Lyon, France, August 2011. IEEE.

8. Mikhail J Atallah, Craig J McDonough, Victor Raskin, and Sergei Nirenburg. Natural language processing for information assurance and security: An overview and implementations. In *Proceedings of the 2001 Workshop on New Security Paradigms*, pages 51–65, Ballycotton, County Cork, Ireland, September 2001.
9. Victor Raskin, Christian F Hempelmann, Katrina E Triezenberg, and Sergei Nirenburg. Ontology in information security: A useful theoretical foundation and methodological tool. In *Proceedings of the 2001 Workshop on New Security Paradigms*, pages 53–59, Cloudcroft, New Mexico, 2001. ACM.
10. Jeffrey Undercoffer, Anupam Joshi, Tim Finin, and John Pinkston. Using DAML+ OIL to classify intrusive behaviours. *The Knowledge Engineering Review*, 18(3):221–241, 2003.
11. Jeffrey Undercoffer, Anupam Joshi, and John Pinkston. Modeling computer attacks: An ontology for intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 113–135. Springer, 2003.
12. Jeffrey Undercoffer, John Pinkston, Anupam Joshi, and Timothy Finin. In *Proceedings of the IJCAI Workshop on Ontologies and Distributed Systems*, pages 47–58, Acapulco, Mexico, August 2004.
13. National Cyber Security Division. National Vulnerability Database (NVD). `https://nvd.nist.gov`, 2017.
14. Tim Finin and Zareen Syed. Creating and exploiting a web of semantic data. In Joaquim Filipe, Ana Fred, and Bernadette Sharp, editors, *Agents and Artificial Intelligence*, pages 3–21, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
15. David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.
16. UMBC Ebiquity. Index of /ontologies/cybersecurity/ids. `http://ebiquity.umbc.edu/ontologies/cybersecurity/ids/`, 2014.
17. MITRE. Common Weakness Enumeration (CWE). `https://cwe.mitre.org/index.html`, 2017.
18. MITRE. Common Attack Pattern Enumeration and Classification (CAPEC). `https://capec.mitre.org/`, 2017.