SplitScreen: Enabling Efficient, Distributed Malware Detection

Sang Kil Cha[†], Iulian Moraru[‡], Jiyong Jang[†], John Truelove[*], David Brumley[‡], David G. Andersen[‡]
Carnegie Mellon University, Pittsburgh, PA
[†] {*sangkilc, jiyongj*}*@cmu.edu,* [‡] {*imoraru, dbrumley, dga*}*@cs.cmu.edu*
[*] *jtruelove@ll.mit.edu*

Abstract

We present the design and implementation of a novel anti-malware system called SplitScreen. SplitScreen performs an additional screening step prior to the signature matching phase found in existing approaches. The screening step filters out most non-infected files (90%) and also identifies malware signatures that are not of interest (99%). The screening step significantly improves end-to-end performance because safe files are quickly identified and are not processed further, and malware files can subsequently be scanned using only the signatures that are necessary. Our approach naturally leads to a network-based anti-malware solution in which clients only receive signatures they needed, not every malware signature ever created as with current approaches. We have implemented SplitScreen as an extension to ClamAV [13], the most popular open source anti-malware software. For the current number of signatures, our implementation is 2× faster and requires 2× less memory than the original ClamAV. These gaps widen as the number of signatures grows.

1 Introduction

The amount of malicious software (malware)—viruses, worms, Trojan horses, and the like—is exploding. As the amount of malware grows, so does the number of *signatures* used by anti-malware products (also called anti-viruses) to detect known malware. In 2008, Symantec created over 1.6 million new signatures, versus a still-boggling six hundred thousand new signatures in 2007 [2]. The ClamAV open-source anti-malware system similarly shows exponential growth in signatures, as shown in Figure 1. Unfortunately, this growth, fueled by easy-to-use malware toolkits that automatically create hundreds of unique variants [1, 20], is creating difficult system and network scaling problems for current signature-based malware defenses.

There are three scaling challenges. First, the sheer number of malware signatures that must be distributed to end-hosts is huge. For example, the ClamAV open-source product currently serves more than 120 TB of signatures per day [14]. Second, current anti-malware systems keep all signatures pinned in main memory. Reducing the size of the pinned-in-memory component is important to ensure operation on older systems and resource constrained devices such as netbooks, PDAs or smartphones, and also to reduce the impact that malware scanning has on other applications running concurrently on the same system. Third, the matching algorithms typically employed have poor cache utilization, resulting in a substantial slowdown when the signature database outgrows the L2 and L3 caches.

We propose *SplitScreen,* an anti-malware architecture designed to address the above challenges. Our design is inspired by two studies we performed. First, we found that the distribution of malware in the wild is extremely biased. For example, only 0.34% of all signatures in ClamAV were needed to detect all malware that passed through our University's e-mail gateways over a 4 month period (§5.2). Of course, for safety, we cannot simply remove the unmatched signatures since a client must be able to match anything in the signature database. Second, the performance of current approaches is bottlenecked by matching regular expression signatures in general, and by cache-misses due to that scanning in particular. Since, in existing schemes, the number of cache-misses grows rapidly with the total number of signatures, the efficiency of existing approaches will significantly degrade as the number of signatures continues to grow. Others have made similar observations [10].

At a high level, SplitScreen divides scanning into two steps. First, all files are scanned using a small, cache-optimized data structure we call a *feed-forward Bloom filter* (FFBF) [18]. The FFBF implements an approximate pattern-matching algorithm that has one-sided error: it will properly identify all malicious files, but may also identify some safe files as malicious. The FFBF outputs: (1) a set of suspect matched files, and (2) a subset of signatures from the signature database needed to confirm that suspect files are indeed malicious. SplitScreen then rescans the suspect matched files using the subset of signatures using an exact pattern matching algorithm.

The SplitScreen architecture naturally leads to a demand-driven, network-based architecture where clients download the larger exact signatures only when needed in step 2 (SplitScreen still accelerates traditional single-host scanning when running the client and the

server on the same host). For example, SplitScreen requires 55.4 MB of memory to hold the current ≈ 533,000 ClamAV signatures. ClamAV, for the same signatures, requires 116 MB of main memory. At 3 million signatures, SplitScreen can use the same amount of memory (55.4 MB), but ClamAV requires 534 MB. Given the 0.34% hit rate in our study, SplitScreen would download only 10,200 signatures for step 2 (vs. 3 million). Our end-to-end analysis shows that, overall, SplitScreen requires less than 10% of the storage space of existing schemes, with only 10% of the network volume (§5). We believe these improvements to be important for two reasons: (1) SplitScreen can be used to implement malware detection on devices with limited storage (e.g., residential gateways, mobile and embedded devices), and (2) it allows for fast signature updates, which is important when counteracting new, fast spreading malware. In addition, our architecture preserves clients' privacy better than prior network-based approaches [19].

SplitScreen addresses the memory scaling challenge because its data structures grow much more slowly than in existing approaches (with approximately 11 bytes per signature for SplitScreen compared to more than 170 bytes per signature for ClamAV). Combined with a cache-efficient algorithm, this leads to better throughput as the number of signatures grows, and represents the major advantage of our approach when compared to previous work that employed simple Bloom filters to speed-up malware detection (§5.9 presents a detailed comparison with HashAV [10]). SplitScreen addresses the signature distribution challenges because users only download the (small) subset of signatures needed for step 2. SplitScreen addresses constrained computational devices because the entire signature database need not fit in memory as with existing approaches, as well as having better throughput on lower-end processors.

Our evaluation shows that SplitScreen is an effective anti-malware architecture. In particular, we show:

- **Malware scanning at twice the speed with half the memory**: By adding a cache-efficient prescreening phase, SplitScreen improves throughput by more than 2× while simultaneously requiring less than half the total memory. These numbers will improve as the number of signatures increases.
- **Scalability**: SplitScreen can handle a very large increase in the number of malware signatures with only small decreases in performance (35% decrease in speed for 6× more signatures §5.4).
- **Distributed anti-malware**: We developed a novel distributed anti-malware system that allows clients to perform fast and memory-inexpensive scans, while keeping the network traffic very low during both normal operation and signature updates. Furthermore, clients maintain their privacy by sending only information about malware possibly present on their systems.

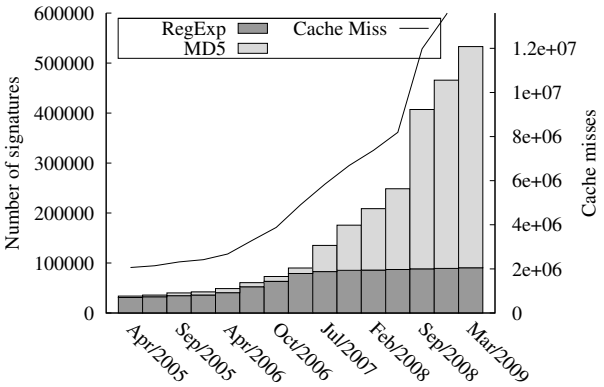

Figure 1: Number of signatures and cache misses in ClamAV from April 2005 to March 2009.

- **Resource-constrained devices**: SplitScreen can be applied to mobile devices (e.g., smartphones[1]), older computers, netbooks, and similar devices. We evaluated SplitScreen on a low-power device similar to an iPhone 3GS. In our experiments, SplitScreen worked properly even with 3 million signatures, while ClamAV crashed due to lack of resources at 2 million signatures.
- **Real-World Implementation**: We have implemented our approach in ClamAV, an open-source anti-malware defense system. Our implementation is available at `http://security.ece.cmu.edu`. We will make the malware data sets used in this paper available to other researchers upon request.

2 Background

2.1 Signature-based Virus Scanning

Signature-based anti-malware defenses are currently the most widely used solutions. While not the only approach (e.g., recent proposals for behavior-based detection such as [15]), there are two important reasons to continue improving signature-based methods. First, they remain technically viable today, and form the bedrock of the two billion dollar anti-malware industry. More fundamentally, signature-based techniques are likely to remain an important component of anti-malware defenses, even as those defenses incorporate additional mechanisms.

In the remainder of this section we describe signature-based malware scanning, using ClamAV [13] as a spe-

[1]Smartphones have many connectivity options, and are able to run an increasingly wide range of applications (sometimes on open platforms). We therefore expect that they will be subjected to the same threats as traditional computers, and they will require the same security mechanisms.

cific example. ClamAV is the most popular open-source anti-malware solution, and already incorporates significant optimizations to speed up matching and decrease memory consumption. We believe ClamAV to be representative of current malware scanning algorithms, and use it as a baseline from which to measure improvements due to our techniques.

During initialization, ClamAV reads a signed signature database from disk. The database contains two types of signatures: whole file or segment MD5 signatures and byte-pattern signatures written in a custom language with regular expression-like syntax (although they need not have wildcards) which we refer to as regular expression signatures (regexs). Figure 1 shows the distribution of MD5 and regular expression signatures in ClamAV over time. Currently 84% of all signatures are MD5 signatures, and 16% are regular expressions. In our experiments, however, 95% of the total scanning time is spent matching the regex signatures.

When scanning, ClamAV first performs several preprocessing steps (e.g., attempting to unpack and uncompress files), and then checks each input file sequentially against the signature database. It compares the MD5 of the file with MD5s in the signature database, and checks whether the file contents match any of the regular expressions in the signature database. If either check matches a known signature, the file is deemed to be malware.

ClamAV's regular expression matching engine has been significantly optimized over its lifetime. ClamAV now uses two matching algorithms [16]: Aho-Corasick [3] (AC) and Wu-Manber [23] (WM).[2] The slower AC is used for regular expression signatures that contain wildcard characters, while the faster WM handles fixed string signatures.

The AC algorithm builds a trie-like structure from the set of regular expression signatures. Matching a file with the regular expression signatures corresponds to walking nodes in the trie, where transitions between nodes are determined by details of the AC algorithm not relevant here. Successfully walking the trie from the root to a leaf node corresponds to successfully matching a signature, while an unsuccessful walk corresponds to not matching any signature. A central problem is that a trie constructed from a large number of signatures (as in our problem setting) will not fit in cache. Walks of such tries will typically visit nodes in a semi-random fashion, causing many cache misses.

The Wu-Manber [23] algorithm for multiple fixed patterns is a generalization of the single-pattern Boyer-Moore [6] algorithm. Matching using Wu-Manber entails hash table lookups, where a failed lookup means the input does not match a signature. In our setting, ClamAV uses a sliding window over the input file, where the bytes in window are matched against signatures by using a hash table lookup. Again, if the hash table does not fit in cache, each lookup can cause a cache miss. Thus, there is a higher probability of cache misses as the size of the signature database grows.

[2] ClamAV developers refer to this algorithm as extended Boyer-Moore.

2.2 Bloom Filters

The techniques we present in this paper make extensive use of Bloom filters [5]. Consider a set S. A Bloom filter is a data structure used to implement set membership tests of S quickly. Bloom filters membership tests may have one-sided errors. A false positive occurs when the outcome of the test is $x \in S$ when x is not really a member of S. Bloom filters will never incorrectly report $x \notin S$ when x really is in S.

Initialization. Bloom filter initialization takes the set S as input. A Bloom filter uses a bit array with m bits, and k hash functions to be applied to the items in S. The hashes produce integers with values between 1 and m, that are used as indices in the bit array: the k hash functions are applied to each element in S, and the bits indexed by the resulting values are set to 1 (thus, for each element in S, there will be a maximum of k bits set in the bit array—fewer if there are collisions between the hashes).

Membership test. When doing a set membership test, the tested element is hashed using the k functions. If the filter bits indexed by the resulting values are all set, the element is considered a member of the set. If at least one bit is 0, the element is definitely not part of the set.

Important parameters. The number of hash functions used and the size of the bit array determine the false positive rate of the Bloom filter. If S has $|S|$ elements, the asymptotic false positive probability of a test is $\left(1-e^{-k|S|/m}\right)^k$ [7]. For a fixed m, $k = ln2 \times |S|/m$ minimizes this probability. In practice however, k is often chosen smaller than optimum for speed considerations: a smaller k means computing a smaller number of hash functions and doing fewer accesses to the bit array. In addition, the hashing functions used affect performance, and when non-uniform, can also increase the false positive rate.

Scanning text. Text can be efficiently scanned for multiple patterns using Bloom filters in the Rabin-Karp [11] algorithm. The patterns, all of which must be of the same length w, represent the set used to initialize the Bloom filter. The text is scanned by sliding a window of length w and checking rolling hashes of its content, at every position, against the Bloom filter. Exact matching requires every Bloom filter hit to be confirmed by running a verification step to weed out Bloom filter false positives (e.g., using a subsequent exact pattern matching algorithm).

3 Design

SplitScreen is inspired by several observations. First, the number of malware programs is likely to continue to grow, and thus the scalability of an anti-malware system is a primary concern. Second, malware is not confined to high-end systems; we need solutions that protect slower systems such as smartphones, old computers, netbooks, and similar systems. Third, signature-based approaches are by far the most widely-used in practice, so improvements to signature-based algorithms are likely to be widely applicable. Finally, in current signature-based systems all users receive all signatures whether they (ultimately) need them or not, which is inefficient[3].

3.1 Design Overview

At a high level, an anti-malware defense has a set of signatures Σ and a set of files F. For concreteness, in this section we focus on regular expression signatures commonly found in anti-malware systems—so we use Σ to denote a set of regular expressions. We extend our approach to MD5 signatures in §3.5.1. The goal of the system is to determine the (possibly empty) subset $F_{malware} \subseteq F$ of files that match at least one signature $\sigma \in \Sigma$.

SplitScreen is an anti-malware system, but its approach differs from existing systems because it does not perform exact pattern matching on every file in F. Instead SplitScreen employs a cache-efficient data structure called a feed-forward Bloom filter (FFBF) [18] that we created for doing approximate pattern matching. We use it in conjunction with the Rabin-Karp text search algorithm (see §2.2). The crux of the system is that the cache-efficient first pass has extremely high throughput. The cache-efficient algorithm is approximate in the sense that the FFBF scan returns a set of suspect files $F_{suspect}$ that is a superset of malware identified by exact pattern matching, i.e., $F_{malware} \subseteq F_{suspect} \subseteq F$. In the second step we perform exact pattern matching on $F_{suspect}$ and return exactly the set $F_{malware}$. Figure 2 illustrates this strategy. The files in $F_{suspect} \setminus F_{malware}$ represent the false positives that we refer to in various sections of this paper, and they are caused by 1) Bloom filter false positives (recall that Bloom filters have one-sided error) and 2) the fact that we can only look for fixed-size fragments of signatures and not entire signatures in the first step (the FFBF scan), as a consequence of how Rabin-Karp operates.

[3]To put things in perspective, suppose there is a new Windows virus, and that the 1 billion computers with Microsoft Windows [4] are all running anti-malware software. A typical signature is at least 16 bytes (e.g., the size of an MD5). If each computer receives a copy of the signature, then that one virus has cost 15,258 MB of disk space worldwide to store the signature.

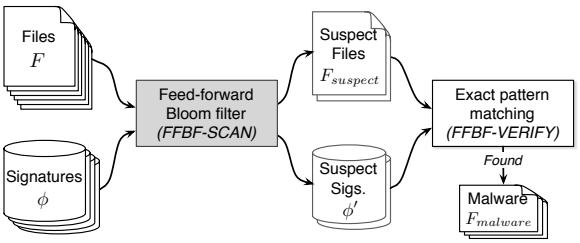

Figure 2: The SplitScreen scanning architecture.

3.2 High-Level Algorithm

The scanning algorithm used by SplitScreen consists of four processing steps called FFBF-INIT, FFBF-SCAN, FFBF-HIT, and FFBF-VERIFY, which behave as follows:

FFBF-INIT$(\Sigma) \to \phi$ takes as input the set of signatures Σ and outputs a bit-vector ϕ which we call the *all-patterns bit vector*. FFBF-SCAN will use this bit-vector to construct an FFBF to scan files.

FFBF-SCAN$(\phi, F) \to (\phi', F_{suspect})$ constructs an FFBF from ϕ and then scans each file $f \in F$ using the FFBF. The algorithm outputs the tuple $(\phi', F_{suspect})$ where $F_{suspect} \subseteq F$ is the list of files that were matched by ϕ, and ϕ' is a bit vector that identifies the signatures actually matched by $F_{suspect}$. We call ϕ' the *matched-patterns bit vector*.

FFBF-HIT$(\phi', \Sigma) \to \Sigma'$ takes in the matched-patterns bit vector ϕ' and outputs the set of regexp signatures $\Sigma' \subseteq \Sigma$ that were matched during FFBF-SCAN.

FFBF-VERIFY$(\Sigma', F_{suspect}) \to F_{malware}$ takes in a set of regular expression signatures Σ', a set of files $F_{suspect}$, and outputs the set of files $F_{malware} \subseteq F_{suspect}$ matching Σ'.

The crux of the SplitScreen algorithm can be expressed as:

$$\begin{aligned}&\text{SCAN}(\Sigma, F) = \\ &\quad \text{let } (\phi', F_{suspect}) = \text{FFBF-SCAN}(\text{FFBF-INIT}(\Sigma), F) \text{ in} \\ &\qquad \text{FFBF-VERIFY}(\text{FFBF-HIT}(\phi', \Sigma), F_{suspect})\end{aligned}$$

Let R denote the existing regular expression pattern matching algorithm, e.g., R is ClamAV. SplitScreen achieves the following properties:

Correctness. SCAN will return the same set of files as identified by R, i.e., $\text{SCAN}(\Sigma, F) = R(\Sigma, F)$.

Higher Throughput. SCAN runs faster than R. In particular, we want the time for FFBF-SCAN plus FFBF-VERIFY plus FFBF-HIT to be less than the time to execute R. (Since FFBF-INIT is an initialization step performed only once per set of signatures, we do not consider it for throughput. We similarly discount in R the time to initialize any data structures in its algorithm.)

Less Memory. The amount of memory needed by SCAN is less than R. In particular, we want $\max(|\phi| + |\phi'|, |\Sigma'|) \ll |\Sigma|$ (the bit vectors are not required to be in memory during FFBF-VERIFY). We expect that the common case is that most signatures are never matched, e.g., the average user does not have hundreds of thousands or millions of unique malware programs on their computer. Thus $|\Sigma'| \ll |\Sigma|$, so the total memory overhead will be significantly smaller. In the worst case, where every signature is matched, $\Sigma' = \Sigma$ and SplitScreen's memory overhead is the same as existing systems's.

Scales to More Signatures. Since the all-patterns bit vector ϕ takes a fraction of the space needed by typical exact pattern matching data structures, the system scales to a larger number of signatures.

Network-based System. Our approach naturally leads to a distributed implementation where we keep the full set of signatures Σ on a server, and distribute ϕ to clients. Clients use ϕ to construct an FFBF and scan their files locally. After FFBF-SCAN returns, the client sends ϕ' to a server to perform FFBF-HIT, gets back the set of signatures Σ' actually needed to confirm malware is present. The client runs FFBF-VERIFY locally.

Privacy. In previous network-based approaches such as CloudAV [19], a client sends every file to a server (the cloud) for scanning. Thus, the server can see all of the client's files. In our setting, the client never sends a file across the network. Instead, the client sends ϕ', which can be thought of as a list of possible viruses on their system. We believe this is a better privacy tradeoff. Furthermore, clients can attain deniability as explained in §3.4. Note our architecture can be used to realize the existing anti-malware paradigm where the client simply asks for all signatures. Such a client would still retain the improved throughput during scanning by using our FFBF-based algorithms.

3.3 Bloom-Based Building Blocks

Bloom filters can have false positives, so a hit must be confirmed by an exact pattern matching algorithm (hence the need for FFBF-VERIFY). Our first Bloom filter enhancement reduces the number of signatures needed for verification, while the second accelerates the Bloom filter scan itself.

3.3.1 Feed-Forward Bloom Filters

An FFBF consists of two bit vectors. The *all-patterns bit vector* is a standard Bloom filter initialized as described in §3.5.1. In our setting, the set of items is Σ. The *matched-patterns bit vector* is initialized to 0.

As with an ordinary Bloom filter, a candidate item is hashed and the corresponding bits are tested against the all-patterns bit vector. If all the hashed bits are set in the all-patterns bit vector, the item is output as a FFBF match. When a match occurs, the FFBF will additionally set each bit used to check the all-patterns bit vector to 1 in the matched-patterns bit vector. In essence, the matched-patterns bit vector records which entries were found in the Bloom filter. This process is shown in Figure 3.

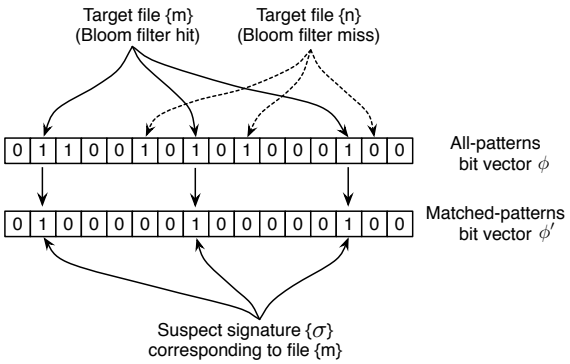

Figure 3: Building the *matched-patterns bit vector* as part of the feed-forward Bloom filter algorithm.

After all input items have been scanned through the FFBF, the matched-patterns bit vector is a Bloom filter representing the patterns that were matched. The user of an FFBF can generate a list of potentially matching patterns by running the input pattern set against the matched-patterns Bloom filter to identify which items were actually tested. Like any other Bloom filter output, the output pattern subset may contain false positives.

In SplitScreen, ϕ is the all-patterns bit vector, and ϕ' is the matched-patterns bit vector created by FFBF-SCAN. Thus, ϕ' identifies (a superset of) signatures that would have matched using exact pattern matching. FFBF-HIT uses ϕ' to determine the set of signatures needed for FFBF-VERIFY.

3.3.2 Cache-Partitioned Bloom Filters

While a Bloom filter alone is more compact than other data structures traditionally used in pattern matching algorithms like Aho-Corasick or Wu-Manber, it is not otherwise more cache-friendly: it performs random access within a large vector. If this vector does not fit entirely in cache, the accesses will cause cache misses which will degrade performance substantially.

SplitScreen uses our cache-friendly partitioned bloom filter design [18], which splits the input bit vector into two parts. The first is sized to be entirely cache-resident, and the first s hash functions map only into this section of the vector. The second is created using virtual memory super-pages (when available) and is sized to be as *large* as possible without causing TLB misses. The FFBF pre-

vents cache pollution by using non-cached reads into the second bloom filter. The mechanisms for automatically determining the size of these partitions and the number of hash functions are described in our technical report [18].

The key to this design is that it is optimized for bloom-filter *misses*. Recall that a Bloom filter hit requires matching each hash function against a "1" in the bit vector. As a result, most misses will be detected after the first or second test, with an exponentially decreasing chance of requiring more and more tests.

The combination of a bloom-filter representation and a cache-friendly implementation provide a substantial speedup on modern architectures, as we show in §5.

3.4 SplitScreen Distributed Anti-Malware

In the SplitScreen distributed model, the input files are located on the clients, while the signatures are located on a server. The system works as follows:

1. The server generates the all-patterns bit vector for the most recent malware signatures and transmits it to the client. It will be periodically updated to contain the latest malware bit patterns, just as existing approaches must be updated.
2. The client performs the pre-screening phase using the feed-forward Bloom filter, generates the matched-patterns bit vector, compresses it and transmits it to the server.
3. The server uses the matched-patterns bit vector to filter the signatures database and sends the full definitions (1% of the signatures) to the client.
4. The client performs exact matching with the suspect files from the first phase and the suspect signatures received from the server.

In this system, SplitScreen clients maintain only the all-patterns bit vectors ϕ (there will be two bit vectors corresponding to two FFBFs, one for each type of signature). Instead of replicating the large signature database at each host, the database is stored only at the server and clients only get the signatures they are likely to need. This makes updates inexpensive: the server updates its local signature database and then sends differential all-patterns bit vector updates[4] to the clients.

Since the clients don't have to use the entire set of signatures for scanning, they also need less in-core memory (important for multi-task systems), and have smaller load times.

SplitScreen does not expose as much private data as earlier distributed anti-malware systems [19], because the contents of clients' files are never sent over the network, instead clients only send compact representations

[4]An all-patterns bit vector update is a sparse—so highly compressible—bit vector that is overlaid on top of the old bit vector.

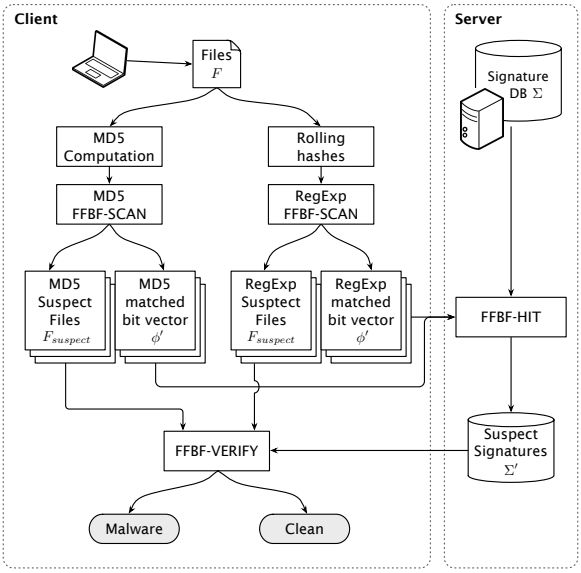

Figure 4: Data flow for distributed SplitScreen.

(bit vectors) of short hashes (under 32 bits) of small (usually under 20 bytes long) parts of undisclosed files and hashes of MD5 signatures of files. Clients concerned about deniability could set additional (randomly chosen) bits in their matched-patterns bit vectors in exchange for increased network traffic.

3.5 Design Details

3.5.1 Files and Signatures Screening

As explained in §2.1, ClamAV uses two types of signatures: regexp signatures and MD5 signatures. We handle each with its own FFBF.

Pattern signatures. The SplitScreen server extracts a fragment of length w from every signature (the way w is chosen is discussed in §5.8, while handling signatures smaller than w bytes and signatures containing wildcards is presented in §3.5.3 and §3.5.2). These fragments will be hashed and inserted into the FFBF. When performing FFBF scanning, a window of the same size (w) is slid through the examined files, and its content at every position is tested against the filter. The hash functions we use in our FFBF implementation are based on hashing by cyclic polynomials [8] which we found to be effective and relatively inexpensive. To reduce computation further, we use the idea of Kirsch and Mitzenmacher [12] and compute only two independent hash functions, deriving all the others as linear combinations of the first two.

MD5 signatures. ClamAV computes the MD5 hash of each scanned file (or its sections) and searches for it in a hash table of malware MD5 signatures. SplitScreen replaces the hash table with an FFBF to save memory. The elements inserted into the filter are the MD5 signatures

themselves, while the candidate elements tested against the filter are the MD5 hashes computed for the scanned files. Since the MD5 signatures are uniform hash values, the hash functions used for the FFBF are straightforward: given a 16-byte MD5 signature $b_1 b_2 ... b_{16}$, we compute the 4-byte hash values as linear combinations of $h_1 = b_1 ... b_4 \oplus b_5 ... b_8$ and $h_2 = b_9 ... b_{12} \oplus b_{13} ... b_{16}$.

3.5.2 Signatures with Wildcards

A small fraction (1.5% in ClamAV) of regular expression signatures contain wildcards, but SplitScreen's Rabin-Karp-based FFBF algorithm operates with fixed strings. Simply expanding the regular expressions does not work. For example, the expression

$$3c666f726d3e\{1-200\}3c696e707574$$

(where "{1-200}" matches any sequence no longer than 200 bytes) generates 256^{200} different byte sequences. It is impractical to put all of them into the Bloom filter.

Instead, SplitScreen extracts the invariant fragments (fixed byte subsequences) of a wildcard-containing signature and selects one of these fragments to put in the FFBF (see §3.5.4 for more details about fragment selection).

3.5.3 Short Signatures

If a regular expression signature does not contain a fixed fragment at least as long as the window size, the signature cannot be added to the feed-forward Bloom filter. Decreasing the window size to the length of the shortest signature in the database would raise the Bloom filter scan false positive rate to an unacceptable level, because the probability of a random sequence of bytes being found in any given file increases exponentially as the sequence shortens.

SplitScreen therefore performs a separate, exact pattern matching step for short signatures concurrently with the FFBF scanning. Short signatures are infrequent (they represent less than 0.4% of ClamAV's signature set for our default choice for the window size—12 bytes), so this extra step does not significantly reduce performance. The SplitScreen server builds the *short signature set* when constructing the Bloom filters. Whenever a SplitScreen client requires Bloom filter updates, the SplitScreen server sends it this short signature set too.

3.5.4 Selecting Fragments using Document Frequency

While malware signatures are highly specific, the fixed-length substrings that SplitScreen uses may not be. For example, suppose that the window size is 16 bytes. Almost every binary file contains 16 consecutive "0x00" bytes. Since we want to keep as few files as possible for the exact-matching phase, we should be careful not to include such a pattern into the Bloom filter.

Σ_i = set of signatures
σ = input signature ($\sigma \in \Sigma$)
w = fixed window size
γ = length w fixed byte sequence (w-gram) in σ
$DF(\gamma)$ = the document frequency of w-gram γ
——— *outputs* ———
ϕ_i = FFBF signatures
Σ_{short} = set of short signatures

for all $\sigma \in \Sigma_{md5}$, put σ into ϕ_{md5}
for all σ in $\Sigma_{fixed} \cup \Sigma_{wild}$
  if $|\sigma| \geq w$
    for all fixed byte w-grams γ in σ
      if $DF(\gamma) = 0$
        put γ into ϕ_{regexp}; GOTO next σ
  //either shorter than w or no zero DF
  put σ into Σ_{short}

Figure 5: Final FFBF-INIT algorithm.

We use the *document frequency (DF)* of signature fragments in clean binary files to determine if a signature fragment is likely to match safe files. The *DF* of a signature fragment represents the number of documents containing this fragment. A high DF indicates that the corresponding signature fragment is common and may generate many false positives.

We compute the DF value for each window-sized signature fragment in clean binary samples. For each signature, we insert into the filter the first fragment with a DF value of zero (i.e., the fragment did not occur in any of the clean binary files). The signatures that have no zero DF fragments are added to the short signature set.

We summarize our signature processing algorithm in Figure 5. The SplitScreen server runs this algorithm for every signature, and creates two Bloom filters—one for MD5 signatures, and one for the regular expression signatures—as well as the set of short signatures.

3.5.5 Important Parameters

We summarize in this section the important parameters that affect the performance of our system, focusing on the tradeoffs involved in choosing those parameters.

Bit vector size. The size of the bit vectors trades scan speed for memory use. Larger bit vectors (specifically, larger non-cache-resident parts) result in fewer Bloom filter false positives, improving performance up to the point where TLB misses become a problem (see §3.3.2).

Sliding window size. The wider the sliding window used to scan files during FFBF-SCAN, the less chance there is of a false positive (see §5.8). This makes FFBF-VERIFY run faster (because there will be fewer files to check). However, the wider the sliding window, the more signatures that must be added to the short signature set. Since we look for short signatures in every input file,

a large number of short signatures will reduce performance.

Number of Bloom filter hash functions. The number of hash functions used in the FFBF algorithm (the k parameter in §2.2) is a parameter for which an optimum value can be computed when taking into account the characteristics of the targeted hardware (e.g. the size of the caches, the latencies in accessing different levels of the memory hierarchy) as described in [18]. Empirically, we found that two hash functions each for the cache-resident part and the non-cache-resident part of the FFBF works well for a wide range of hardware systems.

4 Implementation

We have implemented SplitScreen as an extension of the ClamAV open source anti-malware platform, version 0.94.2. Our code is available at `http://security.ece.cmu.edu`. The changes comprised approximately 8K lines of C code. The server application used in our distributed anti-malware system required 5K lines of code. SplitScreen servers and SplitScreen clients communicate with each other via TCP network sockets.

The SplitScreen client works like a typical anti-malware scanner; it takes in a set of files, a signature database (ϕ in SplitScreen), and outputs which files are malware along with any additional metadata such as the malware name. We modified the existing `libclamav` library to have a two-phase scanning process using FFBFs.

The SplitScreen server generates ϕ from the default ClamAV signatures using the algorithm shown in Figure 5. Note that SplitScreen can implement traditional single-host anti-malware by simply running the client and server on the same host. We use run-length encoding to compress the bit vectors and signatures sent between client and server.

5 Evaluation

In this section we first detail our experimental setup, and then briefly summarize the malware measurements that confirm our hypothesis that most of the volume of malware can be detected using a few signatures. We then present an overall performance comparison of SplitScreen and ClamAV, followed by detailed measurements to understand why SplitScreen performs well, how it scales with increasing numbers of regexp and MD5 signatures, and how its memory use compares with ClamAV. We then evaluate SplitScreen's performance on resource constrained devices and its performance in a network-based use model.

5.1 Evaluation Setup

Unless otherwise specified, our experiments were conducted on an Intel 2.4 GHz Core 2 Quad with 4 GB of RAM and a 8 MB split L2 cache using a 12-byte window size (see §3). When comparing SplitScreen against ClamAV, we exclude data structure initialization time in ClamAV, but count the time for FFBF_INIT in SplitScreen. Thus, our measurements are conservative because they reflect the best possible setting for ClamAV, and the worst possible setting for SplitScreen. Unless otherwise specified, we report the average over 10 runs.

Scanned files. Unless otherwise specified, all measurements reflect scanning 344 MB of 100% clean files. We use clean files because they are the common case, and exercise most code branches. (§5.7 shows performance for varying amounts of malware.) The clean files come from a fresh install of Microsoft Windows XP plus typical utilities such as MS Office 2007 and MS Visual Studio 2007.

Signature sets. We use two sets of signatures for the evaluation. If unspecified, we focus on the current ClamAV signature set (main v.50 and daily v.9154 from March 2009), which contained 530K signatures. We use four additional historical snapshots from the ClamAV source code repository. To measure how SplitScreen will improve as the number of signatures continues to grow, we generated additional regex and MD5 signatures ("projected" in our graphs) in the same relative proportion as the March signature set. The synthetic regexs were generated by randomly permuting fixed strings in the March snapshot, while the synthetic MD5s are random 16-byte strings.

5.2 Malware Measurements

Given a set of signatures Σ, we are interested in knowing how many individual signatures Σ' are matched in typical scenarios, i.e., $|\Sigma'|$ vs. $|\Sigma|$. We hypothesized that most signatures are rarely matched ($|\Sigma'| \ll |\Sigma|$), e.g., most signatures correspond to malware variants that are never widely distribution.

One typical use of anti-malware products is to filter out malware from email. We scanned Carnegie Mellon University's email service from May 1st to August 29th of 2009 with ClamAV. 1,392,786 malware instances were detected out of 19,443,381 total emails, thus about 7% of all email contained malware by volume. The total number of unique signatures matched was 1,825, which is about 0.34% of the total signatures—see figure 6.

Another typical use of anti-malware products is to scan files on disk. We acquired 393 GB of malware from various sites, removed duplicate files based upon MD5, and removed files not recognized by ClamAV using the v.9661 daily and v.51 main signature database. The total number of signatures in ClamAV was 607,988, and the

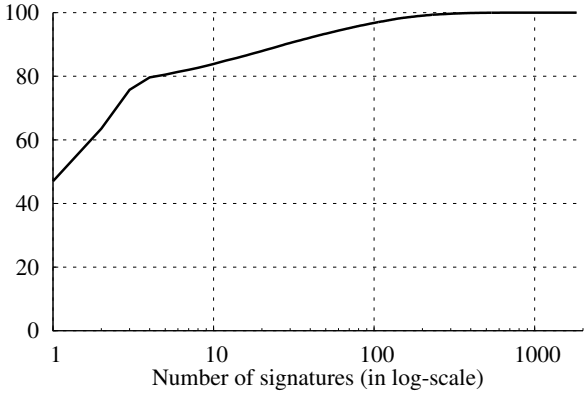

Figure 6: The overall amount of malware detected (y axis) vs. the total number of malware signatures needed (x axis). For example, about 1000 signatures are needed to detect virtually all malware.

total number of unique malware files was 960,766 (about 221 GB). ClamAV reported out of the 960,766 unique files that there were 128,992 unique malware variants. Thus, about 21.2% of signatures were matched.

We conclude that indeed most signatures correspond to rare malware, while only a few signatures are typically needed to match malware found in day-to-day operations.

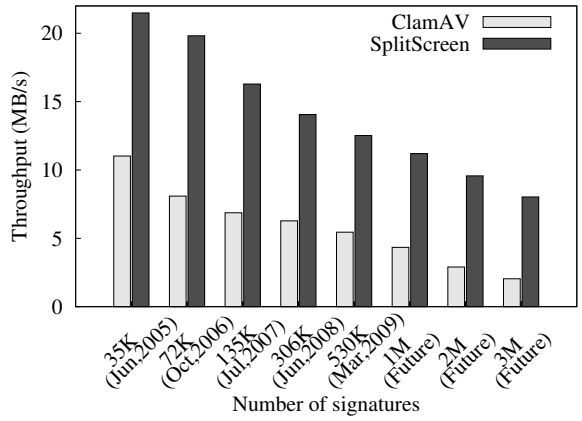

Figure 7: Performance of SplitScreen and ClamAV using historical and projected ClamAV signature sets.

5.3 SplitScreen Throughput

We ran SplitScreen using both historical and projected signature sets from ClamAV, and compared its performance to ClamAV on the same signature set. Figure 7 shows our results. SplitScreen consistently improves throughput by at least 2× on previous and existing signatures, and the throughput improvement factor increases with the number of signatures.

Understanding throughput: Cache misses. We hypothesized that a primary bottleneck in ClamAV was L2 cache misses in regular expression matching. Figure 8 shows ClamAV's throughput and memory use as the number of regular expression signatures grows from zero to roughly 125,000, with *no* MD5 signatures. In contrast, increasing the number of MD5 signatures linearly increases the total memory required by ClamAV, but has almost no effect on its throughput. With no regexp signatures, ClamAV scanned nearly 50 MB/sec, regardless of the number of MD5 signatures.

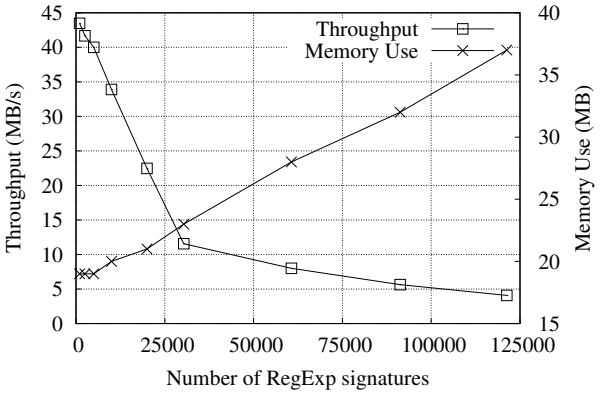

Figure 8: ClamAV scanning throughput and memory consumption as the number of regular expression signatures increases.

Figure 9 compares the absolute number of L2 cache misses for ClamAV and SplitScreen as the (total) number of signatures increases. The dramatic increase in L2 cache misses for ClamAV suggest that this is, indeed, a major source of its performance degradation. In contrast, the number of cache misses for SplitScreen is much lower, helping to explain its improved scanning performance. These results indicate that increasing the number of regex signatures increases the number of cache misses, decreases throughput, and thus is the primary throughput bottleneck in ClamAV.

5.4 SplitScreen Scalability and Performance Breakdown

How well does SplitScreen scale? We measured three scaling dimensions: 1) how throughput is affected as the number of regular expression signatures grows, 2) how FFBF size affects performance and memory use, and 3) where SplitScreen spends time as the number of signatures increases.

Throughput. Figure 10 shows SplitScreen's throughput as the number of signatures grows from 500K (approximately what is in ClamAV now) to 3 million. At 500K signatures, SplitScreen performs about 2.25 times better than ClamAV. **At 3 million signatures, SplitScreen performs 4.5 times better**. The 4.5×

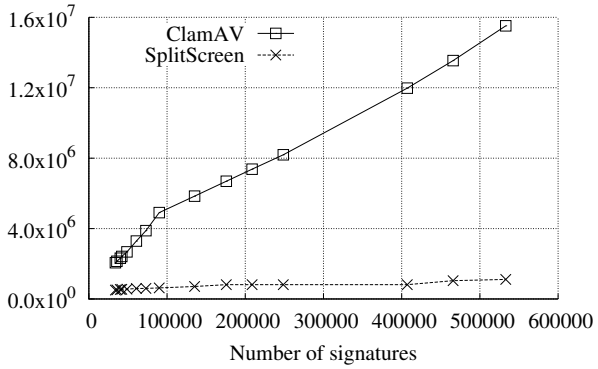

Figure 9: Cache misses.

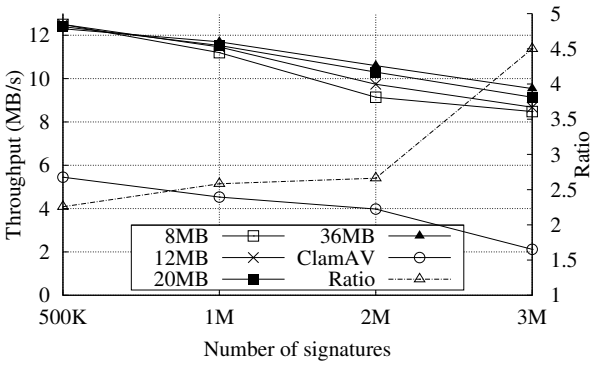

Figure 10: Performance for different size feed-forward Bloom filters, keeping the cache-resident portion constant.

throughput increase is given with a 32 MB FFBF. These measurements are all an average over 10 runs. The worst of these runs is the first when the file system cache is cold, when SplitScreen was only 3× faster than ClamAV (graph omitted due to space).

FFBF Size. We also experimented with smaller FFBF's of size 8, 12, 20, and 36 MB, as shown in Figure 10. The larger the FFBF, the smaller the false positive ratio, thus the greater the performance. We saw no additional performance gain by increasing the FFBF beyond 36 MB.

# sigs	FFBF-SCAN +Short Sigs.	FFBF-HIT + Traffic	FFBF-VERIFY
500K	27.2 (94.7%)	0.7 (2.6%)	0.8 (2.7%)
1M	27.4 (92.4%)	0.9 (3.0%)	1.4 (4.6%)
2M	26.5 (76.0%)	1.3 (3.7%)	7.1 (20.3%)
3M	24.2 (58.3%)	1.7 (4.1%)	15.6 (37.6%)

Table 1: Time spent per step by SplitScreen to scan 1.55 GB of files (in seconds and by percentage).

Per-Step Breakdown. Table 1 shows the breakdown of time spent per phase. We do not show FFBF-INIT which was always $< 0.01\%$ of total time. As noted earlier, we omit ClamAV initialization time in order to provide conservative comparisons.

We make draw several conclusions from our experiments. First, SplitScreen's performance advantage continues to grow as the number of regexp signatures increases. Second, the time required by the first phase of scanning in SplitScreen holds steady, but the exact matching phase begins to take more and more time. This occurs because we held the size of the FFBF constant. When we pack more signatures into the same size FFBF, the bit vector becomes more densely populated, thus increasing the probability of a false positive due to hash collisions. Such false positives result in more signatures to check during FFBF-VERIFY. Thus, while the overall scan time is relatively small, increasing the SplitScreen FFBF size will help in the future, i.e., we can take advantage of the larger caches the future may bring. Note that the size increases to the FFBF need be nowhere near as large as with ClamAV, e.g., a few megabytes for SplitScreen vs. a few hundred megabytes for ClamAV.

5.5 SplitScreen on Constrained Devices

Figure 11 compares the memory required by SplitScreen and ClamAV for FFBF-SCAN. 533,183 signatures in ClamAV consumed about 116 MB of memory. SplitScreen requires only 55.4 MB, of which 40 MB are dedicated to FFBFs. Our FFBF was designed to minimize false positives due to hash collisions but not adversely affect performance due to TLB misses (§3.3.2). At 3 million signatures, ClamAV consumed over 500 MB of memory, while SplitScreen still performed well with a 40 MB FFBF.

We then tested SplitScreen's performance with four increasingly more limited systems. We compare SplitScreen and ClamAV using the current signature set on: a 2009 desktop computer (Intel 2.4 GHz Core 2 Quad, 4 GB RAM, 8 MB L2 cache); a 2008 Apple laptop (Intel 2.4 GHz Core 2 Duo, 2 GB RAM, 3 MB L2 cache); a 2005 desktop (Intel Pentium D 2.8 GHz, 4 GB RAM, 2 MB L2 Cache); and a Alix3c2 (AMD Geode 500 Mhz, 256 MB RAM, 128 KB L2 Cache) that we use as a proxy for mobile/handheld devices.[5]

Figure 12 shows these results. On the desktop systems and laptop, SplitScreen performs roughly 2x better than ClamAV. On the embedded system, SplitScreen performs 30% better than the baseline ClamAV. The modest performance gain was a result of the very small L2 cache on the embedded system.

However, our experiments indicate a more fundamental limitation with ClamAV on the memory-constrained AMD Geode. When we ran using the 2 million signature dataset, ClamAV exhausted the available system memory and crashed. In contrast, SplitScreen successfully oper-

[5]The AMD Geode has hardware capabilities similar to the iPhone 3GS, which has a 600 MHz ARM processor with 128 MB of RAM.

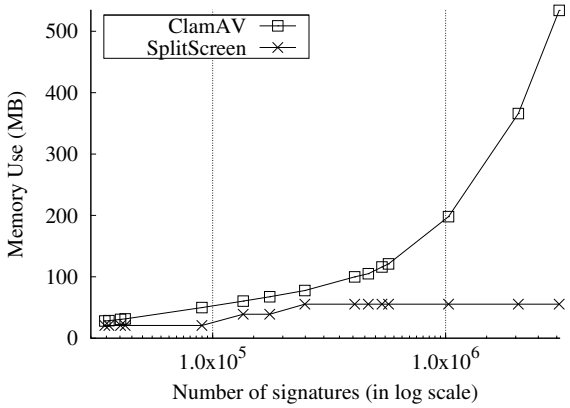

Figure 11: Memory use of SplitScreen and ClamAV.

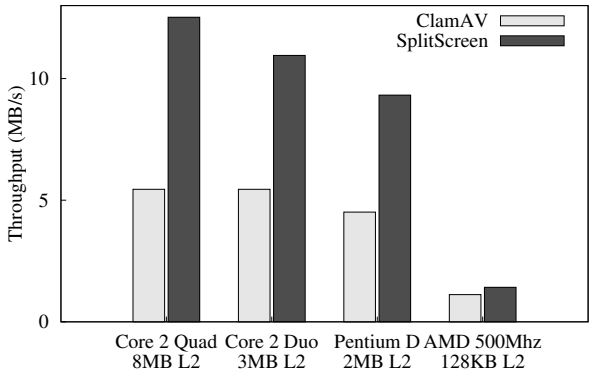

Figure 12: Performance for four different systems (differing CPU, cache, and memory size).

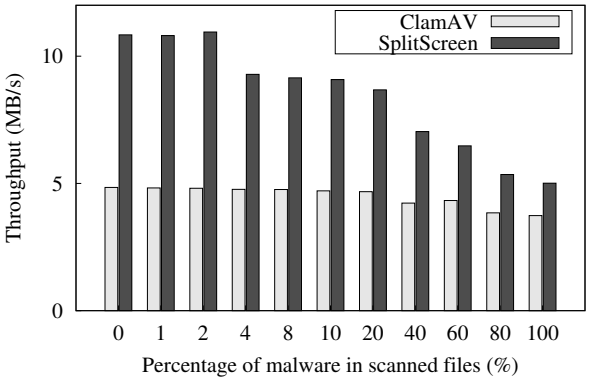

Figure 13: Throughput as % of malware increases (using total scan time including verification).

ated using even the 3 million signature dataset. These results suggest that SplitScreen is a more effective architecture for memory-constrained devices.

5.6 SplitScreen Network Performance

In the network-based setting there are three data transfers between server and client: 1) the initial bit vector ϕ (the all-patterns bit vector) generated by FFBF-INIT sent from the server to the client; 2) the bit vector ϕ' (the matched-patterns bit vector) for signatures matched by FFBF-SCAN sent by the client to the server; and 3) the set of signatures Σ' needed for FFBF-VERIFY sent by the server to the client.

Recall that SplitScreen compresses the (likely-sparse) bit vectors before transmission. The compressed size of ϕ' depends upon the signatures matched and the FFBF false positive rate. Table 2 shows the network traffic and false-positive rates in different cases. The size of both ϕ' and Σ' remains small for these files, requiring significantly less network traffic than transferring the entire signature set.

Table 3 shows the size of the all-patterns bit vector ϕ, which must be transmitted periodically to clients, for increasing (gzipped) ClamAV database sizes. SplitScreen requires about 10% the network bandwidth to distribute the initial signatures to clients.

Overall, the volume of network traffic for SplitScreen ($|\phi| + |\phi'| + |\Sigma'|$) is between 10%-13% of that used by ClamAV on a fresh scan. On subsequent scans SplitScreen will go out and fetch new ϕ' and Σ' if new signatures are matched (e.g., the ϕ' of a new scan has different bits set than previous scans). However, since $|\Sigma'| \ll |\Sigma|$, the total lifetime traffic is still expected to be very small.

5.7 Malware Scanning

How does the amount of malware affect scan throughput? We created a 100 MB corpus using different ratios of malware and clean PE files. Figure 13 shows that SplitScreen's performance advantage slowly decreases as the percentage of malware increases, because it must re-scan a larger amount of the input files using the exact signatures.

5.8 Additional SplitScreen Parameters

In addition to the FFBF size (§5.4), we measured the effect of different hash window sizes and the effectiveness of using document frequency to select good tokens for regular expression signatures.

Fixed string selection and document frequency. The better the fixed string selection, the lower the false positive rate will be, and thus the better SplitScreen performs. We use the document frequency (DF) of known good programs to eliminate fixed strings that would cause false positives. Our experiments were conducted using the known clean binaries as described in §5.1. We found the performance increase in Figure 13 was in part due to DF removing substrings that match clean files. We did a subsequent test with 344 MB of PE files from our data set. Without document frequency, we had a 22% false pos-

Target File Types	Size of target files	Number of target files	$\|\phi'\|$ (Bytes)	$\|\Sigma'\|$ (Bytes)	Total traffic (Bytes)	False-positive rate
Randomly generated	200 MB	1,000	80	405	485	0.50%
Randomly generated	2 GB	10,000	224	223	447	0.14%
Clean PE files	340 MB	1,957	1,829	15,082	16,911	4.19%
Clean ELF files	157 MB	1,319	180	11,766	13,338	9.26%
100% Malware	170 MB	534	17,100	160,828	177,928	N/A
100% Malware	1.1 GB	5277	61,748	648,962	710,710	N/A

Table 2: Network traffic for SplitScreen using 530K signatures.

# signatures	ClamAV CVD (MB)	FFBF + Short Sigs (MB)
130K	9.9	0.77
245K	13.5	1.2
530K	20.8	2.0

Table 3: Signature size initially sent to clients.

Window Size	Avg. F-P	Max. F-P	# Short Sigs
8 bytes	17.3	18.9	1169
10 bytes	11.6	14.3	1350
12 bytes	8.56	9.36	1624
14 bytes	6.70	7.77	2004
16 bytes	5.23	6.31	3203

Table 4: False positive rates for different window sizes. The average and maximum FP rates are from the 10-fold cross validation of DF on 1.55 GB of clean binaries.

itive rate and a throughput of 10 MB/s. With document frequency, we had a 0.9% false positive rate and 12 MB/s throughput. We also performed 10-fold cross validation to confirm that document frequency is beneficial, with the average and max false positive rate per window size shown in Table 4.

Window size. A shorter hash window results in fewer short regexp signatures, but increases the false positive rate. The window represents the number of bytes from each signature used for FFBF scanning. For example, a window of 1 byte would mean a file would only have to match 1 byte of a signature during FFBF-SCAN. (The system ensures correctness via FFBF-VERIFY.)

Using an eight-byte window, hash collisions caused a 3.98% of files to be mis-identified as malware in FFBF-SCAN that later had to be weeded out during FFBF-VERIFY. With a sixteen-byte window, the false positive rate was only 0.46%. The throughput for an 8 and 16 byte window was 9.44 MB/s and 8.67 MB/s, respectively. Our results indicate a window size of 12 seems optimal as a balance between the short signature set size, the false positive rate, and the scan rate.

5.9 Comparison with HashAV

The work most closely related to ours is HashAV [10]. HashAV uses Bloom filters as a first pass to reduce the number of files scanned by the regular expression algorithms. Although there are many significant differences between SplitScreen and HashAV (see §7), HashAV serves as a good reference for the difference between a typical Bloom scan and our FFBF-based techniques.

To enable a direct comparison, we made several modifications to each system. We modified SplitScreen to ignore file types and perform only the raw scanning supported by HashAV. We disabled MD5 signature computation and scanning in SplitScreen to match HashAV's behavior. We updated HashAV to scan multiple files instead of only one. Finally, we changed the evaluation to include only the file types that HashAV supported. *It is important to note that the numbers in this section are not directly comparable to those in previous sections.* HashAV did not support the complex regexp patterns that most frequently show up in SplitScreen's small signatures set, so the performance improvement of SplitScreen over ClamAV appears larger in this evaluation that it does in previous sections.

Figure 14 shows that with 100K signatures, SplitScreen performs about 9× better than HashAV, which in turn outperforms ClamAV by a factor of two. SplitScreen's performance does not degrade with an increasing number of signatures, while HashAV's performance does. One reason is SplitScreen is more cache friendly; with large signature sets HashAV's default Bloom filter does not fit in cache, and the resulting cache misses significantly degrade performance. If HashAV decreased the size of their filter, then there would be many false positives due to hash collisions. Further, HashAV does not perform verification using the small signature set as done by SplitScreen. As a result, the data structure for exact pattern matching during HashAV verification will be much larger than during verification with SplitScreen.

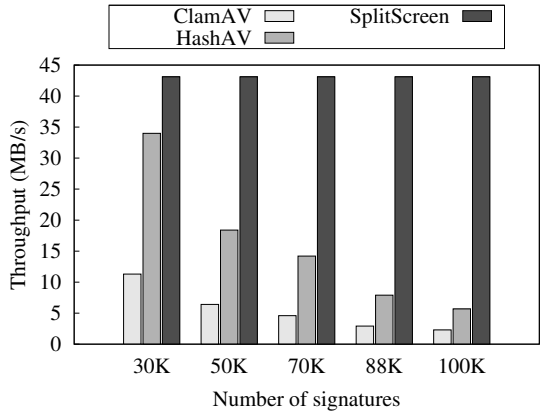

Figure 14: HashAV and SplitScreen scan throughput.

6 Discussion

We see the SplitScreen distributed model providing benefits in several scenarios, beyond the basic speedup provided by our approach. As shown in §5.6, a SplitScreen client requires 10× less data than a ClamAV client before it can start detecting malware. Furthermore, sending a new signature takes 8 bytes for SplitScreen (remember from §3.5.1 that all the FFBF bits corresponding to a signature are generated from just two independent 32-bit hashes) and 20 to 350 bytes on ClamAV. These factors make SplitScreen more effective in responding to new malware because there is less pressure on update servers, and clients get updates faster. The other advantage to dynamically downloading signatures is that SplitScreen can be installed on devices with limited storage space, like residential gateways or mobile devices.

In the SplitScreen distributed anti-malware model, the server plays an active role in the scanning process: it extracts relevant signatures from the signature database for every scan that generates suspect files on a client. Running on an Intel 2.4 GHz Core 2 Quad machine, the unoptimized server can sustain up to 14 requests per second (note that every request corresponded to a scan of 1.5 GB of binary files, so the numbers of suspect files and signatures were relatively high). As such, a single server can handle the virus scanning load of a set of clients scanning 21 GB/sec of data. While this suffices for a proof-of-concept, we believe there is substantial room to optimize the server's performance in future work: (1) Clients can cache signatures from the server by adding them to their short signatures set; (2) the server can use an indexing mechanism to more rapidly retrieve the necessary signatures based upon the bits set in the matched-patterns bit vector; (3) conventional or, perhaps, peer-to-peer replication techniques can be easily used to replicate the server, whose current implementation is CPU intensive but does not require particularly large amounts of disk or memory. These improvements are complementary to our core problem of efficient malware scanning, and we leave them as future work.

7 Related Work

CloudAV [19] applies cloud computing to anti-virus scanning. It exploits 'N-version protection' to detect malware in the cloud network with higher accuracy. Its scope is limited, however, to controlled environments such as enterprises and schools to avoid dealing with privacy. Each client in CloudAV sends files to a central server for analysis, while in SplitScreen, clients send only their matched-patterns bit vector.

Pattern matching, including using Bloom filters, has been extensively studied in and outside of the malware detection context. Several efforts have targeted network intrusion detection systems such as Snort, which must operate at extremely high speed, but that have a smaller and simpler signature set [21]. Bloom filters are a commonly-proposed technique for hardware accelerated deep packet inspection [9].

HashAV proposed using Bloom filters to speed up the Wu-Manber implementation used in ClamAV [10]. They show the importance of taking into account the CPU caches when designing exact pattern matching algorithms. However, their system does not address all aspects of an anti-malware solution, including MD5 signatures, signatures shorter than the window size, cache-friendly Bloom filters when the data size exceeds cache size, and reducing the number of signatures in the subsequent verification step. Furthermore, the SplitScreen FFBF-based approach scales much better for increases in the number of signatures.

A solution for signature-based malware detection in resource constrained mobile devices had previously been presented in [22]. Similarly to SplitScreen, it used signature fragment selection to accelerate the scanning, but could only handle fixed byte signatures, and was less memory efficient than SplitScreen.

The "Oyster" ClamAV extensions [17] replaced ClamAV's Aho-Corasick trie with a multi-level trie to improve its scalability, improving throughput, but did not change its fundamental cache performance or reduce the number of signatures that files must be scanned against.

8 Conclusion

SplitScreen's two-phase scanning enables fast and memory-efficient malware detection that can be decomposed into a client/server process that reduces the amount of storage on, and communication to, clients by an order of magnitude. The key aspects that make this design work are the observation that most malware signatures are never matched—but must still be detectable—combined with the feed-forward Bloom filter that re-

duces the problem of malware detection to scanning a much smaller set of files against a much smaller set of signatures. Our evaluation of SplitScreen, implemented as an extension of ClamAV, shows that it improves scanning throughput using today's signature sets by over 2×, using half the memory. The speedup and memory savings of SplitScreen improve further as the number of signatures increases. Finally, the efficient distributed execution made possible using SplitScreen holds the potential to enable scalable malware detection on a wide range of low-end consumer and handheld devices.

Acknowledgements

We would like to thank Pei Cao and Ozgun Erdogan for helpful discussions and feedback, as well as for making the source code to HashAV available. We would also like to thank Siddarth Adukia, the anonymous reviews and our shepherd for their helpful comments. This work was supported in part by gifts from Network Appliance, Google, and Intel Corporation, by grant CNS-0619525 from the National Science Foundation, and by CyLab at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office. The views expressed herein are those of the authors and do not necessarily represent the views of our sponsors.

References

[1] F-secure: Silent growth of malware accelerates. http://www.f-secure.com/en_EMEA/security/security-lab/latest-threats/security-threat-summaries/2008-2.html.

[2] Symantec global internet security threat report. http://www.symantec.com/about/news/release/article.jsp?prid=20090413_01.

[3] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Comm. of the ACM*, 18: 333–340, 1975.

[4] S. Ballmer. http://www.microsoft.com/msft/speech/FY07/BallmerFAM2007.mspx, 2007.

[5] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Comm. of the ACM*, 13:422–426, 1970.

[6] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Comm. of the ACM*, 20:762–772, 1977.

[7] A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. In *Internet Mathematics*, pages 636–646, 2002.

[8] J. D. Cohen. Recursive hashing functions for n-grams. *ACM Transactions on Information Systems*, 15(3):291–320, 1997.

[9] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood. Deep Packet Inspection Using Parallel Bloom Filters. *IEEE Micro*, 24:52–61, January 2004.

[10] O. Erdogan and P. Cao. Hash-AV: fast virus signature scanning by cache-resident filters. *International Journal of Security and Networks*, 2:50, 2007.

[11] R. M. Karp and M. O. Rabin. Efficient Randomized Pattern-Matching Algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[12] A. Kirsch and M. Mitzenmacher. Less hashing, same performance: Building a better Bloom filter. *Random Structures & Algorithms*, 33(2):187–218, 2008.

[13] T. Kojm. Clamav. URL http://www.clamav.net.

[14] T. Kojm. Introduction to ClamAV. http://www.clamav.net/doc/webinars/Webinar-TK-2008-06-11.pdf, 2008.

[15] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *Proc. of the 18th USENIX Security Symposium*, 2009.

[16] P.-c. Lin, Z.-x. Li, Y.-d. Lin, Y.-c. Lai, and F. Lin. Profiling and accelerating string matching algorithms in three network content security applications. *IEEE Comm. Surveys & Tutorials*, 8:24–37, April 2006.

[17] Y. Miretskiy, A. Das, C. P. Wright, and E. Zadok. Avfs: An on-access anti-virus file system. In *Proc. of the 13th USENIX Security Symposium*, 2004.

[18] I. Moraru and D. G. Andersen. Fast Cache for Your Text: Accelerating Exact Pattern Matching with Feed-Forward Bloom Filters. Technical Report CMU-CS-09-159, Carnegie Mellon University, 2009.

[19] J. Oberheide, E. Cooke, and F. Jahanian. CloudAV: N-Version Antivirus in the Network Cloud. In *Proc. of the 17th USENIX Security Symposium*, 2008.

[20] G. Ollmann. The evolution of commercial malware development kits and colour-by-numbers custom malware. *Computer Fraud & Security*, 2008(9):4 – 7, 2008.

[21] H. Song, T. Sproull, M. Attig, and J. Lockwood. Snort offloader: a reconfigurable hardware NIDS filter. *International Conference on Field Programmable Logic and Applications, 2005.*, pages 493–498, 2005.

[22] D. Venugopal and G. Hu. Efficient signature based malware detection on mobile devices. *Mobile Information Systems*, 4(1):33–49, 2008.

[23] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical Report TR-94-17, University of Arizona, 1994.