Fast Detection of Transformed Data Leaks

Xiaokui Shu, Jing Zhang, Danfeng (Daphne) Yao, *Senior Member, IEEE*, and Wu-Chun Feng, *Senior Member, IEEE*

***Abstract*— The leak of sensitive data on computer systems poses a serious threat to organizational security. Statistics show that the lack of proper encryption on files and communications due to human errors is one of the leading causes of data loss. Organizations need tools to identify the exposure of sensitive data by screening the content in storage and transmission, i.e., to detect sensitive information being stored or transmitted in the clear. However, detecting the exposure of sensitive information is challenging due to data transformation in the content. Transformations (such as insertion and deletion) result in highly unpredictable leak patterns. In this paper, we utilize sequence alignment techniques for detecting complex data-leak patterns. Our algorithm is designed for detecting long and inexact sensitive data patterns. This detection is paired with a comparable sampling algorithm, which allows one to compare the similarity of two separately sampled sequences. Our system achieves good detection accuracy in recognizing transformed leaks. We implement a parallelized version of our algorithms in graphics processing unit that achieves high analysis throughput. We demonstrate the high multithreading scalability of our data leak detection method required by a sizable organization.**

***Index Terms*— Data leak detection, content inspection, sampling, alignment, dynamic programming, parallelism.**

I. Introduction

REPORTS show that the number of leaked sensitive data records has grown 10 times in the last 4 years, and it reached a record high of 1.1 billion in 2014 [3]. A significant portion of the data leak incidents are due to human errors, for example, a lost or stolen laptop containing unencrypted sensitive files, or transmitting sensitive data without using end-to-end encryption such as PGP. A recent Kaspersky Lab survey shows that accidental leak by staff is the leading cause for internal data leaks in corporates [4]. The data-leak risks posed by accidents exceed the risks posed by vulnerable software.

In order to minimize the exposure of sensitive data and documents, an organization needs to prevent cleartext sensitive data from appearing in the storage or communication. A screening tool can be deployed to scan computer file systems, server storage, and inspect outbound network traffic. The tool searches for the occurrences of *plaintext sensitive data* in the *content of files or network traffic*. It alerts users and administrators of the identified data exposure vulnerabilities. For example, an organization's mail server can inspect the content of outbound email messages searching for sensitive data appearing in unencrypted messages.

Data leak detection differs from the anti-virus (AV) scanning (e.g., scanning file systems for malware signatures) or the network intrusion detection systems (NIDS) (e.g., scanning traffic payload for malicious patterns) [5]. AV and NIDS typically employ automata-based string matching (e.g., Aho-Corasick [6], Boyer-Moore [7]), which match static or regular patterns. However, data leak detection imposes new security requirements and algorithmic challenges:

- *Data Transformation:* The exposed data in the content may be unpredictably transformed or modified by users or applications, and it may no longer be identical to the original sensitive data, e.g., insertions of metadata or formatting tags, substitutions of characters, and data truncation (partial data leak). Thus, the detection algorithm needs to recognize different kinds of sensitive data variations.
- *Scalability:* The heavy workload of data leak screening is due to two reasons.
  a) *Long Sensitive Data Patterns:* The sensitive data (e.g., customer information, documents, source code) can be of arbitrary length (e.g., megabytes).
  b) *Large Amount of Content:* The detection needs to rapidly screen content (e.g., gigabytes to terabytes). Traffic scanning is more time sensitive than storage scanning, because the leak needs to be discovered before the message is transmitted.

Directly applying automata-based string matching (e.g., [6], [8], [9]) to data leak detection is inappropriate and inefficient, because automata are not designed to support unpredictable and arbitrary pattern variations. In data leak detection scenarios, the transformation of leaked data (in the description of regular expression) is unknown to the detection method. Creating comprehensive automata models covering all possible variations of a pattern is infeasible, which leads to $O(2^n)$ space complexity (for deterministic finite automata) or $O(2^n)$ time complexity (for nondeterministic finite automata) where n is the number of automaton states. Therefore, automata approaches cannot be used for detecting long and transformed data leaks.

Existing data leak detection approaches are based on set intersection. Set intersection is performed on two sets of n-grams, one from the content and one from sensitive data. The set intersection gives the amount of sensitive n-grams appearing in the content. The method has been used to detect

Manuscript received June 28, 2015; revised October 9, 2015; accepted November 4, 2015. Date of publication November 24, 2015; date of current version December 24, 2015. This work was supported in part by Security and Software Engineering Research Center (S2ERC), an NSF sponsored multi-university Industry/University Cooperative Research Center (I/UCRC). Preliminary versions of this work appeared in *Proceedings of the Third International Workshop on Security and Privacy in Big Data (BigSecurity 2015)* and *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (CODASPY)*. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Vrizlynn L. L. Thing.

The authors are with the Department of Computer Science, Virginia Tech, Blacksburg, VA 24060 USA (e-mail: subx@cs.vt.edu; zjing14@cs.vt.edu; danfeng@cs.vt.edu; feng@cs.vt.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2015.2503271

similar documents on the web [10], shared malicious traffic patterns [11], malware [12], as well as email spam [13]. The advantage of n-grams is the extraction of local features of a string, enabling the comparison to tolerate discrepancies. Some advanced versions of the set intersection method utilize Bloom filter, e.g., [14], which trades accuracy for space complexity and speed. Shu and Yao extended the standard use of n-grams and introduced data-leak detection as a service. They proposed the first solution for detecting accidental data leak with semi-honest providers [15].

However, set intersection is *orderless*, i.e., the ordering of shared n-grams is not analyzed. Thus, set-based detection generates undesirable false alerts, especially when n is set to a small value to tolerant data transformation. In addition, set intersection cannot effectively characterize the scenario when partial data is leaked, which results in false negatives. Therefore, none of the existing techniques is adequate for detecting transformed data leaks.

Our solution to the detection of transformed data leaks is a sequence alignment algorithm, executed on the sampled sensitive data sequence and the sampled content being inspected. The alignment produces scores indicating the amount of sensitive data contained in the content. Our alignment-based solution measures the order of n-grams. It also handles arbitrary variations of patterns without an explicit specification of all possible variation patterns. Experiments show that our alignment method substantially outperforms the set intersection method in terms of detection accuracy in a multitude of transformed data leak scenarios.

We solve the scalability issue by sampling both the sensitive data and content sequences before aligning them. We enable this procedure by providing the pair of a *comparable* sampling algorithm and a *sampling-oblivious* alignment algorithm. The comparable sampling algorithm yields constant samples of a sequence wherever the sampling starts and ends. The sampling-oblivious alignment algorithm infers the similarity between the original unsampled sequences with sophisticated traceback techniques through dynamic programming. The algorithm infers the lost information (i.e., sampled-out elements) based on the matching results of their neighboring elements. Evaluation results show that our design boosts the performance, yet only incurs a very small amount of mismatches.

Existing network traffic sampling techniques, e.g., [16], only sample the content. Our problem differs from existing sampling problems that both sensitive data and content sequences are sampled. The alignment is performed on the sampled sequences. Therefore, the samples of similar sequences should be similar so that they can be aligned. We define a *comparable sampling* property, where the similarity of two sequences is preserved. For example, if x is a substring of y, then x' should be a substring of y', where x' and y' are sampled sequences of x and y, respectively. None of the existing sampling solutions satisfies this comparable sampling requirement. Deterministic sampling, e.g., [17], does not imply comparable sampling, either. The key to our comparable sampling is to consider the local context of a sequence while selecting items. Sample items are selected deterministically within a sliding window. The same sampled items are selected in spite of different starting/ending points of sampling procedures.

Both of our algorithms are designed to be efficiently parallelized. We parallelize our prototype on a multicore CPU and a GPU. We demonstrate the strong scalability of our design and the high performance of our prototypes. Our GPU-accelerated implementation achieves nearly 50 times of speedup over the CPU version. Our prototype reaches 400Mbps analysis throughput. This performance potentially supports the rapid security scanning of storage and communication required by a sizable organization.

We have presented the basic idea and preliminary evaluation results in our workshop paper [1]. In this paper, we formalize and expand the description and analysis of our comparable sampling algorithm and sampling-oblivious alignment algorithm. We conduct new experiments in Section VI to systematically understand how sensitive our system is in response to data transformation in various degrees. We also include the effectiveness evaluation of our sampling design in Section VII.

Our solution detects inadvertent data leaks, where sensitive data may be accidentally exposed. It is not designed for detecting data leaks caused by malicious insiders or attackers. The detection of data leaks due to malicious insiders remains a challenging open research problem.

II. Related Work

Existing commercial data leak detection/prevention solutions include Symantec DLP [14], IdentityFinder [18], GlobalVelocity [19], and GoCloudDLP [20]. GlobalVelocity uses FPGA to accelerate the system. All solutions are likely based on n-gram set intersection. IdentityFinder searches file systems for short patterns of numbers that may be sensitive (e.g., 16-digit numbers that might be credit card numbers). It does not provide any in-depth similarity tests. Symantec DLP is based on n-grams and Bloom filters. The advantage of Bloom filter is space saving. However, as explained in the introduction, Bloom filter membership testing is based on unordered n-grams, which generates coincidental matches and false alarms. Bloom filter configured with a small number of hash functions has collisions, which introduce additional unwanted false positives.

Network intrusion detection systems (NIDS) such as Snort [21] and Bro [22] use regular expression to perform string matching in deep packet inspection [23]. Nondeterministic finite automaton (NFA) with backtracking requires $O(2^n)$ time and $O(n)$ space, where n is the number of automaton states. Deterministic finite automaton (DFA) has a time complexity of $O(n)$ and a space complexity of $O(2^n)$ when used with quantification. Quantification is for expressing optional characters and multiple occurrences in a pattern. DFA's space complexity can be reduced by grouping similar patterns into one automaton [24], reducing the number of edges [25], [26]. These improvements provide a coefficient level of speedup.

However, existing string matching approaches based on DFA or NFA cannot automatically match arbitrary and unpredictable pattern variations. Modified data leak instances cannot be matched or captured if the variation is not manually specified as a matching pattern. Enumerating all potential variation patterns takes exponential time and space with respect to the length of the pattern. Therefore, it is impractical.

In comparison, our sequence alignment solution covers all possible pattern variations in long sensitive data without explicitly specifying them. Another drawback of automata is that it yields binary results. In comparison, alignment provides precise matching scores and allows customized weight functions. Our alignment gives more accurate detection than approximate string matching (e.g., [27], [28]).

Alignment algorithms have been widely used in computational biology applications, and features such as privacy-preserving sequence matching have been studied [29]. In security literature, NetDialign based on the well-known Dialign algorithms is proposed for network privacy [30]. It performs differential testing among multiple traffic flows. Kreibich and Crowcroft presented an alignment algorithm for traffic intrusion detection systems such as Bro [31]. It is a variant of Jacobson-Vo alignment that calculates the longest common subsequence with the minimum number of gaps. Researchers in [32] reported the use of dynamic programming for computing the similarity of network behaviors and presented a technique to handle behavioral sequences with differing sampling rates. Masquerade attacks in the context of user command sequences can be detected with semi-global sequence alignment techniques [33], [34].

Our data leak detection differs from the above network privacy and IDS problems, and it has new requirements as we have explained in the introduction. Our alignment performs complex inferences needed for aligning sampled sequences, and our solution is also different from fast non-sample alignment in bioinformatics, e.g., BLAST [35].

Another approach to the detection of sensitive data leak is to track the data/metadata movement. Several tools are developed for securing sensitive information on mobile platforms [36]–[38]. Nadkarni and Enck described an approach to control the sharing of sensitive files among mobile applications [37]. File descriptors (not the content) are stored, tracked and managed. The access control on files is enforced through policies. Yang et al. presented a method aiming at detecting the transmission of sensitive data that is not intended by smartphone users via symbolic execution analysis [38]. Hoyle et al. described a visualization method for informing mobile users of information exposure [36]. The information exposure may be caused by improper setting or configuration of access policies. The visualization is through an avatar apparel approach. Croft and Caesar expand the data tracking from a single host to a network and use shadow packets to distinguish normal traffic from leaks [39]. The security goals and requirements in all these studies are very different from ours, leading to different techniques developed and used.

iLeak is a system for preventing inadvertent information leaks on a personal computer [40]. It takes advantages of the keyword searching utility present in many modern operating systems. iLeak monitors the file access activities of processes and searches for system call inputs that involve sensitive data. Unlike our general data leak detection approach, iLeak is designed to secure personal data on a single machine, and its detection capability is restricted by the underlying keyword searching utility, which is not designed for detecting either transformed data leaks or partial data leaks.

Bertino and Ghinita addressed the issue of data leaks in database from the perspective of anomaly detection [41]. Normal user behaviors are monitored and modeled in DBMS, and anomalous activities are identified with respect to potential data leak activities. Bertino also discussed watermarking and provenance techniques used in data leak prevention and forensics [41], which is investigated in details by Papadimitriou and Garcia-Molina in [42].

Privacy is a well-known issue in the cloud. Lin and Squicciarini proposed a generic data protection framework to enforce data security in the cloud [43]. A three-tier data protection framework was proposed by Squicciarini et al. to deal with the data leak caused by indexing in the cloud [44]. Privacy-preserving data leak detection was proposed and further developed in [45] and [46], where data leak detection operations are outsourced to a semi-honest third-party. The solution is a specialized set intersection method. Therefore, it is different from this paper.

III. Models and Overview

In our data leak detection model, we analyze two types of sequences: sensitive data sequence and content sequence.

- *Content sequence* is the sequence to be examined for leaks. The content may be data extracted from file systems on personal computers, workstations, and servers; or payloads extracted from supervised network channels (details are discussed below).
- *Sensitive data sequence* contains the information (e.g., customers' records, proprietary documents) that needs to be protected and cannot be exposed to unauthorized parties. The sensitive data sequences are known to the analysis system.

In this paper, we focus on detecting inadvertent data leaks, and we assume the content in file system or network traffic (over supervised network channels) is available to the inspection system. A supervised network channel could be an unencrypted channel or an encrypted channel where the content in it can be extracted and checked by an authority. Such a channel is widely used for advanced NIDS where MITM (man-in-the-middle) SSL sessions are established instead of normal SSL sessions [47]. We do not aim at detecting stealthy data leaks that an attacker encrypts the sensitive data secretly before leaking it. Preventing intentional or malicious data leak, especially encrypted leaks, requires different approaches and remains an active research problem [48].

In our current security model, we assume that the analysis system is secure and trustworthy. Privacy-preserving data-leak detection can be achieved by leveraging special protocols and computation steps [49]. It is another functionality of a detection system, and the discussion is not within the scope of this paper.

A. Technical Challenges

1) High Detection Specificity: In our data-leak detection model, high specificity refers to the ability to distinguish true leaks from coincidental matches. *Coincidental matches* are false positives, which may lead to false alarms. Existing set-based detection is orderless, where the order of matched shingles (n-grams) is ignored. Orderless detection may generate coincidental matches, and thus having a lower accuracy of the detection. In comparison, our alignment-based method has high specificity. For example, a detection system can use 3-grams to represent the sensitive data.

Sensitive data	`abcdefg`
3-grams	`abc`, `bcd`, `cde`, `def`, `efg`

Then, consider the content streams 1 and 2 below. Stream 1 contains a true leak, whereas stream 2 does not.

Content stream 1	`....abcdefg...`
Content stream 2	`....efg...cde...abc...`

However, set intersection between 3-grams of the sensitive data and the 3-grams of content stream 2 results in a significant number of matching 3-grams (`efg`, `cde`, and `abc`), even though they are out of order compared to the sensitive data pattern. This problem is eliminated in alignment, i.e., the content stream 2 receives a low sensitivity score when aligned against the sensitive data.

2) Pervasive and Localized Modification: Sensitive data could be modified before it is leaked out. The modification can occur throughout a sequence (pervasive modification). The modification can also only affect a local region (local modification). We describe some modification examples:

- Character replacement, e.g., `WordPress` replaces every space character with a + in HTTP POST requests.
- String insertion, e.g., HTML tags inserted throughout a document for formatting or embedding objects.
- Data truncation or partial data leak, e.g., one page of a two-page sensitive document is transmitted.

B. Overview of Our Approach

Our work presents an efficient sequence comparison technique needed for analyzing a large amount of content for sensitive data exposure. Our detection approach consists of a comparable sampling algorithm and a sampling oblivious alignment algorithm. The pair of algorithms computes a quantitative similarity score between the sensitive data and the content. Local alignment – as opposed to global alignment [50] – is used to identify similar sequence segments. The design enables the detection of partial data leaks.

Our detection runs on continuous sequences of n bytes (n-grams). n-grams are obtained from the content and sensitive data, respectively. Local alignment is performed between the two (sampled) sequences to compute their similarity. The purpose of our comparable sampling operation is to enhance the analysis throughput. We discuss the tradeoff between security and performance related to sampling in our evaluation sections. Finally, we report the content that bears higher-than-threshold similarity with respect to sensitive patterns. Given a threshold T, content with a greater-than-T sensitivity is reported as a leak.

IV. Comparable Sampling

In this section, we define the sampling requirement needed in data leak detection. Then we present our solution and its analysis.

A. Definitions

One great challenge in aligning sampled sequences is that the sensitive data segment can be exposed at an arbitrary position in a network traffic stream or a file system. The sampled sequence should be deterministic despite the starting and ending points of the sequence to be sampled. Moreover, the leaked sensitive data could be inexact but similar to the original string due to unpredictable transformations. We first define substring and subsequence relations in Definition 1 and Definition 2. Then we define the capability of giving comparable results from similar strings in Definition 3.

Definition 1 (Substring): A substring is a consecutive segment of the original string.

If x is a substring of y, one can find a prefix string (denoted by y_p) and a suffix string (denoted by y_s) of y, so that y equals to the concatenation of y_p, x, and y_s. y_p and y_s could be empty.

Definition 2 (Subsequence): Subsequence is a generalization of substring that a subsequence does not require its items to be consecutive in the original string.

One can generate a subsequence of a string by removing items from the original string and keeping the order of the remaining items. The removed items can be denoted as gaps in the subsequence, e.g., `lo-e` is a subsequence of `flower` (- indicates a gap).

Definition 3 (Comparable Sampling): Given a string x and another string y that x is similar to a substring of y according to a similarity measure M, a comparable sampling on x and y yields two subsequences x' (the sample of x) and y' (the sample of y), so that x' is similar to a substring of y' according to M.

If we restrict the similarity measure M in Definition 3 to *identical relation*, we get a specific instance of comparable sampling in Definition 4.

Definition 4 (Subsequence-Preserving Sampling): Given x as a substring of y, a subsequence-preserving sampling on x and y yields two subsequences x' (the sample of x) and y' (the sample of y), so that x' is a substring of y'.

Because a subsequence-preserving sampling procedure is a restricted comparable sampling, so the subsequence-preserving sampling is deterministic, i.e., the same input always yields the same output. The vice versa may not be true.

In Example 1 with two sequences of integers, we illustrate the differences between a comparable sampling algorithm and a random sampling method, where a biased coin flipping at each position decides whether to sample or not. The input is a pair of two similar sequences. There is one modification (9 to 8), two deletions (7) and (3), and suffix padding (1, 4) in the second sequence. Local patterns are preserved in a comparable sampling method, whereas the random sampling does not. The local patterns can then be digested by our sampling-oblivious alignment algorithm to infer the similarity between the two original input sequences.

Example 1: `Comparable sampling.`

```
Inputs:
1 1 9 4 5 7 3 5 9 7 6 6 3 3 7 1 6
1 1 9 4 5 7 3 5 8 6 6 3 7 1 6 1 4

Comparable sampling may give:
1 1 - 4 - - 3 5 - - - - 3 3 - 1 -
1 1 - 4 - - 3 - - 6 - 3 - 1 - 1 4

Random sampling may give:
1 - - 4 - - 3 5 - 7 - 6 - - 7 1 -
- 1 9 - 5 - - 5 - 6 - 3 7 - 6 1 -
```

B. Our Sampling Algorithm

We present our comparable sampling algorithm. The advantage of our algorithm is its **context-aware selection**, i.e., the selection decision of an item depends on how it compares with its surrounding items according to a selection function. As a result, the sampling algorithm is *deterministic* and *subsequence-preserving*.

Our comparable sampling algorithm takes in $\mathcal{S}$, an input list of items (preprocessed n-grams of sensitive data or content),[1] and outputs $\mathcal{T}$, a sampled list of the same length; the sampled list contains null values, which correspond to items that are not selected. The null regions in $\mathcal{T}$ can be aggregated, and $\mathcal{T}$ can be turned into a *compact representation* $\mathcal{L}$. Each item in $\mathcal{L}$ contains the value of the sampled item and the length of the null region between the current sampled item and the preceding one.

$\mathcal{T}$ is initialized as an empty list, i.e., a list of null items. The algorithm runs a small sliding window w on $\mathcal{S}$. w is initialized with the first $|w|$ items in $\mathcal{S}$ (line 2 in Algorithm 1). The algorithm then utilizes a selection function to decide what items in w should be selected for $\mathcal{T}$. The selection decision is made based on not only the value of that item, but also the values of its neighboring items in w. Therefore, unlike a random sampling method where a selection decision is stochastic, our method satisfies the subsequence-preserving and comparable sampling requirements.

In Algorithm 1, without loss of generality, we describe our sampling method with a specific selection function $f = \min(w, N)$. f takes in an array w and returns the N smallest items (integers) in w. f is *deterministic*, and it *unbiasedly* selects items when items (n-grams) are preprocessed with the min-wise independent Rabin's fingerprint [51]. f can be replaced by other functions that are also min-wise independent. The selection results at each sliding window position determine what items are chosen for the sampled list. The parameters N and $|w|$ determine the sampling rate. `collectionDiff(A,B)` in lines 10 and 11 outputs the collection of all items of collection `A` that are not in collection `B`. The operation is similar to the set difference, except that it works on collections and does not eliminate duplicates.

[1]We preprocess n-grams with Rabin's fingerprint to meet the min-wise independent requirement of selection function f described next. Each item in $\mathcal{S}$ is a fingerprint/hash value (integer) of an n-gram.

Algorithm 1 A Subsequence-Preserving Sampling Algorithm

```
Input: an array $\mathcal{S}$ of items, a size $|w|$ for a sliding window $w$, a
    selection function $f(w, N)$ that selects $N$ smallest items from a
    window $w$, i.e., $f = \min(w, N)$
Output: a sampled array $\mathcal{T}$
 1: initialize $\mathcal{T}$ as an empty array of size $|\mathcal{S}|$
 2: $w \leftarrow \text{read}(\mathcal{S}, |w|)$
 3: let $w.head$ and $w.tail$ be indices in $\mathcal{S}$ corresponding to the
       higher-indexed end and lower-indexed end of $w$, respectively
 4: collection $m_c \leftarrow \min(w, N)$
 5: while $w$ is within the boundary of $\mathcal{S}$ do
 6:     $m_p \leftarrow m_c$
 7:     move $w$ toward high index by 1
 8:     $m_c \leftarrow \min(w, N)$
 9:     if $m_c \neq m_p$ then
10:         item $e_n \leftarrow \text{collectionDiff}(m_c, m_p)$
11:         item $e_o \leftarrow \text{collectionDiff}(m_p, m_c)$
12:         if $e_n < e_o$ then
13:             write value $e_n$ to $\mathcal{T}$ at $w.head$'s position
14:         else
15:             write value $e_o$ to $\mathcal{T}$ at $w.tail$'s position
16:         end if
17:     end if
18: end while
```

$\mathcal{T}$ output by Algorithm 1 takes the same space as $\mathcal{S}$ does. Null items can be combined, and $\mathcal{T}$ is turned into a *compact representation* $\mathcal{L}$, which is consumed by our sampling-oblivious alignment algorithm in the next phase.

We show how our sampling algorithm works in Table I. We set our sampling procedure with a sliding window of size 6 (i.e., $|w| = 6$) and $N = 3$. The input sequence is `1,5,1,9,8,5,3,2,4,8`. The initial sliding window $w =$ `[1,5,1,9,8,5]` and collection $m_c =$ `{1,1,5}`.

Sampling Algorithm Analysis: Our sampling algorithm is deterministic, i.e., given a fixed selection function f: same inputs yield the same sampled string. However, deterministic sampling (e.g., [17]) does not necessarily imply subsequence preserving. One can prove using a counterexample. Consider a sampling method that selects the first of every 10 items from a sequence, e.g., 1-st, 11-th, 21-st, … It is deterministic, but it does not satisfy the subsequence-preserving requirement. Some sampling methods such as coresets [52], [53] do not imply determinism.

Our sampling algorithm is not only deterministic, but also subsequence-preserving as presented in Theorem 1.

Theorem 1: Algorithm 1 (denoted by Ψ) is subsequence-preserving. Given two strings x and y, where x is a substring of y, then $\Psi(x)$ is a substring of $\Psi(y)$.

Proof: Let $L[m : n]$ denote the substring of L starting from the m-th item and ending at the n-th item. Consider strings L_1 and L_2 and their sampled sequences S_1 and S_2, respectively. We prove that the theorem holds in four cases.

Case 1: L_2 equals to L_1. Because our comparable sampling algorithm is deterministic, the same string yields the same sampled sequence. Thus, S_2 is a substring of S_1.

Case 2: L_2 is a prefix of L_1. The sampling of L_1 can be split into two phases.

TABLE I
ILLUSTRATION OUR SAMPLING PROCEDURE

Step	w	m_c	m_p	e_n	e_o	Sampled list
0	[1, 5, 1, 9, 8, 5]	1, 1, 5	N/A	N/A	N/A	<-, -, -, -, -, -, -, -, -, ->
1	[5, 1, 9, 8, 5, 3]	1, 3, 5	1, 1, 5	3	1	<1, -, -, -, -, -, -, -, -, ->
2	[1, 9, 8, 5, 3, 2]	1, 2, 3	1, 3, 5	2	5	<1, -, -, -, -, -, -, 2, -, ->
3	[9, 8, 5, 3, 2, 4]	2, 3, 4	1, 2, 3	4	1	<1, -, 1, -, -, -, -, 2, -, ->
4	[8, 5, 3, 2, 4, 8]	2, 3, 4	2, 3, 4	N/A	N/A	<1, -, 1, -, -, -, -, 2, -, ->

Phase 1 The head of the sliding moves within $L_1[size(win) : size(L_2)]$, i.e., from the start of L_1 to the exact position in L_1 where L_2 ends. Since L_2 is a prefix of L_1, and the window only moves within the scope of the prefix, the sample of L_1 generated in this subprocess is the same as S_2, the final sample of L_2.

Phase 2 The head of the sliding window moves within $L_1[size(L_2) + 1 : size(L_2) + size(win)]$. The tail of the sample window sweeps $L_1[size(L_2) - size(win) + 1 : size(L_2)]$ and yields zero or more sampled items on $S_1[size(L_2) - size(win) + 1 : size(L_2)]$.

$S_1[1 : size(L_2) - size(win)]$ is solely generated in **Phase 1**. Thus, it is the same as $S_2[1 : size(L_2) - size(win)]$. In **Phase 2**, we know that $S_1[size(L_2) - size(win) + 1 : size(L_2)]$ contains zero or more sample items than $S_2[size(L_2) - size(win) + 1 : size(L_2)]$. Thus, $S_2[size(L_2) - size(win) + 1 : size(L_2)]$ is a substring of $S_1[size(L_2) - size(win) + 1 : size(L_2)]$. Thus, S_2 is a substring of S_1.

Case 3: L_2 is a suffix of L_1. The proof is similar to **Case 2**. The sampling of L_1 can be split into two phases.

Phase 1 The tail of the sliding window moves within $L_1[size(L_1) - size(L_2) + 1 : size(L_1) - size(win)]$. The generated sampled sequence is the same as S_2, which is the final sample of L_2.

Phase 2 The tail of the sliding window moves within $L_1[size(L_1) - size(L_2) - size(win) + 1 : size(L_1) - size(L_2)]$. The head of the window sweeps $L_1[size(L_1) - size(L_2) + 1 : size(L_1) - size(L_2) + size(win)]$ and yields zero or more sampled items on $L_1[size(L_1) - size(L_2) + 1 : size(L_1) - size(L_2) + size(win)]$.

$S_1[size(L_1) - size(L_2) + size(win) + 1 : size(L_1) - size(L_2)]$ is the same as $S_2[size(L_1) - size(L_2) + size(win) + 1 : size(L_1) - size(L_2)]$. In addition, $S_2[size(L_1) - size(L_2) + 1 : size(L_1) - size(L_2) + size(win)]$ is a substring of $S_1[size(L_1) - size(L_2) + 1 : size(L_1) - size(L_2) + size(win)]$. Thus, S_2 is a substring of S_1.

Case 4: All others. This case is when L_2 is a substring of L_1, but not a prefix or suffix, i.e., $L_2[1 : size(L_2)] = L_1[m : n]$. We align L_1 and L_2 and cut the two strings at a position where they are aligned. Denote the position in L_2 by k. We obtain $L_2[1 : k]$ as a suffix of $L_1[m : m + k]$ and $L_2[k + 1 : size(L_2)]$ as a prefix of $L_1[m + k + 1 : n]$. Based on the proofs in **Case 2** and **Case 3**, we conclude that $S_2[1 : k]$ is a substring of $S_1[m : m + k]$, and $S_2[k + 1 : size(L_2)]$ is a substring of $S_1[m + k + 1 : n]$. Thus, S_2 is a substring of S_1.

In summary, Theorem 1 holds in every case. □

Our algorithm is unbiased, meaning that it gives an equal probability for every unit in the string to be selected. To achieve bias-free property, we hash inputs using a min-wise independent function, namely Rabin's fingerprint [54]. It guarantees that the smallest N items come equally from any items in the original string. This hashing is performed in PREPROCESSING operation in our prototypes.

The complexity of sampling using the $\min(w, N)$ selection function is $O(n \log |w|)$, or $O(n)$ where n is the size of the input, $|w|$ is the size of the window, The factor $O(\log |w|)$ comes from maintaining the smallest N items within the window w.

Sampling rate $\alpha \in [\frac{N}{|w|}, 1]$ approximates $\frac{N}{|w|}$ for random inputs, where $|w|$ is the size of the sliding window, and N is the number of items selected within the sliding window. For arbitrary inputs, the actual sampling rate depends on the characteristics of the input space and the selection function used. The sampling rate in our evaluations on common Internet traffic is around $1.2\frac{N}{|w|}$.

Sufficient number of items need to be sampled from sequences in order to warrant an accurate detection. Our empirical result in Section VI-B shows that with 0.25 sampling rate our alignment method can detect as short as 32-byte-long sensitive data segments.

V. ALIGNMENT ALGORITHM

In this section, we describe the requirements for a sample-based alignment algorithm and present our solution.

A. Requirements and Overview

We design a specialized alignment algorithm that runs on compact sampled sequences $\mathcal{L}^a$ and $\mathcal{L}^b$ to infer the similarity between the original sensitive data sequence $\mathcal{S}^a$ and the original content sequence $\mathcal{S}^b$. It needs to satisfy the requirement of **sampling oblivion**, i.e., the result of a sampling-oblivious alignment on sampled sequences $\mathcal{L}^a$ and $\mathcal{L}^b$ should be consistent with the alignment result on the original $\mathcal{S}^a$ and $\mathcal{S}^b$.

Conventional alignment may underestimate the similarity between two substrings of the sampled lists, causing misalignment. Regular local alignment without the sampling oblivion property may give inaccurate alignment on sampled sequences as illustrated in Example 2.

Example 2: Sampling-oblivious alignment vs. regular local alignment

```
Original lists:
          5627983857432546397824366
             5627983966432546395
Sampled sequences need to be aligned as:
          --2---3-5---2---3-7-2-3--
             --2---3-6---2---3--
However, regular local alignment may give:
                    23523723
                      23623
```

Because values of unselected items are unknown to the alignment, the decision of match or mismatch cannot be made solely on them during the alignment. We observe that leaked data region is usually consecutive, e.g., spans at least dozens of bytes. Thus, our algorithm achieves sampling oblivion by inferring the similarity between null regions (consecutive sampled-out elements) and counts that similarity in the overall comparison outcomes between the two sampled sequences. The inference is based on the comparison outcomes between items surrounding null regions and sizes of null regions. For example, given two sampled sequences `a-b` and `A-B`, if `a == A` and `b == B`, then the two values in the positions of the null regions are likely to match as well. In case of mismatch surrounding the null region, penalty is applied. Our experimental results confirm that this inference mechanism is effective.

We develop our alignment algorithm using dynamic programming. A string alignment problem is divided into three prefix alignment subproblems: the current two items (from two sequences) are aligned with each other, or one of them is aligned with a gap. In our algorithm, not only the sampled items are compared, but also comparison outcomes between null regions are inferred based on their non-null neighboring values and their sizes/lengths. The comparison results include *match*, *mismatch* and *gap*, and they are rewarded (match) or penalized (mismatch or gap) differently for sampled items or null regions according to a weight function $f_w()$.

Our alignment runs on sampled out elements. We introduce *i)* extra fields of scoring matrix cells in dynamic programming, *ii)* extra steps in recurrence relation for bookkeeping the null region information, and *iii)* a complex weight function estimating similarities between null regions.

Security Advantages of Alignment: There are three major advantages of our alignment-based method for detecting data leaks: order-aware comparison, high tolerance to pattern variations, and the capability of partial leak detection. All features contribute to high detection accuracy.

- *Order-Aware Comparison:* Existing data leak filtering methods based on set intersection are orderless. An orderless comparison brings undesirable false alarms due to coincidental matches, as explained in Section III. In comparison, alignment is order-aware, which significantly reduces the number of false positives.
- *High Tolerance to Pattern Variations:* The optimal alignment between the sensitive data sequence and content sequence ensures high accuracy for data leak detection. The alignment-based detection tolerates pattern variations in the comparison, thus can handle transformed data leaks. The types of data transformation in our model include localized and pervasive modifications such as insertion, deletion, and substitution, but exclude strong encryption.
- *Capability of Detecting Partial Leaks:* Partial data leak is an extreme case of truncation in transformation. In set-intersection methods, the size of sensitive data and that of the inspected content are usually used to diminish the score of coincidental matches, which incurs false negatives when only partial sensitive data is leaked. Local alignment searches for similar substrings in two sequences, thus it can detect a partial data leak.

Algorithm 2 Recurrence Relation in Dynamic Programming

```
Input: A weight function $f_w$, visited cells in $H$ matrix that are adjacent to $H(i,j)$: $H(i-1,j-1)$, $H(i,j-1)$, and $H(i-1,j)$, and the $i$-th and $j$-th items $\mathcal{L}^a_i$, $\mathcal{L}^b_j$ in two sampled sequences $\mathcal{L}^a$ and $\mathcal{L}^b$, respectively.
Output: $H(i,j)$
1: $h^{up}.score \leftarrow f_w(\mathcal{L}^a_i, -, H(i-1,j))$
2: $h^{left}.score \leftarrow f_w(-, \mathcal{L}^b_j, H(i,j-1))$
3: $h^{dia}.score \leftarrow f_w(\mathcal{L}^a_i, \mathcal{L}^b_j, H(i-1,j-1))$
4: $h^{up}.null_{row} \leftarrow 0$
5: $h^{up}.null_{col} \leftarrow 0$
6: $h^{left}.null_{row} \leftarrow 0$
7: $h^{left}.null_{col} \leftarrow 0$
8: $h^{dia}.null_{row} \leftarrow \begin{cases} 0, & \text{if } \mathcal{L}^a_i = \mathcal{L}^b_j \\ H(i-1,j).null_{row} + \mathcal{L}^a_i.span + 1, & \text{else} \end{cases}$
9: $h^{dia}.null_{col} \leftarrow \begin{cases} 0, & \text{if } \mathcal{L}^a_i = \mathcal{L}^b_j \\ H(i,j-1).null_{col} + \mathcal{L}^b_j.span + 1, & \text{else} \end{cases}$
10: $H(i,j) \leftarrow \arg\max_{h.score} \begin{cases} h^{up} \\ h^{left} \\ h^{dia} \end{cases}$
11: $H(i,j).score \leftarrow \max \begin{cases} 0 \\ H(i,j).score \end{cases}$
```

B. Recurrence Relation

We present the recurrence relation of our dynamic program alignment algorithm in Algorithm 2. For the i-th item $\mathcal{L}_i$ in a sampled sequence $\mathcal{L}$ (the compact form), the field $\mathcal{L}_i.value$ denotes the value of the item and a new field $\mathcal{L}_i.span$ denotes the size of null region between that item and the preceding non-null item. Our local alignment algorithm takes in two sampled sequences $\mathcal{L}^a$ and $\mathcal{L}^b$, computes a non-negative score matrix H of size $|\mathcal{L}^a|$-by-$|\mathcal{L}^b|$, and returns the maximum alignment score with respect to a weight function. Each cell $H(i,j)$ has a score field $H(i,j).score$ and two extra fields recording sizes of neighboring null regions, namely $null_{row}$ and $null_{col}$.

The intermediate solutions are stored in matrix H. For each subproblem, three previous subproblems are investigated: *i)* aligning the current elements without a gap, which leads to a *match* or *mismatch*, *ii)* aligning the current element in $\mathcal{L}^a$

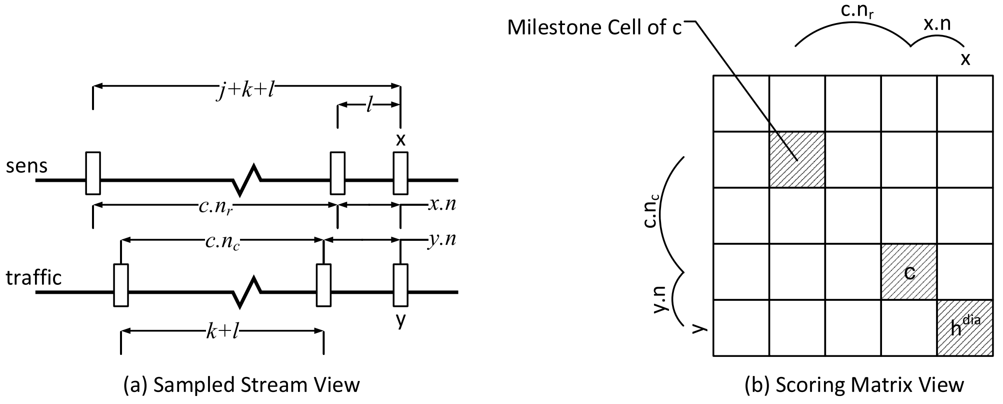


Fig. 1. Illustration of the notation used in the weight function $f_w()$ for the match case (i. e., $x.value = y.value$) in the alignment view (a) and matrix view (b). The milestone cell in (b) is for inference due to sampling.

with a gap, and *iii)* aligning the current element in $\mathcal{L}^b$ with a gap. A cell candidate h is generated for each situation; its score $h.score$ is computed via the weight function f_w (lines 1 to 3 in Algorithm 2). The other two fields, $null_{row}$ and $null_{col}$, are updated for each candidate cell (lines 4 to 9). This update may utilize the null region value stored in the *span* field of an item. All three cell candidates h^{up}, h^{left}, and h^{dia} are prepared. The cell candidate having the highest score is chosen as $H(i, j)$, and the score is stored in $H(i, j).score$.

C. Weight Function

A weight function computes the score for a specific alignment configuration. Our weight function $f_w()$ takes three inputs: the two items being aligned (e.g., $\mathcal{L}_i^a$ from sensitive data sequence and $\mathcal{L}_j^b$ from content sequence) and a reference cell c (one of the three visited adjacent cells $H(i-1, j-1)$, $H(i, j-1)$, or $H(i-1, j)$). It then outputs a score of an alignment configuration. One of $\mathcal{L}_i^a$ and $\mathcal{L}_j^b$ may be a gap $(-)$ in the alignment. The computation is based on the penalty given to mismatch and gap conditions and reward given to match conditions. Our weight function differs from the one in Smith-Waterman algorithm [55] in its ability to infer comparison outcomes for null regions. This inference is done efficiently accordingly to the values of their adjacent non-null neighboring items. The inference may trace back to multiple preceding non-null items up to a constant factor.

In our $f_w()$, r is the reward for a single unit match, m is the penalty for a mismatch, and g is the penalty for a single unit aligned with a gap. As presented in Section V-B, the field $value$ is the value of a sampled item (e.g., $x.value$ or $y.value$ in $f_w()$ below), and the field $span$ stores the length of the null region preceding the item. For the input cell c, the fields n_r (short for $null_{row}$) and n_c (short for $null_{col}$) record the size of the accumulated null regions in terms of row and column from the nearest milestone cell (explained next in our traceback strategy) to the current cell. $\text{diff}(m, n) = |m - n|$. Values p, q, l, k, and j serve as weight coefficients in our penalty and reward mechanisms. We detail our weight function $f_w()$ below and illustrate the lengths l, k and j for the match case in Figure 1.

1) (Gap) h^{up}

$$f_w(x, -, c) = c.score + m \times p + g \times q$$

where

$$p = \min(c.n_r + x.span + 1, c.n_c)$$
$$q = \text{diff}(c.n_r + x.span + 1, c.n_c)$$

2) (Gap) h^{left}

$$f_w(-, y, c) = c.score + m \times p + g \times q$$

where

$$p = \min(c.n_r, c.n_c + y.span + 1)$$
$$q = \text{diff}(c.n_r, c.n_c + y.span + 1)$$

3) (Mismatch) $h^{dia}|x.value \neq y.value$

$$f_w(x, y, c) = cell.score$$

4) (Match) $h^{dia}|x.value = y.value$

$$\begin{aligned} f_w(x, y, c) = cell.score \\ + r \times l \\ + m \times k \\ + g \times j \end{aligned}$$

where

$$\begin{aligned} l &= \min(x.span, y.span) + 1, \\ k &= \min(c.n_r, c.n_c) - l, \\ j &= \text{diff}(c.n_r, c.n_c) + \text{diff}(x.span, y.span) + l \end{aligned}$$

Traceback in our weight function is for inferring matching outcomes based on preceding null regions, including the adjacent one. Our traceback operation is efficient. It extends to a constant number of preceding null regions. To achieve this property, we define a special type of cells (referred to as milestone cells) in matrix H with zero $null_{row}$ and $null_{col}$ fields. These milestone cells mark the boundary for the traceback inference; the subproblems (upper left cells) of a milestone cell are not visited. A milestone cell is introduced in either match or gap cases in f_w.

TABLE II
DATASETS IN ACCURACY & SCALABILITY EXPERIMENTS

Dataset	Size	Details
A. Enron [56]	2.6 GB	517,424 email (with full headers and bodies) of 150 users
B. Source-code	3.8 MB	288 source files in projects `tar`, `net-tools`, `gzip`, `procps`, and `rsync`
C. Outbound HTTP requests	12 MB	HTTP requests of 20 users (30-minute Internet activities recorded for each user)
D. Outbound/inbound MiscNet	500MB	Miscellaneous web traffic containing text and multimedia content

D. Algorithm Analysis

The complexity of our alignment algorithm is $O(|\mathcal{L}^a||\mathcal{L}^b|)$, where $|\mathcal{L}^a|$ and $|\mathcal{L}^b|$ are lengths of compact representations of the two sampled sequences. The alignment complexity for a single piece of sensitive data of size l is the same as that of a set of shorter pieces with a total size l, as the total amounts of matrix cells to compute are the same.

In a real-world deployment, the overall sensitive data sequence $\mathcal{S}^a$ is usually close to a fixed length, and more attention is commonly paid to the length of the content sequence $\mathcal{S}^b$. In this case, the complexity of our alignment is $O(|\mathcal{L}^b|)$ where $\mathcal{L}^b$ is the sampled list of $\mathcal{S}^b$. We experimentally evaluate the throughput of our prototype in Section VII, which confirms the $O(|\mathcal{L}^b|)$ complexity in the analysis.

The correctness of our alignment is ensured by dynamic programming and the recurrence relation among the subproblems of string alignment. The preciseness of similarity inference between sampled-out elements is achieved by our specifically designed weight function. Empirical results show that the alignment of sampled sequences $\mathcal{L}^a$ and $\mathcal{L}^b$ is very close to the alignment of original sequences $\mathcal{S}^a$ and $\mathcal{S}^b$, confirming the sampling oblivion property.

Our alignment of two sampled sequences achieves a speedup in the order of $O(\alpha^2)$, where $\alpha \in (0, 1)$ is the sampling rate. There is a constant damping factor due to the overhead introduced by sampling. The expected value is 0.33 because of the extra two fields, besides the score field, to maintain for each cell in H. We experimentally verify the damping factor in our evaluation.

Permutation-based data transformation (e.g., position swaps) affects the alignment precision and reduces the overall detection accuracy.

VI. EVALUATION ON DETECTION ACCURACY

We extensively evaluate the accuracy of our solution with several types of datasets under a multitude of real-world data leak scenarios. Our experiments in this section aim to answer the following questions.

1) Can our method detect leaks with pervasive modifications, e.g., character substitution throughout a sensitive document?
2) Can our method detect localized modifications, especially partial data leaks?
3) How specific is our detection, that is, the evaluation of false positives?
4) How does our method compare to the state-of-the-art collection intersection method in terms of detection accuracy?

TABLE III
SEMANTICS OF TRUE AND FALSE POSITIVES AND TRUE AND FALSE NEGATIVES IN OUR MODEL

	True Leak	No Leak
Leak detected	*TP*	*FP*
No leak detected	*FN*	*TN*

A. Implementation and Experiment Setup

We implement a single-threaded prototype (referred to as *AlignDLD* system) and a collection intersection method (referred to as *Coll-Inter* system), which is a baseline. Both systems are written in `C++`, compiled using `g++ 4.7.1` with flag `-O3`. We also provide two parallel versions of our prototype in Section VII for performance demonstration.

- *AlignDLD:* our sample-and-align data leak detection method with sampling parameters $N = 10$ and $|w| = 100$. 3-grams and 32-bit Rabin's fingerprints[2] are used.
- *Coll-Inter:* a data leak detection system based on collection intersection,[3] which is widely adopted by commercial tools such as GlobalVelocity [19] and GoCloudDLP [20]. 8-grams and 64-bit Rabin's fingerprints are used, which is standard with collection intersection.

We use four datasets (Table II) in our experiments. *A. Enron* and *B. Source-code* are used either as the sensitive data or the content to be inspected. *C. Outbound HTTP requests* and *D. MiscNet* are used as the content. Detailed usages of these datasets are specified in each experiment.

We report the detection rate in Equation (1) with respect to a certain threshold for both *AlignDLD* and *Coll-Inter* systems. The detection rate gives the percentage of leak incidents that are successfully detected. We also compute standard false positive rate defined in Equation (2). We detail the semantic meaning for primary cases, true positive (TP), false positive (FP), true negative (TN), and false negative (FN), in Table III.

$$\text{Detection rate (Recall)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{1}$$

$$\text{False positive rate} = \frac{\text{FP}}{\text{FP} + \text{TP}} \tag{2}$$

We define the *sensitivity* $\mathbb{S} \in [0, 1]$ of a content sequence in Equation (3). It indicates the similarity of sensitive data D

[2]Rabin's fingerprint is used for unbiased sampling discussed in Section IV-B.

[3]Set and collection intersections are used interchangeably.

and content $C_{D'}$ with respect to their sequences $\mathcal{S}^a$ and $\mathcal{S}^b$ after PREPROCESS. ξ is the maximum score in the alignment, i.e., the maximum score calculated in the scoring matrix of our dynamic programming alignment. r is the reward for one-unit match in the alignment (details in Section V-C).

$$\mathbb{S} = \frac{\xi}{r \times \min(|\mathcal{S}^a|, |\mathcal{S}^b|)} \tag{3}$$

We reproduce four leaking scenarios in a virtual network environment using VirtualBox. We build a virtual network and deploy the detection systems at the gateway of the virtual network. The detection systems intercept the outbound network traffic, perform deep packet inspection, and extract the content at the highest known network layer.[4] Then the detection systems compare the content with predefined sensitive data to search for any leak.

1) *Web Leak:* a user publishes sensitive data on the Internet via typical publishing services, e.g., `WordPress`,
2) *FTP:* a user transfers unencrypted sensitive files to an FTP server on the Internet,
3) *Backdoor:* a malicious program, i.e., `Glacier`, on the user's machine exfiltrates sensitive data,
4) *Spyware:* a `Firefox` extension `FFsniFF` [57] exfiltrates sensitive information via web forms.

It is not a challenge to detect intact data leaks. Our *AlignDLD* system successfully detects intact leaks in all these leaking scenarios with a small sampling rate between 5% and 20%. In the following subsections, we analyze the detection accuracy to answer the questions at the beginning of this section.

B. Detecting Modified Leaks

We evaluate three types of modifications: *i)* real-world pervasive substitution by `WordPress`, *ii)* random pervasive substitution, and *iii)* truncated data (localized modifications).

1) Pervasive Substitution: We test *AlignDLD* and *Coll-Inter* on content extracted from three kinds of network traffic.

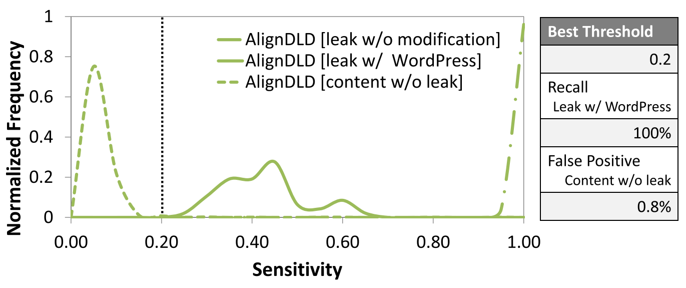
1) Content without any leak, i.e., the content does not contain any sensitive data.
2) Content with unmodified leak, i.e., sensitive data appearing in the content is not modified.
3) Content with modified leaks caused by `WordPress`, which substitutes every space with a "+" in the content.

The sensitive dataset in this experiment is English text, 50 randomly chosen email messages from the Enron dataset.[5] The content without leak consists of other 950 randomly chosen Enron email messages. We compute the sensitivities of the content according to Equation (3).

We evaluate and compare our *AlignDLD* method with the *Coll-Inter* method. The distributions of sensitivity values in all 6 experiments are shown in Figure 2. The table to the right of each figure summarizes the detection accuracy under a chosen threshold. The dotted lines in both

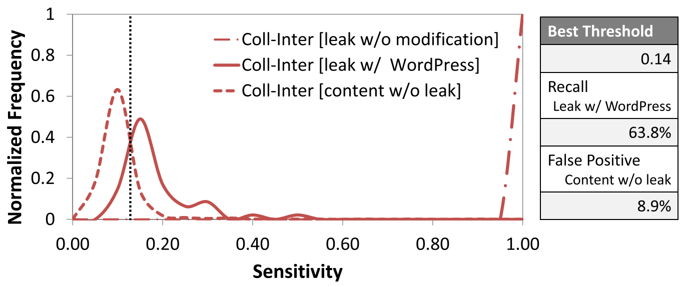
[4]The content is obtained at the TCP layer when unknown protocols are used at higher network layers.

[5]Headers are included.



Best Threshold
0.2
Recall Leak w/ WordPress
100%
False Positive Content w/o leak
0.8%

(a) AlignDLD (our method)



Best Threshold
0.14
Recall Leak w/ WordPress
63.8%
False Positive Content w/o leak
8.9%

(b) Collection Intersection (Coll-Inter, baseline)

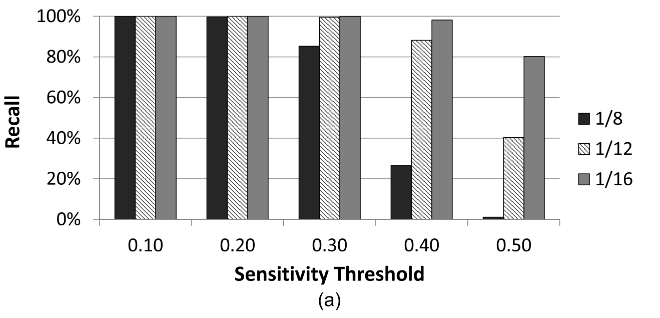
Fig. 2. Detection comparison of leak through WordPress in AlignDLD (a) and collection intersection (b). In each subfigure, each of the 3 curves shows the distribution of sensitivity values under one of 3 scenarios: leak without transformation, leak with WordPress transformation, or content without leak. With a threshold of 0.2, AlignDLD detects all the leaks. In comparison, collection intersection performs worse as shown in the table on the right.

Figure 2(a) and (b) (on the left) represent the content without leak. Low sensitivities are observed in them by both systems as expected. The dashed lines (on the right) represent the content with unmodified leak. High sensitivities are reported by both systems as expected.

The solid lines in Figure 2 represent the detection results of leaks with `WordPress` modifications. Our *AlignDLD* method (in Figure 2(a)) gives much higher sensitivity scores to the transformed data leak than the *Coll-Inter* method. ***AlignDLD* detects all transformed email leaks with a threshold of 0.2,** i.e., it achieves 100% recall. The false positive rate is low. In contrast, *Coll-Inter* in Figure 2(b) results in a significant overlap of sensitivity values between messages with no leak and messages with transformed leaks. Its accuracy is much lower than that of *AlignDLD*, e.g., 63.8% recall and a 10 times higher false positive rate. Further analysis of false positives caused by coincidental matches (dotted lines on the left) is given in Section VI-C.

2) Random and Pervasive Substitution: The sensitive data in this experiment is the same as above, i.e., randomly chosen 50 Enron emails (including headers). For the content sequences, we randomize one byte out of every m bytes, where $m \in \{8, 12, 16\}$. The smaller m is, the harder the detection is, as the similarity between the content and sensitive data becomes lower. The detection results with respect to various thresholds are shown in Figure 3.

The recall values decrease as the substitution frequency increases for both the alignment and collection intersection methods as expected. **Our alignment method degrades more gracefully under the pervasive substitution scenario.** For example, under threshold 0.3, the detection rate is over 80% even when one out of every 8 bytes is substituted.



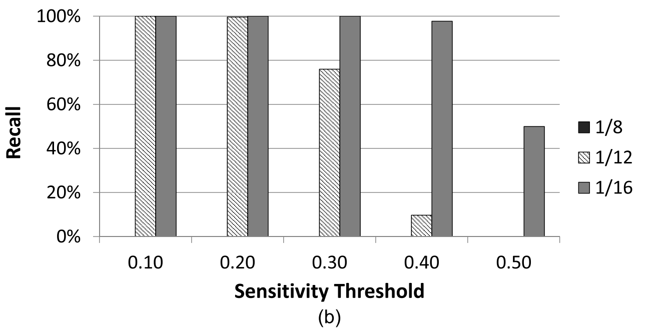


Fig. 3. Sensitivity values of the content under various transformation ratios with *A. Enron* dataset. Transformation ratio (X-axis) denotes the fraction of leaked sensitive data that is randomized. (a) AlignDLD (our method). (b) Collection Intersection (Coll-Inter, baseline).

The collection intersection cannot detect the leak (0% detection rate) in the same scenario.

3) Data Truncation: In data truncation or partial data leak scenarios, consecutive portions of the sensitive data are leaked. In this experiment, a content sequence contains a portion of sensitive text. The total length of the sensitive text is 1KB. The size of the leaked portion appearing in the content sequence ranges from 32 bytes to 1KB. Each content sequence is 1KB long with random padding if needed.

We measure the *unit sensitivity* $\tilde{\mathbb{S}} \in [0, 1]$ on segments of content sequences. Unit sensitivity $\tilde{\mathbb{S}}$ is the normalized *per-element* sensitivity value for the aligned portion of two sequences. It is defined in Equation (4), where $\tilde{\xi}$ is the maximum local alignment score obtained between aligned segments $\tilde{\mathcal{S}}^a$ and $\tilde{\mathcal{S}}^b$, which are sequence segments of sensitive data D and content $C_{D'}$. The higher $\tilde{\mathbb{S}}$ is, the better the detection is. Threshold l is a predefined length describing the shortest segment to invoke the measure. $l = 16$ in our experiments.

$$\tilde{\mathbb{S}} = \frac{\tilde{\xi}}{r \times \min(|\tilde{\mathcal{S}}^a|, |\tilde{\mathcal{S}}^b|)} \quad \text{where } \min(|\tilde{\mathcal{S}}^a|, |\tilde{\mathcal{S}}^b|) \geq l \qquad (4)$$

The detection results are shown in Figure 4, where X-axis shows the threshold of sensitivity, and Y-axis shows the recall rates of AlignDLD. Content with longer sensitive text is easier to detection as expected. Nevertheless, **our method detects content with short truncated leaks as small as 32 bytes with high accuracy.** The detection rate decreases with higher thresholds. We observe that high thresholds (e.g., higher than 0.6) are not necessary for detection when 8-byte shingles are used; false positives caused by coincidental matches are low in this setup. These experiments show that our detection is resilient to data truncation.

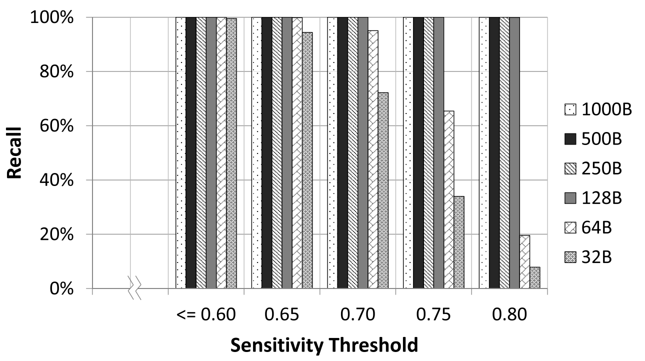


Fig. 4. The detection success rate of AlignDLD in partial data leaks under various detection thresholds. Each content sequence contains a consecutive portion of a 1KB sensitive text, ranging from 32 bytes to 1KB. AlignDLD achieves 100% detection rates when the threshold is equal or smaller than 0.6.

TABLE IV

SAMPLING RATES OF AlianDLD ON A.ENRON AND B.SOURCE-CODE DATA SETS $|w| = 100$

N	2	3	5	10	20	40
Enron	2.83%	4.14%	6.67%	12.32%	22.78%	43.1%
S.Code	2.81%	4.01%	6.30%	11.81%	22.33%	43.04%

C. Low False Positive Rate

The purpose of this experiment is to evaluate how specific our alignment-based data leak detection is, i.e., reporting leaks and only leaks. We compute and compare the amount of coincidental matches (defined in Section III) found by our method and the collection intersection method. We conduct two sets of experiments using *A. Enron* and *B. Source-code* datasets. In *A. Enron*, we use 50 random email messages (including headers) as the sensitive data and other 950 messages as the content. In *B. Source-code*, we use 5 random files as the sensitive data and other 283 files as the content. None of the contents contain any intentional leaks. Sensitivity scores are computed for each email message and source code file. Small amounts of coincidental matches are expected in these two datasets, because of shared message structures and C/C++ code structures.

We test the impact of sampling in this experiment. We chose screen size $N = 2, 3, 5, 10, 20, 40$ and window size $|w| = 100$. The sampling rates (Table IV) on the two datasets are similar when rounded to percentages. This is because Rabin's fingerprint maps any n-gram uniformly to the item space before sampling.

We measure the signal-to-noise ratios (SNR$_{\text{dB}}$) between sensitive scores of real leaks and sensitive scores of non-leak traffic. We calculate SNR$_{\text{dB}}$ as in Equation 5, where the signal value is the averaged sensitivity score of traffic containing leaks, and the noise value is the averaged sensitivity score of regular traffic with no leaks.

$$\text{SNR}_{\text{dB}} = 10 \log_{10} \frac{\text{Signal}}{\text{Noise}} \qquad (5)$$

Our results in Figure 5 show that the sensitivities are equal or less than 0.1 for almost all detection using our AlignDLD system. With a reasonable threshold (e.g., 0.2), none of these coincidental matches triggers a false alarm. The detection

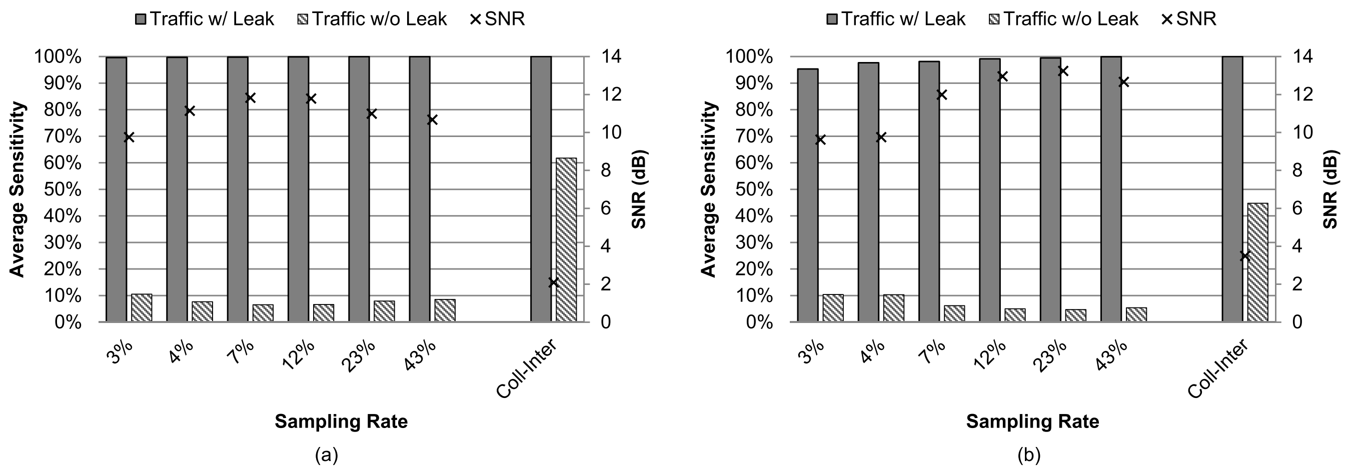


Fig. 5. Capability of differentiating real leak from coincidental matches of our AlignDLD approach with different sampling rates and Coll-Inter. (a) Enron. (b) Source-code.

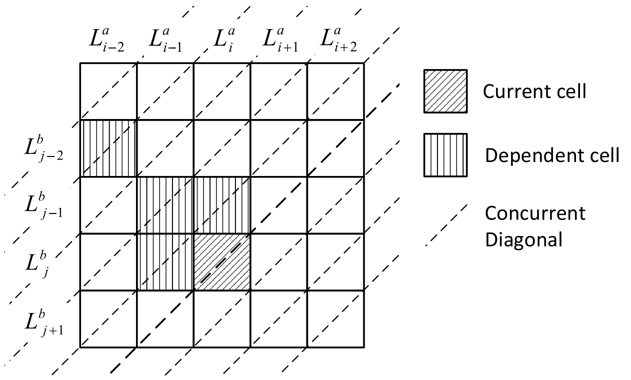


Fig. 6. Parallel realization of our alignment algorithm. $\mathcal{L}^a_i$ and $\mathcal{L}^b_j$ are the current items to be aligned. All cells on the concurrent diagonal of $(\mathcal{L}^a_i, \mathcal{L}^b_j)$ can be computed simultaneously.

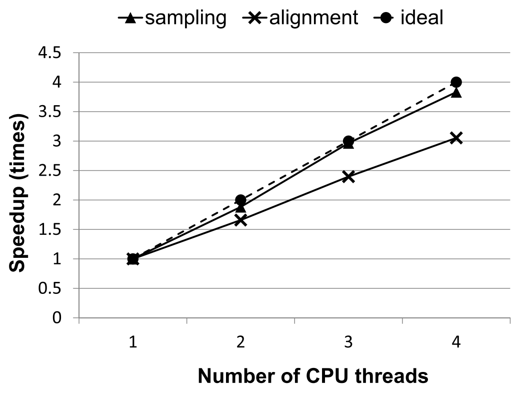


Fig. 7. High scalability of parallel sampling and alignment algorithms.

capability of our approach is generally stable with respect to different sampling rates. We observe that sampling rates have a noticeable but insignificant impact on the results. SNR_{dB} slightly increases when the sampling rate is small, e.g., 3%.

Our previous experiments in Section VI-B show that thresholds $\geq$ 0.2 give a strong separation between true leaks and coincidental matches. Thus, the evidence shows that our method achieves high recall with zero or low false positive rate. In comparison, the collection intersection method reports higher sensitivity scores for the content without any leak, e.g., 62% for Enron emails. High sensitivity scores in coincidental matches lead to a high false positive rate for the collection intersection method as illustrated in Figure 2.

Summary: The experimental results provide strong evidences supporting that our method is resilience against various types of modifications evaluated. Our alignment algorithm provides a high specificity (i.e., low number of coincidental matches), compared to the collection intersection method. Our approach is capable of detecting leaks of various sizes, ranging from tens of bytes to megabytes.

VII. Parallelization and Evaluation

In order to achieve high analysis throughput, we parallelize our algorithms on CPU as well as on general-purpose GPU platforms. In this section, we aim to answer the following questions:

1) How well does our detection scale? (Sections VII-B and VII-C)
2) What is the speedup of sampling? (Section VII-D)

A. Parallel Detection Realization

We implement two parallel versions of our prototype on a hybrid CPU-GPU machine equipped with an Intel Core i5 2400 (Sandy-Bridge micro-architecture) and an NVIDIA Tesla C2050 GPU (Fermi architecture with 448 GPU cores):

1) a multithreading AlignDLD program on CPU,[6]
2) a parallel AlignDLD program on GPU.[7]

Smith-Waterman alignment was parallelized in OpenGL [58] and CUDA [59]. Our parallel alignment algorithms differ from the existing ones, as we address several implementation issues in parallel computing due to our complex weight function.

In the multithreading CPU version, we parallelize both the Sampling and Alignment procedures with the `pthread` library. We parallelize the Sampling operation by loading different strings onto different threads. Long streams are split

[6]The multithreaded CPU version is written in C, compiled using gcc 4.4.5 with flag -O2.

[7]The GPU version is written in CUDA compiled using CUDA 4.2 with flag -O2 -arch sm 20 and NVIDIA driver v295.41.

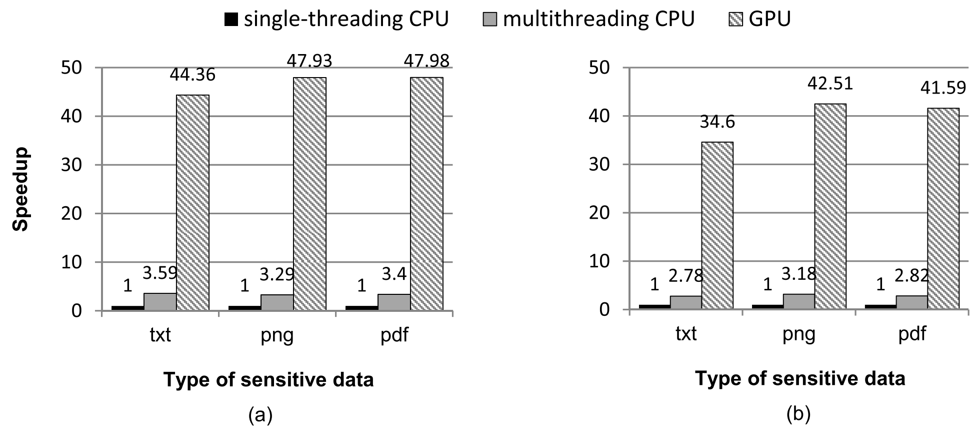


Fig. 8. Speedup of multithreading alignment and GPU-accelerated alignment over the single thread version on different combinations of sensitive data and traffic data. (a) Enron. (b) MiscNet.

into multiple substrings. Substrings are sampled in parallel by different threads and then assembled for output. ALIGNMENT is the most time-consuming procedure and is made parallel on both CPU and GPU.

We use a parallelized score matrix filling method to compute a diagonal of cells at the same time. Our parallel matrix filling strategy is illustrated in Figure 6. The scoring matrix is filled from the top left corner to the bottom right corner. At any stage of the process, cells on the concurrent diagonal (dashed lines in Figure 6) can be computed simultaneously. Our strategy is a variant of the standard Smith-Waterman parallelism [60]. Dependent cells in our algorithm include traditional three adjacent cells as well as all previous cells on the diagonal that is orthogonal to the concurrent diagonal.

The alignment between a sampled sensitive data sequence and a sampled content sequence is assigned to a block of threads on the GPU, and every thread in the block is responsible for an element on the moving diagonal. This method consumes linear space and is efficient. It allows us to fully utilize the memory bandwidth, putting reusable data into fast but small (32KB in our case) shared memory on GPU.

B. Scalability

In this experiment, we parallelize SAMPLING and ALIGNMENT in AlignDLD through various numbers of threads. The times of speedup in analyzing *A. Enron* dataset are reported in Figure 7. The results show the close-to-ideal scalability for SAMPLING when parallelized onto an increasing number of threads. Our unoptimized multithreaded CPU ALIGNMENT scales up less well in comparison, which we attribute to poor memory cache utilization. The score matrices are too large to fit into the cache for some alignments. The interaction between threads may evict reusable data from the cache. These operations in turn may cause cache misses. An optimized program should possess better data locality to minimize cache misses, and the optimization can be achieved in real-world detection products.

C. GPU Acceleration

We evaluate the performance of the most time-consuming ALIGNMENT procedure on a GPU with 448 cores grouped in 14 stream multiprocessors and a quad-core CPU. Times of speedup in detecting sensitive data of types `txt`, `png`, or `pdf`[8] against *A. Enron* or *D. MiscNet* traffic, respectively, are shown in Figure 8. The result shows that the GPU-accelerated ALIGNMENT achieves over 40 times of speedup over the CPU version on large content datasets (for both *A. Enron* and *D. MiscNet*). GPU speedup with *A. Enron* data is nearly 50 times of the CPU version.

Due to the limited bandwidth between CPU and GPU, data transfer is the bottleneck of our GPU implementation and dominates the execution time. A common strategy to solve the issue is to overlap data transfer and kernel execution or to batch the GPU input [61]. Another possible approach from the hardware perspective is to use a CPU-GPU integrated platform, such as AMD APU or Intel MIC, which benefits from the shared memory between CPU and GPU [62].

We report the throughput of ALIGNMENT in our GPU implementation under various parameters. Other procedures – that are faster than alignment – can be carried out in parallel with ALIGNMENT in real-world deployment. We randomly generate sensitive data pieces, 500 bytes for each, and run the detection against 500MB misc network traffic (*D. MiscNet*). The results in Table V show that we can achieve over 400Mbps throughput on a single GPU. This throughput is comparable to that of a moderate commercial firewall. More optimizations on data locality and memory usage can be performed in real-world detection products.

TABLE V
THROUGHPUT (IN Mbps) OF THE ALIGNMENT OPERATION ON GPU

Sensitive data size (KB)	250	500	1000	2500
Sampling rate				
0.03	426	218	110	44
0.12	23	11	5	2

D. Sampling Speedup

We measure the performance gain brought by sampling and compare the empirical results with the theoretical expectation. Measurements are performed on *A. Enron*, *C. HTTP*, and

[8]43KB txt data, 21KB png data, and 633KB pdf data.

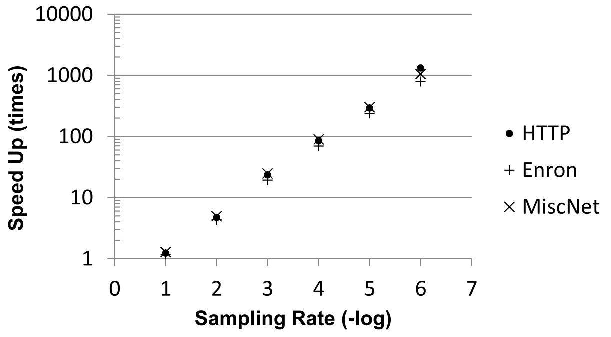


Fig. 9. Alignment speedup through sampling.

D. MiscNet datasets. Figure 9 shows the speedup of ALIGNMENT through different sampling rates α $(0.5, 0.25, 0.125, \ldots)$. $-\log_2 \alpha$ is shown on the X-axis. The well fitted lines (R^2 at 0.9988, 0.9977 and 0.9987) from the results have slope coefficients between 1.90 and 2.00, which confirms the α^2 speedup by our sampling design. We calculate the damping factor 0.33 from intercept coefficients of fitted lines.

VIII. CONCLUSIONS AND FUTURE WORK

We presented a content inspection technique for detecting leaks of sensitive information in the content of files or network traffic. Our detection approach is based on aligning two sampled sequences for similarity comparison. Our experimental results suggest that our alignment method is useful for detecting multiple common data leak scenarios. The parallel versions of our prototype provide substantial speedup and indicate high scalability of our design. For future work, we plan to explore data-movement tracking approaches for data leak prevention on a host.

ACKNOWLEDGMENT

The authors thank David Evans, Trent Jaeger, Jaeyeon Jung and Michael Wolfe for their suggestions and feedback on the paper.

REFERENCES

[1] X. Shu, J. Zhang, D. Yao, and W.-C. Feng, "Rapid and parallel content screening for detecting transformed data exposure," in *Proc. 3rd Int. Workshop Secur. Privacy Big Data (BigSecurity)*, Apr./May 2015, pp. 191–196.

[2] X. Shu, J. Zhang, D. Yao, and W.-C. Feng, "Rapid screening of transformed data leaks with efficient algorithms and parallel computing," in *Proc. 5th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, San Antonio, TX, USA, Mar. 2015, pp. 147–149.

[3] (Feb. 2015). *Data Breach QuickView: 2014 Data Breach Trends*. [Online]. Available: https://www.riskbasedsecurity.com/reports/2014-YEDataBreachQuickView.pdf, accessed Feb. 2015.

[4] Kaspersky Lab. (2014). *Global Corporate IT Security Risks*. [Online]. Available: http://media.kaspersky.com/en/business-security/Kaspersky_Global_IT_Security_Risks_Survey_report_Eng_final.pdf

[5] L. De Carli, R. Sommer, and S. Jha, "Beyond pattern matching: A concurrency model for stateful deep packet inspection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 1378–1390.

[6] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, Jun. 1975.

[7] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762–772, Oct. 1977.

[8] S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese, "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia," in *Proc. 3rd ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, 2007, pp. 155–164.

[9] L. Yang, R. Karim, V. Ganapathy, and R. Smith, "Improving NFA-based signature matching using ordered binary decision diagrams," in *Proc. 13th Int. Symp. Recent Adv. Intrusion Detect.*, Sep. 2010, pp. 58–78.

[10] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *Proc. 11th Annu. Symp. Combinat. Pattern Matching*, 2000, pp. 1–10.

[11] M. Cai, K. Hwang, Y.-K. Kwok, S. Song, and Y. Chen, "Collaborative Internet worm containment," *IEEE Security Privacy*, vol. 3, no. 3, pp. 25–33, May/Jun. 2005.

[12] J. Jang, D. Brumley, and S. Venkataraman, "BitShred: Feature hashing malware for scalable triage and semantic analysis," in *Proc. 18th ACM Conf. Comput. Commun. Secur. (CCS)*, 2011, pp. 309–320.

[13] K. Li, Z. Zhong, and L. Ramaswamy, "Privacy-aware collaborative spam filtering," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 725–739, May 2009.

[14] Symantec. (2015). *Symantec Data Loss Prevention*. [Online]. Available: http://www.symantec.com/data-loss-prevention, accessed Feb. 2015.

[15] X. Shu and D. Yao, "Data leak detection as a service," in *Proc. 8th Int. Conf. Secur. Privacy Commun. Netw. (SecureComm)*, Padua, Italy, Sep. 2012, pp. 222–240.

[16] D. Ficara, G. Antichi, A. Di Pietro, S. Giordano, G. Procissi, and F. Vitucci, "Sampling techniques to accelerate pattern matching in network intrusion detection systems," in *Proc. IEEE Int. Conf. Commun.*, May 2010, pp. 1–5.

[17] U. Vishkin, "Deterministic sampling—A new technique for fast pattern matching," in *Proc. 22nd Annu. ACM Symp. Theory Comput. (STOC)*, 1990, pp. 170–180.

[18] (2015). *Identity Finder*. [Online]. Available: http://www.identityfinder.com/, accessed Feb. 2015.

[19] Global Velocity Inc. (2015). *Cloud Data Security From the Inside Out—Global Velocity*. [Online]. Available: http://www.globalvelocity.com/, accessed Feb. 2015.

[20] GTB Technologies Inc. (2015). *GoCloudDLP*. [Online]. Available: http://www.goclouddlp.com/, accessed Feb. 2015.

[21] M. Roesch, "Snort—Lightweight intrusion detection for networks," in *Proc. 13th USENIX Conf. Syst. Admin. (LISA)*, 1999, pp. 229–238.

[22] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Proc. 7th Conf. USENIX Secur. Symp. (SSYM)*, vol. 7. 1998, p. 3.

[23] P.-C. Lin, Y.-D. Lin, Y.-C. Lai, and T.-H. Lee, "Using string matching for deep packet inspection," *Computer*, vol. 41, no. 4, pp. 23–28, Apr. 2008.

[24] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, "Deterministic memory-efficient string matching algorithms for intrusion detection," in *Proc. Int. Conf. Comput. Commun.*, vol. 4. Mar. 2004, pp. 2628–2639.

[25] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2006, pp. 339–350.

[26] S. Kumar, J. Turner, and J. Williams, "Advanced algorithms for fast and scalable deep packet inspection," in *Proc. 2nd ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, Dec. 2006, pp. 81–92.

[27] M. J. Atallah, F. Chyzak, and P. Dumas, "A randomized algorithm for approximate string matching," *Algorithmica*, vol. 29, no. 3, pp. 468–486, Mar. 2001.

[28] M. J. Atallah, E. Grigorescu, and Y. Wu, "A lower-variance randomized algorithm for approximate string matching," *Inf. Process. Lett.*, vol. 113, no. 18, pp. 690–692, Sep. 2013.

[29] M. Blanton, M. J. Atallah, K. B. Frikken, and Q. Malluhi, "Secure and efficient outsourcing of sequence comparisons," in *Proc. 17th Eur. Symp. Res. Comput. Secur.*, 2012, pp. 505–522.

[30] J. Jung, A. Sheth, B. Greenstein, D. Wetherall, G. Maganis, and T. Kohno, "Privacy oracle: A system for finding application leaks with black box differential testing," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2008, pp. 279–288.

[31] C. Kreibich and J. Crowcroft, "Efficient sequence alignment of network traffic," in *Proc. Internet Meas. Conf.*, 2006, pp. 307–312.

[32] S. E. Coull, F. Monrose, and M. Bailey, "On measuring the similarity of network hosts: Pitfalls, new metrics, and empirical analyses," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2011, pp. 1–16.

[33] S. E. Coull and B. K. Szymanski, "Sequence alignment for masquerade detection," *Comput. Statist. Data Anal.*, vol. 52, no. 8, pp. 4116–4131, Apr. 2008.

[34] H. A. Kholidy, F. Baiardi, and S. Hariri, "DDSGA: A data-driven semi-global alignment approach for detecting masquerade attacks," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 2, pp. 164–178, Mar./Apr. 2015.

[35] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *J. Molecular Biol.*, vol. 215, no. 3, pp. 403–410, Oct. 1990.

[36] R. Hoyle, S. Patil, D. White, J. Dawson, P. Whalen, and A. Kapadia, “Attire: Conveying information exposure through avatar apparel,” in *Proc. Conf. Comput. Supported Cooperat. Work Companion (CSCW)*, 2013, pp. 19–22.

[37] A. Nadkarni and W. Enck, “Preventing accidental data disclosure in modern operating systems,” in *Proc. 20th ACM Conf. Comput. Commun. Secur.*, 2013, pp. 1029–1042.

[38] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, “AppIntent: Analyzing sensitive data transmission in Android for privacy leakage detection,” in *Proc. 20th ACM Conf. Comput. Commun. Secur.*, 2013, pp. 1043–1054.

[39] J. Croft and M. Caesar, “Towards practical avoidance of information leakage in enterprise networks,” in *Proc. 6th USENIX Conf. Hot Topics Secur. (HotSec)*, 2011, p. 7.

[40] V. P. Kemerlis, V. Pappas, G. Portokalidis, and A. D. Keromytis, “iLeak: A lightweight system for detecting inadvertent information leaks,” in *Proc. 6th Eur. Conf. Comput. Netw. Defense*, Oct. 2010, pp. 21–28.

[41] E. Bertino and G. Ghinita, “Towards mechanisms for detection and prevention of data exfiltration by insiders: Keynote talk paper,” in *Proc. 6th ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, 2011, pp. 10–19.

[42] P. Papadimitriou and H. Garcia-Molina, “Data leakage detection,” *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 1, pp. 51–63, Jan. 2011.

[43] D. Lin and A. Squicciarini, “Data protection models for service provisioning in the cloud,” in *Proc. 15th ACM Symp. Access Control Models Technol.*, 2010, pp. 183–192.

[44] A. Squicciarini, S. Sundareswaran, and D. Lin, “Preventing information leakage from indexing in the cloud,” in *Proc. 3rd IEEE Int. Conf. Cloud Comput.*, Jul. 2010, pp. 188–195.

[45] X. Shu, D. Yao, and E. Bertino, “Privacy-preserving detection of sensitive data exposure,” *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 5, pp. 1092–1103, May 2015.

[46] F. Liu, X. Shu, D. Yao, and A. R. Butt, “Privacy-preserving scanning of big content for sensitive data exposure with MapReduce,” in *Proc. 5th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2015, pp. 195–206.

[47] Y. Jang, S. P. Chung, B. D. Payne, and W. Lee, “Gyrus: A framework for user-intent monitoring of text-based networked applications,” in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 79–93.

[48] K. Borders and A. Prakash, “Quantifying information leaks in outbound Web traffic,” in *Proc. 30th IEEE Symp. Secur. Privacy (SP)*, May 2009, pp. 129–140.

[49] S. Jha, L. Kruger, and V. Shmatikov, “Towards practical privacy for genomic computation,” in *Proc. IEEE Symp. Secur. Privacy*, May 2008, pp. 216–230.

[50] V. Polyanovsky, M. A. Roytberg, and V. G. Tumanyan, “Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences,” *Algorithms Molecular Biol.*, vol. 6, no. 1, p. 25, 2011.

[51] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, “Min-wise independent permutations,” *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 630–659, Jun. 2000.

[52] P. K. Agarwal and R. Sharathkumar, “Streaming algorithms for extent problems in high dimensions,” in *Proc. 21st Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*, 2010, pp. 1481–1489.

[53] D. Feldman, M. Monemizadeh, C. Sohler, and D. P. Woodruff, “Coresets and sketches for high dimensional subspace approximation problems,” in *Proc. 21st Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*, 2010, pp. 630–649.

[54] M. O. Rabin, “Fingerprinting by random polynomials,” Center Res. Comput. Technol., Harvard Univ., Cambridge, MA, USA, Tech. Rep. 15-81, 1981.

[55] T. F. Smith and M. S. Waterman, “Identification of common molecular subsequences,” *J. Molecular Biol.*, vol. 147, no. 1, pp. 195–197, Mar. 1981.

[56] C. Kalyan and K. Chandrasekaran, “Information leak detection in financial e-mails using mail pattern analysis under partial information,” in *Proc. 7th WSEAS Int. Conf. Appl. Informat. Commun. (AIC)*, vol. 7. 2007, pp. 104–109.

[57] C. Wüest and E. Florio, “Firefox and malware: When browsers attack,” Symantec Corp., Mountain View, CA, USA, White Paper, Oct. 2009. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/firefox_and_malware.pdf, accessed Feb. 2015.

[58] W. Liu, B. Schmidt, G. Voss, A. Schroder, and W. Muller-Wittig, “Bio-sequence database scanning on a GPU,” in *Proc. 20th Int. Parallel Distrib. Process. Symp.*, Apr. 2006, pp. 1–8.

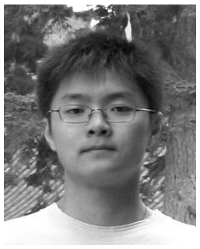
[59] S. A. Manavski and G. Valle, “CUDA compatible GPU cards as efficient hardware accelerators for Smith–Waterman sequence alignment,” *BMC Bioinformat.*, vol. 9, no. 2, p. 10, 2008.

[60] Y. Liu, D. L. Maskell, and B. Schmidt, “CUDASW++: Optimizing Smith–Waterman sequence database searches for CUDA-enabled graphics processing units,” *BMC Res. Notes*, vol. 2, no. 1, p. 73, May 2009.

[61] M. A. Jamshed *et al.*, “Kargus: A highly-scalable software-based intrusion detection system,” in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 317–328.

[62] K. Lee, H. Lin, and W.-C. Feng, “Performance characterization of data-intensive kernels on AMD fusion architectures,” *Comput. Sci.-Res. Develop.*, vol. 28, no. 2, pp. 175–184, May 2013.

Xiaokui Shu received the bachelor’s degree in information security from the University of Science and Technology of China (USTC), in 2010. He is currently pursuing the Ph.D. degree in computer science with Virginia Tech. His research focuses on system and network security. Being the top student in the class, he graduated with the Guo Moruo Award and was awarded the SIMIT Chinese Academy of Sciences Scholarship in 2008. He succeeded at his first penetration test at USTC and won the first prize in Virginia Tech Inaugural Cyber Security Summit Competition in 2011.



Jing Zhang received the B.S. degree from the College of Computer, National University of Defense Technology (NUDT), China. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Virginia Tech. His research interests are high performance computing, bioinformatics, and cloud computing. His major research topics focus on the optimization of irregular applications on heterogeneous platforms.

Danfeng (Daphne) Yao received the Ph.D. degree in computer science from Brown University, in 2007. She is an Associate Professor and an L-3 Faculty Fellow with the Department of Computer Science, Virginia Tech, Blacksburg. She received the NSF CAREER Award in 2010 for her work on human-behavior driven malware detection, and most recently ARO Young Investigator Award for her semantic reasoning for mission-oriented security work in 2014. She received the Outstanding New Assistant Professor Award from the Virginia Tech College of Engineering in 2012. She has received several best paper awards (ICICS ’06, CollaborateCom’09, and ICNP’12). She was given the Award for Technological Innovation from Brown University in 2006. She held a U.S. patent for her anomaly detection technologies. She is an Associate Editor of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING. She serves as a PC Member in numerous computer security conferences, including ACM CCS.

Wu-Chun Feng received B.S. (Hons.) degrees in computer engineering and music and the M.S. degree in computer engineering from Penn State University, in 1988 and 1990, respectively, and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, in 1996. He is a Professor with the Department of Computer Science, Virginia Tech (VT). He leads the Synergy Lab and serves as a Site Codirector of the NSF Center for High-Performance Reconfigurable Computing with VT. His research interests encompass a broad range of topics in efficient parallel computing, including high-performance computing and networking, energy-efficient (or green) supercomputing, accelerator-based computing, cloud computing, grid computing, bioinformatics, and computer science pedagogy for K-12.