Automatically Inferring Malware Signatures for Anti-Virus Assisted Attacks

Christian Wressnegger*, Kevin Freeman†, Fabian Yamaguchi*, and Konrad Rieck*

* Institute of System Security, TU Braunschweig
† Institute of Computer Science, University of Göttingen

Abstract
Although anti-virus software has significantly evolved over the last decade, classic signature matching based on byte patterns is still a prevalent concept for identifying security threats. Anti-virus signatures are a simple and fast detection mechanism that can complement more sophisticated analysis strategies. However, if signatures are not designed with care, they can turn from a defensive mechanism into an instrument of attack. In this paper, we present a novel method for automatically deriving signatures from anti-virus software and discuss how the extracted signatures can be used to attack sensible data with the aid of the virus scanner itself. To this end, we study the practicability of our approach using four commercial products and exemplary demonstrate anti-virus assisted attacks in three different scenarios.

Keywords
Anti-Virus; Malware; Signatures; Attacks

1. INTRODUCTION
Virus scanners are one of the most common defenses against security threats—despite well-known weaknesses and shortcomings. Millions of end hosts run these scanners on a regular basis to check files for infections and detect malicious code. Individuals, companies and even government organizations employ anti-virus software for fending off attacks at their desktop systems. The success and prevalence of these products largely build on their simple yet reliable functionality: Files are matched against a database of known detection patterns (signatures) which is regularly updated by the vendor to account for novel threats. Such pattern matching can be implemented very efficiently and is able spot all sorts of threats if appropriate and up-to-date signatures are available [see 3, 58].

Signature-based detection suffers from a well-known drawback: Unknown threats for which no signatures exist can easily bypass the detection. This problem is further aggravated by the frequent use of obfuscation in malicious code that obstructs static signature matching [14, 34, 37]. As a consequence, a large body of research has focused on developing alternative approaches for analyzing and detecting malicious software, for example, using semantic models [e.g., 10, 11], behavioral analysis [e.g., 13, 28, 31] or network traffic monitoring [e.g., 20, 21, 54]. Over the last decade, anti-virus vendors have increasingly adopted these detection mechanisms to keep up with malware evolution. Still, signatures based on byte patterns remain an integral part of security products and complement more sophisticated detection mechanisms.

Attackers are well aware of the prevalence of virus scanners and the detection mechanisms they employ. Thus, an adversary might not only seek means for evasion but also take advantage of their presence, for example, by injecting signatures into benign content. In 2014 the Bitcoin blockchain contained the signature of the DOS/STONED malware, tricking the Microsoft Security Essentials into falsely identifying the file as malware and removing it [62]. The nature and origin of the incident is unclear and so is the fact whether this has been an isolated proof-of-concept, driven by the manual effort of reverse engineering malware signatures or part of an automated attack.

In this paper, we systematically explore the feasibility of such anti-virus assisted attacks. We introduce a novel method for automatically deriving signatures from commercial virus scanners which is agnostic to the particular implementation and provides an adversary with byte patterns that approximate the original signatures without the need to reverse engineer the analysis engine or signature database. With these patterns at hand, the attacker can draw a virus scanner’s attention to benign data and flag chosen content as malicious, and thereby selectively block access or delete files. Moreover, we show that such malicious markers do not have to be injected directly, but may be infiltrated as unprivileged user in a way that these end up in the local data store, that then gets flagged by the virus scanner.

To assess the feasibility of this attack in practice, we apply our method for signature derivation to four commercial anti-virus products in an empirical study. We find that on average 38% of the derived signatures can be approximated by simple combinations of byte patterns. Several of these patterns match text strings, artifacts of packers or environment checks, and are mostly unrelated to the semantics of the considered malicious code. Furthermore our study shows that only 8% of such signatures match patterns, that are detected by more than one anti-virus product considered in our experiments, enabling an attacker to play off gateway and end-user security solutions against each other by crafting
target-specific inputs. Finally, we make use of the inferred malicious markers to demonstrate anti-virus assisted attacks in three different scenarios: 1) covering up password guessing, 2) deleting a user’s emails and 3) facilitating web-based attacks by removing browser cookies.

In summary we make the following contributions:

- **Anti-virus assisted attacks.** We introduce a new class of anti-virus assisted attacks that limit access to benign data by abusing byte-pattern based signatures and demonstrate their feasibility in different scenarios.

- **Automatically deriving signatures.** We present a novel method for automatically deriving pattern-based signatures from virus scanners without the need for manually reverse engineering the software.

- **Identification of inadequate signatures.** In an empirically study with four commercial anti-virus products, we identify overly simplistic signatures that build on short byte patterns of resource or code artifacts and thus are well-suited for the described attacks.

The remainder of the paper is structured as follows: In Section 2 we describe the attack vector and present a brief overview of anti-virus signatures. We introduce our method for deriving signatures in Section 3 and empirically study its application and results in Section 4. In Section 5 we then present specific attacks based on malicious markers and discuss mitigations in Section 6. Limitations and related work are discussed in Section 7 and Section 8, respectively. Section 9 concludes the paper.

2. ANTI-VIRUS ASSISTED ATTACKS

Over the last years, numerous software flaws have been identified in popular anti-virus products [e.g., 25, 42]. For example, Ormandy disclosed vulnerabilities in products of ESET [43], Kaspersky [44], AVG [46], FireEye [45] and more recently Symantec/ Norton [47]. These findings are especially critical as the corresponding products are employed in many network and host defenses. According to leaked internal documents, even the NSA and GHCQ have set their hands on anti-virus software to infiltrate networks [16].

We advance this line of work and present a novel class of anti-virus assisted attacks, called malicious markers, that does not rely on exploiting vulnerabilities but is based on the weak design of pattern-based signatures. Although modern anti-virus products comprise several different detection mechanisms, signatures based on byte patterns are still widely used as a complementary tool. In the following, we introduce malicious markers as attack vector (Section 2.1) and look upon anti-virus signatures that can be used to execute this unique type of attack (Section 2.2).

2.1 Attack Vector

Any initially benign data that at some point in time contains malware or its payload clearly has to be considered harmful by an AV product. A file can thus be trivially flagged as malicious by injecting malware. However, in case only a couple of carefully selected byte patterns are added to a file that do not represent malicious functionality but happen to match a signature deployed by virus scanners, the situation is not necessarily as clear anymore.

Ideally, malware signatures capture malicious functionality. In practice however, byte-pattern based signatures often match resources, wrongly aligned code, import tables or compressed data. Even worse, if the virus scanner does not consider additional context for its decision, signatures can be freely moved to other files. Malicious markers rely on implanting such overly simplistic malware signatures into benign data to raise false alarms.

In the most trivial case an attacker would achieve such an implant by directly manipulating the targeted files. However, having direct (privileged) access would give rise to several other, way more powerful attacks than deleting or restricting access to data. Malicious markers on the contrary allow to operate on a more subtle setting where a remote attacker has no direct access but is able to remotely “contribute” to a local file. Inadequate signatures that do not consider the context of the matched bytes are smuggled through input validation of common network protocols and end-user applications to trigger false-positives on the client-side.

Furthermore, such minimal triggers enable an attacker to play off gateway and end-user security solutions against each other if products from distinct vendors are used.

2.2 Anti-Virus Signatures

Anti-virus software comprises a wide range of different analysis and detection techniques, including batch processing, on-access analysis, behavioral blocking and scheduled updating of signatures. While the underlying analysis engines often build on sophisticated concepts, such as bytecode interpreters [42] and other Turing-complete representations [8], the actual signature matching typically boils down to three basic strategies: byte patterns, hash sums, and in the widest sense of the phrase, similarity measures. In the following, we describe each of these strategies in more detail to identify types that are suitable for malicious markers and set the scope for our approach of deriving signatures.

2.2.1 Signatures based on Byte Patterns

The most common strategy for signature-based detection is the use of byte patterns. These signatures contain one or more constant sequence of bytes, possibly separated by gaps of varying size, that need to match with the content of a file to trigger a detection. These signatures can be efficiently matched using classic string-processing techniques, such as the Aho-Corasick algorithm [1, 22]. To this end, sets of strings are represented as a Trie that serves as a finite state machine for determining matches. This representation allows to efficiently model wildcards, ranges and character classes, thereby providing capabilities similar to those of regular expressions.

*Example of byte patterns.* The open-source scanner ClamAV defines a simple format for representing byte patterns, including disjunctions (aa|bb), gaps (n=\(a\)), wildcards ‘?’ and character classes [57]. Figure 1(a) shows a simplified version of such a signature for the Virut malware family. While ClamAV can generally match arbitrary data, in case of the provided example the signature describes x86 code that corresponds to a simple decryption routine. Figure 1(b) shows one instance matched by the above signature.

Formally, pattern-based signatures simply are a subset of regular expressions, yet for the purpose of signature derivation, we choose a different formal description that focuses entirely on their appearance. We observe that pattern-based signatures are given by sequences of disjunctions over bytes,
possibly separated by gaps of varying size. Expressing each
disjunction as a set containing its alternatives, a signature
becomes a sequence of sets. We can describe this formally by
defining a symbolic alphabet \( S = \{0, \ldots, 255\} \), where \( \mathcal{P} \)
is the power set and corresponds to all possible subsets of byte
values. Additionally, we use \( \ast \) as a shortcut for \( \{0, \ldots, 255\} \)
to represent irrelevant bytes. Each word \( w \in \mathcal{S}^* \) of the corre-
sponding language then already expresses the format of a
signature. However, these words alone do not account for
the varying sizes of gaps. We hence introduce two functions,
\( t : \{1, \ldots, |w|\} \rightarrow \mathbb{N} \) and \( h : \{1, \ldots, |w|\} \rightarrow \mathbb{N} \) that assign a
minimum and maximum number of repetitions to each of
\( w \)’s symbols. A pattern-based signature is then simply given
by the tuple \( s = (w, t, h) \).

2.2.2 Signatures based on Hash Sums
A second strategy frequently implemented by anti-virus
software is matching based on hash sums. To this end, hash
sums over complete files or parts of files are calculated and
stored in the signature database. Simple checksums such as
CRC32 or cryptographic hash functions such as MD5 or
SHA1 are often used due to the availability of fast implemen-
tations in both software and hardware [24, 42, 57]. While
hash collisions may theoretically result in false positives, in
practice this is not an issue in this particular field of appli-
cation, making it an attractive choice for many vendors.
In comparison to byte patterns, hash sums enable matching
large regions in a file with a compact signature. This
approach, however, does not allow for wildcard characters
or gaps, and therefore provides no means to match largely
similar files with a single signature. In consequence, individ-
ual hash-based signatures are required for even the slightest
variations of known malware samples.

Conceptually, hash-based signatures can however be inter-
preted as continuous byte patterns and thus the signatures
can be described using the same formal description presented
in Section 2.2.1. In particular, a hash sum with given offset
and length can be represented as a continuous byte pattern
that is preceded by a gap of fixed size.

Example of hash sums. The open-source scanner ClamAV
enables to define hash-based signatures either as hash sum
of the complete file or specifically for PE files over individual
sections [57]. Figure 2 illustrates both types of hash sums for
the malware Kido, where in the first case the length of the
matching region is specified in the second parameter (162970)
and in the latter case in the first parameter (81920). The
type of the hash function is derived by ClamAV based on
the size of the hash sum or the database file it is stored in.

2.2.3 Similarity Measures
Aside from byte patterns and hash sums, anti-virus soft-
ware often employs other measures to identify security threats,
that allow to more generally describe a malware binary or its
behavior. These may include the inspection of instruction
counts, the analysis of API usage, n-gram detection models,
and in some cases even machine learning techniques. In com-
parison to signatures based on byte patterns and hash sums,
these detection approaches are often bound to a concrete
execution context and can capture complex semantics, such as
different events necessary to trigger an infection. Due to
this complexity such detection schemes are less suitable for
constructing malicious markers and we thus put our focus
solely on pattern-based signatures.

3. INFERRING MALWARE SIGNATURES
A classic approach for deriving signatures from virus scan-
cers is reverse engineering the respective analysis engines
and dumping the signature database [42]. Such reverse engi-
nering, however, relies on tedious manual work and needs
to be repeated for each individual scanner. We in contrast
introduce a simple and intuitive, yet generic method that is
gnostic to the inspected virus scanner: Given a set of known
malware samples, we strategically create modifications of
these files and scan them to see whether the scanner still
flags them as malicious. This procedure allows us to derive
a signature by piecing together bytes that are observed to be
relevant for detection over different samples and runs.
Although simple in design this method is sufficient to obtain
suitable markers for anti-virus assisted attacks as introduced
in Section 2.

Our method for the derivation of signatures executes the
following three steps which can be applied to any possible
virus scanner:

1. \textit{Detecting relevant bytes}. First, we determine relevant
bytes in each malware sample by utilizing feedback
from the virus scanner over multiple runs (Section 3.1).

2. \textit{Sequence alignment and merging}. We proceed to align
the relevant bytes from samples with the same signature
and merge them into a single sequence (Section 3.2).

3. \textit{Creation of signatures}. Finally, we transform the merged
sequences into a valid signature format, yielding the
final signature (Section 3.3).

In the following, we discuss each of these steps in detail and
describe the optimizations we perform compared to a naive
implementation.
3.1 Detecting Relevant Bytes

We begin our analysis by processing each malware sample independently to derive bytes relevant for detection. To achieve this, the main idea is to simply flip—bitwise negate—one byte after another and run the target virus scanner on the resulting file. If the modified file does not trigger an alarm, the changed byte is relevant for detection and thus we include the original value of that byte in our signature. If in contrast, the scanner shows no reaction to our modification, the byte seems to be irrelevant.

There are three problems with this approach in practice. First, determining the exact signature requires to exhaustively test all possible combinations rather than simply flipping bytes. Second, the largest portion of a virus scanner’s runtime is taken up by its initialization. Passing each file variation to the scanner separately therefore induces a high runtime overhead. Third, a naïve implementation of this approach requires quadratic disk space and its runtime is dominated by the disk’s I/O operations.

As a remedy, we restrict the derivation to signatures that match all samples in our dataset of the same malware family but are not necessarily complete. For the application as malicious markers this however is a sufficiently good approximation. Moreover, we reduce the runtime overhead induced by the scanner’s initialization by passing samples to the scanner in large batches rather than separately. We finally address the third problem by formulating the following algorithm that follows a divide-and-conquer approach to perform byte flipping: The complete file is first divided into \( k \) partitions. For each partition, all bytes are flipped and the virus scanner is applied to the modified file. If a partition does not contain relevant bytes (the scanner still detects the malware) no further inspection is needed and the complete partition is considered to be irrelevant for the signature. Otherwise, the same procedure is recursively applied until partitions can no longer be divided. In effect, large portions of a file can be quickly marked as irrelevant and dismissed.

The procedure presented thus far is well suited for signatures based on byte patterns but inefficient when dealing with hash-based signatures, particularly if the hash sum is calculated over the entire file. Our approach would flip all \( n \) bytes individually and requires to scan the resulting \( n \) files in order to derive this relation. To speed up this process we introduce two thresholds \( t_b \) and \( t_p \). The first relates to the ratio of bytes considered relevant during one iteration of the divide-and-conquer process. The second specifies the partition sizes for which our heuristic is applied: if \( t_b = 99\% \) of a partition with at least \( t_p = 25,000 \) bytes are marked as relevant, we conclude that we are dealing with a hash-based signature and focus the search on the edges of the partition to determine the exact region the hash is calculated on. This allows us to decide whether or not we are dealing with a hash-based signature at an early stage and accelerate the process significantly.

As a result of this analysis, we obtain a preliminary signature \( s = (w, l, h) \) for each sample \( x \) composed of \( m \) bytes \( x_1, \ldots, x_m \). We construct this signature by first creating a format description \( w \) where \( w_i = \ast \) if \( x_i \) corresponds to an irrelevant byte, and \( w_i = \{ x_i \} \) if \( x_i \) is a relevant byte. We proceed to create a compressed form of \( w \) referred to as \( w' \) by scanning \( w \) from left to right and merging consecutive irrelevant bytes (gaps). Relevant bytes, however, are preserved for the sake of signature alignment as described in Section 3.2. We additionally store information about the length of the gaps as the minimum and maximum number of repetitions. That is, for each byte \( w_i \), we set \( l(i) \) and \( h(i) \) to the number of symbols of \( w \) merged to obtain \( w_i \). The resulting signature may still be changed in the next step to account for information contained in other samples tagged with the same signature label.

3.2 Sequence Alignment

We proceed by grouping malware samples according to the label assigned by the scanner. For a given group of corresponding signatures \( X \), we then create a joint signature by employing the Needleman-Wunsch algorithm [39]. That is, we align their corresponding format descriptions given by the set \( \{ w \mid (w, l, h) \in X \} \). During this procedure we ignore the minimum and maximum number of repetitions encoded by the functions \( l \) and \( h \) at first and merely compare the signature’s format descriptions.

Given two strings \( u \) and \( w \), the Needleman-Wunsch algorithm attempts to align these strings, that is, it creates new strings \( \hat{u} \) and \( \hat{w} \) from \( u \) and \( w \) respectively by introducing an arbitrary number of irrelevant bytes denoted as \( \ast \) between bytes of \( u \) and \( w \). These additional irrelevant bytes serve as padding, ensuring that \( \hat{u} \) and \( \hat{w} \) are of same size. The algorithm attempts to ensure that \( \hat{u}_i \) is equal to \( \hat{u}_i \) for as many of the strings’ positions \( i \) as possible. For merging the resulting alignment we combine the length specifications denoted by \( l \) and \( h \) and extend the bounds such that the range is maximized. Moreover, for positions \( i \) where \( \hat{u}_i \) and \( \hat{w}_i \) are unequal, we simply merge the sets \( \hat{u}_i \) and \( \hat{w}_i \) by applying the union operator. To create a signature for an entire set of strings, we iteratively apply the Needleman-Wunsch algorithm to merge one signature at a time with the preliminary version of our final signature.

To further detail the range specifications one can employ a similar approach as used for identifying relevant bytes (see Section 3.1). We insert or delete dummy bytes at the locations of irrelevant bytes to determine the number of bytes that may occur in between sequences of relevant bytes and apply the virus scanner to it. To this end, we again make use of a (binary) divide-and-conquer strategy in order to speed up the process. While this approach provides a precise solution, this strategy is time consuming and thus we omit this step for our experiments and exclusively rely on the alignment described above.

As a result of this step, we obtain a signature for each group of samples, that is, a tuple \( (w, l, h) \) where \( w \) describes the signatures format as a sequence of disjunctions over bytes, and \( l(i) \) and \( h(i) \) are the minimum and maximum number of repetitions of the \( i \)-th symbol in \( w \).

3.3 Creation of Signatures

We finally construct signatures compliant with those used by ClamAV and in accordance with our formal definition. To this end, we join the values of each symbol \( w_i \) using the alternate operator ‘|’ surrounded by brackets, which may be omitted if a symbol contains one value only. Additionally, we annotate this specification of a symbol with its minimum and maximum number of repetitions expressed as ranges \( \langle n-m \rangle \), where \( n \) and \( m \) are specified by a signature’s functions \( l \) and \( h \). Strictly specified ranges such as \( \langle n-n \rangle \) are simplified to \( \langle n \rangle \). Repetitions of exactly one, \( \langle 1-1 \rangle \) and \( \langle 1 \rangle \) respectively, are omitted yielding a final signature as shown in Figure 3.
4. EMPIRICAL STUDY

Equipped with a method for automatically deriving signatures from virus scanners, we conduct an empirical study with four popular anti-virus products and the open-source scanner ClamAV. As we are not interested in comparing individual security vendors but in gaining insights into used signatures, we reference these scanners in the following as AV1 to AV5, with ClamAV being AV1 as our baseline. In particular, we carry out four experiments on a recent malware dataset that is detailed in Section 4.1:

- First, we demonstrate the viability of the proposed method for signature derivation in a controlled experiment using ClamAV (Section 4.2).
- Second, we apply our method to the virus scanners we do not have ground truth for and perform a quantitative study of the derived signatures (Section 4.3).
- Third, we investigate whether or not signatures from different vendors overlap, that is, cover identical malware bytes, and if so, to which extend (Section 4.4).
- Fourth, we study the quality of deployed signatures with special focus on semantics and whether or not they are bound to a specific context (Section 4.5).

4.1 Malware Dataset

The quality of derived signatures hinges on a representative dataset of malware for deriving corresponding signatures. We thus collect 9,969 malware samples that are detected by all five scanners considered in our evaluation. In particular, we have been given access to submissions to the VirusTotal service, allowing us to gather a broad selection of recently submitted files. A brief overview of the dataset is presented in Table 1. The vast majority of the files are applications or dynamic libraries in the Portable Executable format. The remaining files correspond to archives, Windows shortcuts and other carriers of malicious code. Depending on the applied virus scanner, the files in the dataset are assigned to roughly 250 malware families, where the concrete number of different signature labels assigned by the scanners ranges from 277 up to 1,327 (see the first column in Table 2).

4.2 Controlled Experiment: ClamAV

In the first experiment, we evaluate our approach in a controlled setup using the open-source scanner ClamAV for which all signatures are publicly available and we thus have ground truth to assess the reconstruction capabilities of our approach. To this end, we first derive signatures for all samples in our datasets and then compare the output to the corresponding signatures of ClamAV using the string edit distance [32]. In this experiment we focus on static pattern-based signatures as returned by our method, but skip hash-based signatures as these are not suitable for being used as malicious markers (cf. Section 2). Moreover, we replace gaps in all signatures with a generic wildcard to account for minor differences in the gap ranges.

Figure 4 presents the results of this comparison as the distribution of the observed distances to the original signatures. To achieve a comparison between signatures of different lengths, we normalize the edit distance $d(x, y)$ between two signatures $x$ and $y$ as follows:

$$d(x, y) = \frac{d(x, y)}{\max(|x|, |y|)} \in [0, 1]$$

where $|x|$ and $|y|$ correspond to the lengths of the signatures. The resulting distribution of normalized distances has a sharp peak close to 0.0 and roughly 50% of all derived signatures match their counterpart in the ClamAV database almost perfectly. Moreover, 75% of the derived byte patterns differ from the original signature by at most 4% of the content and only a minor fraction cannot be correctly derived by our method.

As an example of this experiment, let us re-consider the derived signature shown in Figure 3. The two expressions are almost identical except for the third disjunction, where the derived signature misses two out of five possible bytes (indicated as spaces). As shown in Figure 1, this signature corresponds to a simple decryption loop that is varied by the malware authors using different instructions and registers. Our dataset comprises samples of the Virut malware with three of the five variants encoded in the ClamAV signature and thus the derived signature does not match its counterpart exactly. Still, the rest of the byte patterns and their disjunctions are precisely uncovered by our approach. In addition to the successful derivations, we also inspect...
cases where our method fails to arrive at a good approximation of the original signature. One example is a variant of the malware family \textit{Adware.Somoto} which \textit{ClamAV} identifies using a compact byte pattern that our approach misses. A closer investigation reveals that the corresponding sample is a self-extracting archive of the \textit{Nullsoft Scriptable Install System} (NSIS) for which \textit{ClamAV} introduced an unpacker in version 0.91. Dynamic analyses and unpacking are out of scope of our method.

### 4.3 Quantitative Study: AV2–AV5

We proceed to derive signatures for the remaining four commercial anti-virus products. The setup is kept identical to the previous experiment with the only exception that we do not have access to the ground truth for the original signatures. We therefore focus on statistically assessing the quality and content of the derived signatures.

<table>
<thead>
<tr>
<th>Derived Signatures</th>
<th>#Labels</th>
<th>Hash sums</th>
<th>Byte patterns</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{AV1}</td>
<td>1327</td>
<td>940 (71%)</td>
<td>377 (28%)</td>
<td>10 (1%)</td>
</tr>
<tr>
<td>\textit{AV2}</td>
<td>277</td>
<td>1 (0%)</td>
<td>255 (99%)</td>
<td>21 (8%)</td>
</tr>
<tr>
<td>\textit{AV3}</td>
<td>522</td>
<td>323 (62%)</td>
<td>69 (13%)</td>
<td>130 (25%)</td>
</tr>
<tr>
<td>\textit{AV4}</td>
<td>282</td>
<td>178 (63%)</td>
<td>93 (33%)</td>
<td>11 (4%)</td>
</tr>
<tr>
<td>\textit{AV5}</td>
<td>586</td>
<td>177 (30%)</td>
<td>353 (60%)</td>
<td>56 (10%)</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>598</td>
<td>323 (54%)</td>
<td>229 (38%)</td>
<td>45 (8%)</td>
</tr>
</tbody>
</table>

**Table 2: Results of signature derivation.**

Table 2 provides a first overview of the derived signatures, where \textit{AV1} corresponds to \textit{ClamAV} and the remaining scanners to commercial products. The scanners considerably differ in type and amount of signatures that can be derived. On average, our approach is able to infer 38% pattern-based signatures and 54% hash-based signatures, whereas only the remaining 8% cannot be uncovered.

\textit{AV2} especially stands out with 92% byte patterns and only a single hash-based signature, while for the other scanners at least one third of all signatures corresponds to hash sums. On the other end of the scale, for \textit{AV3} we fail to extract one forth of all signatures. This may indicate the use of advanced heuristics or that multiple pattern-based signatures are assigned to the same label, implicitly constructing a signature with long disjoint patterns. We discuss this in more detail in Section 7.

To illustrate the quality of the derived signatures, let us consider the byte patterns shown in Figure 5 and 6, respectively. Figure 5 shows a signature for the malware family \textit{Swizzor}. Clearly, an effort has been made to pinpoint the decryption loop of the malware (starting at line 7) that unveils the malicious payload at runtime. In particular a 5-byte long key is used to decrypt data using the XOR operator (line 8).

By contrast, the signature shown in Figure 6 contains an ASCII string that corresponds to the Windows product key of the Anubis sandbox. The instructions at lines 6–11 are part of a call to the \texttt{RegOpenKeyExA} API function. This signature captures a simple environment check used by several backdoors that compares the product key of the execution environment against known sandbox systems.

Figure 5: Derived signature for the \textit{Swizzor} malware: The byte patterns match a XOR-based decryption loop (line 7) with a 5-byte key (line 11).

Figure 6: Derived signature for a generic backdoor: The byte patterns correspond to the Windows product key of the Anubis sandbox (line 2–5).

Not all of the signatures we derive can be interpreted as clearly as these examples. Many signatures match resources, wrongly aligned code, import tables or even compressed data. To get a better understanding for the quality of these signatures, we statistically analyze the characteristics of the corresponding byte patterns derived by our method. Figures 7(a) to 7(c) show for each of the five scanners the distribution of the signature size, the number of gaps and the entropy of the patterns.

With respect to the size, \textit{AV2} again draws attention: It appears that this scanner makes use of very compact and equally sized patterns that contain no or only a few gaps. While short signatures generalize well, they may also induce more false positives which presumably is the reason why other scanners such as \textit{AV3}–\textit{AV5} use longer signatures—partly also with more gaps between the byte patterns. As demonstrated in Section 2 short signatures are good candidates for malicious markers.

As malicious markers might be restricted to a specific character set, it is also interesting to see how the entropy of the byte patterns is distributed. An entropy of 0.0 indicates uniform byte patterns, such as paddings, while values close to 8.0 are reached by random, encrypted or compressed data. Matched import tables reside on the lower end of the scale due to the identical significant bytes of addresses and x86 code is mostly located in the middle and upper half of the scale. Three of the scanners cover almost the full entropy range, indicating the diversity of the byte patterns in the signatures.
Finally, we inspect how well the pattern-based signatures have been derived by our method. Since we do not have ground truth, we rate the quality of the signatures by clustering them and comparing the clusters to the groups of labels the signatures correspond to. This conformity is plotted in Figure 7(d) as the F-measure for the individual anti-virus products. On average the derived signatures yield an agreement with the labels of 96%, that is, for most of the dataset, similar signatures are assigned to the same label, whereas dissimilar signatures are associated with unequal labels. While this experiment cannot prove that the derived signatures exactly match their counterparts, it demonstrates that we infer a representation that is at least closely related to the original signatures.

4.4 Signature Overlap

Although anti-virus vendors openly share samples, concrete signatures usually are not exchanged as these represent a company’s trade secret. We investigate to which extent signatures nevertheless overlap between competing anti-virus products. To this end we measure the number of (half-)bytes in relation to a signature’s length that overlap between products. Figure 8 shows the average overlap of the five considered engines to each other. Only 5.2% of the derived signatures fully overlap with the signature of another vendor in a way that both products raise an alarm on the same substring of bytes. Consequently, for 94.8% of the signatures a virus scanner raises an alarm while others do not. 21.9% even have no resemblance among different anti-virus products at all.

4.5 Context and Semantics of Signatures

As final experiment of our study, we analyze the quality of the derived signatures. We are interested in determining whether the signatures are bound to a specific context within the file and if they model semantics of the targeted malicious code. To this end, we examine how well the derived signatures can be implanted in benign files that have nothing in common with the original malware. Since our dataset is mainly comprised of PE executables, we consider a set of benign applications of the same format for this task, which we have verified not to be flagged as malicious by any of the scanners. We proceed to implant the signatures derived by our method into these applications, irrespective of the relative position to PE sections or existing code, and apply the virus scanners to the resulting files. A breakdown of the detection results Table 3. On average 68% of the implants are successful. Apparently, several of the pattern-based signatures are applied without checking the context or semantics of the matching region for plausibility, allowing the direct transfer of signatures from one file to another. The situation is especially troublesome for AV1, AV2 and AV3 that simply flag the majority of implants as malicious. AV5 appears to be the only one to use semantics-aware matching in most of the cases. Still, the overall quantity of working implants is higher than for AV3 or AV4.

<table>
<thead>
<tr>
<th>Virus scanners</th>
<th>Implantable</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV1</td>
<td>377</td>
</tr>
<tr>
<td>AV2</td>
<td>255</td>
</tr>
<tr>
<td>AV3</td>
<td>69</td>
</tr>
<tr>
<td>AV4</td>
<td>93</td>
</tr>
<tr>
<td>AV5</td>
<td>353</td>
</tr>
<tr>
<td>Average</td>
<td>229</td>
</tr>
</tbody>
</table>

Table 3: Detection results on pattern-based signatures implanted in benign PE files.

5. CASE STUDIES

We proceed to demonstrate the threat of anti-virus assisted attacks and feasibility of implanting malicious markers in three different scenarios: 1) covering up password guessing, 2) deleting a user’s emails and 3) facilitating web-based attacks by removing browser cookies. For each scenario we choose a different anti-virus product to demonstrate the generality of the approach.
Similar to shellcodes in network attacks, useful malicious markers need to consist of certain bytes to pass input validation of common network protocols and end-user applications. In these scenarios ASCII or printable characters are especially promising. We thus consider 200 PHP files from VirusTotal that are detected by all five virus scanners. Depending on the applied virus scanner the samples correspond to 28–37 malware families. For each family, we derive signatures and inspect their suitability for the use as malicious markers. Table 4 summarizes the different categories of character sets.

Although we are only operating on 30 to 40 different signatures, we are able to find suitable malicious markers for all virus scanners and character sets. The actual effort for an adversary to find suitable markers for a given application thus has to be considered as being rather low.

No logs, no crime. Similar to the empirical study presented earlier, we begin by considering the open-source scanner ClamAV as an attack target. We aim at removing traces of password guessing, for example, over SSH, POP3 or IMAP. For this purpose, we exploit that Linux and other UNIX systems keep track of failed login attempts in a log file, often called auth.log as shown in Figure 10. Note that the username which is controlled by the attacker is stored in clear in the log file. It turns out that a large range of printable characters (Printable-4) can be used as username and is written verbatim to the file.

As a consequence, an attacker may finish each iteration over a list of guessed passwords with a set of malicious markers, i.e., specially crafted login names that correspond to anti-virus signatures. If the attacked host is running a virus scanner configured to delete or quarantine viruses, any file containing such a malicious marker is deleted or at least moved to a different location. This not only makes manual investigation of the attack hard but may also inhibit the functionality of tools analyzing log files to stop password guessing, such as fail2ban [26].

Figure 9 exemplarily shows a signature used by ClamAV that is accepted as login name and allows to tag the authentication log as malicious. Once the SSH daemon writes the authentication log file, it turns out that a large range of printable characters (Printable-4) can be used as username and is written verbatim to the file.

Deletion of emails. With a slight twist, malicious markers can also be used to obstruct the delivery of emails. To illustrate this setting, we consider the commercial anti-virus product AV2 operated on Windows. As target we choose Mozilla Thunderbird, which stores the user’s emails in a variation of the mbox format family [5]. While attachments and binary data are encoded, the email body is stored verbatim in this format. This enables an attacker to smuggle in malicious markers by sending crafted emails. The adversary is free to use any ASCII encoded characters, including non-printable, but excluding ASCII extensions and the NUL-character (Printable-1). Figure 11 shows a suitable candidate for AV2 that can be used as implant in this setting.

It suffices that the attacker delivers a single email to the victim to trigger quarantining or deleting the inbox database. The crafted email does not even need to look suspicious, as the attacker may use ASCII control characters, such as \t (NP form feed, new page) or a sequence of newline characters or whitespaces, to hide the malicious marker from being displayed in clear sight in Mozilla Thunderbird and other mail clients. Note that it is not possible to simply use a complete malware binary for this attack as it likely contains non-printable characters and thus is incorrectly stored in the email database. Moreover, chances are high that the malware binary would be filtered out by the email gateway already, whereby the malicious marker most probably would not. We discuss this in more detail in Section 6.

Removing browser cookies. The presented attack vector can also be adapted to assist in web-based attacks. In this scenario our goal is to force a user to re-login into a webpage application and thereby enable tampering with the authentication process. For this setting, we use Google Chrome as the target application and consider AV2 as our partner in crime. Google Chrome stores its cookies in a simple SQLite3 database called User Data\Default\Cookies. Interestingly, while the actual value of the cookie is strictly encoded when stored in this database, the name of the cookie is not: A cookie name may hence consist out of any printable character in the range from \x20 to \x7E, except for semicolons and the equality sign (Printable-3).

An attacker may hence simply provide a specially crafted cookie, with the malicious marker as its name, on a website and lure its victim into visiting it (no evolved web-based attack, e.g., MITM, needed). On access of the webpage the anti-virus product blocks access to the cookie database, thereby forcing the user to re-authenticate with certain web

<table>
<thead>
<tr>
<th>Families</th>
<th>Derived signatures</th>
<th>ASCII characters *</th>
<th>Printable-1</th>
<th>Printable-2</th>
<th>Printable-3 *</th>
<th>Printable-4 *</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV1</td>
<td>37</td>
<td>35</td>
<td>34</td>
<td>34</td>
<td>20</td>
<td>7</td>
</tr>
<tr>
<td>AV2</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>26</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>AV3</td>
<td>28</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>AV4</td>
<td>37</td>
<td>33</td>
<td>31</td>
<td>31</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>AV5</td>
<td>36</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>12</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 9: ClamAV signature labeled PHP.ShellExec

Figure 11: AV2 signature labeled JS:Decode-BHU [Trj]
applications. Figure 12 exemplarily shows a signature used by AV products that meets the mentioned criteria of allowed characters.

The same attack vector can be used to tamper with any data that contains user-controlled information and is stored in files by the browser. Mozilla Firefox, for instance, also stores `HTML5 local storage objects` in a SQLite database and thus is also a suitable target for malicious markers in web-based communication.

For example, storing each entry of an authentication network setup is difficult and might even be impossible in the same vendor, as signatures usually are not shared among applications. Figure 12 exemplarily shows a signature used by AV5 that matches the mentioned criteria of allowed characters.

For example, storing each entry of an authentication network setup is difficult and might even be impossible in the same vendor, as signatures usually are not shared among applications. Figure 12 exemplarily shows a signature used by AV5 that matches the mentioned criteria of allowed characters.

For example, storing each entry of an authentication network setup is difficult and might even be impossible in the same vendor, as signatures usually are not shared among applications. Figure 12 exemplarily shows a signature used by AV5 that matches the mentioned criteria of allowed characters.
Repititive byte-sequences. We run into a similar problem whenever one signature matches a sequence of bytes multiple times within a file. For example, if we consider a file that contains the same signature twice, one signature always matches if we apply our derivation algorithm, as we only flip single bytes during the derivation. Consequently, it is not possible for our approach to reveal the used signature in this case. Our empirical study, however, shows that such cases are rare and we are able to retrieve byte patterns for a third of the considered signatures.

8. RELATED WORK

The analysis and detection of malware is a very vivid area of research both in academia and industry. Consequently, implementations of such systems are under high scrutiny, subject to security audits and target of adversarial attacks. The analysis and detection of malware is a very vivid area of research both in academia and industry. Consequently, implementations of such systems are under high scrutiny, subject to security audits and target of adversarial attacks. Also, due to the widespread use of signature-based detection in commercial products, a significant body of research has particularly studied the merits and deficiencies of these systems. In the following, we discuss each of these related strains of research in detail.

Attacking anti-virus products. Many researchers have dealt with vulnerabilities in anti-virus products and point out implementation flaws that allow to bypass defensive mechanisms or hijack execution [2, 29, 42, 64]. Over the past year Ormandy, for instance, called attention to several flaws in wide-spread commercial security products [43–47] demonstrating the large attack surface such systems expose. Our method in contrast does not rely on implementation flaws, but addresses a conceptual issue in the use of poorly designed signatures.

Similar in spirit, Min and Varadharajan [35] make use of anti-virus products as an ally for attacks and introduce “AV-Paramware”, an advanced malware piggybacking virus scanners. In particular, the device driver of the virus scanner is tricked into starting the malicious code on every boot and ensures a stealthy operation by providing the malware with the same high privileges as itself. “AVLeak” [6], on the other hand, also employs repeated scans of AV products, but uses the results to carefully leak fingerprints of used emulators.

Evasion of detection. Of course, exploits in anti-virus products can also be used to bypass detection. However, also the underlying programming logic of file format parsers, unpackers or search algorithms might be used to evade detection. Jana and Shmatikov [25], for instance, describe two classes of bypasses: Chameleon attacks where a malware sample appears as a different type than it actually is and werewolf attacks that exploit differences in parsing logic of specific file types. Both, are based on a difference in “perception” between the product at the end-host and the virus scanner. A popular example in this context is the Portable Document Format (PDF): The Adobe Reader is to such an extent liberal in processing malformed documents that do not comply the specifications, that it is difficult to exactly reproduce its behavior in a security product [49, 63]. If a virus scanner’s type-inference fails, there often is no alternative but reverting to static pattern-based signatures.

Moreover, methods for evading signature-based detection have been proposed and studied in various contexts [e.g., 18, 36, 52, 53, 65]. In mobile security research for instance, Zheng et al. [65] assess the robustness of virus scanners for mobile malware against simple transformations. Continuing this line of research, DroidChameleon [52] implements more advanced transformations, showing that in 2013, ten commercial Android anti-malware products can be bypassed using common obfuscation techniques. In a follow-up work, the authors additionally show that even one year later, these systems can still be bypassed with the same techniques [53].

Deriving signatures. In intrusion detection, testing signatures has received substantial attention [30, 38, 55, 56, 60]. To this end some authors attempt to manually [38] and automatically [30] derive signatures from such systems. In contrast to our method the latter however inspects the execution of the matching process on a binary level, while we observe the outcome of the matching completely passively but strategically modify the input.

Deriving signatures from anti-virus software on the contrary has received very little attention so far. Christodorescu and Jha [9] explore possibilities for evaluating virus scanners, and in particular, provide first insights into the feasibility of signature extraction. Filiol [15] extends this and defines a naive algorithm similar to the underlying idea described in this paper, and presents an extraction scheme to determine the disjunctive normal form of a signature. The computational complexity of the latter however prevents a practical application. Our approach notably extends these works by inferring more precise signatures, in a computational feasible manner, which not only represent simple byte sequences, but also express the relation to each other.

Improving signature-based systems. Another strain of research has focused on improving traditional methods as well as devising alternative detection approaches. Christodorescu et al. [12] present a method to reverse malware obfuscations and use it as preprocessing step for AV products. Filiol [15] proposes a signature matching scheme that explicitly limits black-box analysis of virus scanners, and Christodorescu et al. [10] suggest to move away from simple byte-based signatures in favour of the detection based on code semantics [50, 51].

Several authors have proposed the use of machine learning techniques for automatically generating detectors [e.g., 4, 13, 27, 28] as well as signatures [e.g., 19, 33, 40]. While some of these approaches provide excellent detection capabilities and runtime performance, the application of machine learning can induce additional security problems [17, 41, 48, 59] whose elimination is an ongoing research effort [7, 23, 61].

9. CONCLUSIONS

Despite several advanced detection techniques, such as behavioral blocking and packer-agnostic unpacking, most anti-virus products still rely on static signature matching as a fail-back mechanism. This signature-based detection is an effective and efficient tool, if appropriate and up-to-date signatures are available. However, the widespread use of this detection approach can also play into the hands of an attacker, if she gains access to the signatures and exploits them as an instrument for attack.

We inspect this threat and analyze the feasibility of anti-virus assisted attacks, where signatures derived from malware samples are implanted into benign network communication. Based on a novel method for deriving signatures from virus scanners, we show that a considerable amount of the deployed signatures corresponds to simple byte patterns that
are not bound to particular file types or contexts. We then demonstrate how such rather carelessly designed signatures can be used by an attacker. To this end, we introduce the concept of malicious markers and present three scenarios in which an adversary is able to use such markers to remotely instruct a virus scanner to block or delete content on her behalf, turning the anti-virus product into an ally behind defense lines. While our study shows that simple signature matching can be a vulnerability rather than a protection, the concept of fending off attacks directly at the end host still is promising. To be effective in this task, the employed signatures however need to incorporate context and semantics of matched code to be used without side-effects. Dynamic approaches and heuristics, for instance, implicitly include these relations, such that we plead in favor of further advancing the use of more sophisticated and precise detection approaches in anti-virus products.

Acknowledgments

The authors gratefully acknowledge funding from the German Federal Ministry of Education and Research (BMBF) under the projects APT-Sweeper (FKZ 16KIS0307) and VAMOS (FKZ 16KIS0534).

References


