Stego-Malware Attribution: Simple Signature and Content-based Features Derived and Validated from Classical Image Steganalysis on Five Exemplary Chosen Algorithms

Bernhard Birnbaum
Dept. of Computer Science
Otto-von-Guericke University
Magdeburg, Germany
email: bernhard.birnbaum@ovgu.de

Christian Krätzer
Dept. of Computer Science
Otto-von-Guericke University
Magdeburg, Germany
email: christian.kraetzer@ovgu.de

Jana Dittmann
Dept. of Computer Science
Otto-von-Guericke University
Magdeburg, Germany
email: jana.dittmann@ovgu.de

Abstract*—Stego malware, which hides malicious functionality using steganographic communication channels, is becoming increasingly common in today's attack scenarios. Cybersecurity capabilities against such malware include prevention, detection, response and attribution tasks. In this paper, we focus on JPEG images and the attribution task by investigating a set of very simple signature-based steganalysis features for stego-malware attribution by attempting to identify the embedding algorithm used in a multi-class problem. First, the communication scenario in stego-malware is discussed by showing how the warden (observer) setting differs from the typical communication setup in steganography (known as the 'Alice and Bob (A-B) scenario') to be used for a simple (non-blind) cover-stego pair analysis besides blind steganalysis. For our considered stego-malware case, the stego communication is redefined as an attacker-to-attacker (A-A) scenario by extending the capabilities of the warden. Second, due to the very simple nature of stego approaches often used in malware, basic assumptions in steganography are not well incorporated in the malware design. This motivates us to study simple, classically known steganography approaches to simulate stego-malware attribution capabilities using five long-standing, well-known steganography tools. Four simple signature-based and two content-based features are derived for the attribution of five stego algorithms and their performance is validated in a multi-class comparison. Using a test set of 1000 randomly selected original cover images from the Alaska2 dataset, the feature set for attribution of the five algorithms used and their individualisation properties are investigated exemplarily for two different capacities (low: 26 bytes and high: 2.1 kBytes) and two different embedding keys (one long and one short), also considering a recompression case for the low capacity. A single and double recompression of the 1000 Alaska2 images used and the Flickr dataset with its 31,783 images are performed to determine the false positive detection performance within image data without steganographic embedding. The results show the differences in stego-algorithm attribution performance per feature and algorithm.

***Keywords*—stego-malware communication scenario; multi-class steganalysis and attribution.**

I. INTRODUCTION

According to [1], attackers started to use information hiding techniques to make malicious software (malware) stealthier and harder to detect more than a decade ago. In the last 10 years, the volume of malware using steganography and information hiding to prevent detection (bypass security mechanisms), implement evasion or anti-forensics techniques, as well as create hidden communication channels to orchestrate attacks, has been growing on a yearly basis [2]. For such malware using information hiding the term *stego-malware* was created and the following taxonomy was proposed [1]:

- Group 1) malware hiding information by modulating shared resources (e.g., a CPU register)
- Group 2) malware hiding information within network traffic
- Group 3) malware hiding information in digital media objects (covers, e.g., digital images in JPEG formats)

This taxonomy partially reflects the goal of the information hiding mechanism implemented by the attacker: Methods belonging to group 1 are primarily used to allow two processes to exchange data within the same machine or to bypass hardware isolation, methods in group 2 are primarily used to implement Internet-wide covert communication, and methods belonging to group 3 are used for data infiltration, exfiltration or storage. For instance, images modified via steganographic techniques have been used to store information on the local file system of the infected host, to conceal configuration files and malicious code when spreading the infection, or to implement simple command and control (C&C) channels by making them available in social networks or other network services.

Today's cybersecurity capabilities against malware include prevention, detection, response and attribution tasks. For stego-malware detection, for example, the authors from MalJPEG [3] provide an overview of existing work and show that JPEG images are often used. Further [4] summarises available stego-malware approaches by also concluding that JPEG is often used as cover media type. For the detection, the authors in [5] propose an approach to locate stego content in JPEG images by analysing JPEG header markers. In our paper, we also have selected to follow this idea of using JPEG header analysis (file/header integrity as well as the characteristics of specific markers). We focus on an attribution task: Finding traces indicating on the source of the malware. Such a source can be for example malware creation kits on one or more computers, malicious cyber activities of a human intruder or an ultimately responsible party, see e.g., in [6]. As summarized in [7], attribution contains the identification of such sources, the collection of artefacts, extracting relevant information from the data and answering attribution questions besides the source, such as e.g., the time of a malware infestation and activities that where performed on the target. This is similar to the attribution approach presented by Jennifer Newmans group in [8].

In this paper, we address in particular the following in attribution: The identification of the source and collection of artefacts to try to determine the used stego algorithm in stego-malware that is hiding information in digital images (media objects used as covers group 3 in the taxonomy discussed above).

Starting in 2015, the volume of attacks observed 'in the wild' using such methods increased in numbers but also reduced in terms of variety. In fact, the majority of malware exploiting steganographic techniques seems to only take advantage of images as the preferred type of cover media object. As detailed in surveys on that topic (e.g., [2] and [4]), it seems that attackers are capitalizing on the techniques offered by related literature, publicly available source code and libraries, or third-party information-hiding-capable malicious routines offered on a Crime-as-a-Service basis (usually malware creation kits which also contain steganography modules/plugins).

In contrast to the academic research on traditional (end-to-end) steganography (as discussed, e.g., in [9] or [10]), stego-malware relies on much simpler basic assumptions on the communication scenario that will be discussed in more detail in Section II. As done in recent work in [5], we also use existing simple steganography tools from [11], easily available to potential stego-malware creators: *jphide*, *jsteg*, *outguess*, *steghide* and *f5*. Amongst other tools *f5* was also used in [5].

Focusing on stego-malware that uses JPEG images as cover, the paper contributions are as follows to identify the embedding algorithm used in a multi-class problem:

- The disscussion of the warden setting in the stego-malware communication scenario, calling it attacker-to-attacker (A-A) setup by also showing differences in the corresponding basic assumptions for traditional steganographic end-to-end communication (also known as the 'Alice and Bob (A-B) scenario').
- Considerations on the attribution of steganographic methods in stego-malware, aiming at providing indicators of compromise (IoC) for malware detection by trying to identify the algorithm used in a multi-class attribution problem on the example of five algorithms.
- Introduction of a set of light-weight (i.e., easy to compute) features derived from observed artefacts during embedding: Four simple, blind signature-based features and two non-blind, content-based features. The signatures are derived by using existing forensic tools (here *foremost* and *binwalk*) as well as by header analysis on the JPEG files (considering the JFIF version from the JPEG APP0 marker segment as well as a string-search in the JPEG COM marker segments). While the first is novel to this paper, the second follows the methodology in [3] and [5] but determines novel signatures for the string-search performed. The content-based, non-blind features, which analyse in our case the (re-)compression behaviour as well as the embedding impact to the colour distribution in the image, are motivated by the typical, content-focused steganalysis methods discussed, e.g., in [10]. The feature extraction and attribution functionalities are presented in detail in Section III and in Figure 2.
- An empirical investigation on five simple, classical steganography approaches with two different capacities and key sizes based on 1000 randomly chosen images from the Alaska2 image steganalysis reference database [12] to derive a tendency for malware detection and algorithm individualisation by additionally testing for false positives (wrongful stego attribution on cover data with no embedding) with (1) single and double re-compressed Alaska2 images without embeddings and (2) the Flickr30k data set from [13] with 31,783 images in total. The results show in the multi-class decision the following: Unique attribution of *jsteg* is possible with two blind features with no errors and one blind feature with 0.18 percent errors, four stego algorithms can be attributed using file header signatures. *jphide* is difficult to attribute with our features set. Those promising results motivate further work for stego-malware detection and attribution focusing on JPEG image header analysis.

The rest of the paper is structured as follows: First, the A-A communication scenario of stego-malware and corresponding attribution challenges for the Warden are introduced in Section II. In Section III, the attribution concept with the summary of the used steganography tools, the used image sets, the embedding options and the set of attribution features are discussed. Section IV contains implementation details and Section V summarizes the results, followed by a last section with a summary and conclusion.

II. Scenario and Attribution Background for Stego-Malware

Stego-malware has firmly established itself as a dangerous and still growing malware trend since 2015. A recent example for such malware relying on steganographic channels is an incident that has been reported in November 2022 by (among many other sources) [14]. In [15] this stego-malware is discussed in some detail by security specialists that where reacting to this attack early on and who were responsible for limiting its spread by providing involved actors with indicators of compromise (IoC). The details in [15] provide first insights, with more detailed information presented here:

In late October 2022 malicious source code projects from this attack started to appear on the Python code repository PyPi. They were claiming to be libraries to be useful for web development tasks and were soon added (in a kind of supply chain attack) as includes to other repositories on PyPi and GitHub. In those malicious packages, the existing steganography library 'judyb' [16] (itself a fork from the well known Python steganography module 'stegano' [17]) was used to manage the communication. On *infestation* on a target machine, the malware tried to post-fetch/infiltrate malicious routines steganographicaly hidden in PNG images from a fixed remote source (in this case a channel on imgur.com), *establish on site control* (this only worked on MS Windows machines due to the execution methods used), *execute the actual malicious function* (a fork of W4SP-Stealer [18], trying to steal saved passwords, two-factor-authentication tokens, wallet keys, etc. and uploading them to the command & control (C&C) server; here, this W4SP fork is different from other forks by the fact that it uses the steganographic channel to exfiltrate the stolen data instead of simply posting it on Discord) and *communicate it back to the C&C server*. The code of the malicious libraries also contained hookups for a spreader module to try to infest also other MS Windows machines in the same Active Directory domain, but corresponding code was defunct in the sources analyzed here. Figure 1 illustrates the activities of this stego-malware.

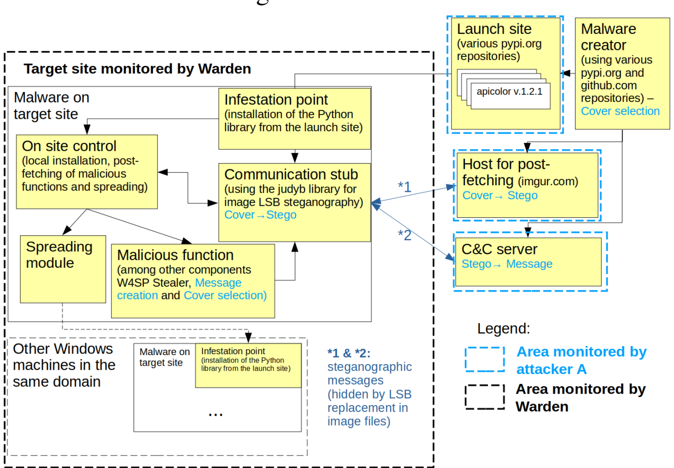

Fig. 1. Activities of the exemplary discussed stego-malware and the A-A communication scenario discussed in this paper.

In their initial report [15], the authors summarize that '*little more than 80 projects containing the malicious packages*' were detected in this supply chain attack. It can be assumed that this number is too low, since over more than two weeks many different versions of the initial libraries were created on PyPi under different names and were then be used to poison other projects (on PyPi and GitHub) by adding them as includes in user contributions. The attack wave stopped after PyPi managed to find a way to automatically derive IoCs and thereby effectually take down new versions before they could be used to poison other projects.

For this representative stego-malware scenario using image steganography, the following points shall be highlighted:

- All discussed software components (the steganography library, the W4SP-Stealer as actual payload, etc.) were and still are **publicly available on platforms like GitHub**.
- Attackers make extensive use of **third party resources** (e.g., repositories like PyPi or GitHub as well as social media sites / image hosts like imgur.com) in infiltration and exfiltration activities, making blocking by application level firewalls difficult.
- The steganographic method used in the discussed example is **trivial least significant bit (LSB) replacement for PNG images** (which is still good enough to prevent usual end-point security solutions from raising an alarm during infiltration).
- The **key** used for securing the communication is not pre-shared but is instead infiltrated together with other attack components, **violating Kerckhoffs' principle**.
- The attacker embeds steganographic messages in PNG images found in **the target system observed by the warden** with the intention to communicate these stego images back to him/herself, making the whole scenario prone to cover-stego image comparison attacks.

This stego-malware example is representative for the current state-of-the-art, where the significant deviations from the traditional 'Alice and Bob' (A-B) end-to-end communication scheme can be identified:

- As the **attacker A activities at the target system are fully observable**, all attacker actions should be as limited as possible to **raise no suspicion**.
- Therefore **Non-Kerckhoffs'** setups (key-less techniques, hard-coded keys or key infiltrated together with the payload of the hidden communication) are often used.
- **Simple embedding and retrieval techniques** are used natively in the target domain or are put in place with supply-chain-attacks like the one discussed above.
- The classical 'Alice and Bob' (A-B) end-to-end communication scenario with a 'Warden' monitoring the channel is changed to a scenario with the **attacker A controlling both ends of the communication** and the **Warden observing the target system and its incoming and outgoing communication** - the A-A scenario shown in Figure 1.
- The cover selection and/or cover embedding at the target system can be observed by the warden. Blind as well as **non-blind analysis of cover and stego objects used** become possible.

For in-depth justifications on these generalizing claims, the reader is referred to [2] and [4].

III. Attribution Concept

In the A-A communication scenario of stego-malware, the Warden has (potentially) full access to the malware injected into the target side and to all communication channels, including the steganographic communication, due to the Non-Kerckhoffs' setups. **As a consequence**, the Warden would be capable to perform **blind** steganalysis (as in the

 ISBN: 978-1-68558-092-6

traditional Warden in an 'Alice and Bob' (A-B) end-to-end steganography scenario) but potentially also **non-blind** (stego-cover comparison) analysis, since the media used as cover in an outgoing communication would be originating within his observed domain. In stego-malware setups, the attacker A relies primarily on two things for the security/confidentiality of his malicious communication: a) a lack of suspicion from the Warden (i.e., security by obscurity) and b) the fact that an in-depth analysis of every in- and outgoing object by dedicated steganalysis engines is far to expensive (in terms of run-time, communication delays and other QoS aspects, false positives, etc.), so such an analysis would only be used as an on demand service for the scaling of methods for evidence gathering in case of a suspicions (e.g., when an IoC was found by an endpoint security solution). As a consequence, the attacker will assumedly try to perform only **simple, innocently looking operations** (like, e.g., accessing PNG files on an image hosting service as in the example discussed in the previous section) to avoid raising suspicion.

In this paper, the research idea is to **define and use a set of light-weight attribution features** that could be checked by the Warden for each communicated media object as part of the **continuous perimeter defense of the target site** (e.g., as rules in a next-generation firewall). As starting point for our empirical work, **five existing, simple steganography tools**, easily available to potential stego-malware creators, are used here, together with **simple signature- and content based-steganalysis methods** to provide a set of light-weight (i.e., easy to compute) features (five simple structural features plus two content-based features). This set of light-weight attribution features is applied on a selected image test set, processed with a fixed embedding option using two selected payload capacities and two different keys as described in the following sub-sections.

A. Steganography and general analysis tools

Our goal is to use very simple approaches to simulate malware steganography. Therefore the well known **Steganography Toolkit** [11] is used for the empirical evaluations in this paper. It is maintained by the GitHub user 'DominicBreuker' and is one of the most popular steganography repositories on GitHub. At the time of writing of this paper it has been forked (and extended) more than 300 times by other users. The reason why it is so popular lies in the fact that it provides a large number of popular steganography and steganalysis tools in a Docker image, making them easily deployable on many platforms without complicated installation procedures. The **steganography tools selected for this paper** from this toolkit are limited to provided image steganography tools for JPEG images. The corresponding set of steganography methods contains the following five tools: *jphide*, *jsteg*, *outguess*, *steghide* and *f5*.

The following **general analysis tools** are selected from the Steganography Toolkit to compute the features/attributes for this paper: *exiftool*, *binwalk*, *foremost*, *strings*, and the *imagemagick* modules 'identify' and 'compare'. All tools used in this paper are listed in Table I.

B. Image sets for evaluation

For a first empirical evaluation, the quality of the test data (esp. the amount of relevant and representative data) is important to obtain generalizable results. To ensure that the data used here is representative as well as diverse, 1000 randomly chosen specimen are sampled as covers from the established image steganalysis reference dataset 'Alaska2' [12]. To provide a significant amount of wide variance image data to establish potential false positive rates for the attribution in an 'in the field' scenario, additionally the 'Fickr30k' dataset from [13] is used in our evaluations.

C. Embedding options used

In this first evaluation, an attribution based on two different message capacities (embedding data: ASCII text of 26 Bytes ('low' capacity scenario) and 2.1 kBytes ('high' capacity scenario) length) and two keys of different length (4 Bytes (='short') and 128 Bytes (='long') are used. Only *jsteg* does not support a key as a parameter and therefore the embedding that case is key-less ('no key').

D. Our set of light-weight attribution features

Motivated from the idea to design an easy to compute feature set, the tools selected (see Section III-A) are analysed and a set of features is identified for potential attribution. Based on in-depth tool output analysis, the following set of light-weight attribution features are implemented from Table I, using pre-existing analysis tools (see marked in cursive) and used in this paper. This table also encodes for each of the attribution features whether it is relevant (`r`), unique (`u`), motivated from (`m`) or not applicable (`n.a.`) for a specific steganographic tool.

Our six features are motivated from the following observations:

- $\mathtt{ba}_1$: The feature extracted by *exiftool* is considered anomalous if the value cannot be successfully retrieved. As can be seen in Figure 2, the 2 bytes reserved for the JFIF version in the APP0 marker segment are zero, which is the case for all *jsteg* embedding attempts.
- $\mathtt{ba}_2$: For correctly written JPEG images, *binwalk* can also determine the data type by extracting image data, but similar to the feature $\mathtt{ba}_1$, this does not apply to *jsteg* embeddings, as the file header is corrupted by this stego tool.
- $\mathtt{ba}_3$: The tool *foremost* can produce successful output for all manipulations by carving the input image except for *jsteg*, since the *jsteg* image headers appear to be damaged, which violates JPEG image format integrity.
- $\mathtt{ba}_4$: All tools seem to leave specific traces in the COM sections of the JPEG file header. This is a weakness that many stego tools share, because they use in many cases non-standard JPEG libraries and do not write correct or plausible JPEG/JFIF metadata. As listed in Table II, $\mathtt{ba}_4$

TABLE I
THE ATTRIBUTION FEATURES USED IN THIS PAPER WITH THE CORRESPONDING ANALYSIS TOOLS AND THE ENCODING OF THE RELEVANCE FOR THE FIVE CONSIDERED STEGANOGRAPHY TOOLS (r: RELEVANT, u: UNIQUE, m: MOTIVATED FROM, n.a.: NOT APPLICABLE)

feature id	feature name	feature type	analysis tools	*jphide*	*jsteg*	*outguess*	*steghide*	*f5*
signature-based features								
$\mathtt{ba}_1$	*JFIF version*	blind	exiftool	n.a.	r, u, m	n.a.	n.a.	n.a.
$\mathtt{ba}_2$	*binwalk extraction*	blind	binwalk	n.a.	r, u, m	n.a.	n.a.	n.a.
$\mathtt{ba}_3$	*foremost carving*	blind	foremost	n.a.	r, u, m	n.a.	n.a.	n.a.
$\mathtt{ba}_4$	*file header*	blind	strings	r	r, u	r	r	r, u, m
content-based features								
$\mathtt{nba}_1$	*file size*	non-blind	exiftool	r	n.a.	n.a.	r, m	n.a.
$\mathtt{nba}_2$	*difference color mean*	non-blind	imagemagick	r	n.a.	n.a.	r, m	n.a.

TABLE II
SIGNATURES FOR FEATURE $\mathtt{BA}_4$ - CHECKED ON ITS PRESENCES: IF $\mathtt{sig}_1$, $\mathtt{sig}_3$ AND $\mathtt{sig}_5$ ARE NOT PRESENT AND $\mathtt{sig}_2$ AND $\mathtt{sig}_4$ ARE PRESENT THE CORRESPONDING STEGO ALGORITHM IS DETECTED

signature id	stego algorithm	signature content
$\mathtt{sig}_1$	*jsteg*	JFIF
$\mathtt{sig}_2$	*f5*	((
$\mathtt{sig}_3$	*jphide*	56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
$\mathtt{sig}_4$	*outguess, steghide*	!22222222222222222222222222222222222222222222222222
$\mathtt{sig}_5$	*outguess, steghide*	Exif

is based on different signatures, which are also shown in Figure 2. First, the identifier field in the JFIF-APP0-marker-segment is checked for $\mathtt{sig}_1$. If it is not present, a *jsteg* embedding is assumed. Further in the file header, the COM-marker-segment is also checked for the following signatures: the presence of $\mathtt{sig}_2$ indicates a *f5* embedding; *jphide* embeddings usually do not contain $\mathtt{sig}_3$; whenever $\mathtt{sig}_4$ is found but $\mathtt{sig}_5$ is not, an *outguess* or *steghide* embedding is expected.

- $\mathtt{nba}_1$: The idea behind this feature is to compare original and stego-image file sizes, as the stego tools *jphide* and *steghide* do not perform an image recompression during embedding. This results in a stego file size that is much closer to the original compared to other stego algorithms. The following rule was derived by empirical analysis: $\mathtt{diff.\ size} < \frac{\mathtt{orig.\ size}}{2} \wedge \mathtt{stego\ size} > \frac{\mathtt{orig.\ size}}{2}$.
- $\mathtt{nba}_2$: This feature is also motivated by the fact that *jphide* and *steghide* do not apply a recompression. So, when creating a differential image of the stego-manipulated image (by *jphide* or *steghide*) and the original, all visible changes are directly related to the embedded data. Usually, the amount of changes is much smaller than the changes related to a re-compression, except when embedding large amount of data. As stego manipulations are also visible in the different colour channels of the image, but changes in the RGB channels due to a re-compression are much higher than for stego embeddings, the following detection rule is derived by empirical analysis: $\mathtt{diff.\ mean} > 127 \wedge \mathtt{diff.\ mean} == \mathtt{rgb\ diff.\ mean} \pm 1\%$.

IV. IMPLEMENTATION OF THE ATTRIBUTION

Analysis and attribution feature extraction: The process of feature extraction is done by first executing all given analysis tools (as outlined in Section III-A) in a so-called '**screening phase**' in the following order: *exiftool*, *binwalk*, *strings*, *foremost* and *imagemagick*. In the '**parsing phase**', the attribution features from Table I are parsed into a CSV file, where each row contains the data of one investigated JPEG files.

Implementation of the attribution: The attribution was implemented in two different modes, **blind** and **non-blind**. As **blind** attribution is the most common case in reality, **non-blind** attribution – often plausible for stego-malware as discussed for the A-A scenarion and the warden monioring capabilities – allows to take advantage of more powerful features (e.g., *steghide* embeddings are not detectible with only the blind attributes presented in Table III-A). Finally, multiple queries with rules/signatures specific for each steganography tool are used to interpret the extracted feature values, which performs the actual stego-tool attribution. Figure 2 illustrates the attribution process in a flowchart.

V. RESULTS AND EVALUATION

In Table III at the end of the document, the evaluation results are summarized. The table is divided into three main column parts:

The **first** part, containing the first and second column, the features from Table I and the test configuration are listed. In the **second** part, the attribution results in terms of correct or incorrect entries to the result list as well as (true/false) positive (=stego) hits and (true/false) negative (=unmodified cover) hits are given. In the **third** main column part, all stego tool identification results are included for the corresponding test configuration.

Starting with the results for *jphide*, Table I discusses attribution results for each of the blind or non-blind attributes deemed relevant for a stego tool, using the described stego embeddings on the Alaska2 data set. This is done for all five stego tools. Second, false positive rate tests on re-compressed genuine Alaska2 and Flickr30k images are presented.

The results presented in the table can be summarised as follows:

- **Regarding different capacities and keys** for all algorithms:

(1) The **blind** simple signatures show:
   - **no influence from different capacities and keys** in the performance for ***jphide***, ***jsteg*** and ***f5***, while *jphide* and *f5* have false classifications for the file header signatures of $\mathtt{ba}_4$: *jphide* with the re-compressed Alaska2 set and Flickr30k, *f5* with Flickr30k only;
   - for ***outguess*** **and** ***steghide*** the $\mathtt{ba}_4$ - file header signatures are sensitive: The high capacity with the short key influences the *outguess* file header signature (for the 2.1KB message, in 437 of the 1000 cases *outguess* could not successfully embed, which results for this tool in an empty (0 Byte) output file without a header – in the evaluations these cases are counted as false negatives since they are no genuine image files any longer but are also not flagged to be the output of this steganography tool). For *steghide*, the $\mathtt{ba}_4$ file header signature is (besides embedding problems that result in only 881 stego files being successfully created for the 2.1KB message) not only resulting in large numbers of false negatives for all cases except the re-compressed Alaska2 images with short capacity with the short key but it also lacks discriminatory power in regard to *steghide* and *outguess*.

(2) The **non-blind** (content-based) features motivated from *steghide* characteristics show the following:
   - both features are relevant for ***steghide*** **and** ***jphide*** but not in a unique manner;
   - the first content-based feature ($\mathtt{nba}_1$) of **file size** is **in most cases capacities and keys independent** but attributes *steghide* as well as *jphide* at the same time (in a not unique manner) with errors only in high capacity with the short key and with wrong classifications in the Alaska2 re-compression tests;
   - the second non-blind feature of different **color mean** attributes ($\mathtt{nba}_2$) also *steghide* as well as *jphide* and is **error prone to high capacity with the short key too** and less sensitive for all other settings with wrong classifications in the Alaska2 re-compression tests.

- **Regarding the individual algorithm identification performances**:

(1) ***jsteg***: Features $\mathtt{ba}_1$, $\mathtt{ba}_2$ and $\mathtt{ba}_3$ are motivated from the artefacts observed after embedding and also $\mathtt{ba}_4$ file header motivated from *f5* can be used with $\mathtt{sig}_1$ in the file header to identify *jsteg* in a unique manner with best results. There are only a minor error in high capacity with the short key for $\mathtt{ba}_2$ and JFIF signature errors for $\mathtt{ba}_1$ in the Flickr30k tests of 58 errors.

(2) ***f5***: Feature file header $\mathtt{ba}_4$ with signature $\mathtt{sig}_2$ allows a unique identification of *f5* with only low errors in the Flickr30k test of 624 similar cases in $\mathtt{sig}_2$.
In summary of (1) of (2): $\mathtt{ba}_1$ and $\mathtt{ba}_4$ signatures $\mathtt{sig}_2$ seem to be partially occurring also in JPEG data on the example on Flickr30k causing classification errors: for $\mathtt{ba}_1$ of 0.18% (58/31783) and $\mathtt{ba}_4$ of 1.96% (624/31783).

(3) ***outguess***: Feature file header $\mathtt{ba}_4$ with signature $\mathtt{sig}_4$ is relevant for identification of *outguess* but it is not unique with errors in the high capacity embedding; it overlaps with the signature for *steghide* in the file header. For stego malware detection without the need of algorithm identification the $\mathtt{sig}_4$ usage is possible. There are no errors in the re-compression and Flickr30k tests.

(4) ***steghide***: Feature file header $\mathtt{ba}_4$ with signature $\mathtt{sig}_4$ is relevant for identification of *steghide* but also *outguess* is attributed and therefore no unique algorithm identification is possible, but it allows with $\mathtt{sig}_4$ the general stego-malware detection. Only the re-compressed embedding has no errors. As for *outguess* there are also no errors in the re-compression and Flickr30k tests.
The two blind content-based features $\mathtt{nba}_1$ and $\mathtt{nba}_2$ are motivated from *steghide* artefacts in JPEG files. Both features are relevant but do not allow algorithm individualization as also *jphide* causes similar artefacts in JPEG. Further it causes false positives in the re-compression and in the Flickr30k tests.
In summary from (3) and (4): the $\mathtt{ba}_4$ $\mathtt{sig}_4$ allows stego-malware detection but no algorithm individualization.

(5) ***jphide***: Feature file header $\mathtt{ba}_4$ with signature $\mathtt{sig}_3$ is relevant and unique for identification of algorithm but the lack of signature $\mathtt{sig}_3$ (and the attribution based on this fact) also appears in the Flickr30k tests.
The two non-blind content-based features $\mathtt{nba}_1$ and $\mathtt{nba}_2$ are motivated from *steghide* artefacts in JPEG files. Both features are also relevant but does not allow algorithm individualization as also *steghide* causes similar artefacts. Further it also causes as for *steghide* false positives in the re-compression and in the Flickr30k tests.
In summary for (5): the based on $\mathtt{ba}_4$ $\mathtt{sig}_3$ is unique for the algorithm *jphide*, but also occurs in non stego data in re-compression of Alaska2 and Flickr30k. Content based features are also relevant, but also occur during normal re-compression. Therefore, for these three features, it is difficult to used them for stego-malware detection.

For the tested attribution features, the following summary can be drawn:

- the signature-based features $\mathtt{ba}_2$ and $\mathtt{ba}_3$ are capacity and key independent and perform best for *jsteg* algorithm identification with no false positives in re-compression as well as in the Flickr30k tests;
- feature $\mathtt{ba}_1$ (JFIF Version) has a similar performance for *jsteg* with few errors of 0.18% in the Flickr30k tests;
- $\mathtt{ba}_4$ COM signatures allow algorithm identification with:

- $\texttt{sig}_1$: *jsteg* – unique with no errors,
- $\texttt{sig}_2$: *f5* – unique with $\texttt{sig}_2$ but 1.96% errors in Flickr30k test,
- $\texttt{sig}_3$: *jphide* – relevant but with 100% errors in Alaska2 re-compression and 98.04% errors in the Flickr30k test,
- $\texttt{sig}_4$ and $\texttt{sig}_5$: *outguess* and *steghide* - relevant with errors depending on capacity and keys.

The content-based features have high error rates when the images are re-compressed and are therefore not applicable if re-compression needs to be considered.

VI. Summary and Conclusion

Summarizing the results presented in Section V, it can be said that the set of light-weight attribution features used in this paper in an initial and also very simple evaluation shows a first positive tendency to potentially identify the stego algorithm used in a stego-malware scenario with the tested set of five different existing algorithms. The promising results motivate further work on attribution approaches, especially for a generalization for stego-malware detection and prevention scenarios. One research perspective might be the combination of our approach with its multi-class attribution with the localization work for embedding artefacts as discussed in [5]. The interest in this field is caused (as pointed out in Section II) by non-Kerckhoffs' setups, which are often found in combination with simple embedding techniques (like the ones practically evaluated in this paper, or even more trivial, like LSB embedding in pixel domain image formats such as PNG). Furthermore, the Warden can usually monitor all relevant communication channels as well as the potential cover objects available in the target domain. This last characteristic of such stego-malware scenarios also enables non-blind analysis and attribution methods which significantly simplify the detection and attribution tasks.

The first aspect for potential **future work** would be the **definition of additional attribution characteristics** for individualization of the attack, respectively to characterise the attacker A more precisely. An extension should cover further aspects such as different stego key usage, different capacities, message- and stego-encoding, etc. as well as further forensic approaches. During the interpretation of the results, additional knowledge was derived, that could be used for future attribution features: With the tested steganography tools, no metadata, such as geodata, used camera, or timestamps were available in the generated stego image files. In addition, all stego objects were generated by the stego tools in baseline encoded JPEGs, even if their covers were progressive DCT-based JPEGs. For non-blind attribution, initial tests with image entropy showed for a stego object a slightly larger entropy than for a double compressed version of the same cover (using the same quality factor as in the first compression again). Also, the location of the changes in the difference image indicate for some tools whether a stego embedding or a re-compression might have created the artefacts.

Second, the **list of steganography tools and methods targeted in the attribution should be significantly extended**, e.g., by including into the evaluations also the steganographic tools represented in the StegoAppDB [19]. Analyzing **different embedding capacities** might bring more individualization, like malicious content included as hidden message could potentially be classified by message length into different classes.

An extension into **inter-media attribution** would be beneficial. In this paper only JPEG images were considered, but also image formats (esp. PNG and BMP for the still popular LSB-embedders) or other media formats like audio file formats should be addressed. The Stego-Toolkit [11] would also provide a good starting point for such developments.

Source code analysis for stego tools would give very valuable attribution characteristics, including the used JPEG-libraries with details on quantization tables and other header details to be expected in the output of the created stego objects as a kind of software-based fingerprinting or signature-based detection.
Since some of the stego methods are also content-sensitive in the embedding (e.g., ignoring low texture regions and embedding only into high texture regions) evaluations with a focus on **content selection** and different content classes should be performed.

Author Contributions: Initial idea & attribution methodology: Jana Dittmann (JD); Stego-Malware scenario modelling and domain adaptation from end-to-end steganalysis: Christian Krätzer (CK); Evaluation (setup and realisation): Bernhard Birnbaum (BB); Writing – original draft: BB; Writing – review, enhancements, finalisation & editing: CK, JD and BB.

References

[1] W. Mazurczyk and L. Caviglione, "Information hiding as a challenge for malware detection," *IEEE Security & Privacy*, vol. 13, no. 2, pp. 89–93, 2015.

[2] L. Caviglione, M. Choraś, I. Corona, A. Janicki, W. Mazurczyk, M. Pawlicki, and K. Wasielewska, "Tight arms race: Overview of current malware threats and trends in their detection," *IEEE Access*, vol. 9, pp. 5371–5396, 2021.

[3] A. Cohen, N. Nissim, and Y. Elovici, "Maljpeg: Machine learning based solution for the detection of malicious jpeg images," *IEEE Access*, vol. 8, pp. 19997–20011, 2020.

[4] R. Chaganti, V. Ravi, M. Alazab, and T. D. Pham, "Stegomalware: A systematic survey of malwarehiding and detection in images, machine learningmodels and research challenges," *CoRR*, vol. abs/2110.02504, 2021.

[5] V. Verma, S. K. Muttoo, and V. B. Singh, "Detecting stegomalware: Malicious image steganography and its intrusion in windows," in *Security, Privacy and Data Analytics* (U. P. Rao, S. J. Patel, P. Raj, and A. Visconti, eds.), (Singapore), pp. 103–116, Springer Singapore, 2022.

[6] H. S. Lin, "Attribution of malicious cyber incidents: From soup to nuts," *Legal Perspectives in Information Systems eJournal*, 2016.

[7] F. Skopik and T. Pahi, "Under false flag: using technical artifacts for cyber attack attribution," *Cybersecurity*, vol. 3, pp. 1–20, 2020.

[8] W. Chen, Y. Wang, Y. Guan, J. Newman, L. Lin, and S. Reinders, "Forensic analysis of android steganography apps," in *Advances in Digital Forensics XIV* (G. Peterson and S. Shenoi, eds.), (Cham), pp. 293–312, Springer International Publishing, 2018.

[9] J. Fridrich, *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge, MA.: Cambridge University Press, 2009.

[10] R. Böhme, *Advanced Statistical Steganalysis*. Information Security and Cryptography, Springer Berlin Heidelberg, 2010.

[11] D. Breuker, “Steganography-toolkit,” *https://github.com/DominicBreuker/stego-toolkit - last accessed: August 21st, 2023*, 2020.

[12] Kaggle, “Alaska2 image steganalysis set,” *https://www.kaggle.com/competitions/alaska2-image-steganalysis/data - last accessed: August 21st, 2023*, July 2020.

[13] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier, “From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions,” *TACL*, vol. 2, pp. 67–78, 2014.

[14] F. Y. Rashid, “Malicious python package relies on steganography to download malware,” *https://www.darkreading.com/threat-intelligence/malicious-pypi-package-steganography-download-malware/ - last accessed: August 21st, 2023*, 2022.

[15] Spectralops, “Check point cloudguard spectral exposes new obfuscation techniques for malicious packages on pypi,” *https://research.checkpoint.com/2022/check-point-cloudguard-spectral-exposes-new-obfuscation-techniques-for-malicious-packages-on-pypi/ - last accessed: August 21st, 2023*, 2022.

[16] JUDYB, “Judyb steganography library,” *https://pypi.org/project/judyb/ - last accessed: August 21st, 2023*, 2020.

[17] Stegano, “Stegano steganography library,” *https://sr.ht/ cedric/stegano/ - last accessed: August 21st, 2023*, 2020.

[18] W4SP, “W4sp-stealer,” *https://github.com/Im4wasp/W4SP-Stealer-Sourcecode - last accessed: August 21st, 2023*, 2020.

[19] J. Newman, “Stegoappdb,” *https://forensicstats.org/stegoappdb/ - last accessed: August 21st, 2023*, 2020.

APPENDIX A

Fig. 2. Flowchart of attribution

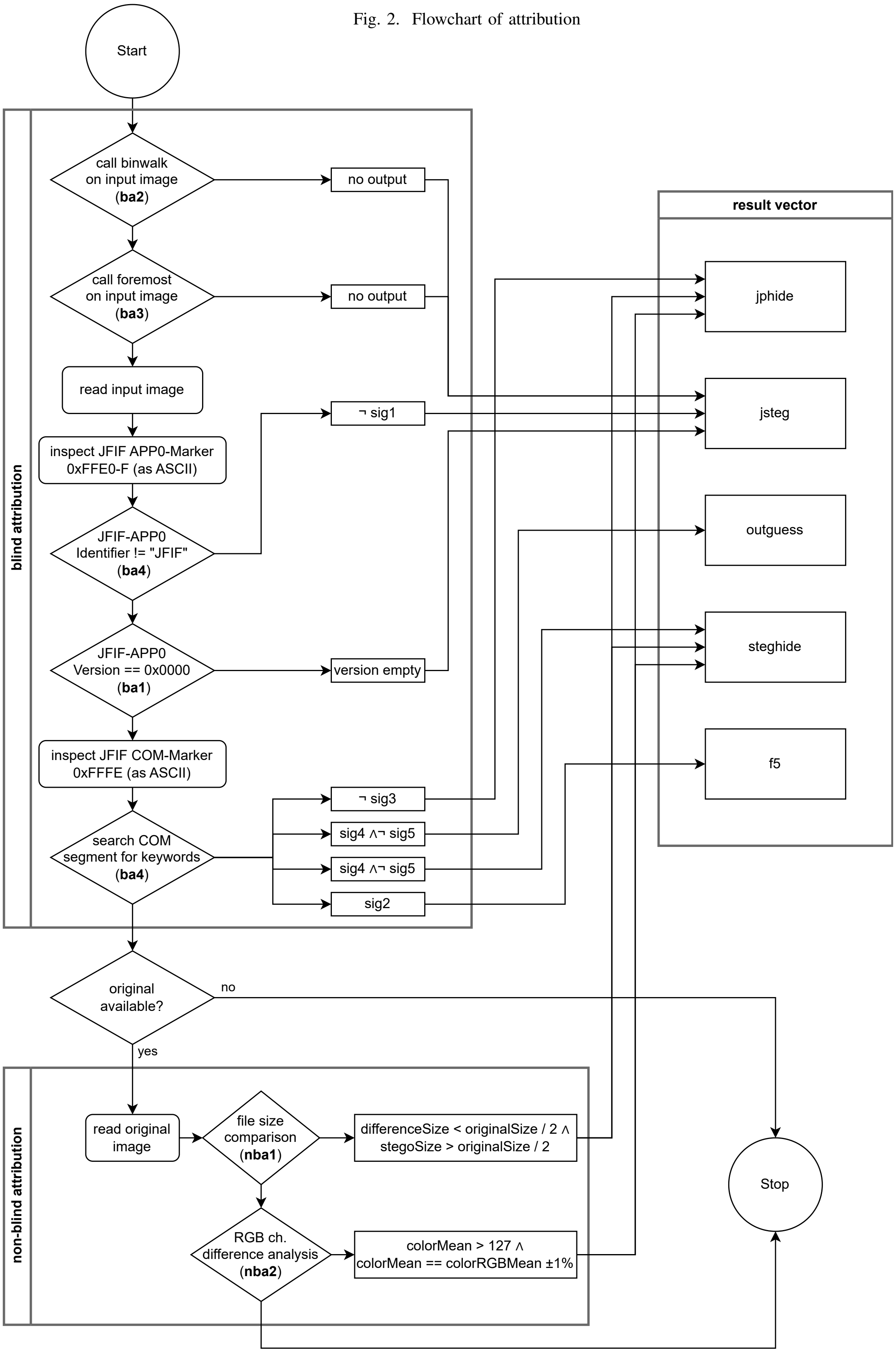

TABLE III
ATTRIBUTION RESULTS FOR THE FIVE TESTED STEGO ALGORITHMS AND THE IMPLEMENTED FEATURES

feature	**test config**	**in result list**		**positives**		**negatives**		**identification results**				
		correct	incorrect	true p.	false p.	true n.	false n.	*jphide*	*jsteg*	*outguess*	*steghide*	*f5*
jphide (1000 images from Alaska2)												
$\texttt{ba}_4$	26B, long Key	1000	0	**1000**	0	0	0	**1000**	0	0	0	0
	rec; 26B, short Key	1000	0	**1000**	0	0	0	**1000**	0	0	0	0
	26B, short Key	1000	0	**1000**	0	0	0	**1000**	0	0	0	0
	2.1KB, short Key	1000	0	**1000**	0	0	0	**1000**	0	0	0	0
$\texttt{nba}_1$	26B, long Key	1000	1000	**0**	0	0	0	**1000**	0	0	1000	0
	rec; 26B, short Key	1000	1000	**0**	0	0	0	**1000**	0	0	1000	0
	26B, short Key	1000	1000	**0**	0	0	0	**1000**	0	0	1000	0
	2.1KB, short Key	1000	1000	**0**	0	0	0	**1000**	0	0	1000	0
$\texttt{nba}_2$	26B, long Key	923	923	**0**	0	0	77	**923**	0	0	923	0
	rec; 26B, short Key	775	775	**0**	0	0	225	**775**	0	0	775	0
	26B, short Key	921	921	**0**	0	0	79	**921**	0	0	921	0
	2.1KB, short Key	0	0	**0**	0	0	1000	**0**	0	0	0	0
jsteg (1000 images from Alaska2)												
$\texttt{ba}_1$	26B, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
	rec; 26B, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
	2.1KB, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
$\texttt{ba}_2$	26B, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
	rec; 26B, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
	2.1KB, keyless	999	0	**999**	0	0	1	0	**999**	0	0	0
$\texttt{ba}_3$	26B, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
	rec; 26B, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
	2.1KB, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
$\texttt{ba}_4$	26B, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
	rec; 26B, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
	2.1KB, keyless	1000	0	**1000**	0	0	0	0	**1000**	0	0	0
outguess (1000 images from Alaska2)												
$\texttt{ba}_4$	26B, long Key	1000	1000	**0**	0	0	0	0	0	**1000**	1000	0
	rec; 26B, short Key	1000	1000	**0**	0	0	0	0	0	**1000**	1000	0
	26B, short Key	1000	1000	**0**	0	0	0	0	0	**1000**	1000	0
	2.1KB, short Key	563	563	**0**	0	0	437	0	0	**563**	563	0
steghide (1000 images from Alaska2)												
$\texttt{ba}_4$	26B, long Key	338	338	**0**	0	0	662	0	0	338	**338**	0
	rec; 26B, short Key	1000	1000	**0**	0	0	0	0	0	1000	**1000**	0
	26B, short Key	338	338	**0**	0	0	662	0	0	338	**338**	0
	2.1KB, short Key	241	241	**0**	0	0	640	0	0	241	**241**	0
$\texttt{nba}_1$	26B, long Key	1000	1000	**0**	0	0	0	1000	0	0	**1000**	0
	rec; 26B, short Key	1000	1000	**0**	0	0	0	1000	0	0	**1000**	0
	26B, short Key	1000	1000	**0**	0	0	0	1000	0	0	**1000**	0
	2.1KB, short Key	881	881	**0**	0	0	0	881	0	0	**881**	0
$\texttt{nba}_2$	26B, long Key	992	992	**0**	0	0	8	992	0	0	**992**	0
	rec; 26B, short Key	971	971	**0**	0	0	29	971	0	0	**971**	0
	26B, short Key	991	991	**0**	0	0	9	991	0	0	**991**	0
	2.1KB, short Key	0	0	**0**	0	0	881	0	0	0	**0**	0
f5 (1000 images from Alaska2)												
$\texttt{ba}_4$	26B, long Key	1000	0	**1000**	0	0	0	0	0	0	0	**1000**
	rec; 26B, short Key	1000	0	**1000**	0	0	0	0	0	0	0	**1000**
	26B, short Key	1000	0	**1000**	0	0	0	0	0	0	0	**1000**
	2.1KB, short Key	1000	0	**1000**	0	0	0	0	0	0	0	**1000**
all other		n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
imagemagick re-compression (1000 images from Alaska2, quality factor 75%)												
$\texttt{ba}_1$	genuine recompressed	0	0	0	0	**1000**	0	0	0	0	0	0
	genuine recompressed twice	0	0	0	0	**1000**	0	0	0	0	0	0
$\texttt{ba}_2$	genuine recompressed	0	0	0	0	**1000**	0	0	0	0	0	0
	genuine recompressed twice	0	0	0	0	**1000**	0	0	0	0	0	0
$\texttt{ba}_3$	genuine recompressed	0	0	0	0	**1000**	0	0	0	0	0	0
	genuine recompressed twice	0	0	0	0	**1000**	0	0	0	0	0	0
$\texttt{ba}_4$	genuine recompressed	0	1000	0	1000	**0**	0	1000	0	0	0	0
	genuine recompressed twice	0	1000	0	1000	**0**	0	1000	0	0	0	0
$\texttt{nba}_1$	genuine recompressed	0	1134	0	0	**433**	0	567	0	0	567	0
	genuine recompressed twice	0	2000	0	0	**0**	0	1000	0	0	1000	0
$\texttt{nba}_2$	genuine recompressed	0	650	0	0	**675**	0	325	0	0	325	0
	genuine recompressed twice	0	1960	0	0	**20**	0	980	0	0	980	0
genuine Flickr30k (31783 images from flickr)												
$\texttt{ba}_1$	original	0	58	0	58	**31725**	0	0	58	0	0	0
$\texttt{ba}_2$	original	0	0	0	0	**31783**	0	0	0	0	0	0
$\texttt{ba}_3$	original	0	0	0	0	**31783**	0	0	0	0	0	0
$\texttt{ba}_4$	original	0	31783	0	31783	**0**	0	31159	0	0	0	624