Evaluation of Applying LDA to Redacted Documents in Security and Safety Analysis

Katsuyuki Umezawa
Dept. of Information Science
Shonan Institute of Technology
Kanagawa, Japan
umezawa@info.shonan-it.ac.jp

Sven Wohlgemuth
Intelligent Systems Laboratory
SECOM Co., Ltd.
Tokyo, Japan
s-wohlgemuth@secom.co.jp

Keisuke Hasegawa
Intelligent Systems Laboratory
SECOM Co., Ltd.
Tokyo, Japan
keisu-hasegawa@secom.co.jp

Kazuo Takaragi
Chief Executive Officer
HISAFE
Kanagawa, Japan
kazuo.takaragi.wg@gmail.com

***Abstract*—Cyber attacks are often executed by imitating existing attacks and combining them. Using existing vulnerability databases, we have presented a way to semi-automatically determine the presence of vulnerabilities in the design documents of products under development. We have calculated the similarity between documents using the Latent Dirichlet Allocation (LDA) technology and compared the design document of the new product with the vulnerability database. When this comparison processing is conducted by a third party as a service, it may be desirable to not inadvertently disclose a part of the design document of the new product to the third party. In this study, we used the LDA technique to experimentally verify that the calculated similarity value does not deteriorate even when a portion of the design document is encrypted or obfuscated. In conclusion, we discovered no substantial difference in similarity with the original document; however, there are changes in numerical values depending on the words to be encrypted/obfuscated. In particular, the degradation of similarity is very small when the version number is encrypted/obfuscated.**

***Index Terms*—vulnerability analysis, natural language processing, latent dirichlet allocation, cosine similarity**

I. Introduction

As information systems are increasing in complexity, there are more instances where cyber attacks are carried out by exploiting numerous vulnerabilities instead of targeting a single vulnerability alone. Furthermore, NIST SP800-53 Rev.5 [1] states "it is important to incorporate new and up-to-date controls based on empirical attack data." In the wake of a rising number of cyber attacks, this has served to highlight the value of "empirical" knowledge. We have proposed a threat analysis method that can utilize attack cases discovered in existing systems in actual operation [2]. In particular, our new method evaluates the risk of complex systems by combining a fault tree analysis, which traditionally used for reliability analysis; attack tree analysis, which has been introduced to analyze malicious attack patterns, and vulnerability database.

Figure 1 depicts an overview of our research. When vulnerable information is discovered, it is published in reports, research papers, etc., and is stored in databases so that it can be referenced (see (1) in Figure 1). Moreover, new attacks are often implemented by imitating or combining existing attacks (see (2) in Figure 1). However, when creating a new product, developers want to be made aware of vulnerabilities in the design document. (see (3) in Figure 1). Thus, we used LDA technology to compare the design document to the vulnerability database semi-automatically determine whether or not the new design document contains existing vulnerabilities (Fig. (4) of 1).

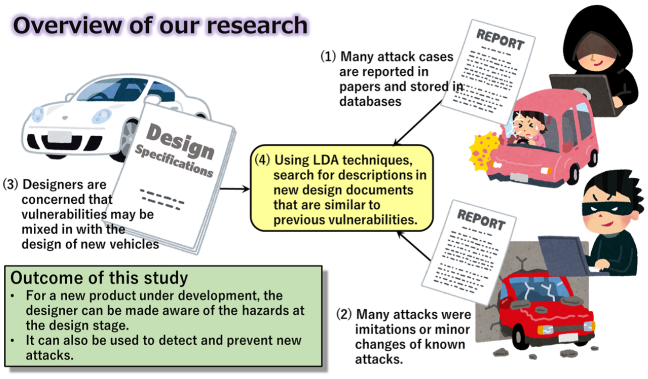

Fig. 1. Overview of Our Research

This study is related to the application that the above-mentioned design document and vulnerability database matching processing is conducted as a service by a third-party organization. When a third-party organization provides a judgment service, it can be assumed that there are words and numbers that we do not wish to disclose in the design documents that contain trade secrets. In other words, it has been experimentally verified that even if some of the design document is encrypted or obfuscated, the similarity may still be calculated in the same manner as it would be in the original document.

This study is based briefly on the presentation at SCIS2023 [3] and then expands on the discussion about LDA.

II. Background

Interference with safety owing to security threats is recognized as a major issue plaguing safety-oriented systems. In the EVITA project [4], a risk analysis was conducted using attack trees to ensure the security of in-vehicle communication. One approach to analyzing the causal relation between safety and security is to express the relation as a combination of a fault tree and an attack tree [5]. Security analysts typically describe

the nodes in the attack and fault tree structures using natural language. As a result, utilizing natural language processing can effectively expedite security analysis.

MITRE, a US-based company, provides several forms of vulnerability databases. Common vulnerability and exposure (CVE) [6] is a database comprising individual software vulnerabilities. The common weakness enumeration (CWE) [7] catalogs common vulnerabilities by focusing on their underlying causes. Additionally, common attack pattern enumeration and classification (CAPEC) [8] is a database that classifies attacks according to their patterns.

CVE reports $> 10,000$ vulnerabilities annually, and sophisticated attacks occur using multiple combinations based on these vulnerabilities. Hence, creating an attack tree that exhaustively covers all possible attack scenarios is difficult. Focusing on such problems, we have proposed a threat analysis method using a vulnerability database [9] [10]. This study extends previous studies by investigating the threat analysis using partially redacted documents to prevent disclosure of sensitive information.

Researchers have reported various methods to ensure the authenticity of sanitized electronic documents for a long time [11] [12], and one proposed solution involves specifying who sanitized the original document [13]. Furthermore, researchers have proposed methods to search encrypted documents by checking search tags [14] and accelerating the retrieval process [15]. Conventional retrieval technologies for redacted documents enable the retrieval of sensitive information from encrypted or obfuscated documents by assigning retrieval IDs and tags.

Herein, we investigate a novel technology, which can be used to search for documents similar to confounded documents by matching them with over 100,000 vulnerability database documents. Particularly, we propose a method for estimating a model using latent Dirichlet allocation (LDA) for revealing similarities between documents using cosine similarity.

III. LDA Implementation

In this study, we implemented LDA using Gensim, a Python library (Figure 2) [16].

Gensim is licensed under the LGPL and is an open-source library for unsupervised topic modeling and natural language. This study used the source code published in the previous research [17] after adding a data reading unit and processing that excludes particles and adverbs of documents called stopwords. The details of the algorithm of the LDA generation process are as follows:

(1–1) Read the description text of the base document group as array data. For every word, use the Morphy method of the wordnet class of nltk.corpus, which is a Python library, to restore the word to its original form. Furthermore, specify English in words method of the stopwords class of nltk.corpus and remove the stop words from the read sentences. Remove words that appear less than an arbitrary number of times (twice or less in this study) from the remaining text data.

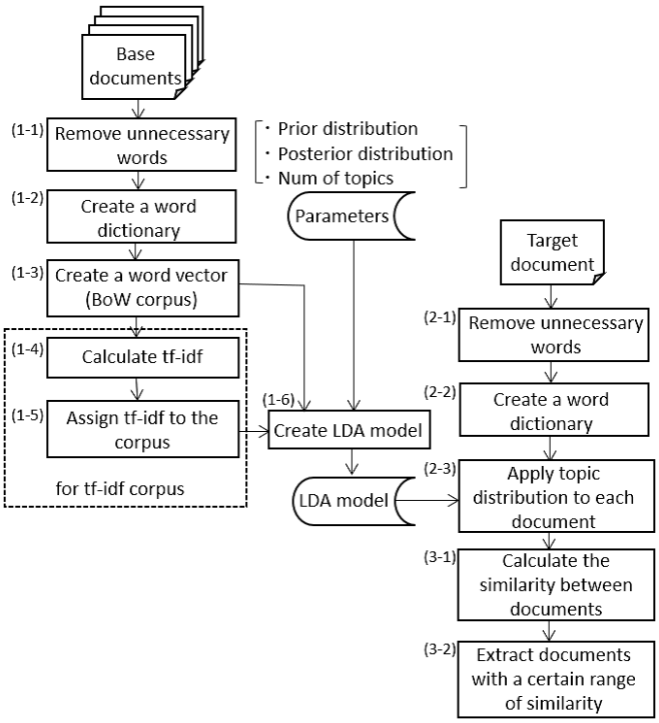

Fig. 2. Overall processing with the proposed tool

(1–2) Create a word dictionary from the documents that have completed the above processing using the Dictionary method of the corpora class.

(1–3) Each document is converted into a vector of words from the created dictionary and document group by the doc2bow method of dictionary class (this process is called creating BoW corpus). If we want to perform weighting using tf-idf, perform the following (1–4) and (1–5).

(1–4) Create TfidfModel by TfidfModel method in Gensim's models class.

(1–5) The TF-IDF corpus is obtained by passing the BoW corpus created in (1–3) to the TfidfModel.

(1–6) Create an LDA model by specifying the dictionary and corpus (BoW corpus or tf-idf corpus) created in the LdaModel method of Gensim's ldamodel class and parameters.

After creating the LDA model, assign a topic distribution to the evaluation target. The flow is as follows:

(2–1) Perform the same processing as (1–1) to read the comparison document.

(2–2) Perform the same processing as (1–2), and set the evaluation target as a word vector.

(2–3) A vectorized document is given as an argument to the get_document_topics method of the LdaModel class, and a probability distribution is assigned.

At the end of the process in Figure 2, the similarity is calculated.

(3–1) The similarity of the topic distribution vectors $\vec{x}, \vec{y}$ of each document obtained by the flow from the base document and the flow from the evaluation target is the cosine similarity $cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|}$.

(3–2) Finally, the documents whose similarity exceeds the preset threshold value are extracted.

IV. Experimental method

A. Documents to be inspected

The two design documents to be examined were the section on LOCAL PRIVILEGE ESCALATION (referred to as DD1) and the section on BROWSER HACKING (referred to as DD2) in document [18]. Portions of each document are depicted in Figures 3 and 4.

LOCAL PRIVILEGE ESCALATION

Though we got a remote shell based on our browser hacking, it's also impossible to get arbitrary permission because of AppArmor. We need another vulnerability to escape from AppArmor and get a higher privilege than browser's process context. It seems that the Linux kernel version of CID is very old, there is nearly no exploiting mitigations on Linux kernel 2.6.36.

Fig. 3. Part of DD1

BROWSER HACKING

Since the User Agent of Tesla web browser is "Mozilla/ 5.0 (X11; Linux) AppleWebKit/ 534.34 (KHTML, like Gecko) QtCarBrowser Safari/ 534.34 ", it can be deduced that the version of QtWebkit is around 2.2.x. In such old version, there are many vulnerabilities in QtWebkit. Our exploit utilizes two vulnerabilities to achieve arbitrary code execution. The first vulnerability exists in function JSArray::sort(). This function will be called when the method function sort() of an array be called in JavaScript code. The function JASrray::sort() mainly do three things:

Fig. 4. Part of DD2

Here, the words in dark gray are removed because they are proper nouns. Light gray words are obfuscated because they appear frequently. Furthermore, the numbers enclosed in frames are to be obfuscated.

B. About obfuscation

This section describes the obfuscation conducted in this experiment. The purpose of this experiment is to confirm the LDA matching precision for the obfuscated or encrypted documents. Thus, the security strength related to this obfuscation is outside the scope of this paper. Regarding the security strength, we believe that any encryption or obfuscation processing should be used.

Lists of obfuscated words for DD1 and DD2 are depicted in Tables I and II.

TABLE I
OBFUSCATED WORDS FROM DD1

Target word	Occurrences	Random number r
kernel	4	1
privilege	3	4
arbitrary	2	6
context	2	10
exploit	2	4
vulnerability	2	9

TABLE II
OBFUSCATED WORDS FROM DD2

Target word	Occurrences	Random number r
structure	20	15
function	12	5
address	9	12
element	8	4
storage	8	1
pointer	6	8
type	6	10
array	5	12
memory	5	16
tag	5	8
vulnerability	5	7

TABLE III
OBFUSCATED VERSION NUMBER FROM DD1

Target version number	Occurrences	Random number r
(Linux kernel) 2.6.36	1	3

TABLE IV
OBFUSCATED VERSION NUMBER FROM DD2

Target version number	Occurrences	Random number r
(Mozilla) 5.0	1	4
(AppleWebKit) 534.34	1	12
(Safari) 534.34	1	16
(QtWebkit) 2.2.x	1	2

Here we take "vulnerability" in Table I as an example. The random number is $r = 9$. A table obtained by randomly rearranging the publicly available word table [19] (which shall also be made public) is called a random word list. Suppose that "vulnerability" appears in the kth random word list. At that time, the target word is replaced with 16 words that are

$k-(r-1) \leq l \leq k+(16-r)$. Specifically, "vulnerability" is replaced by the following 16 words: "achievement, ninny, atrocity, drawback, feet, partial, mutuality, optimizes, **vulnerability**, vest, bosoms, sumptuously, goethe, asylums, algebraically, unpredictable." In addition, refer to the list of version numbers for each product and change the version numbers to the 16 numbers before and after the intended version number. A list of obfuscated words and version numbers are illustrated in Tables IX and X.

C. About LDA parameters

Several parameters need to be calculated for model generation in LDA. First, for the corpus, the Bag of Words (BoW) corpus (Step (1-3) in Figure 2) and the term frequency - inverse document frequency (tf-idf) corpus (Step (1-4)(1-5)) can be selected. We have selected the if-idf corpus. Furthermore, we have truncated words with a frequency of two or less in the document. Also, it is important to determine the number of topics. Proposals have been made to automate the process of counting topics and to count topics using the metrics of coherence and perplexity, which measure the quality of subjects and predictability respectively. [20] [21]. Nevertheless, as a preliminary experiment, we changed the number of topics from 1 to 20 and adopted the number of topics with the highest accuracy. The number of topics was 13 in the analysis of DD1, and 11 in the analysis of DD2.

D. Experimental results (cosine similarity)

We employed the LDA method to match obfuscated DD1 (145 words) and DD2 (728 words) with CVE [6] (119,479 documents) to determine the cosine similarity. For comparison, we also estimated the similarity with the CVE for the original DD1 and DD2 before obfuscation. Tables V and VII show the results of obfuscating words. Tables VI and VIII reveal the obfuscated version numbers. The number of filtered items is the number of CVEs with a higher similarity than the CVE targeted by each design document (CVE-2013-6282 in DD1, CVE-2011-3928 in DD2). In other words, the number of filtered items represents how many of the 119,479 documents were narrowed down by the LDA technology. The specifications of the PC utilized in the experiment are OS: Windows 10, CPU: Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz, Memory: 24GB.

V. Consideration

As illustrated in Tables V and VII, if 6 or 11 words are obfuscated, the degree of similarity does not vary substantially, but the accuracy of filtering becomes worse. However, when each word is obfuscated individually, there is not a noticeable difference in the number of filtered items.

Tables VI and VIII confirm that filtering accuracy does not deteriorate even when multiple version numbers are obfuscated at the same time. Furthermore, the execution time is around 25 seconds, which is a practical amount of time.

From Tables V and VIII, some obfuscated documents have less filtering (better accuracy) than the original documents. Here, we concentrate on the number of filtering in Table VI and examine it in more detail. Figure 5 reveals the breakdown of the number of filtering.

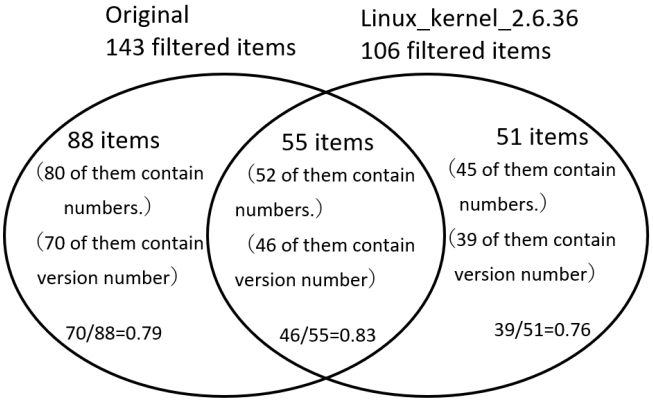

Fig. 5. Breakdown of number of filtered items

We hypothesized that if the version numbers were included in the CVEs, they would have an impact on filtering accuracy. However, from Figure 5, the percentage of CVEs containing version numbers among the filtered CVEs did not change much. It is unclear why the number of narrowed-down cases was reduced from 88 to 51. It has been observed that the number filtered varies greatly depending on the words being obfuscated. The fact that the number to be filtered has decreased this time (performance has increased as a result of obfuscation) appears to be within the margin of error.

VI. Conclusion and Future Work

In this study, we proposed a case where threat analysis using LDA is conducted by a third party. To prevent the disclosure of specifications containing trade secrets before development to third parties, we experimentally demonstrated that even if we employ obfuscated specifications, there is no significant difference in the similarity of matching results with the existing vulnerability case database (CVE).

In future studies, we plan to continue to conduct verification using real design documents (specifications), automatically determine the number of topics in LDA analysis, employ techniques other than LDA, use vulnerability databases other than CVE, and apply them to further cases.

References

[1] SP 800-53 Rev.5, "Security and Privacy Controls for Information Systems and Organizations" Sep. 2020. https://doi.org/10.6028/NIST.SP.800-53r5

[2] K. Umezawa, H. Koyanagi, S. Wohlgemuth, Y. Mishina, and K. Takaragi, "Safety and Security Analysis using LDA based on Case Reports: Case Study and Trust Evaluation Method," Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES 2022), Article No.: 154, pp. 1-7, Aug. 2022.

[3] K. Takaragi, T. Kubota, S. Wohlgemuth, K. Umezawa, K. Hasegawa, and Y. Mishina, "Privacy–enhanced AI natural language processing for KYC –Applying LDA to redacted documents with range proofs by DAC/ZKRP–, " Proceedings of the Symposium on Cryptography and Information Security (SCIS 2023), Jan, 2023..

TABLE V
SIMILARITY BETWEEN CVE AND DD1 WITH OBFUSCATED WORDS

CVE	original DD1	obfuscated kernel	obfuscated privilege	obfuscated arbitrary	obfuscated context	obfuscated exploit	obfuscated vulnerability	obfuscated all 6 words
CVE-1999-0001	0.9680	0.9497	0.9677	0.9556	0.9697	0.9716	0.9481	0.9415
CVE-1999-0002	0.8291	0.9096	0.7982	0.8252	0.8300	0.8325	0.8182	0.9104
CVE-1999-0003	0.8557	0.9335	0.8426	0.8241	0.8565	0.8781	0.8023	0.9011
Omitted (originally, there are 119479 lines in total)								
CVE-2019-9976	0.9725	0.9628	0.9734	0.9571	0.9742	0.9805	0.9459	0.9535
CVE-2019-9977	0.9362	0.9447	0.9372	0.9131	0.9378	0.9503	0.8973	0.9262
CVE-2019-9978	0.9466	0.9333	0.9464	0.9342	0.9486	0.9517	0.9254	0.9395
Diff. from the original (max)	—	0.1800	0.0482	0.0603	0.0248	0.0472	0.0988	0.1701
Diff. from the original (ave.)	—	0.0260	-0.0021	-0.0242	0.0009	0.0158	-0.0408	0.0105
Execution time (s)	24.710	25.074	26.154	24.861	24.557	24.590	24.946	24.449
Number of filtered items	**143**	**12782**	**66**	**692**	**73**	**155**	**2922**	**6894**

TABLE VI
SIMILARITY BETWEEN CVE AND DD1 WITH OBFUSCATED VERSION NUMBERS

CVE	original DD1	obfuscated Linux kernel 2.6.36
CVE-1999-0001	0.9680	0.9690
CVE-1999-0002	0.8291	0.8003
CVE-1999-0003	0.8557	0.8539
Omitted (originally, there are 119479 lines in total)		
CVE-2019-9976	0.9725	0.9771
CVE-2019-9977	0.9362	0.9443
CVE-2019-9978	0.9466	0.9545
Diff. from the original (max)	—	0.0551
Diff. from the original (ave.)	—	0.0090
Execution time (s)	24.710	24.360
Number of filtered items	**143**	**106**

[4] A. Ruddle et al., "Deliverable D2.3: Security requirements for automotive on-board networks based on dark-side scenarios," Seventh Research Framework Programme of the European Community, July 2008.

[5] I. N. Fovino et al., "Integrating cyber attacks within fault trees," Reliability Engineering and System Safety 94 (2009) p.p.1394–1402.

[6] MITRE Corporation, "CVE-Common Vulnerability and Exposure," https://cve.mitre.org/, (Last accessed April 11, 2023).

[7] MITRE Corporation, "CWE List - Common Weakness Enumeration," https://cwe.mitre.org/data/, (Last accessed April 11, 2023).

[8] MITRE Corporation, "CAPEC - Common Attack Pattern Enumeration and Classification," https://capec.mitre.org/, (Last accessed April 11, 2023).

[9] K. Umezawa, Y. Mishina, K. Taguchi and K. Takaragi, "A Prop osal of Threat Analyses using Vulnerability Databases," 2018 Symposium on Cryptography and Information Security (SCIS), 1C2-6, Jan. 2018.

[10] Y. Mishina, K. Takaragi and K. Umezawa "A Proposal of Threat Analyses for Cyber-Physical System using Vulnerability Databases," 2018 IEEE International Symposium on Technologies for Homeland Security (IEEE HST), 2018.

[11] K. Miyazaki, S. Susaki, M. Iwamura, T. Matsumoto, R. Sasaki and H. Yoshiura, "Digital Document Sanitizing Problem," Information Processing Society of Japan (IPSJ) Computer Security Group (CSEC) technical reports, pp. 61-67, 2003.

[12] Y. Hatano, K. Miyazaki and S. Tezuka, "An Information Disclosure System of Digital Documents with Protecting Personal Information," Journal of Information Processing Society of Japan (IPSJ), Vol. 47, No.3, pp. 667–675, 2006.

[13] T. Izu, N. Kanaya, M. Takenaka and T. Yoshioka, "A Sanitizable Signature Scheme with Sanitizer Identification," Journal of Information Processing Society of Japan (IPSJ), Vol.48, No.9, pp. 2990–2998, 2007.

[14] D. Boneh and B. Waters, "Conjunctive, Subset, and Range Queries on Encrypted Data." In: Vadhan, S.P. (eds) Theory of Cryptography. TCC 2007. Lecture Notes in Computer Science, vol 4392. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-70936-7_29

[15] N. Matsuda, T. Ito, Hi. Shibata, M. Hattori and T. Hirano, "Efficient Searchable Encription and Its Application to Web Services," Proceedings of the Multimedia, Distributed, Cooperative, and Mobile Symposium (DICOMO), pp. 2067–2074, 2013.

[16] H. Koyanagi, Y. Mishina, K. Takaragi, S. Wohlgemuth, and K. Umezawa, "Research on attack cases via topic-model analysis and selection of vulnerability candidates from large-scale vulnerability database," International Workshop on Security (IWSEC) Poster Session, Fukui, Japan, Sep. 2020.

[17] Spooky_Maskman, "Topic analysis by LDA with Gensim," https://qiita.com/Spooky_Maskman/items/0d03ea499b88abf56819. (Last accessed Feb. 28, 2023).

[18] S. Nie, L. Liu, and Y. Du, "FREE-FALL: HACKING TESLA FROM WIRELESS TO CAN BUS," Briefing, Black Hat USA 2017, July 2017.

[19] http://www.mieliestronk.com/corncob_lowercase.txt (Last accessed Feb. 28, 2023).

[20] J. Chang, S. Gerrish, C. Wang, J. L. Boyd-Graber, and D. M Blei, "Reading tea leaves: How humans interpret topicmodels," Advances in NIPS, p.p. 288–296, 2009.

[21] D. Newman, J. H. Lau, K. Grieser, and T. Baldwin, "Automatic Evaluation of Topic Coherence," p.p. 100–108, 2010.

TABLE VII
SIMILARITY BETWEEN CVE AND DD2 WITH OBFUSCATED WORDS

CVE	original DD2	obfuscated structure	obfuscated function	obfuscated address	obfuscated element	obfuscated storage	obfuscated pointer
CVE-1999-0001	0.9343	0.9342	0.9344	0.9159	0.9448	0.9397	0.9430
CVE-1999-0002	0.9427	0.9426	0.9431	0.9222	0.9548	0.9482	0.9528
CVE-1999-0003	0.9654	0.9654	0.9657	0.9602	0.9662	0.9706	0.9722
Omitted (originally, there are 119479 lines in total)							
CVE-2019-9976	0.8019	0.8020	0.8010	0.7982	0.8016	0.8065	0.8015
CVE-2019-9977	0.8990	0.8990	0.8988	0.8838	0.9073	0.9043	0.9057
CVE-2019-9978	0.9695	0.9695	0.9696	0.9732	0.9626	0.9751	0.9748
Diff. from the original (max)	—	0.0001	0.0016	0.0656	0.0530	0.0324	0.0309
Diff. from the original (ave.)	—	0.0000	-0.0003	0.0009	-0.0036	0.0040	0.0007
Execution time (s)	26.104	26.191	26.147	25.844	26.211	25.618	25.968
Number of filtered items	**1775**	**1768**	**1778**	**422**	**8086**	**1191**	**1724**

CVE	original DD2	obfuscated type	obfuscated array	obfuscated memory	obfuscated tag	obfuscated vulnerability	obfuscated all 11 words
CVE-1999-0001	0.9343	0.9498	0.9379	0.9343	0.9467	0.9406	0.9744
CVE-1999-0002	0.9427	0.9502	0.9463	0.9427	0.9489	0.9500	0.9705
CVE-1999-0003	0.9654	0.9732	0.9689	0.9654	0.9718	0.9691	0.9889
Omitted (originally, there are 119479 lines in total)							
CVE-2019-9976	0.8019	0.8124	0.8049	0.8019	0.8105	0.7976	0.8169
CVE-2019-9977	0.8990	0.9187	0.9025	0.8990	0.9146	0.9032	0.9431
CVE-2019-9978	0.9695	0.9729	0.9732	0.9695	0.9724	0.9703	0.9820
Diff. from the original (max)	—	0.0449	0.0199	0.0000	0.0352	0.0208	0.1308
Diff. from the original (ave.)	—	-0.0025	0.0027	0.0000	-0.0017	-0.0030	-0.0068
Execution time (s)	26.104	25.854	25.883	25.500	25.680	25.843	26.188
Number of filtered items	**1775**	**3298**	**1389**	**1775**	**2789**	**2703**	**9781**

TABLE VIII
SIMILARITY BETWEEN CVE AND DD2 WITH OBFUSCATED VERSION NUMBERS

CVE	original DD2	obfuscated Moziila 5.0	obfuscated AppleWebKit 534.34	obfuscated QtCarBrowser Safari 534.34	obfuscated QtWebkit 2.2.x	obfuscated all 4 numbers
CVE-1999-0001	0.9343	0.9319	0.9343	0.9343	0.9334	0.9310
CVE-1999-0002	0.9427	0.9373	0.9427	0.9427	0.9416	0.9362
CVE-1999-0003	0.9654	0.9635	0.9654	0.9654	0.9649	0.9629
Omitted (originally, there are 119479 lines in total)						
CVE-2019-9976	0.8019	0.8113	0.8019	0.8019	0.8026	0.8120
CVE-2019-9977	0.8990	0.9002	0.8990	0.8990	0.8986	0.8997
CVE-2019-9978	0.9695	0.9678	0.9695	0.9695	0.9693	0.9675
Diff. from the original (max)	—	0.0317	0.0000	0.0000	0.0025	0.0332
Diff. from the original (ave.)	—	0.0051	-0.0000	-0.0000	0.0003	0.0054
Execution time (s)	26.104	26.529	26.066	26.297	26.002	26.458
Number of filtered items	**1775**	**1884**	**1775**	**1775**	**1662**	**1834**

TABLE IX
OBFUSCATED WORDS AND VERSION NUMBERS FROM DD1

Target	Occurrences	Random number r	Replacement
kernel	4	1	**kernel**, laity, amnesiac, wast, expiration, posters, phooey, elderly, outpourings, sunscreens, cross, ripening, totting, unfreezing, trunking, overhears
privilege	3	4	recommencement, scorched, axons, **privilege**, opportune, unannounced, fairness, axillary, checkup, overstepping, upcast, cohered, paraffin, televisual, cuneiform, corkscrew
arbitrary	2	6	perspectives, drover, execute, fingerprinted, deviating, **arbitrary**, populate, lathered, hoofs, humiliate, honorary, merrymaking, congruences, dan, owns, sphincters
context	2	10	example, textually, bivalves, canvas, zeppelin, replant, thunderbolt, bemusedly, explosions, **context**, discounted, boudoirs, conflictingly, disinclination, seasonally, convenience
exploit	2	4	treasurer, bolstering, truthfully, **exploit**, meeting, lightning, grooving, battlegrounds, discriminated, condemnable, amalgamates, genomic, auditions, communique, uniformly, retrench
vulnerability	2	9	accomplishment, ninny, atrocity, drawback, feet, partial, mutuality, optimizes, **vulnerability**, vest, bosoms, sumptuously, goethe, asylums, algebraically, unpredictable
(Linux kernel) 2.6.36	1	3	2.6.34, 2.6.35, **2.6.36**, 2.6.37, 2.6.38, 2.6.39, 2.6.9, 3.0, 3.1, 3.10, 3.11, 3.12, 3.13, 3.14, 3.15, 3.16

TABLE X
OBFUSCATED WORDS AND VERSION NUMBERS FROM DD2

Target	Occurrences	Random number r	Replacement
structure	20	15	hatless, redeployed, gushing, languishing, hacks, cheaply, handyman, gab, withstand, dynamics, crayons, loaned, catastrophic, common, **structure**, classicist
function	12	5	untangle, intimidated, selectivity, communally, **function**, mishandles, fiesta, hyperbole, downsized, birthdays, undisturbed, practically, macaw, basketry, grappling, halfheartedly
address	9	12	quotidian, borehole, dot, bobtail, limped, fez, compensates, growls, replanting, redisplayed, ruled, **address**, shifter, rightward, unattainably, hinges
element	8	4	loves, choker, philosophizing, **element**, horrifically, groper, evaporates, pioneer, roadblocks, harridan, swooned, witchdoctors, grimaced, affliction, tablespoonfuls, thorium
storage	8	1	**storage**, pup, undulated, cuddly, headnote, pose, recursive, espresso, frank, barkers, harshly, democratic, malfunctioning, freshener, colonise, numerology
pointer	6	8	fatally, predisposes, illustrators, sows, trained, tailpiece, bankrupted, **pointer**, peck, cooperative, lunchtimes, macroscopic, spectators, consist, lower, transducers
type	6	10	treasuries, tights, toothmarks, charters, jesuit, entity, lofts, videoconferencing, intensities, **type**, enhances, trendiness, relativistic, dreadlocks, stratifies, defibrillator
array	5	12	rumblings, ichneumon, revenging, calorimeter, crests, sepulchres, lodges, blockage, pains, undid, ibis, **array**, layering, dormitory, slurped, therms
memory	5	16	cobs, bejewelled, pelting, featuring, egomaniac, saris, collectivity, brasses, fauns, reorganisations, teeming, tenths, spline, snorer, weep, **memory**
tag	5	8	protactinium, dart, answerer, safeness, snowier, betters, hosier, **tag**, annulment, others, groupings, estuary, bricklayer, lithography, funerary, divide
vulnerability	5	7	aatrocity, drawback, feet, partial, mutuality, optimizes, v**ulnerability,** vest, bosoms, sumptuously, goethe, asylums, algebraically, unpredictable, mackintosh, impalas
(Mozilla) 5.0	1	4	4.0.1, Aurora 5.0a2, 5.0 Beta 2, **5.0**, 5.0.1, Nightly 6.0a1, Aurora 6.0a2, 6.0 Beta 1, 6.0, 6.0.1, 6.0.2, Nightly 7.0a1, Aurora 7.0a2, 7.0 Beta 1, 7.0, 7.0.1
(AppleWebKit) 534.34	1	12	533.17.9, 533.18.1, 533.19.4, 533.2+, 533.20.25, 533.21.1, 533.4+, 533+, 534.1+, 534.15+, 534.16+, **534.34**, 534.55.3, 534.57.2, 537.13+, 537.75.14
(Safari) 534.34	1	16	532.0+, 532.3+, 532+, 533.17.8, 533.17.9, 533.18.1, 533.19.4, 533.2+, 533.20.25, 533.21.1, 533.4+, 533+, 534.1+, 534.15+, 534.16+, **534.34**
(QtWebkit) 2.2.x	1	2	2.1.1, **2.2**, 2.3, 2.3.4, 2.3.4.dfsg, 2.3.4.dfsg.1, 3.0, 4.8.6, 5.0, 5.1.1, 5.2.1, 5.212.0, 5.212.0.1+dde, 5.212.0.2 alpha4+dde, 5.212.0 alpha2, 5.212.0 alpha3