THE JOURNAL OF DIGITAL FORENSICS, SECURITY AND LAW

Journal of Digital Forensics, Security and Law

Volume 9 | Number 2 Article 3

2014

File Detection on Network Traffic Using Approximate Matching

Frank Breitinger
University of New Haven

Ibrahim Baggili
University of New Haven

Recommended Citation

Breitinger, Frank and Baggili, Ibrahim (2014) "File Detection on Network Traffic Using Approximate Matching," *Journal of Digital Forensics, Security and Law*: Vol. 9 : No. 2 , Article 3.
DOI: https://doi.org/10.15394/jdfsl.2014.1168
Available at: https://commons.erau.edu/jdfsl/vol9/iss2/3

FILE DETECTION ON NETWORK TRAFFIC USING APPROXIMATE MATCHING

Frank Breitinger and Ibrahim Baggili
University of New Haven
300 Boston Post Rd, New Haven, CT, 06511
{FBreitinger,IBaggili}@newhaven.edu

ABSTRACT

In recent years, Internet technologies changed enormously and allow faster Internet connections, higher data rates and mobile usage. Hence, it is possible to send huge amounts of data / files easily which is often used by insiders or attackers to steal intellectual property. As a consequence, data leakage prevention systems (DLPS) have been developed which analyze network traffic and alert in case of a data leak. Although the overall concepts of the detection techniques are known, the systems are mostly closed and commercial.
Within this paper we present a new technique for network traffic analysis based on approximate matching (a.k.a fuzzy hashing) which is very common in digital forensics to correlate similar files. This paper demonstrates how to optimize and apply them on single network packets. Our contribution is a straightforward concept which does not need a comprehensive configuration: hash the file and store the digest in the database. Within our experiments we obtained false positive rates between 10^{-4} and 10^{-5} and an algorithm throughput of over 650 Mbit/s.

Keywords: Approximate matching, Bloom filter, mrsh-v2, data loss prevention, network traffic analysis

1. INTRODUCTION

The opportunities offered by the Internet are changing our lives. New technologies allow faster connectivity, higher data rates and mobile usage. As a result, companies now provide video-on-demand, cloud computing and online storage. Furthermore, e-mail is one of the most important communication mechanisms (Radicati & Hoang, 2011) superseding all other forms of communication. This is coupled with the increase in the number of Internet users and the volume of data (IEEE 802.3 Ethernet Working Group, 2012).

With the advent of new technologies comes new challenges, such as virus attacks, spam mail and data leakage, especially for companies which maintain sensitive data. As *data is valuable* – for legitimate and criminal purposes – there has been a constant increase in data breach incidents since 2009. For instance, according to the Open Security Foundation (OSF), 1605 incidents were reported last year which is an increase of over 45% compared to 2011. We list below some major data breach incidents in the last few years[1]:

- Attackers were successful in compromising 77 million records at the Sony Corporation in April 2011.
- LinkedIn was allegedly compromised, with 6.5 million password hashes stolen in June 2012.

In order to secure their networks, companies install intrusion detection and prevention sys-

[1] `http://datalossdb.org/`{index/largest, statistics} last accessed 2014-May-20

tems (IDPS), firewalls and virus scanners. However, only two-thirds of all data breaches are the result of hacking attacks. According to OSF, 36%[1] of all recorded incidents involve insiders, as shown in the popular Edward Snowden case. This has inspired research and practice in data leakage prevention. Vendors have now come up with data leakage prevention systems (DLPS) which analyze the network stream. These tools are often based on deep packet inspection (DPI, see Sec. 2.1) which means that network packets are parsed.

In another sub-discipline, the digital forensics community has pursued research in *approximate matching* (a.k.a fuzzy hashing or similarity hashing) allowing investigators to find similar files. This area of research has gained more popularity over the last couple of years (Garfinkel, 2010). Approximate matching is a rather new working field but has been proven to be useful for similar input detection (e.g., different versions of a file) or embedded objects detection (e.g., a jpg within a Word document). Furthermore, approximate matching can be used for fragment detection where a fragment is a small piece of a file (Breitinger, Guttman, McCarrin, & Roussev, 2014).

Combining both the areas of DLP and approximate matching, in this work, we present a novel technique using approximate matching to detect files in network traffic. Compared to existing techniques, our proposed approach is straight forward and does not need comprehensive configuration. It can be easily deployed and maintained since only fingerprints (a.k.a. similarity digest) are required. Our approach does not require machine learning, or rule generation. The main contribution is to demonstrate that it is possible to use approximate matching on network traffic by changing the algorithms slightly although algorithms where never designed to handle such small pieces. To the best of our knowledge, this the first paper describing a technique for file identification using approximate matching in network traffic.

The rest of the paper is organized as follows: In Sec. 2 we present the state of the art in DPI, DLPS, and approximate matching. Next, we explain the problem and summarize our solution in Sec. 3. In Sec. 4 we describe our testing methodology before we present our experimental results in Sec. 5. The last two sections are the ongoing research, future work as well as the conclusion.

2. FOUNDATIONS & RELATED WORK

This section explains the foundations and presents related literature. In Sec. 2.1 we discuss the different levels of network packet analysis. The current techniques for data leakage prevention systems are explained in Sec. 2.2. Bloom filters are an essential concept of that work and is presented in Sec. 2.3. Sec. 2.4 introduces approximate matching in general and highlights two algorithms called `mrsh-v2` and `sdhash`. The last section presents a concept for an efficient database lookup for similarity digests.

2.1 Packet Inspection

Static packet inspection is the most obvious mechanism. It treats each packet as a 'standalone' packet and decisions are made based on information contained in packet headers. Rules can be created based on destination IPs, source IPs, ports and protocols. This was then advanced to *stateful packet inspection* (SPI) which "shares many of the inherent limitations of the static packet filter with one important difference: state awareness. [...] The typical dynamic packet filter is aware of the difference between a new and an established connection" (Tipton & Krause, 2003, p77++). Hence, a device such as a firewall maintains a table to be aware of the connections.

The next step in the evolution was *deep packet inspection* (DPI). Besides analyzing packet headers, it examines the actual payload. "DPI engines parse the entire IP packet, and make forwarding decisions by means of rule-based logic that is based on signature or regular expression matching. That is, they compare the data within a packet payload to a database of predefined attack signatures (a string of bytes).

[... However,] searching through the payload for multiple string patterns within the datastream is a computationally expensive task" (Proter, 2010).

2.2 Techniques for Data Leakage Prevention

According to (Lawton, 2008), an organization's data can be classified into three states: a) *Data in Motion* (DIM): data in the process of being transmitted over the network, b) *Data at Rest* (DAR): data in file systems, FTP servers, and c) *Data in Use* (DIU): data at a network endpoint, like a desktop computer or a USB device. *Data Loss Prevention Systems* (DLPS) were created to identify sensitive information by content in DIM, DAR or DIU, and to prevent its leakage outside of an organization

The main idea behind these systems is to perform deep packet inspection (DPI) for automatic network analysis. In other words, it tries to detect protected information or files within network traffic.

According to (SANS Institute, 2010), the following approaches are used in DPI:

Regular expressions are effective in case of structured data like credit card numbers or social security numbers, however, they cannot be used efficiently for file identification.

Database fingerprint analyzes network packets for exact strings. Instead of looking for all credit card numbers, one may only look for specific ones. In addition, it is very common to identify documents that are tagged with buzzwords like 'confidential' or 'secret', however, this approach fails if buzzwords are omitted or if the data is binary.

Exact file matching uses hash functions to find exact matches which is file type independent. However, it is trivial to evade (alter at any position). Another drawback is that all packets need to be captured, the files needs to be reconstructed and then hashed.

Statistical analysis is based on machine learning approaches. This approach includes comprehensive training in the beginning and only works reliably for with the availability of a large training dataset. Furthermore, if new protected data is added, the training step needs to be restarted. Lastly, this approach is prone to false positives and false negatives.

Besides the four aforementioned techniques, there is *conceptual/lexicon* which is a combination of rules, directories and other analysis methods[2] and *categories* which is also based on rules and dictionaries.

The most promising approach for automatic file identification is *partial document matching* which looks for a complete or partial match (e.g., a few sentences of a document) of protected content. This technique often uses a rolling hash to compare documents against network packet payloads. Unfortunately, we were not able to find seminal published research on this approach as most DLPS are commercial and closed source.

2.3 Bloom Filter

Bloom filters (Bloom, 1970) are commonly used to represent elements of a finite set S. A Bloom filter is an array of m bits initially all set to zero. In order to 'insert' an element $s \in S$ into the filter, k independent hash functions are needed where each hash function h outputs a value between 0 and $m-1$. Next, s is hashed by all hash functions h. The bits of the Bloom filter at the positions $h_0(s), h_1(s), \ldots h_{k-1}(s)$ are set to one.

To answer the question if s' is in S, we compute $h_0(s'), h_1(s'), \ldots h_{k-1}(s')$ and analyze if the bits at the corresponding positions in the Bloom filter are set to one. If this holds, s' is assumed to be in S, however, we may be wrong as the bits may be set to one by different elements from S. Hence, Bloom filters suffer from a non-trivial false positive rate. Otherwise, if at least one bit is set to zero, we know that $s' \notin S$. It is obvious that the false negative rate is equal to zero.

In case of uniformly distributed data, the probability that a certain bit is set to 1 during the insertion of an element is $1/m$, i.e., the probability that a bit is still 0 is $1 - 1/m$. After inserting n elements into the Bloom filter, the probability of a given bit position to be 1 is

[2] As this technique is very complex, we would like to refer the reader to (SANS Institute, 2010, p9).

$1 - (1 - 1/m)^{k \cdot n}$. In order to have a false positive, all k array positions need to be set to one. Hence, the probability p for a false positive is

$$p = \left[1 - (1 - 1/m)^{k \cdot n}\right]^k \approx (1 - e^{-kn/m})^k. \quad (1)$$

2.4 Approximate Matching

Approximate matching is a rather new area and probably had a breakthrough in 2006 with an algorithm called context triggered piecewise hashing (Kornblum, 2006). Since then, a few more algorithms were presented which were summarized in (Breitinger, Liu, et al., 2013). A main remark of the authors is that currently only two algorithms have the ability to correlate small fragments and the original file–`sdhash` (Roussev, 2010) and `mrsh-v2` (Breitinger & Baier, 2012).

A brief runtime comparison showed that `mrsh-v2` outperforms `sdhash` by a factor of 8 (Breitinger, Liu, et al., 2013). Hence, our research focuses on `mrsh-v2` while we use `sdhash` for an initial verification.

2.4.1 sdhash

This algorithm was proposed in 2010 by Roussev (Roussev, 2010) and attempts to pick characteristic features for each object that are unlikely to be appear by chance in other objects based on results from an empirical study. In the baseline implementation, each feature is hashed with SHA-1 (Gallagher & Director, 1995) and inserted into a Bloom filter (Bloom, 1970) (details are given at the end of this section) where a feature is a sequence of 64 bytes. The similarity digest of the data object is a sequence of 2048-bit filters, each of which represents approximately 10 KiB[3] of the original data, on average.

2.4.2 mrsh-v2

The algorithm was proposed by Breitinger & Baier (Breitinger & Baier, 2012) in 2012 and is based on multi resolution similarity hashing (Roussev, III, & Marziale, 2007) and context triggered piecewise hashing (a.k.a. `ssdeep`, (Kornblum, 2006)). The overall idea is quite simple: divide an input into chunks using a pseudo random function, hash each chunk and insert the chunk-hashes into a Bloom filter.

2.4.3 Similarity Digest

To insert a chunk-hash[4] into a $m = 2048 = 2^{11}$ bit Bloom filter, the algorithms take 55 bits from the chunk-hash, splits it into $k = 5$ sub-hashes of 11 bits and sets the corresponding bit. For instance, the sub-hash 000 1000 $1100_2 = 8C_{16} = 140_{10}$ will set bit 140 in the Bloom filter. The implementations have an upper limit of chunks per Bloom filter. If this limit is reached, a new Bloom filter is created. Hence, the final similarity digest is a sequence of Bloom filters. To identify the similarity between two digests, all Bloom filters of fingerprint a are compared against all Bloom filters of fingerprint b with respect to the Hamming distance as metric[5].

2.5 Improving the Database Lookup Complexity for Approximate Matching Algorithms

In (Breitinger, Baier, & White, 2014) the authors motivated, discussed and evaluated a technique to overcome the lookup complexity for similarity digests. Let x denote the amount of digests in a database, then the complexity for a single digest is $O(x)$. In other words, the digest has to be compared against all digests in the database. Regarding network packets, this means that each packet needs to be compared against the complete database; each packet has a complexity of $O(x)$.

Instead of having multiple small Bloom filters for a single input, the authors suggest to insert all files into a huge Bloom filter. Thus, the lookup complexity per chunk is $O(1)$.

Having only one Bloom filter comes with two disadvantages. Due to efficiency reasons, the Bloom filter has to be in memory completely

[3]Note, KiB is kibibyte which is different to KB which is kilobyte. While kibi has a base of 1024, kilo uses the base 1000.

[4]Note, with respect to `sdhash` this is actually a feature hash, however, we use both terms as synonyms and use chunk from here on.

[5]The actual comparison is not important for the remainder of this paper.

and therefore one limiting factor is the physical available memory. However, their approach only requires 32 MiB of memory to monitor 2 GiB of data; 100 GiB requires about 2 GiB of memory. We claim that in case of most mid-size businesses, the data will not exceed 100 GiB when dealing with office documents, source-code and blueprints (images).

On the other hand, our approach is a *packet-against-set* comparison. That is, the answer of a packet-query is either yes (there is a similar file to that packet in the set) or no. Nevertheless, we claim that in case of data leakage prevention or virus identification this is sufficient.

A detailed description of the exact procedure is beyond the scope of this paper and hence we will not discuss the parameters and default values in detail.

3. WORK FLOW AND IMPLEMENTATION

The upcoming subsections provide an overview of our solution. We explain the exact procedure, explain the implementation and additional benefits for our approach.

3.1 Procedure Overview

Besides lookup complexity, algorithms have to be optimized to handle fragments of MTU size. In contrast to their original purpose (handle inputs of kilobytes or megabytes), they have to handle fragments of approximately 1500 bytes.

In order to handle network packets, algorithms need finer granularity. Thus, we reduced the blocksize from the original 160 bytes to 64 bytes which results in more chunks. However, finer granularity increases the chance for false positives as the decision is based on less data. That is why we deploy a filter mechanism that eliminates non-relevant chunks from consideration, e.g., long runs of zero.

Therefore, the overall procedure requires two phases:

1. *Database generation:* Divide file into chunks, filter out non-relevant chunks, hash chunks and fill the Bloom filter.
2. *Network packet analysis:* Divide packet into chunks, hash chunks and compare against the Bloom filter.

Note, the filter mechanism is only necessary for creating the Bloom filter. During the network analysis phase the packets are hashed and only compared. To sum it up, no matter how complex the filter mechanisms are, the performance of the network analysis is not influenced.

3.2 Result Presentation

Conventional algorithms print a match score between 0 and 100 to show the communality between two files. In the current approach we decided to have a binary decision, either a packet is in the filter or not. If at least $r \geq 8$ consecutive features of a packet are found in the Bloom filter, `mrsh-net` detected a match and outputs

```
packet: 12 of 18 (longest run: 10)
```

which means that a packet was found in the underlying Bloom filter. In total, the packet consisted of 18 chunks where 12 were found. The longest run was 10. As we changed the blocksize from 160 to 64 bytes, 8 consecutive features meant, that $8 \cdot 64 = 512$ bytes of the packet are found.

3.3 Chunk Filter

As mentioned in Sec. 3.1, not all chunks are of the same quality. In order to decide if it is an important or an unimportant chunk, we consider two values called entropy and randomness. While the former one is based on the well-known Shannon entropy (Shannon, January 2001), randomness considers two neighboring bytes. If two consecutive bytes are equal or differ by one, then anti-randomness R is increased. More formally, let a chunk of length L be given where B_i denotes the byte at position i. Then, anti-randomness is calculated as follows:

$$R = \sum_{i=0}^{L-2} ar(B_i) \text{ where } ar(B_i) = \begin{cases} 1, & \text{if } |B_i - B_{i+1}| \leq 1 \\ 0, & \text{else} \end{cases}$$

A chunk has sufficient randomness if $R \cdot 2 < L$ (we identified this value by best practice working on 100 random files).

3.4 Implementation Details

To validate our findings, we released a prototype called `mrsh-net` which is basically a modification of the latest `mrsh-v2` version. Currently there is only one branch, thus a lot of testing-code-pieces are included making the code harder to understand. For instance, we implemented counting Bloom filters which were necessary for testing purposes and still an ongoing research project (see Sec. 6). The prototype can be downloaded on from website[6].

3.4.1 Commandline Arguments

-g generates a Bloom filter from DIR and prints it to std. Usage:
`./mrshnet -g DIR/* > dbFile`.

-i reads Bloom filter BF-FILE and compares DIR/FILE against it. Usage:
`./mrshnet -i BF-FILE FILE/DIR/*`.

-f generates the 'false positive' matches for a list of files.
`./mrshnet -f DIR/*` .

-e sets the minimum entropy a chunks needs to have. Usage:
`./mrshnet -e 2.8 -g DIR/* > dbFile`.

-t excludes a file type for the -f option.

'False positive' means that all files are added to the Bloom filter, next file f is removed (we implemented counting Bloom filters for testing purposes) and finally f is compared against the filter. If combined with -t, the filter excludes all files of -t type and compares them against it. In both cases matches are printed to stdio.

3.5 Additional Benefits

One aspect which might not be immediately obvious is privacy. Due to the usage of Bloom filters, the monitored data is stored in a preimage resistant format (also we use the non-cryptographic hash function FNV). Hence, it is possible to maintain sensitive data at a central point, fill the Bloom filter and distribute it with no information leaks. For instance, anti virus vendors may provide a database containing the newest malware.

Another point is that Bloom filters can be easily combined with each other by ORing both filters which allows us to update filters. In the case of counting Bloom filters, one may even remove elements.

4. TEST METHODOLOGY

All tests run on 'simulated' network traffic where simulated means that all files are split into 1460 bytes sequences. This size results from the MTU size of 1500 bytes minus 20 bytes IP header and minus 20 bytes TCP header. On real network traffic, it is easily possible to skip 40 bytes in the beginning and thus we claim this simulates a real life circumstances. Our experiments are divided into two parts: synthetic and real-world data.

4.1 Throughput

A fundamental property of network analysis tools is throughput – huge delays are not acceptable. According to Sec. 3.1, the throughput considers phase 2. We neglect the database generation process as it is independent from the network analysis.

Our assessment is based on our C-implementation on a usual workstation. We capture the time for processing a given set of files, i.e., reading, hashing and comparing. Note, there is neither a hardware implementation of our algorithm nor are we programing experts. However, there exist ideas how to build high-throughput hardware implementations for Bloom filters e.g., (Dharmapurikar, Krishnamurthy, Sproull, & Lockwood, 2003).

4.2 Random Data

This controlled test utilizes `/dev/urandom` to create a pseudo random file set. Thus, it is possible to distinguish between true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). In order to have a real life distribution of file sizes, we took the t5-corpus (more details see Sec. 4.4) as a model and created 4457 files with identical sizes which is in

[6] `http://wp1187348.server-he.de/z_downloads/mrsh-net.zip`; anonymous for review.

total 1.78 GiB.

To determine the detection rates, we did the following tests:

TP: Fill all files from the corpus to the Bloom filter. Next, we compare all packets against the Bloom filter. All identified packets are true positive. (This can be solved using the `-g` and `-i` options).

TN: All files are added to the Bloom filter. Next, we remove file f from the Bloom filter and compare f against it (solved by `-f` option).

FP: 1-TN.

FN: 1-TP.

As approximate matching algorithms work on random data (Breitinger, Stivaktakis, & Roussev, 2013), we expect a TP rate close to 100% and a TN rate of exactly 100%. The TP rate will most certainly differ from 100 (slightly) as the last packet of a file might be too small and does not produce a longest run of $r \geq 8$.

4.3 Similarity

Before describing the test methodology for real data, we note our definition of similar and dissimilar:

TP: means that two files share a significant and interesting amount of data, e.g., same text passages, pictures or copyright information.

FP: means that two files have nothing in common or only unrelated information, e.g., file headers in common, but they are matched.

TN: means that two files have nothing in common.

FN: means that two files are not matched although they share significant and interesting data.

Thus, it is necessary to analyze the general communality between files, i.e., which data is file type specific and not related to the actual content. Possible samples are common headers, long runs of zeros, or file structure information. This is the input for creating useful filters.

Since there is no ground truth available, it is hard to categorize a match. To classify the genuiness (true positive + false positives), it is necessary to assess all obtained matches. In order to verify the impostors (true negative + false negative) we have to evaluate the whole corpus. Thus, we mainly focus on true positives and false positives for the remainder of this paper.

Recall, the output of our procedure is not an exact match but a statement that there is a similar file in the underlying set. Thus, it is necessary to compare all positive matches against the underlying file set. In order to handle this, we apply `sdhash` to identify 'possible true positives' which then are partly inspected manually.

4.4 Real World Data

For the real world data test we choose the *t5-corpus*[7] (Roussev, 2011) which contains 4457 files having a total size of 1.78 GiB. Thus, the average file size is almost 420 KiB and the file type distribution is given in Table 1. The t5-corpus is a subset of the govdocs[8] which was obtained by crawling web servers in the .gov domain. Due to the gathering process, we anticipate to have related files in the corpus.

Table 1 Statistics of t5-Corpus

jpg	gif	doc	xls	ppt	pdf	text	html
362	69	533	250	368	1073	711	1093

4.4.1 Result Presentation

When assessing our results we consider the packet and the file levels. The *packet level* describes the relation between all sent packets and matched / non-matched ones which is especially important for false positives. However, with respect to true positives, we suggest the *file level* which requires that at least one packet matches.

[7] `http://roussev.net/t5/` (last accessed 2014-May-20).

[8] `http://digitalcorpora.org/corpora/files` (last accessed 2014-May-20).

For instance, two large Microsoft office documents that share only a single graphic will not have many packets in common but are similar and the graphic is monitored content.

4.4.2 Cross Matching File Types

The first test examines the detection behavior among different file types. Let Y denote the file type, and let SET_Y be all files of type Y. Then, DB_{-Y} is a database that contains all files of $t5 \backslash SET_Y$, i.e., all files except the files of type Y. Furthermore, let our concept be denoted by $mrshnet(SET, DB)$ which is a function that returns all files in SET that matches the database DB. According to that, cross matching runs $S = mrshnet(SET_Y, DB_{-Y})$ for all file types Y. The output of a run is a set S which contains all cross matched files.

To distinguish between TP and FP, we compared SET_Y and $t5 \backslash SET_Y$ by `sdhash` and received a list of possible matches. Next, we manually proofed the similarity starting with the best matches, i.e., if a file is matched two files, we consider the higher score first.

For instance, set $Y = doc$. Then, all *.docs files are compared against the DB_{-doc}. The result could be a set like $S = \{f1.doc, f2.doc, f3.doc\}$ which serves as input for `sdhash`. Here, the output is a list like

```
file1.doc matches fileA.ppt (50)
file2.doc matches fileB.xls (10)
...
```

According to this list, we picked matches and compared them manually, i.e., open file1.doc and fileA.ppt and compare them. The selection process mostly focused on borderline matches. For instance, two files yielding a `sdhash` score of 50 are most likely similar where two files having a score of 5 are more critical.

The motivation for this test is that the expected number genuiness should be small. We assume hits between Microsoft office documents or between html and text.

4.4.3 True Positives

We have introduced techniques to minimize the chance of false positive called entropy and randomness. In general, these approaches filter out chunks in advance so that we come closer to random data and do make decisions based on long runs of zeros.

Within this test we studied the impact of different configurations with respect to the true positive rate. Also chunks are filtered out, there should be a high true positive rate on both levels, file level and packet level.

4.4.4 Mixed File Types

For this test we randomly selected 100 files out of the t5-corpus which serve as 'monitored files' and hashed to DB. The remaining 4357 files are used to produce the traffic. In contrast to cross matching, there are also comparisons among equal file types.

5. EXPERIMENTAL RESULTS & ASSESSMENT

For our testing, we used the configuration a Bloom filter with $m = 2^{28}$ bits (32 MiB) and a blocksize of 64 bytes, i.e., the approximate length of a chunk.

5.1 Throughput

To determine the throughput we processed the t5-corpus which has exactly 1823.11 MiB (1.78 GiB). The measured result includes the time for reading the input from disk, splitting it into packets, processing (hashing) it and performing the Bloom filter comparison. The time was measured by the Linux `time`-command where we selected the user time: 23.474 s. Overall, this comes to $^{1823.11\ \text{MiB}}/_{23.474\ \text{s}} = 77.67$ MiB/s which corresponds $77.67\ \text{MiB} \cdot 2^{20} \cdot 8 = 651,501,914$ bits/s – approximately 650 Mbit/s.

The test was performed on a 2 GHz Intel Core i7 CPU, single threaded. However, the approached allows for easy parallelism of the complete approach without any synchronization. We only need to hash packets and compare them to the Bloom filter.

5.2 Detection Rates on Random Traffic

The results are as expected. On the packet level, the true positive rate is at 99.6% and the true

negative rate is at 100.0%. The few false negative result from the length of the last packet which could be very small and thus contain less than 8 runs.

5.3 File Analysis and Similarity

This section explores the kind of basic-communality between files on the basis of the statements from Sec. 4.3. According to the work, we rate a match as a false positive if two underlying files only share common headers, long runs of zeros, or file structure information.

For this test we randomly selected files from the t5-corpus, correlated them by hand to ensure they are dissimilar and then compared them automatically with respect to the longest common subsequence. We decided for the longest common sequences as it is very similar to our approach – we require a minimum run length.

Basically we made two observations which yield long runs of non-significant data / information. In other words, possible false positives according to our definition.

The first kind of strings are a low entropy sequences which are very common throughout most types. Especially Microsoft office documents share long common low entropy sequences, but also types like gif or jpg. Note, some files can be regarded as containers, capable of embedding other types, e.g., Microsoft office documents can contain embedded pictures. Besides low entropy, we also discovered long runs of 'non-random' sequences which have a normal to high entropy.

We argue that these kind of chunks represent non-relevant information for the user, and thus packets comprised of these chunks can be safely neglected. To sum it up, it is reasonable to apply a filter mechanism as described in Sec. 3.3.

5.4 Detection Rates for Cross Matches

This section analyzes the cross matches and thus the behavior of $mrshnet(FILES_Y, DB_{\neg Y})$ where in total 1,311,576 packets have been sent.

A summary of our findings is given in Table 2 which shows the matches on the packet and the file level. For instance, row 1 states that in total 10,288 packets matched the database and belonged to 809 different files. Since the entropy is 0.0, all packets are considered. SDM is the `sdhash` match rate which expresses that `sdhash` also identified similarity when comparing $FILES_Y$ against the monitored set, e.g., 66.8% of 10,288 packets have a high probability to be a true positive. Rows highlighted with a star are outputs where randomness mode is on, i.e., packets marked as non-random are filtered out.

When rating the settings, we mostly considered the packet-file-ratio (abbreviated pf-ratio) which is the relation between packet matches and file matches. For instance, a high pf-ratio (close to 1) like for gif indicates that only single packets matched the database which is an indicator for less interesting results and vice versa. In the specific case of gif, all files included the same, low entropy sequence that matched the database.

Regarding the table, an entropy over 2.8 with the randomness test seems to be promising. In total there are 4510 packets where the SDM rate is 99.0%. It is obvious that the higher the required entropy, the less overall matches.

In the following we consider the true positive and false positives for row *2.8**. Totally, 4510 packets matched which belong to 77 different files. Out of this 4510 packets, 99.0% coincide with the results of `sdhash`. If we consider all `sdhash` hits as true positives, our approach yields a false positive rate per packet of $^{4510 \cdot 0.01}/_{1,311,576} = 3.44 \cdot 10^{-5}$.

To verify the `sdhash` results, we manually proofed some of the 52 file pairs (62.7% of the 77). Most matches where between text and html or between the Microsoft office documents. For instance, `mrsh-net` returned a match for 003344.doc which was correlated by `sdhash` to 003358.ppt. Examining these two files showed that both contain equal graphics. We found true positives only.

Next, we reviewed those files where `sdhash` could not find a similar file. First, it was conspicuous that almost all hits were caused by a

Table 2 Impact of Introducing an Entropy

e	jpg	gif	doc	xls	ppt	pdf	text	html	Packet level matches	Packet level SDM	File level matches	File level SDM
	(matches on packet level / matches on file level)											
0.0	133/53	17/17	2386/233	1249/117	5251/335	1118/29	48/8	86/17	10288	66.8%	809	21.5%
0.0*	0/0	5/5	1147/31	504/29	2984/227	932/14	38/6	77/12	5687	87.9%	324	31.8%
2.4	131/53	14/14	1573/118	674/60	3753/242	1018/53	40/8	85/17	7288	80.8%	538	32.3%
2.4*	0/0	0/0	1017/26	191/25	2654/73	930/14	38/6	77/12	4907	94.8%	156	47.4%
2.8	131/53	0/0	1136/35	148/5	2856/72	968/20	39/7	83/17	5361	96.7%	209	74.2%
2.8*	0/0	0/0	961/17	10/6	2497/22	927/14	38/6	77/12	4510	99.0%	77	67.5%
3.2	130/53	0/0	1124/35	148/5	2837/72	960/20	39/7	82/17	5320	96.7%	209	74.2%
3.2*	0/0	0/0	955/17	8/6	2479/22	925/14	38/6	76/12	4481	99.1%	75	67.5%

* Here the randomness check is turned on.

single packet which means files have only small communality. Since our implementation works deterministically, i.e., if A matches B, then also B matches A., we compared all remaining files to each other. In fact, we could detect file pairs that are similar. For instance, some html files where falsely identified as text files and had the cascading style sheet (css) included. Thus, these files have similar layout and tables. It is questionable if this is relevant or non-relevant information. Assume an internal webpage which contains secret business numbers. Then, an employee could download it, update it with the latest numbers and mail it to someone. Another example are translated documents which are still in the same layout. Actually, this depends on the scenario / use case and an admin needs to consider this when creating the database.

Nevertheless, even if we rate all these matches as false positives we obtain a false positive rate of $3.44 \cdot 10^{-5}$ for cross matching which is acceptable for a prototype.

5.5 True Positive Analysis

Deploying filter mechanism to reduce the false positive matches imply reducing true positives as well. Therefore, this section studies the impact of different settings to the true positive rate. The results are given in Table 3.

Equal to the random-test, there is no 100% true positive rate if both filtering mechanisms are turned off. The reason is the rolling hash. For instance, one undetected file was an almost empty Word document. The byte structure was composed of many zero runs and thus most packets contained zeros. The rolling hash is not able to determine enough trigger sequences and the run will not exceed the minimum.

Regarding our chosen setting 2.8*, about 1/4 of all packets are filtered out due to less entropy and randomness but we still detect 98.7% of all files. To conclude this section: monitored files should consist of a few kilobytes of data having an entropy over 2.8.

5.6 Detection Rates on Mixed File Types

This test simulates the case where the database contains mixed content. We randomly selected 100 files out of the t5-corpus which serve as the monitored files. The remaining files are used to produce the traffic. We capture all packets that provoke a match and performed an equal analysis then in the previous sections.

In total we sent 4357 files producing 1,284,738 packets. The settings are based on our previous findings: $e = 2.8*$. A summary of the results is presented in Table 4. Note, gif and jpg are not listed because there were no matches.

Row 1 shows the overall results. Overall 833 packets are genuines with a `sdhash` match rate of 97.6%. It was conspicuous that pdf had many matches. Studying the results shows that there are multiple false positives which seem due to overlapping pdf structure information. In contrast to Office documents and images, the structure information has a high entropy and over-

Table 3 True Positive Rates for Different Settings Sending 4457 Files / 1,311,576 Packets in Total

e	0.0	0.0*	2.4	2.4*	2.8	2.8*	3.2	3.2*
pkt. (%)	91.9	82.0	81.2	80.1	76.4	75.9	69.4	69.2
files (%)	99.9	98.9	99.3	98.9	98.8	98.7	98.0	98.0

* Here the randomness check is turned on.

Table 4 SDM Rates for All File Types

Packet level		File level		doc	xls	ppt	pdf	text	html
matches	SDM	matches	SDM						
883	97.6%	109	88.1%	10	1	11	59	2	26
731	99.5%	50	92.0%	10	1	11	/	2	26

Note, gif and jpg are not listed because there were no matches.

comes our filter out techniques. Nevertheless, there are also near duplicates, e.g., 000740.pdf and 000743.pdf.

Row 2 considers the results without pdf. The 731 packets come to a `sdhash` match rate of 99.5%. Our manual review showed that all files are true positives (most of them are similar webpages like 002224.html, 002758.html). Hence, the false positive rate here is 0.

Considering the packet-file-ratio of both rows shows that in average a match of pdf is due to $^{883-731}/_{59} \approx 2.5$ packets. Further investigation revealed that most matches are due to the same byte sequence. In other words, the database contained one sequence which a lot of pdfs included. However, we could not find interesting similarity among these matches.

If we classify all pdfs as false positives (which is an upper limit as we also had true positives), the overall false positive rate becomes $^{883-731}/_{1,284,738} = 1.1 \cdot 10^{-4}$.

6. ONGOING RESEARCH AND FUTURE WORK

The previous section demonstrated that there are sequences that overcome our current filter methods which is the basis for good detection rates. Hence, our current work aims at improving filtering techniques. One idea is to provide a list of sequences which are very common and should be ignored. In order to create such a list, one may study each file format intensively or use counting Bloom filters which is the focus at the moment.

Instead of analyzing the file types independently, we extend `mrsh-net` through a learning phase based on a file set. The files should share no significant similarity (according to our definition they are true negatives) and only have non-relevant communality, e.g., headers or share the company logo. Next, all files are hashed and inserted into a counting Bloom filter. In contrast to traditional Bloom filter, a counting one has a memory and knows how often a bit is set to one. By design, dissimilar files / content is supposed to set the different bits. Put it the other way round, bits that are set often are most likely to be structure information or headers. Hence, we can use counting Bloom filters to unset bits or block bits from setting.

A second idea is the consideration of more than one packet. This requires a connection table containing the information that userA has a connection to userB. Then, we easily can count the amount of database hits and react, e.g., the connection is interrupted if three hits appear.

Besides improving the filtering mechanism, we have to consider ways to evaluate impostors. However, we argue that an approach detecting not all true positives but having a false positive rate close to zero is already useful.

7. CONCLUSION

In this paper, we considered the challenge of similar file identification on network traffic using approximate matching (a.k.a. fuzzy hashing). We argued that data leakage is an increasing problem which requires straightforward open source tools and techniques to solve this problem.

We demonstrated that with some minor changes approximate matching can reliably work on network traffic and obtain good results. Our tests showed that random data can be detected perfectly while real-world data has an acceptable false positive rate of between 10^{-4} and 10^{-5}. The challenge of real-world data is the filtering of 'common substrings' which was solved using entropy and anti-randomness. However, this needs further research so that the considered packets are akin to random data.

Compared to existing algorithms, our approach is very simple and straightforward: hash the file and add it to the database. Furthermore, we used only open source technologies to achieve our goals.

REFERENCES

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*, 422–426.

Breitinger, F., & Baier, H. (2012, October). Similarity Preserving Hashing: Eligible Properties and a new Algorithm MRSH-v2. *4th ICST Conference on Digital Forensics & Cyber Crime (ICDF2C)*.

Breitinger, F., Baier, H., & White, D. (2014). On the database lookup problem of approximate matching. *1st Digital Forensics Research Conference EU (DFRWS-EU'14)*.

Breitinger, F., Guttman, B., McCarrin, M., & Roussev, V. (2014). Approximate matching: Definition and terminology. *NIST Special Publication 800-168*, *DRAFT*.

Breitinger, F., Liu, H., Winter, C., Baier, H., Rybalchenko, A., & Steinebach, M. (2013, Sept). Towards a process model for hash functions in digital forensics. *5th International Conference on Digital Forensics & Cyber Crime*.

Breitinger, F., Stivaktakis, G., & Roussev, V. (2013, Sept). Evaluating detection error trade-offs for bytewise approximate matching algorithms. *5th ICST Conference on Digital Forensics & Cyber Crime (ICDF2C)*.

Dharmapurikar, S., Krishnamurthy, P., Sproull, T., & Lockwood, J. (2003). Deep packet inspection using parallel bloom filters. In *High performance interconnects, 2003. proceedings. 11th symposium on* (p. 44-51).

Gallagher, P., & Director, A. (1995). *Secure Hash Standard (SHS)* (Tech. Rep.). National Institute of Standards and Technologies, Federal Information Processing Standards Publication 180-1.

Garfinkel, S. L. (2010, August). Digital forensics research: The next 10 years. *Digitial Investigation*, *7*, 64–73. Retrieved from `http://dx.doi.org/10.1016/j.diin.2010.05.009` doi: 10.1016/j.diin.2010.05.009

IEEE 802.3 Ethernet Working Group. (2012, July). *Industry Connections Ethernet Bandwidth Assessment* (Tech. Rep.). IEEE.

Kornblum, J. (2006, September). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, *3*, 91–97. Retrieved from `http://dx.doi.org/10.1016/j.diin.2006.06.015` doi: 10.1016/j.diin.2006.06.015

Lawton, G. (2008). New technology prevents data leakage. *Computer*, *41*(9), 14–17.

Proter, T. (2010, October). The perils of deep packet inspection. *Symantec.com*. Retrieved from `http://www.symantec.com/connect/articles/perils-deep-packet-inspection`

Radicati, S., & Hoang, Q. (2011, May). *Email*

statistics report, 2011-2015 (Tech. Rep.). 1900 Embarcadero road, suite 206., Palo Alto, CA, 94303: Radicati Group, INC. Retrieved from http://www.radicati.com/wp/wp-content/uploads/2011/05/Email-Statistics-Report-2011-2015-Executive-Summary.pdf

Roussev, V. (2010). Data fingerprinting with similarity digests. In K.-P. Chow & S. Shenoi (Eds.), *Advances in digital forensics vi* (Vol. 337, pp. 207–226). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-15506-2_15 doi: 10.1007/978-3-642-15506-2_15

Roussev, V. (2011, August). An evaluation of forensic similarity hashes. *Digital Investigation*, *8*, 34–41. Retrieved from http://dx.doi.org/10.1016/j.diin.2011.05.005 doi: 10.1016/j.diin.2011.05.005

Roussev, V., III, G. G. R., & Marziale, L. (2007, September). Multi-resolution similarity hashing. *Digital Investigation*, *4*, 105–113. doi: 10.1016/j.diin.2007.06.011

SANS Institute. (2010). *Understanding and selecting a data loss prevention solution.* Securosis, L.L.C.

Shannon, C. E. (January 2001, January). A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3–55.

Tipton, H. F., & Krause, M. (2003). *Information security management handbook* (5th ed.). Auerbach Publications.

