

---

# IDENTIFYING AUTHORSHIP STYLE IN MALICIOUS BINARIES: TECHNIQUES, CHALLENGES & DATASETS

---

Jason Gray\*, Daniele Sgandurra†, and Lorenzo Cavallaro‡

## ABSTRACT

Attributing a piece of malware to its creator typically requires threat intelligence. Binary attribution increases the level of difficulty as it mostly relies upon the ability to disassemble binaries to identify authorship style. Our survey explores malicious author style and the adversarial techniques used by them to remain anonymous. We examine the adversarial impact on the state-of-the-art methods. We identify key findings and explore the open research challenges. To mitigate the lack of ground truth datasets in this domain, we publish alongside this survey the largest and most diverse meta-information dataset of 15,660 malware labeled to 164 threat actor groups.

**Keywords** adversarial · malware · authorship attribution · advanced persistent threats · datasets

## 1 Introduction

Malicious software (malware) remains one of the biggest threats to organizations, and there seems no sign of this changing in the near future [114]. Identifying malware authors to a person, group or country provides evidence to analysts of the wider goals of threat actors. Furthermore, it provides a method to counter cyber attacks and disrupt the malware economy through public indictment [100, 90].

The current and only method for authorship attribution used by analysts involves prolonged analysis of the threat actor over a long duration and within different phases of the killchain [72]. Part of this process includes gathering features such as network analysis and exploitation techniques referred to as *indicators of compromise* as well as relying on known databases of Tactics, Techniques and Procedures (TTPs).

Sometimes there exists no wider context, especially if the threat actor is unknown to the victim. In very few cases, analysts discover the malware source code and use this to determine attribution through source code authorship attribution [40, 63, 23, 67]. However, released source code leads to copycat attacks or the malware no longer used [41]. This means defenders often find themselves with only the malware binary as evidence. The quicker the defenders analyse the malware and identify a probable threat actor, the quicker they can understand and contextualize an attack (including if they must contact an authority and which relevant authority), which leads to a quicker response time and attack mitigation.

The specific problem of identifying an author of piece of malware is known as Malware Authorship Attribution (MAA). However, using the binary alone represents a difficult problem due to the complexities of program provenance [107]. Despite this, the binary still provides interesting artifacts on author style, *e.g.*, implementation of encryption, propagation, mutation or even the setup of command and communication servers within the malware infrastructure. Even though the demand for malware attribution continues to increase, we notice few publications detailing the methods of malware authorship attribution.

Recent work [87, 21, 4, 59, 20] informed the wider authorship attribution field. Neal et al. [87] wrote a survey on the wider topic of stylometry focusing on de-anonymizing text. The survey by Burrows et al. [21] focuses on the attribution of source code up until 2011 and highlights the positive use of Machine Learning techniques in the

---

\*Royal Holloway University of London

†Royal Holloway University of London

‡King's College London

authorship attribution field. Brennan et al. [20] introduce the notion of exploring the adversarial approach toward the stylometry problem and provide novel datasets to aide this research direction. Kalgutkar et al. [59] provide further insight on the code authorship attribution problem by exploring the use of features in benign source and binary code attribution as well as the attribution models and methods. They also present the challenges in the research field and incorporate the field of plagiarism detection. Finally, Alrabaee et al. [4] discuss three state-of-the-art techniques for the single-author binary authorship problem [22, 3, 106] and provide promising results from applying malware to these systems.

The current state-of-the-art systems show promising results on attributing programs where author style remains unaltered apart from compilation techniques. However, there exist few attempts to extend these systems to consider author masking techniques such as those used by some Advanced Persistent Threat (APT) groups). This limitation to the current state-of-the-art systems opens them to attack and thus there exists the need to fully understand the adversarial challenges to authorship attribution of malware.

**Contributions** Our contributions include a thorough systematization of the malware authorship attribution problem focusing on the data modeling techniques, datasets and features used for attribution to allow the community to understand how each paper builds upon each other and the shortcomings within the current research. We review eighteen attribution systems. We compare them in terms of techniques, features, efficacy, functionality and adversarial robustness.

We discover there exist only two publicly available author-labeled malware datasets, both of which contain significant flaws such as non unique labels. Furthermore, we found the current features used for author style remain varied with no clear consensus on authorship style (42 of a total of 72 features were used separately by research groups). The current state-of-the-art systems remains inapplicable to real world use cases. The majority of systems fail to take into account modern malware development methods, *e.g.*, assuming multiple authors. Additionally, researchers use non-representative datasets of the real world which introduce adversarial issues surrounding open world assumptions, continuous learning, concept drift and obfuscation. On top of this, the majority of attribution systems from existing research lack reproducibility owing to system unavailability, systems no longer working, or the literature omitting fundamental details.

Focusing on the dataset problem, we contribute by publishing a labeled meta-information dataset of 15,660 malware. We extensively use open-source intelligence to build a list of APT groups and then gather hashes of malware to which we verify their legitimacy against VirusTotal. We use Natural Language Processing techniques to gather the most high likely label for a hash from various open-source intelligence material. This dataset is the largest verified APT labeled malware dataset to date. We searched 896 files made up of a mixture of PDFs, CSVs, rules, and indicator of compromise files. We found 15,660 unique hashes which we have labeled to 164 APT groups. Furthermore, we identified an additional 7,485 unique hashes. For these unlabeled hashes, we record the top 5 keywords from the file and the keywords of the metadata.

Our work complements Kalgutkar et al. [59] and Brennan et al. [20] by extending the application of authorship attribution to malware by including a full detailed analysis of malware author style, features and adversarial approach. We expand upon the work by Alrabaee et al. [4] and incorporate the multiple author attribution problem into the conversation of MAA. We also note the survey by Xue et al. [119] which focuses on general Machine Learning based program analysis whereas we focus purely on authorship style gained from program analysis using multiple data modeling techniques.

Section 2 presents the background to threat actors, authorship attribution and adversarial techniques. Section 3 systematizes the MAA problem, focusing on the data modeling techniques, authorship style, features and datasets. Section 4 discusses real-world application of the current state-of-the-art, looking at the challenges and recommendations for future work. Finally, in Section 5, we present the method we used to create a new APT malware dataset for the research community.

## 2 Background: Threats Actors, Authorship Attribution, & Adversarial Techniques

In this section, we set out the background to the *malware authorship attribution (MAA)* problem. MAA is the identification of the author of an unknown malicious file<sup>4</sup>. In particular, we define the authors we wish to identify as *Threat Actors*. We also explore the more wider form of the MAA problem and consider the types of adversary attacks which MAA systems are likely to face.

---

<sup>4</sup>Throughout the paper, we refer to malicious files as “malware”, “malware binaries” or “malicious binaries” interchangeably.

## 2.1 Threat Actors

Although the threat actors with the greater skill level tend to use better adversarial techniques [14], they also tend to possess unique styles when using custom-made tools. Naturally, if any attackers use commercially available or open source tools, then the author of the tool is not necessarily the threat actor. As we wish to focus on identifying author style of threat actors, we shall look to focus on where style exists *i.e.* within custom tools. These tools are generally produced by Advanced Persistent Threats (APTs).

**Advanced Persistent Threats** APTs represent the most sophisticated attackers. The US National Institute of Standards and Technology (NIST) provides an in-depth definition of an APT [86]. For the purpose of this paper, we consider APT groups as state and state-sponsored threats. For instance, the Daqu, Flame and Gauss are examples of malware used by allegedly state funded APT groups as part of espionage campaigns [16]. Additionally, these campaigns, alongside Stuxnet and Red October, display the difficulty of detecting state and state-sponsored APT threats [118]. At the moment, there exists only sparse information on APT groups, and the data remains unstructured and difficult to automatically analyze. Lemay et al. [69] created a survey which contains several pieces of information on APT groups retrieved from public sources, such as the various aliases used for group names and the alleged campaigns conducted.

Table 1: Revised list of the top 10 APT groups. We gathered information from AT&T Cybersecurity [13], MITRE [84] and CCN-CERT [28] to create this list. The table also reports the alleged group location and the number of unique and shared tools linked to each group.

Rank 2020	Rank 2018	Group Name	Number of Aliases	Aliases	Suspected Location	Number of Unique Tools	Number of Shared Tools
1	1	Lazarus Group	4	HIDDEN of Peace, COBRA, ZINC, ACADEMY, Guardians NICKEL	DPRK	16	2
2	*	Gamaredon Group	0	N/A	N/A	1	0
3	7	Kimsuky	1	Velvet Chollima	DPRK	0	0
4	3	MuddyWater	2	TEMP.Zagros, Seedworm	Iran	2	6
5	*	TA505	1	Hive0065	N/A	5	3
6	2	Sofacy	11	SNAKEMACKEREL, APT 28, Sednit, Pawn Storm, Group 74, Tsar Team, Fancy Bear, Strontium, Swallowtail, SIG40, Threat Group-4127	Russia	20	4
7	*	PROMETHIUM	1	StrongPity	N/A	2	0
8	10	Turla	5	Snake, Venomous Bear, Waterbug, WhiteBear, Krypton	Russia	10	10
9	4	Oil Rig	3	IRN2, HELIX KITTEN, APT 34	Iran	9	11
10	*	Emissary Panda	6	TG-3390, BRONZE UNION, Threat Group-3390, APT27, Iron Tiger, LuckyMouse	China	3	12

\* Not in the 2018 top 10 APT groups.

In addition, there exists a publicly available spreadsheet containing APT groups and their aliases [113]. Various cyber-experts from several reputable cyber-threat intelligence sources, such as FireEye, CrowdStrike and MITRE [84], regularly contribute to the spreadsheet and it quickly gained popularity amongst the research community for the *ground truth*. There also exist a few open-source sharing methods such as STIX [88] and TAXII [89] to help researchers, but most threat intelligence options require payment [19].

To further highlight the issues surrounding APT groups, we gathered information from MITRE [84], AT&T Cybersecurity [13] and CCN-CERT [28] to create a list, Table 1, of the top ten APT groups along with the alleged group location and tools linked to each group. From the table, we see the vast number of aliases and the lack of samples linked to each APT group. For example, there currently exists no known malware linked to the group *Kimsuky*. We also observe the majority of the APTs use both unique malware and open source/shared tools *e.g.*, *Turla* and *Oil Rig* use PsExec but allegedly, different Nation States sponsor them. Therefore, MAA becomes increasingly harder if all groups use identical tools. Furthermore, we remark there exist no APT groups on the list allegedly sponsored by a

Western or Five Eye nation<sup>5</sup>. We believe the source of the data, predominantly American Threat Intelligence companies, might introduce some bias to the list as their focus aligns to the threat actors of Western or Five Eye Nation. However, threat actors target and belong to a variety of countries. Finally, we remark from 2018 to 2020 six APT groups remain in the top 10. This shows the longevity of the groups despite an increase in public attribution.

## 2.2 Binary Similarity and YARA Rules

Currently, malware analysts use YARA rules<sup>6</sup> for recognizing and attributing malware samples. YARA rules tend to identify shellcode and code reuse for linking samples and not authorship style which is akin to the binary similarity problem, *i.e.* comparing how much shared code exists between binaries [48] or searching binaries for code cloning [37]. Using similarity for attribution is not foolproof and in many cases can be lead to false accusations [14]. It usually also requires analyzing all of the binary whereas identifying author style can be performed on smaller code fragments.

Even though an analyst must write a rule based on their research of each unique sample (meaning YARA rules remain as labor-intensive to most manual malware analysis methods), they provide a much easier and quicker solution to the current MAA systems. Research by Bassat and Cohen [15] shows the ease of using YARA rules in the “wild” for clustering malware similarities between alleged Russian APTs. However, the same research also shows YARA rules rely on unpacked samples to trigger the identified traits within the YARA rules and this is similar to current MAA systems. More recently, Raff et al. [96] tackle the labor-intensive problem and develop the state-of-the-art to automatically generate YARA rules using malware. Similar to the research by Bassat and Cohen [15], Kaspersky developed a Threat Attribution tool based on APT malware binary similarity [60].

## 2.3 Binary Authorship Attribution

MAA is a subset of the binary authorship attribution (BAA) problem. BAA applies to other tasks, such as plagiarism and intellectual property rights. In these cases, we know all the authors beforehand, *e.g.*, the students in a programming class. In contrast, malware authors wish to remain undisclosed due to the illegality and secrecy of the underground market within which they operate [1]. When we know all the possible authors, we call this *Closed World (Assumption)*, otherwise *Open World (Assumption)* [85]. All of the authorship systems reviewed in this work use *Closed World Assumption (CWA)*. From a data modeling perspective, this prevents understanding the real world context of authorship attribution. However, the CWA mitigates some of the challenges such as quantifying some of the unknowns, *i.e.*, the total number of authors. Mitigating some of challenges can help with exploring other authorship objectives.

There exists varying objectives of authorship attribution set out by Kalgutkar et al. [59]. These consist of *identification* - linking a binary to an author, *clustering* - grouping stylistic similarities, *evolution* - tracking stylistic changes over time, *profiling* - understanding stylistic characteristics, and *verification* - checking for adversarial tampering. We focus on identification as the other objectives can be a by-product of the research on authorship *identification* and *identification* forms the basis of understanding authorship style which the other objectives rely on.

Within all authorship attribution objectives, we must consider if the goal is *Single* or *Multiple* authorship. The single authorship attribution problem assumes only one author for every piece of binary. Conversely, the multiple authorship attribution problem assumes multiple authors created the binary. Assuming single authorship of binaries which multiple authors created is likely to make any attribution system incorrectly learn authorship style and lead to potential attacks on the system which we explore in the next section.

## 2.4 Adversary Techniques

Most BAA work assumes the authors are unaware of an attribution system being in place. Few works consider authors using adversarial techniques to influence the output of the attribution system. This requires attackers to first identify which features appear easier to manipulate to affect the output of the attribution system. Some of these attacks are aimed at the learning phase (*e.g. training set poisoning*). However, most existing binary modification attacks are aimed at evading the attribution system at run-time (*evasive attacks*). Meng et al. [81] describe three evasive attacks namely; (i) the confidence-loss attack, (ii) the untargeted attack and (iii) the targeted attack. The confidence-loss attack defeats an attribution system by removing any traces of author style to ensure it predicts no author label for the binary. The untargeted attack attempts to make the prediction of the attribution system as any other author than itself. The targeted attack tries to convince the attribution system the binary belongs to a pre-chosen author other than the attacker. We deem the confidence-loss attack as unsophisticated as most malware authors try this by default to remain anonymous

---

<sup>5</sup>The Five Eyes consist of United States, United Kingdom, Canada, Australia and New Zealand

<sup>6</sup>YARA is a pattern matching tool with a rule based syntax which allows the discovery of specific signatures [9].

and maintain their privacy. Whereas we class the other two attacks as sophisticated and we believe APT groups are more likely to implement these attacks.

#### 2.4.1 Unsophisticated attacks.

We deem these attacks to be obfuscation techniques authors use to hide their identity and fool malware detection systems. Common obfuscation techniques include *encryption* and *packing*. The use of encryption prevents easy analysis. The adversary encrypts the main function to prevent static analysis on the malware. The program initially calls a function to decrypt itself upon runtime. This function requires a decryption key which the author either stores at a remote location (such as a Communication and Control server) or hides in the malware delivery method (such as a phishing email). Otherwise, storing the key in the malware file allows for the malware analysts to decrypt it.

Malware authors use *packing* to evade analysis and detection systems. The developer compresses the binary to hide the functionality of the binary. A packed binary contains a small amount of code which enables the binary to decompress itself at runtime. A packed version of a binary appears as a completely different version to the original binary, which allows adversaries to trick defense systems such as anti-virus software. The majority of packed binaries require manually unpacking before applying static analysis techniques. However, there exist automatic tools such as Un{i}packer which unpacks common packing tools such as UPX, ASPack, PEtite and FSG [75]. Authorship attribution systems either require the samples unpacked to extract author style or they apply their process to packed binaries to test if authorship style remains after packing.

#### 2.4.2 Sophisticated attacks.

*False flags* used by APTs to imitate other groups [14] are the primary example of sophisticated attacks currently in use. Simko et al. [112] considered the idea of *imitating programmer style* for source code authorship attribution. This led to the definitions of *Forgery* and *Masking* techniques. The *Forgery* technique describes the process an adversary employs to create a program which the attribution system outputs as a different APT group. For example, we describe a targeted attack by A on B (involving an innocent party C) when A successfully convinces B that C performed the attack. If A convinces B any other attacker executed the attack, then we class the attack as untargeted. *Masking* is when an adversary manages to hide as the original author of a program it has modified. For example, an attacker wants to add malicious code into an open source project without the original authors knowing. Similarly to Forgery, masking can either be targeted or untargeted. Matyukhina et al. [76] develop such an attack to five state-of-the-art source code authorship attribution models by learning authorship style from data collected from open source repositories. They create three types of source code transformation attacks based on capturing author style to create both targeted and untargeted attacks. Similarly, Quiring et al. [94] construct a Monte-Carlo Tree search to transform source code for both targeted and untargeted attacks on two state-of-the-art source code authorship attribution systems. Interestingly they both circumvent the authorship attribution system by [23] using different approaches. These attacks on source code authorship attribution systems show MAA systems are likely to face similar attacks and so any system must consider such attacks.

### 3 Malware Binary Authorship Attribution

We reviewed papers in the subject field over the last decade to identify relevant systems and research applicable to MAA. Our search criteria looked for work which addressed the problem of binary and malware authorship attribution. We omitted any papers which performed a binary classification on malware and contained no significant contribution on authorship styles to malicious files, *e.g.*, we omit the paper [65] as this classifies malware into APT group or non-APT group but we include [66] as this classifies malware into specific APT groups. We identified eighteen papers which possess a significant relationship with MAA, and we contacted all authors whose systems were not publicly available. We received a mixture of responses. Some systems had contractual obligations to prevent them from being shared, others did not wish to share their system or said their system shall be made available in the future. On top of the eighteen papers, we identified the survey by Alrabaee et al. [4] which evaluates the systems in [106, 3, 22]. Although this paper provides no new system it helps provide added insight on the systems they evaluated in the context of malware. We focus on: (i) *data modeling techniques*, (ii) *datasets*, and (iii) *features*. We decided on these three areas as they represent the key components in building analytical systems for understanding large data.

In Section 3.1, we first classify the *data modeling techniques* used in these works into five categories: (i) *classification* techniques identify whether a piece of malware belongs to known set of groups; (ii) *clustering* techniques enable us to group malware into authors based on underlying data trends; (iii) *anomaly detection* methods allow us to label malware based on malware not conforming to a known group or category; (iv) *structured prediction* methods predict structured

Table 2: A list of known Data Modeling Techniques used to tackle the binary authorship problem published between 2011 and 2019. There exist five categorizes of techniques: (i) Classification; (ii) Clustering; (iii) Anomaly detection; (iv) Structured prediction; and (v) Non-machine learning methods.

Data Modeling	Algorithm	Attribution System
Classification	Deep/Artificial Neural Networks (DNN/ANN)	[103], [104], [78], [7], [8]
	Tree Bagging (TB)	[54]
	Random Forests (RF)	[50], [22], [54], [44]
	Support Vector Machine (SVM)	[106], [77], [80], [22], [54], [58], [78]
	Bayesian Classifiers ( <i>e.g.</i> , Naïve Bayes (NB))	[50], [54]
Clustering	Large Margin Nearest Neighbor (LMNN)	[106]
	K-Means Clustering	[106], [7], [8]
Anomaly Detection	Multi-View Fuzzy Clustering	[47]
	Isolated Forests (IF)	[66]
Structured Prediction	Conditional Random Fields (CRFs)	[80], [78]
	Dissimilarity Algorithm	[3]
Non-Machine Learning	Manual Analysis	[74]
	Attribution Weighting	[6]

objects for example within a binary file we can identify a structure for an author based on assembly language; (v) *non-machine learning methods* include alternative probabilistic or manual methods.

In Section 3.2, we categorize the works based on the *datasets* used within the systems. Specifically, we divide the *datasets* into benign source code, benign binaries and malware binaries to match the current approach by researchers. The benign software approach uses compiled source code from known authors and the malware approach uses predominantly APT malware. In Section 3.3, we explore malware author style and derive a categorization of author features which we use to compare the eighteen BAA systems.

### 3.1 Data Modeling Techniques

We present all the techniques used from the reviewed papers in Table 2. From the table, we see fifteen of the systems use various Machine Learning (ML) methods. We also notice the majority of ML methods favor the classification problem. We believe the reason for this lies in the easier approach of solving the closed-world problem using labeled source code data which we show in Section 3.2 and Section 4.1. Research on source code authorship attribution mirrors the same pattern [59].

Hong et al. [54] uniquely explore more than two classification algorithms and conclude Random Forest (RF) and Support Vector Machine (SVM) as the most suitable candidates for solving the problem due to their enhanced performance against the other five techniques they tested. This concurs with the rest of the field [50, 22, 58, 77, 44]. Seven papers consider three alternative ML methods: clustering, anomaly detection and structured prediction techniques [106, 66, 80, 78, 7, 8, 47]. We explore these further as they show promise towards the open-world problem.

In detail, Rosenblum et al. [106] use a SVM classifier within their single-author closed-world model and they extend this solution to the open-world problem by using a k-mean clustering technique to cluster binaries based on previously built author profiles. For this, they change their original classifier to the Large Margin Nearest Neighbor (LMNN) as this aids building author profiles. Laurenza et al. [66] approach APT triaging by identifying outliers of APT style within malware using Isolated Forests (IF).

Meng et al. [80], Meng and Miller [78] extend the multiple author feature discovery work ([77]) by using Conditional Random Fields (CRFs) applied to the assumption multiple authors code consecutive basic blocks. In this scenario, CRFs outperform SVMs. Continuing this assumption, Meng and Miller [78] explore the use of Deep Neural Networks directly on the binaries' raw bytes without any analysis or feature extraction process. Rosenberg et al. [103, 104] also consider the use of Artificial Neural Networks for classifying binaries to authors. Alrabaee et al. [7, 8] use convolutional neural networks to cluster author style and then use a classifier to determine if a piece of malware belongs to an author cluster. Finally, Haddadpajouh et al. [47] choose a multi-view fuzzy clustering model to group malware into APT groups based on identifying loosely defined patterns among binary artifacts.

Alternative non-ML methods used to solve the BAA problem also use features which identify author style. Both Alrabaee et al. [3] and Alrabaee et al. [6] use probabilistic methods such as dissimilarity algorithms and a novel

attribution weighting formula respectively. Marquis-Boire et al. [74] propose a pipeline driven from manual malware analysis.

## 3.2 Current Datasets

Datasets remain a key part of any analysis process due to the necessity of identifying binary specific trends within the data. We summarize the current sources used within the eighteen systems reviewed. We split the dataset analysis into two sections: (i) Benign Source Code and Binaries; and (ii) Malware Binaries. Afterwards, we provide an overall comparison of the datasets.

### 3.2.1 Benign Source Code and Binaries

Due to the lack of author labeled binaries, the majority of the research in BAA uses source code from student competitions and then compiles it using a variety of compilers to create a ground truth binary dataset. This approach allows researchers more control on the *cleanliness* of the dataset. Specifically, this provides researchers with greater certainty on the verification of the ground truth. In addition, this provides the ability to choose which complexities the toolchain process introduces, artificially create larger datasets by using multiple toolchain processes and link author styles learned from source code stylometry. Consequently, this approach leads to datasets which fail to represent the real world. They tend to remain static and not evolve alongside author styles. Additionally, these datasets add extra time to consider all the different toolchain combinations to account for the various compilation methods. Researchers also choose the datasets to consist of only C and C++ languages due to the popularity of the programming languages [116]. However, malware generally consists of various languages. We describe the four main sources below.

**Google Code Jam (GCJ) [45]** Since 2008, this worldwide student competition runs annually and the organizers publish all the problems and solutions for anyone to download. There exist multiple benefits for using the GCJ dataset for authorship identification. Firstly, all the participants code similar programs and this allows researchers to focus purely on author style and not program functionality. Secondly, the dataset consists of diverse authors from all over the world. Thirdly, GCJ offers substantial prizes to the participants meaning they must know their identity. Hence, there is no necessity for the participants to hide their author style unlike malware creators. In general, the overall quality of the submissions varies as not all the samples compile meaning researchers must clean the dataset before using it.

Hendrikse [50] uses the script written by Caliskan et al. [22] to obtain the GCJ dataset. However, they both use different subsets of the same dataset for testing and training their attribution systems. Alrabaee et al. [8] use the GCJ dataset to build synthetic binaries from multiple authors by combining the source codes of the various entries. They construct binaries consisting of between two and eight authors. However, this method introduces the issue of distinct separation between the various author styles within the binaries. Therefore, we believe this method constructs a poor dataset for training BAA systems due to the cleanliness allowing the systems to easily distinguish between the authors. However, the dataset provides an opportunity to test systems and evaluate whether they actually perform highly on such a clean dataset.

**GitHub [42]** This is a hosting site for software development which uses *git*, an open source version control platform. GitHub encourages agile development for software projects and allows multiple authors to edit and contribute to various repositories whilst recording the contribution of each user. Meng et al. [79] created the tool *git-author* to tackle the attribution of GitHub repositories to each author. This enabled them to create a labeled dataset for multiple authorship attribution. Three works use *git-author* for the ground truth of their attribution system [77, 80, 78]. Additionally, the GitHub community ranks each repository out of five stars which Caliskan et al. [22] use to judge programmer ability. In this work, they build their GitHub dataset using only repositories containing at least two hundred lines of code and they omit any forked repositories or any named “Linux”, “kernel”, “OSX”, “LLVM” or “next”. They state this ensures a sufficient amount of code exists to learn author style and it also reduces the amount of shared code within their GitHub dataset. Alrabaee et al. [8] collect fifty C/C++ projects where between 50 and 1,500 authors contributed to each project. Introducing a high number of authors potentially saturates author style boundaries as there exists some natural cross-over with author style making it even harder to distinguish between the distinct authors.

Plenty of disadvantages exist from using this data source for malware attribution. Firstly, the majority of repositories are benign projects and malware authors are unlikely to use popular open source repositories for malware development. Secondly, the openness of GitHub allows anyone to clone the code and in turn author style. Finally, it opens up the code to the potential attack where an adversary modifies the code without the repository owner noticing through author style imitation [112].

**Planet Source Code [93]** This platform hosts source code and claims to host 4.5 million lines of code and this includes approximately 200,000 lines of C/C++ code. When a user uploads their code to the site, they rank their own skill level choosing the option of unranked, beginner, intermediate or advanced. Other site members then rank each submission for the various awards the site offers. The combination of both these ranking methods provides site users with confidence in the coding standard. Similar to previous data sources, there exists the assumption any uploaded source code belongs to the user who uploads the code.

**Other Benign Sources.** In addition to the three public repositories above, Rosenblum et al. [106], Alrabae et al. [6] use student coursework. Alrabae et al. [6] assume the source code author refers to the student who submitted the coursework, whereas the dataset used by Rosenblum et al. [106] included submissions where the students worked in pairs. To mitigate this issue, they performed manual analysis to identify a single author for each program. Alternatively, academics use plagiarism detectors on coursework submissions to identify where students cheated and this provides a form of “authorship attribution”. However, plagiarism checkers fail to check for contributions from unknown third parties [2, 39]. In comparison, for MAA we must consider methods to identify unknown programmers/malware authors due to the “underground” behavior exhibited [1]. Rosenblum et al. [106] state the students in their dataset received skeleton code which potentially influenced the students’ programming style even though they attempted to remove all the skeleton code from the samples. In comparison to malware authors, the students must identify themselves to receive a score for their coursework and therefore are likely to refrain from implementing methods to hide their author style. Unfortunately, data protection policies prevent both Rosenblum et al. [106], Alrabae et al. [6] from sharing the datasets.

Kalgutkar et al. [58], Gonzalez et al. [44] created a benign Android application dataset using applications from stores such as Google Play Store, AppLand, Anzhi, Aptoide, Fdroid, MoboMarket, Nduoa, Tinent and Xiaomi for which they attribute by using the private certificates from the signed APK files. Additionally, Gonzalez et al. [44] use the store called 3gyu. As well as the previous application stores, both these papers use APK files from GitHub and an on-line collaborative system called Koodous.

**Toolchain** The common toolchain approach uses multiple compilers and optimization levels. However, every combination of compiler and optimization level used produces a unique binary sample from the same source code. This generic approach excludes the use of varying obfuscation tools and modifications which create further unique binaries. There exist six papers [50, 78, 4, 6, 7, 8] which create a dataset using multiple compilers from both open source and commercial sources, such as Clang [30], GNU [43], ICC [57], LLVM [71], Microsoft Visual Studio [82] and Xcode [11]. A sophisticated malware developer might create a customized compiler yet this remains unlikely due to the deterrence of the complexities of compiler design and it is a unique identifier. The optimization functionality of compilers decreases the program’s runtime, but at the same time it increases the compilation duration. Programmers consider a cost-benefit analysis when deciding which level of optimization to perform. Similar to using different compilers, using varying optimization levels affects author style. Eight papers consider at least one optimization level within their research to account for the effect of optimization on author style [22, 77, 80, 50, 78, 4, 6, 8]. However, there still requires further understanding of the impact of toolchains on author style.

### 3.2.2 Malware Binaries

Creating an author labeled malware dataset echoes similar difficulties in creating a malware family labeled dataset [109]. We show particular interest in APT malware as APT groups tend to use sophisticated adversarial techniques. To the best of our knowledge, there exist two attempts to create a large APT labeled dataset. Laurenza et al. [66] create a list of APT groups and use these to scan publicly available reports written by threat intelligence companies, government departments, anti-virus and security companies for related malware hashes. They use these hashes to download the samples from sources such as VirusTotal. They store this dataset on GitHub [64]. For the purpose of their paper Laurenza et al. [66] use a subset of [64] consisting of 19 APT groups and over 2000 malware samples. Due to the unavailability of the exact dataset used in Laurenza et al. [66], we analyzed the GitHub dataset [64]. The second attempt to create an APT malware dataset is by “*cyber-research*” which they store on GitHub [33]. The dataset contains 3594 malware samples<sup>7</sup> which are related to twelve APT groups and are allegedly sponsored by five different nation-states. Similarly to Laurenza et al. [66], “*cyber-research*” collect the malware samples using open source threat intelligence reports from multiple vendors and then downloaded from VirusTotal. However, “*cyber-research*” omit the method they used to label the malware hashes from the 29 sources and so researchers have no assurances on the validity of the label. We note Haddadpajouh et al. [47] use a subset of [33], focusing on five groups namely APT1, APT3, APT28, APT33, and APT37. In both cases, we observed general issues with creating labeled APT malware datasets:

---

<sup>7</sup> “*cyber-research*” also include information on a further 855 samples which they could not obtain.



- **APT group names used for a single APT group often differ which leads to multiple aliases and not knowing which common name to use as the label.** In some cases, different groups share the same aliases. Either researchers linked multiple APT groups to the same nation or multiple APT groups potentially collaborated together. This makes it difficult to create a single list which contains a one-to-one relationship between sample and group. This problem relates to the one solved by Hurier et al. [56], who produce a distinct naming dataset for malware family names as anti-virus vendors use their own naming conventions.
- **Reports on APT groups often reference multiple groups when the researchers compare or link groups.** Therefore, researchers must take extra care when automatically extracting labels from the reports. For example, within [64] there exist the same reports linked to differing APT groups.

Due to these problems and the availability of APT datasets, some authors use alternative malware datasets or obtain datasets from private sources. Alrabaee et al. [4] obtain malware from their own Security Lab (Zeus and Citadel malware), from Contagio (Flame and Stuxnet malware) and from VirusSign (Bunny and Babar malware). They omit the method they use to determine the ground truth for this dataset. It appears Alrabaee et al. [6] use the same dataset and they state they manually determined the labeling. Alrabaee et al. [7] use a similar dataset but they add samples of the Mirai botnet to the dataset. The Microsoft Malware Classification Challenge dataset [102] provides an alternative popular malware source. Three works use subsets of this dataset [3, 7, 8]. The dataset by Ronen et al. [102] contains nine malware families<sup>8</sup> and currently there exist no links between the nine malware families and APT groups [84].

Four papers omit their malware sources and they all use cyber security experts to label their datasets [74, 103, 54, 104]. Only the works by Rosenberg et al. [103, 104] use datasets with labels representing the nation states which the APT groups are allegedly from or backed by. In particular, they use malware allegedly from or backed by two countries, namely Russia and China. Rosenberg et al. [104] state the dataset consists of four unique malware families in the training set<sup>9</sup>, with 400 samples from each family, and they use two unique malware families in the testing set<sup>10</sup>, with 500 samples from each family. Marquis-Boire et al. [74] use the smallest dataset containing only three samples (NBOT, Bunny and Babar) which they claim belong to the APT group named Snowglobe<sup>11</sup>. The alternative approaches to MAA by Kalgutkar et al. [58] and Gonzalez et al. [44] look to explore Android malware datasets. These works offer an interesting approach towards labeling the malware by the private certificates from the signed APK files. This approach is unique to Android malware and therefore fails to generalize. Gonzalez et al. [44] also perform manual analysis as they consider a lot more malware including APK files from Virus Total, Hacking Team and the Drebin dataset.

### 3.2.3 Comparison of Datasets

Table 3 provides an overview of all the different datasets used within the current research. We organized Table 3 as follows: we clustered all the columns relating to *benign source code and binaries* and then incorporate our discussion on *malware binaries* under the same titled column; we kept the *Ground Truth* column separate to highlight the various methods across both benign and malware datasets; finally, we recorded the largest number of authors and binaries considered in each work. We note the work by Alrabaee et al. [8] appears twice in the table due to the work using two distinct datasets for tackling the single and multiple author problem.

Overall, we observe the lack of systems using malware as the sole dataset. Among those papers which use malware, researchers use binaries collected from various sources and samples. This variety means there exists little overlap between the different datasets preventing true system comparison. In most cases, few samples exist for each author which makes it extremely hard for an attribution system to pick up on author style trends.

Limited datasets exist for the multiple authorship problem. Currently, researchers use benign source code from GitHub repositories to compile multiple author binaries or they synthetically create them from single author benign source code. Both these methods create binaries which represent the extremes of author style within a binary: the GitHub binaries contain many author styles distributed across the binary [34] and the synthetic binaries contain multiple author style separated into distinct sections within the binary [8]. Additionally, both datasets lack specific malware author style traits.

<sup>8</sup>Ramnit, Lollipop, Kelihos\_ver3, Vundo, Simda, Tracur, Kelihos\_ver1, Obfuscator.ACY, Gatak.

<sup>9</sup>Net-Traveler and Winnti/PlugX both allegedly China and Cosmic Duke and Sofacy/APT28 both allegedly Russia.

<sup>10</sup>Derusbi allegedly China and Havex allegedly Russia.

<sup>11</sup>This group allegedly associates with France.

Table 3: A summary of the largest datasets and sources used within the papers we reviewed published between 2011 and 2019. We include the toolchain process for the datasets created from source code and the method of author labeling to determine the “Ground Truth”.

Paper	Year	Benign Source Code and Binaries					Compilers <sup>c</sup>	Malware Binaries	Ground Truth <sup>d</sup>	Authors <sup>e</sup>	Binaries <sup>e</sup>	
		G CJ	Git Hub	Planet	Other <sup>a</sup>	Languages						Optimization <sup>b</sup>
Rosenblum et al. [106]	2011	✓			✓	C/C++	G		*	191	1,747	
Alrabaee et al. [3]	2014	✓				C/C++			*	7	□	
Marquis-Boire et al. [74]	2015							✓	•	3	3	
Meng [77]	2016		✓			C/C++	1	G	<	282	170	
Meng et al. [80]	2017		✓			C/C++	1	G	<	284	169	
Rosenberg et al. [103]	2017							✓	•	2	4,200	
Hendrikse [50]	2017	✓				C/C++	2	GLM	*	14	1,863	
Alrabaee et al. [4]	2017	✓	✓			C/C++	1	GIM <sup>f</sup> X	✓	*◇▷	1,000	□
Caliskan et al. [22]	2018	✓	✓			C/C++	3	G	*	600	5,400	
Meng and Miller [78]	2018		✓			C/C++	5	GIM	<	700	1,965	
Hong et al. [54]	2018							✓	•	7	1,088	
Alrabaee et al. [6]	2018	✓	✓	✓	✓	C/C++	2	GICM	✓	*◇	23,000	103,800
Rosenberg et al. [104]	2018				✓			✓	•	2	4,200	
Kalgutkar et al. [58]	2018		✓		✓	Java <sup>g</sup>		✓	◇•	40	1,559	
Gonzalez et al. [44]	2018		✓		✓	Java <sup>g</sup>		✓	◇•	30	420	
Laurenza et al. [66]	2018							✓	•	19	2,000+	
Alrabaee et al. [7]	2019	✓	✓			C/C++		GICM	✓	*•	21,050	428,460
Alrabaee et al. [8]	2019	✓	✓			C/C++	2	GICM	✓	*•	1,900	31,500
Alrabaee et al. [8]	2019	✓	✓			C/C++	4	GICM	✓	*	350	50
Haddadpajouh et al. [47]	2020							✓	•	5	1200	

<sup>a</sup> Other sources for benign datasets where ✓ means they state the source

<sup>b</sup> Number of Optimization Levels used (blank means paper does not state/consider)

<sup>c</sup> Compilers used: G - GCC/g++ I - ICC L - LLVM C - Clang M - Microsoft Visual Studio X - Xcode

<sup>d</sup> Ground Truth Method: \* - Source Code Author ◇ - Manually Determined < - *git-author* [79] ▷ - Undisclosed • - Cyber Security Experts/Malware Analysis Reports

<sup>e</sup> Largest Dataset Stated

<sup>f</sup> Alrabaee et al. [4] state they use Visual Studio in their methodology but include no dataset details.

<sup>g</sup> Android APK Files

□ - Not Disclosed.

In terms of authorship attribution, the researchers treat the APT binaries as single author which importantly introduces false author style links. The three most promising APT datasets created by Laurenza et al. [66], cyber-research [33] and Rosenberg et al. [104] exhibit flaws. The dataset by Laurenza et al. [66] contains many APT groups but few samples per group whereas the datasets by Rosenberg et al. [104] contains fewer groups but more samples per group. The dataset by cyber-research [33] lacks assurances surrounding the labeling process.

The issue of verifying the ground truth of the labels of the malware datasets still requires investigating. Source code authors appear easier to distinguish [59]. In comparison, the majority of malware requires manual analysis or cyber security experts. In the case of malware from the campaign titled ‘Olympic destroyer’, the threat actor used *false flags* to trick analysts into arriving at multiple attribution hypotheses. The original malware authors included specific code reuse from previous campaigns by other attackers. Additionally, they tried to confuse malware analysts by using different spoken language within the comments, user interface and function names. Various analysts discovered the various artifacts throughout the malware at different times and this led to attribution to groups from Russia, Iran, China and North Korea [14].

Fundamentally, using different datasets means each approach answers slightly different research questions. Furthermore, this suggests a lack of sharing and effort across the research field to try to solve the same problems. In Section 5, we hope to change this through the creation of an APT malware dataset which addresses the limitations and shortcomings we identify and publish this for the community to use for future research.

### 3.3 Author Features

Capturing author style provides the key to identification. The majority of the state-of-the-art methods determine author style through extracting multiple features and then completing feature ranking experiments using their data modeling techniques (Table 2) on their chosen ground truth dataset (Table 3). Researchers tend to extract various features based on domain expert knowledge or previous research. In some cases, the papers [8, 80] use features extracted directly from the binary through either vector or image representation for some experiments. In these cases, it remains unclear which features the model actually uses for author style which presents a gap in the research area. Going forward we review only those specific features which the papers explicitly stated.

Many of the state-of-the-art BAA systems rely on the area of *code stylometry* research to provide a starting point for features related to author style. Code stylometry features belong to three categories of *lexical*, *syntactic* and *semantic*. However, they omit any features from code execution. To mitigate this, Kalgutkar et al. [59] propose that researchers capture author style from *behavioral and application dependent* characteristics.

#### 3.3.1 Malware Author Style

Malware authors tend to have unique goals [99] which we can use to help determine the author style and extract features aimed towards capturing the goal of the malware author. Marquis-Boire et al. [74] remain the only paper to specifically consider malware features for author style through their aim of identifying credible links between APT malware. In particular, they pick up on malware programming style by APT groups such as the use of stealth, evasion and data ex-filtration techniques. Kaspersky [60] discuss similar themes from their binary similarity research but also widen the search to toolkits, exploits and targeted victim. We extend the ideas from these works with our previous discussion to devise five macro-categories, namely *strings*, *implementation*, *infrastructure*, *assembly language* and *decompiler* to compare the state-of-the-art systems in Section 3.3.3 along with providing further explanations of each category.

#### 3.3.2 Feature Extraction Tools

All these categories require tools able to extract features from varying aspects of the binary. We collated all the tools used in the eighteen systems in Table 4. This allows us to assess the popularity of each tool and understand why some tools are used more than others. We note most of the tools used are for static extraction. We observe the most popular tool as Dyninst [92] closely followed by IDA Pro/Hex-Rays [52]. The reason Dyninst most likely edges IDA Pro is due to Dyninst being open source. Five of the systems use multiple tools to extract different features [6, 7, 8, 22, 106], and this appears to be the best approach for extracting features from the five macro-categories we recommend in Section 3.3.1. Two unpackers, UPX [91] and a custom Android app, were used by Alrabaee et al. [7, 8] and Kalgutkar et al. [58], Gonzalez et al. [44] respectively. This shows the lack of interest in applying the current state-of-the-art methods to the malware domain. Only two dynamic analysis tools were used in total. Rosenberg et al. [103, 104] and Haddadpajouh et al. [47] both use Cuckoo Sandbox [46] and Hendrikse [50] uses DECAF [49]. Unfortunately, the tool used by Hong et al. [54] is undisclosed.

Table 4: A list of the tools used during the feature extraction process

Tool	Type	Extraction Technique	Attribution System
angr [111]	<i>ds</i>	○	[6]
BinComp [97]	<i>cp</i>	○	[7]
BinShape [110]	<i>o</i>	○	[6]
bjoern [73]	<i>ds</i>	○	[22]
Cuckoo Sandbox [46]	<i>s</i>	●	[106], [104], [47]
Custom Android App	<i>u, p</i>	○	[58], [44]
DECAF [49]	<i>s</i>	●	[50]
Dyninst [92]	<i>ds</i>	○	[106] <sup>1</sup> , [77], [80], [78], [7], [8] <sup>1</sup>
FLOSS [38]	<i>se</i>	○	[66]
FOSSIL [5]	<i>o</i>	○	[6]
IDA Pro/Hex-Rays [52]	<i>ds, d</i>	○	[3], [50], [22], [7], [8]
Jakstab [61]	<i>ds</i>	○	[7], [8]
Manually	<i>o</i>	◐	[74]
Netwide Assembler [35]	<i>ds</i>	○	[22]
Nucleus [10]	<i>ds</i>	○	[8]
pefile [27]	<i>o</i>	○	[66], [7], [8]
radare2 [95]	<i>ds</i>	○	[22]
Unknown tool used	<i>o</i>	◐	[54]
UPX [91]	<i>u</i>	◐	[7], [8]

<sup>1</sup> [106], [7] and [8] use *ParseAPI* which is now included within Dyninst [92].

**Key:-** ○ - Static Analysis ◐ - Static and Dynamic Analysis ● - Dynamic Analysis  
*ds* - disassembler *o* - other *s* - sandbox *u* - unpacker *p* - parser *d* - decompiler  
*se* - string extractor *cp* - compiler provenance

Overall, a total of nineteen tools were used. This shows there exists limited knowledge on whether extracting the same features via different tools affects the ability to capture authorship style.

### 3.3.3 Feature Comparison

We collated all the features from the eighteen systems and organized them into the five feature macro categories related to malware author style. In total, we collated 72 features. We structured the features by cross-referencing them against the systematization of the data modeling techniques from Section 3.1 and present the results in Table 5 and Table 6. Where possible, we condensed any papers into single columns which used exactly the same features. We also include the column “extraction techniques” to indicate the programming analysis techniques required to extract each feature. Due to all the “assembly” features requiring only static analysis extraction techniques, we present the categorization from the multiple author works [77, 80, 78] alongside the single author works. We include the column “Authorship Problem” in Table 6 for easier comparison across both the single and multiple author problem.

From this results, we remark there exists no favorable feature set for which the research field currently agrees upon. In fact, we recorded 42 unique features. In terms of feature extraction, researchers show a clear preference towards static analysis (36 features) and in terms of favorable macro-category then there exists a clear preference towards “assembly language”. In the following, we provide additional insight into the five macro-categories in terms of the application of these features for MAA. We use the macro-categories due to the vast number of features.

Table 5: State-of-the-art strings, implementation, infrastructure and decompiler features used in binary and malware authorship attribution research.

String Features	Extraction Technique	Classifying						Clustering		Anomaly Detection		Non ML
		[22]	[103]	[54]	[104]	[58]	[7]	[8]	[47]	[66]	[74]	
Artifact naming schemes/Algorithms	●										✓	
C&C Commands	●										✓	
Cuckoo Sandbox Report (Treated as Words)	●		✓		✓							
Encryption Keys	●										✓	
Errors	○										✓	✓
File Header	●							✓	✓			✓
Function Names	○										✓	✓
Grammar Mistakes	○										✓	
MS-DOS Header	○								✓			
N-Grams (Words)	○	✓				✓	✓				✓	
Optional Header	○								✓			
Operating System	●											✓
Programming Language Keywords	○	✓										✓
Timestamp Formatting	○										✓	
<b>Implementation Features</b>												
Binary Data Directories	○									✓		
C&C Parsing Implementation	●										✓	
Code Re-use	○										✓	
Compiler	○										✓	✓
Configuration Techniques	●										✓	
Constructor Design	●										✓	
Cyclometric Complexity	○							✓				
Execution Traces	●							✓				
File Interactions Traits (Locations, Modified, etc)	●			✓							✓	✓
Function Lengths	○									✓		✓
Multithreading Model (Use of Mutexes)	○			✓							✓	
Obfuscated String Statistics	○									✓	✓	
Obfuscation Functions	●										✓	
Propagation Mechanisms	●										✓	
Registry Keys	●			✓								
System API Calls	●			✓				✓	✓		✓	✓
System/OS Version Determination technique	○										✓	
Software Architecture & Design	●										✓	
Stealth and Evasion Techniques	●										✓	
Use of Global Variables	○										✓	
<b>Infrastructure Features</b>												
DNS URLs	●			✓							✓	
IP addresses (C&C Servers)	●			✓							✓	
Network Communication	●										✓	✓
User Agent/Beaconing Style	●										✓	
<b>Decompiler Features</b>												
Abstract Syntax Tree	○	✓										

Key:- ○ - Static Analysis ● - Static and Dynamic Analysis ● - Dynamic Analysis

Table 6: State-of-the-art assembly features used in binary and malware authorship attribution research. All assembly features are extracted using static analysis.

Assembly Features	Authorship Problem		Classifying					Clustering		Anomaly Detection	Structured Prediction	Non ML			
	[106]	[22]	[77, 80, 78]	[50]	[54]	[44]	[7]	[8]	[106]	[47]	[66]	[80, 78]	[3]	[74]	[6]
Annotated Control Flow Graph	○						✓								
Backward Slices of Variables	●		✓									✓			
Block Catches Exceptions	●		✓									✓		✓	
Block Position Within a Function CFG	●		✓									✓			
Block Throws Exceptions	●		✓									✓		✓	
Byte Codes	○											✓			
Call Graphlets	○	✓							✓	✓			✓	✓	
CFG Edge Types	●		✓									✓			
Constant Values	●		✓									✓			✓
Control Flow Graph Edges & Node Unigrams	○		✓												
Control Flow Graph Hashes	○				✓										
Data Flow Graph	○						✓								
Exact Syntax Template Library	○												✓		
Function (Opcode Chunks)	○				✓										✓
Function CFG Width & Depth	●		✓									✓			
Graphlets	○	✓							✓				✓		
Idioms (Instructions)	○	✓	✓						✓		✓	✓	✓		
Imports & Exports (Shared Libraries, Method Names)	○				✓					✓		✓	✓		
Inexact Syntax Template Library	○												✓		
Instruction Operand Sizes & Prefixes	●		✓									✓			
Library Calls	○	✓	✓						✓			✓	✓	✓	✓
Loop Nesting Level	●		✓									✓			
Loop Size	●		✓									✓			✓
N-Grams (Opcodes)	○	✓	✓				✓	✓	✓			✓	✓	✓	✓
Number of Basic Blocks	○														✓
Number of Input/Output/internal registers of a block	●		✓									✓			
Number of Live Registers at Block Entry & Exit	●		✓									✓			
Number of Used & Defined Registers	●		✓									✓			
Opcodes	○									✓					
Register Flow Graph	○				✓								✓		
Stack Height Delta of the Block	●		✓									✓			✓
Stack Memory Accesses	●		✓									✓		✓	✓
Super Graphlets	○	✓							✓				✓	✓	✓

Key:- ○ Single Authorship Problem ● Multiple Authorship Problem

**Strings.** These features capture any strings, artifacts and values within the malicious binary. The author influences any embedded *strings* and so there exists a wealth of knowledge on the author to gain from extracting strings. For example, strings may infer the native language of the author and therefore their potential location. Any naming conventions for both functions and artifacts infer any author personality and choices. Other significant choices for the author which strings help infer include programming language, encryption techniques and error handling messages.

Malware authors understand how much information can be leaked from strings and therefore continue to research methods for either removing author style or changing them to imitate another author. Freely available tools such as packers, obfuscators and strippers allow any author to remove their author style from strings/constants. The simple task of adding false artifacts or function names shall change author style within strings. We note the special case of the works by Rosenberg et al. [103, 104], who convert the MAA problem into a Natural Language Processing (NLP) problem through the use of analyzing Cuckoo Sandbox reports [46].

**Implementation.** Features in this category describe author choice involving both malware design and execution. Predominately, researchers extract these features during dynamic analysis which makes them much harder to obfuscate and mask. For example, the approach the author takes to interact with the victim (*e.g., propagation method*). Some of these features can be mimicked (*e.g., toolchain process*) and this potentially allows authors to imitate other authors. If dynamic analysis fails, then analysts must rely on much harder and more manual techniques to identify implementation features. These features also change depending on both the malware authors' development environment and victim's system making it harder to automate across varying types of malware.

**Infrastructure.** We use this category to describe any feature which relates to specific infrastructure choices made by the author, *e.g., choice of IP for command and control server*. If a threat actor reuses the same infrastructure, then it may offer an easy attribution decision. However, it might not be straightforward: for example, authors might attack other authors to use their infrastructure to imitate them or the author may loan out their infrastructure. We also expect sophisticated authors to change their infrastructure for each attack, or at the very least, mask identifiers such as IP addresses through methods such as IP spoofing or proxies.

**Assembly Language.** We collate any feature extracted from the assembly language representation of the binary. Researchers mainly extract them using static analysis which make them amenable to automation. These features focus on capturing instructions, control flow, data flow, external interactions and register flow. This can be either at the function level of the binary or much more fine-grained through the basic blocks of the program. Capturing both the program flow and more fine-grained features makes it harder for the author to modify them for adversarial purposes. The assembly language also presents an opportunity to feed it directly as a raw input into a Deep Neural Network (DNN) [80].

A malware analyst relies on the state-of-the-art disassembler to maintain author style. Otherwise, the authorship attribution problem morphs into the binary similarity problem. Furthermore, there exist multiple methods to build *basic blocks* and then *control flow graphs* (CFG) to understand the flow of the program. CFG include important program aspects such as *error handling, functions* and *library and system interactions*. Even when built, graphs provide further complications as the problem of subgraph isomorphism remains an NP complete problem [31]. Therefore, alternative representations must be sought. However, these alternatives then become approximations through statistical representations which increases the likelihood of losing authorship style. Finally, choices in the toolchain process (*e.g., CFG flattening*) present even more difficulties to overcome when building flow graphs.

**Decompiler.** This process attempts to recover the source code from the binary and remains an unsolved problem [101]. State-of-the-art decompilers such as IDA [52] recover code which closely represents the original source code, especially when no optimization or other code modifications occurred during the toolchain process. Source code recovery allows researchers to extract author style features determined from *source code authorship attribution, e.g., abstract syntax trees* [22]. However, these features rely heavily on the state-of-the-art decompilers and any binary modifications tend to highly impact the ability to recover them [36].

## 4 Real-world Application of State-of-the-art Systems

In this section, we provide an evaluation of the eighteen BAA systems identifying key findings and open research challenges (Section 4.1) and research recommendations (Section 4.2). We present our results in Table 7 and group the systems into the five data modeling techniques from Section 3.1. Our systems evaluation consists of reviewing the efficacy and functionalities of the eighteen systems. This comparison considers the applicability of the current state-of-the-art techniques to malware binaries and enables us to set out the future research directions. Here, we

define *Efficacy* as the accuracy of a system achieving its desired goal. We compare the efficacy of three experiments: (i) compiled source code, (ii) obfuscation, and (iii) malware on the largest datasets systematized in Table 3. For operational capability, we must consider the overall implementation of the system. Thus, we devise the following five categories to compare system *functionality* (based on the availability and reproducibility of a system):

- ⊖ **System currently not available.** We received no reply from our correspondence or the authors were unable to share the system.
- ⊘ **System partially available.** We were able to locate part of the system online but core components were missing.
- ⊙ **System does not compile.** We attempted to modify the source code and install previous dependencies. However, this ultimately was not possible.
- ⊗ **System contains errors at runtime.** If we managed to construct the system, we found errors occurred whilst attempting to run evaluation experiments which we were unable to patch.
- ⊕ **System completes.** The system was able to run a malware evaluation test.

In addition to this, we devise various categories for further comparison of the systems. We provide indication of the *ground truth* used to perform the experiment, namely source code, binaries or Android applications. We compare the systems by the addressed *authorship problem* (*single or multiple*), and the *feature extraction techniques* used (static, dynamic or a combination of both). We also compare whether the researchers implemented *parallelization* or *cross-validation* evaluations, and whether they took into account *toolchains* or *shared libraries*. From our author style feature systematization (Section 3.3), we compare the systems over the five macro-categories (*i.e.*, *strings*, *implementation*, *infrastructure*, *assembly* and *decompiler*). Finally, using the discussion in Section 2.4, we explore whether any researchers consider *adversarial* challenges and *privacy* implications to their authorship attribution systems using the following categories we devised:

*Adversarial:*

- Researchers do not consider any attacks
- Researchers consider unsophisticated attacks
- Researchers consider sophisticated attacks

*Privacy:*

- Researchers do not consider privacy implications
- Researchers mention privacy implications
- Researchers discuss the privacy implications

#### 4.1 Key Findings and Open Research Challenges

From the criteria above, and from the results shown in Table 7, we categorize our key findings as follows. First, we explore *System Goal and Datasets* focusing on ground truth and authorship problem solved. Then, we examine the effect of *Languages*, *Code Re-use and Toolchains* and *Attribution Features and Extraction Methods* on BAA systems. Next, we examine the *System Functionality* and in particular consider the importance of training, reproducibility of results and availability for the state-of-the art systems. Finally, we explore the open challenges with regards to *System Efficacy* and *Adversarial Considerations* as well contextualizing the implications of these systems on *Privacy and Ethics*.

**System Goal and Datasets.** We note the majority of the systems focus on the single authorship problem. All the systems use *closed-world assumption* and in the majority of cases use classification systems. This makes them impractical for use in the “wild”. Rosenblum et al. [106] remains the only paper to partially consider the open world problem by training a model based on the closed world model. However, the datasets used to train, test and evaluate the systems contain minimal consistency especially for the datasets containing malware. Although the use of compiled source code repositories provides a ground truth, it brings extra complexities with the necessity to consider all toolchain possibilities. There exists no verifiable, publicly available and sufficiently large APT dataset which researchers can use to build MAA systems. All of the current attribution systems use datasets which are static, leading to a discontinuous learning model which is likely to experience concept drift. This is a wider problem within machine learning, deep learning and artificial intelligence models. Kolosnjaji et al. [62] show concept drift occurs within malware detection models based on unevolving datasets.



Table 7: Comparison of the Analyzed Systems between 2011 and 2019. Organized by data modeling technique and cross-referenced against ground truth, authorship problem, features, system efficacy, system functionality and adversarial considerations.

	Paper	Year	Features										Efficacy <sup>d e</sup>			Functionality <sup>f</sup>	Adversarial <sup>g</sup>	Privacy <sup>h</sup>
			Original Groundtruth <sup>a</sup>	Authorship Problem <sup>b</sup>	Analysis <sup>c</sup>	Parallelization	Cross-Validation	Toolchains	Shared Libraries	Strings	Implementation	Infrastructure	Assembly	Decompiler	Source Code (%)			
Classifying	[106]	2011	S	○	○		✓					✓	51	F 58	ACC 34	⊗	□	□
	[77]	2016	S	●	○		✓		✓			✓	52	—	—	⊗	□	□
	[80]	2017	S	●	○	✓	✓		✓	✓		✓	58	—	—	⊗	□	□
	[103]	2017	M	○	●		✓			✓			—	—	94.6	⊗	□	□
	[50]	2017	S	○	●		✓	✓	✓			✓	95.3	94.1	—	⊗	□	□
	[22]	2018	S	○	○		✓			✓		✓	83	88	ACC 70	⊗	□	■
	[78]	2018	S	●	○	✓	✓	✓	✓			✓	71	—	—	⊗	□	□
	[54]	2018	M	○	●		✓			✓	✓	✓	—	—	AF 88.2	⊗	□	□
	[104]	2018	M	○	●		✓			✓			—	—	99.75	⊗	□	□
	[58]	2018	A	○	○		✓			✓			98	77	96	⊗	□	□
	[44]	2018	A	○	○		✓					✓	86.74	—	66.92	⊗	□	□
	Clustering	[7]	2019	S	○	●		✓	✓	✓	✓		✓	F 94	—	CC 96.9	⊗	□
[8]		2019	S	○	○		✓	✓	✓		✓	✓	P 84	—	P 45	⊗	□	□
[8]		2019	S	●	○		✓	✓	✓		✓	✓	P 89	—	—	⊗	□	□
[106]		2011	S	○	○		✓				✓		AMI 45.6	—	—	⊗	□	□
[47]		2020	M	○	●				✓	✓		✓	—	—	95	⊗	□	□
[66]		2018	M	○	○		✓		✓	✓		✓	—	—	98	⊗	□	□
Structured Prediction	[80]	2017	S	●	○	✓	✓		✓	✓		✓	65	—	—	⊗	□	□
	[78]	2018	S	●	○	✓	✓	✓	✓			×	—	—	—	⊗	□	□
Non-ML	[3]	2014	S	○	○				✓			✓	84	F 25	ACC 69.75	⊗	□	□
	[74]	2015	M	○	●					✓	✓	✓	n/a	n/a	n/a	n/a	□	□
	[6]	2018	S	○	○		✓	✓	✓	✓	✓	✓	P 49	P 95	Re 68	⊗	□	□

<sup>a</sup> S - Source Code M - Malware A - Android Applications

<sup>b</sup> ○ Single Authorship Problem ● Multiple Authorship Problem

<sup>c</sup> ○ Static Analysis ● Static and Dynamic Analysis ● Dynamic Analysis

<sup>d</sup> × Experiment Incomplete — No Experiment Considered na Not Applicable

<sup>e</sup> All accuracy unless precedes with: F -  $F_1$  measure [except for Alrabaee et al. [7] who define and use  $F_{0.5}$ ] AF - Average  $F_1$  score AMI - Adjusted Mutual Information ACC - Average Correctly Clustered P - Precision CC - Correctly Clustered

Re - The average accuracy in relation to a malware analysis report

<sup>f</sup> ⊗ System Not Available ⊙ System Partially Available ⊖ System Does Not Compile

⊗ System Contains Errors At Runtime n/a Not Applicable

<sup>g</sup> □ Researchers do not consider any attacks □ Researchers consider unsophisticated attacks ■ Researchers consider sophisticated attacks

<sup>h</sup> □ Researchers do not consider privacy implications □ Researchers mention privacy implications ■ Researchers discuss the privacy implications

Malware development follows a similar agile work flow process to benign software development where multiple authors collaborate [24], as in the recent GandCrab ransomware campaign [51]. However, there exists limited research exploring multiple authorship within MAA. Even in BAA, only four out of the eighteen papers consider multiple authorship for a program. From the four multiple author focused papers, there still remain research gaps such as applying these techniques to malware. However, there exist many challenges with this approach. This includes overcoming both obfuscation and packing techniques. Additionally, it remains unclear whether the features they use help with clustering multiple malware authors.

**Languages, Code Re-use and Toolchains.** Code re-use from other software and libraries impact attribution systems and this can lead to the incorrect author attributed. We observe only eight systems attempt to account for the effect of shared libraries on authorship style. Previous works all attempt to remove standard libraries from the binaries before extracting author features [77, 80, 78, 3, 6, 7, 8, 50]. These works only focus on removing C/C++ libraries due to their datasets containing binaries compiled from C/C++ source code. In fact, none of the state-of-the-art systems consider any other programming languages.

However, authors write malware in multiple languages [25] and thus a programming language gap exists when it comes to identifying the malware author. Therefore, we believe using systems trained only on compiled source code datasets to label unknown malware hinders a MAA system. Further issues exist if the programmer adheres to language standards where a strict format must be followed, *e.g.* the style guide for Python (PEP 8 [117]). Standards are most likely to significantly reduce the amount of author style within a program as everyone will produce similar looking code. However, the speed of malware development must match the speed at which it requires deploying<sup>12</sup> and this determines the likelihood of a malware author following standards. In any case, future research should consider features which are robust against any standards to prevent this becoming an attack method to the attribution systems themselves.

The re-use of code within benign programs is common practice and malware development is no different. Within malware, there exists a lot of code re-use from both open and closed sources due to the pressure of beating vulnerability patching or meeting the demands of cyber warfare to complete mission objectives. Code re-use can be both helpful and unhelpful. In fact, a lot of code reuse from other authors contaminates samples and leads to an even smaller dataset to learn author style. For example, if someone leaks the source code then this quickly leads to multiple copycat attackers. On the other hand, code reuse of the actual malware author helps identify malware written by the same authors [105]. Only three papers within the current research consider the effect of toolchains on their attribution system [78, 50, 6], meaning there still exist questions regarding the impact of compilers on author style. However, if we consider a malware dataset then the choice of compiler is predefined by the author and so by default we automatically would train upon a dataset which potentially used various compilers. This may answer why using a model trained on compiled source code provides limited aid when applying it to malware.

**Features, Extraction and Style.** We observe *strings* and *assembly language* as the two most popular feature macro-categories for author style and this also correlates with static analysis as the most popular technique to extract features. These popular macro-categories omit key malware specific features and traits which experts tend to discover among APT author style. The common extraction process used involves static analysis, likely due to the “quickness” it provides over dynamic analysis. Furthermore, there exist multiple tools for binary analysis which achieve similar tasks. This leads to further research questions surrounding the effect of extraction tools on author style.

There exists limited research around finding malware author style. The goals of benign software programmers clearly differ to malicious software programmers and yet most of the research focuses on only the benign stylometry approach of lexical, syntactic and semantic features on assembly language and these methods ignore malware specific features. In the case of multiple authors, the state of the art mainly identifies fine-grained features (*e.g.*, basic block exception handling) [77, 80, 78, 8] and this differs to the features identified by Marquis-Boire et al. [74] for linking malware authors (*e.g.*, languages used, command and control server setups and obfuscation techniques used).

**System Functionality.** The majority of systems we tested, retrained their systems to perform each of the evaluation tests<sup>13</sup>. The system which showed higher performance capability in some cases required a training time of a week [80] with the most likely explanation of using no parallelization techniques. After spending a considerable amount of time and effort, none of the eighteen systems we tested fell into the “System completes” category and this provides the main reason for a shortage of further research within this field. Although the published results show promise, the lack of consistency with the evaluation metrics makes it hard to validate the results without further testing.

<sup>12</sup>The window of deployment depends on the availability of a vulnerability patch.

<sup>13</sup>Unfortunately, continuously retraining your system to account for new discoveries in the “wild” remains a resource intensive task.

**System Efficacy.** Not all the papers performed experiments using source code, obfuscation or malware experiments<sup>14</sup> and in some unique cases the system takes too long to complete [78] or the method used is not applicable [74]. In terms of presented results we gathered, only 19 out of 34 used the accuracy metric. Therefore, we considered an alternative metric for the remaining 15 results. This highlights the lack of consistency on evaluation metrics across the field. Let us examine the three types of experiments:

- **Source Code.** The results vary considerably and this makes it difficult to compare the systems. The later systems seem to perform better. This appears to be down to the progress of extraction techniques which allow researchers to remove some external noise (*e.g.* system libraries) from the binaries.
- **Obfuscation.** From the very few experiments, it remains impossible to tell whether the problem is solved due to the inconsistency of results. The result by Hendrikse [50] appears the most promising. This is due to the thoroughness of obfuscation techniques considered, and even though they considered the fewest number of features, the prominent difference to the other systems is the inclusion of dynamic features.
- **Malware.** Comparing the malware experiments is much harder, as the goals of the systems differ slightly. The datasets used were also considerably smaller, and the researchers undertook considerable efforts to clean the datasets. It is these reasons which explain the considerably high accuracy attained. This is not necessarily bad as it could help malware analysts examine a small subset of malware which they believe originate from the same author. Even if we were to consider much larger and dirtier datasets, then the state-of-the-art systems remain unlikely to produce the same levels of accuracy.

We note the inconsistency of datasets encumbers the comparison of the systems. A prime example of this inconsistency is Caliskan et al. [22], who report better efficacy on obfuscation than Alrabaee et al. [4] despite appearing to use the same method for obfuscation experiments. Even though both papers report different metrics, we state the reasons we think there exists a higher accuracy in the later results. Firstly, we believe Alrabaee et al. [4] used an older version of the system from [22] as they published their paper first. Secondly and most importantly, they both use different datasets.

**Adversarial Considerations.** From the table, only four<sup>15</sup> of the eighteen systems [22, 50, 58, 6] considered basic attacks, *e.g.* obfuscation. This highlights the lack of adversarial considerations towards any binary attribution system. Even those researchers who implemented unsophisticated attacks (*e.g.*, obfuscation) on their systems, reported an increase in the amount of manual assistance needed to de-obfuscate the binaries. This meant the systems became more semi-automated. Out of all the single authorship methods, Hendrikse [50] provides the most comprehensive evaluations using readily available obfuscation tools which range from very easy to hard techniques. However, their attribution system uses the fewest amount of features which opens itself to targeted and untargeted attacks. This is because their system uses fewer features than Caliskan et al. [22] and Meng et al. [81] show the binary attribution system created by Caliskan et al. [22] is open to both: targeted and untargeted attacks. Meng et al. [81] extend the attacks by Carlini and Wagner [26] designed for DNNs trained for image labeling. Meng et al. [81] generated a method to modify the feature vector and the binary. When they modify the binary they ensure the binary still executes which is a fundamental requirement for a successful binary modification attack. We predict this method of attack works for all the other single author systems too. Therefore, the majority of single author state-of-the-art systems remain open to both unsophisticated and sophisticated attacks.

**Privacy and Ethics.** The majority of systems use author style features developed from benign source code author identification rather than focusing on malicious author styles. This means these systems and techniques can be used to identify benign software developers who might create programs to avoid detection in nations which prevent freedom of speech. Furthermore, these authors may have previously submitted software to the benign sources used by many of the systems. The authors may be unaware of researchers using their software. This not only violates their privacy rights but this raises ethical questions surrounding the further use of the benign datasets.

## 4.2 Recommendations

**Real-world Application.** None of the MAA systems we reviewed appear immediately ready for implementation in the “wild”. There exists a lack of sufficient details to replicate the systems. Anyone wishing to join this research field must start from scratch and redo the majority of previous work. Furthermore, limited results on malware exist

---

<sup>14</sup>We utilized the results of the survey by [4] to incorporate the obfuscation and malware experiments using the systems from [106, 22, 3]. However, [4] omit the accuracy results for these experiments and instead use  $F_{0.5}$  for the F-measure as they claim the systems in [106, 22, 3] are extra sensitive to false positives.

<sup>15</sup>The obfuscations experiments for systems [106] and [3] were computed in the survey by Alrabaee et al. [4].

meaning it is unknown whether the current techniques are effective for real-world use. Additionally, most systems require intense manual analysis and significant training times further showing these systems are unready for operational deployment.

**Privacy.** Although these systems are aimed at detecting malicious authors, they can be used to detect benign software users and this raises privacy concerns. This provides further evidence future research must focus purely on malicious author styles. Few of these works consider the privacy and anonymity implication of the developed tools. Therefore, we believe MAA systems should also be tested in other contexts than that of malware written by a threat actor to measure their efficacy and impact in benign scenarios.

**Adversarial Approach.** None of the analyzed papers consider sophisticated adversarial testing. We suggest any MAA system must undergo adversarial testing before deployment. In particular, it must show robustness to sophisticated attacks like the one described by Meng et al. [81] which we predict works for all current single author binary attribution systems. There also exists no research into adversarial attacks on multiple authorship attribution systems. In the future, we predict all APT malware authors shall implement sophisticated attacks to remain anonymous and avoid law enforcement.

**Datasets.** In general, there lacks both a consistency of performance metrics and datasets used across the research field. Systems which trained upon source code and were then used to identify malware author performed worse than those systems which originally trained upon malware. The datasets used played a pivotal role in these systems and most of them lacked a variety of programming languages or ability to cope with the effect of shared libraries and compilers. We hope the creation of our APT malware dataset in Section 5 allows a fair comparison among future systems.

**Multiple Authors.** The single author assumption fundamentally hinders the ability to determine the author of binaries developed by multiple authors (*agile software development*), especially as the commercialized malware industry uses agile work flow methods to speed up the development process to both increase profits and beat vulnerability discovery time. Being able to see if authors are used across multiple malware development projects shall provide insight within the malware development industry and introduce a new method of tracking malicious threats, especially APT groups. To further aide this we suggest all future work should adopt our approach of considering features from the five feature macro-categories of: *strings, implementation, infrastructure, assembly language* and *decompiler*. This allows for all aspects of malware author styles to be captured.

## 5 APTClass: Creation of an APT Malware Dataset

From our discussion in Section 3.2 on datasets, we deemed it a high priority to ensure there exists a sufficiently large and diverse dataset accessible to research for use in discovering malware authorship style and creating malware authorship identification systems. In this section, we set out how we created **APTClass**, a meta-information dataset consisting of 15,660 labeled malware samples. Our overall approach follows a similar method to Laurenza and Lazeretti [64]: we gather a large amount of open-source intelligence (OSINT) and then we perform preprocessing on the data before extracting information. In addition, we propose a novel method for label identification and extraction to solve the issues discovered in Section 3.2.2 and because of our focus on labeling we only extract malware hashes. This can be extended to include *URLs, IP Addresses, or Tactics, Techniques and Procedures* as shown by previous works [70, 120]. Our novel label extraction method uses a matching algorithm which combs the OSINT in a systematic process to match against a list of 1,532 APT group names. We describe this process in detail in Section 5.1.

### 5.1 Method

APTClass follows five steps: (i) create a list of APT groups and group them by alleged nation; (ii) gather OSINT, mainly PDF reports of attack campaigns; (iii) extract hashes and label from the gathered intelligence; (iv) clean the dataset by removing duplicate malware hashes and use VirusTotal [29] to verify the legitimacy of the samples gathered in step (iii); and finally (v) filtering for executable binaries. For the purpose of this dataset, APTClass considers executable files as ELF, Windows 32 EXE, Windows 16 EXE, Windows Installation file and Windows DLL.

#### 5.1.1 Creating a consistent list of APT labels

To overcome the issues of multiple aliases introduced by various analysts, APTClass treats each name as a unique group. Although this initially inflates the number of groups and introduces some duplication (*e.g., group 123* and

Table 8: List of sources used for creating a consistent list of APT labels.

Source Name	Last Updated
MISP [83]	October 2020
APT Operation Tracker [113]	October 2020
MITRE ATT&CK [84]	October 2020
sapphire00 [108]	Nov 2018
Thailand CERT [115]	October 2020
Council on Foreign Relations [32]	October 2020

*group123* are listed separately), we believe this to be the correct approach as often analysts cannot reach a consensus regarding groups and may use different names within OSINT when referring to the same group. APTClass still captures any nation link for a group. From our experience, analysts tend to have a higher confidence on linking groups to nations. APTClass also records whether a group is linked to multiple nations to account for mis-attribution. APTClass extracts the nation and group names from six sources in Table 8 using the process set out in Algorithm 1. Essentially, APTClass extracts from the sources a *dictionary* with *nations* as *keys* and a *list of group names* as *values*. APTClass then standardizes the names and removes duplicates over the six dictionaries. This approach identified 1,532 names. We are aware there exists duplication among sources, however, this helps further validate the list of names as well as increase the varying aliases for each group.

---

**Algorithm 1:** Creating list of APT names

---

**Input:** *sources\_list*

**Output:** *final\_list*

**Function Main:**

```

for source in sources_list do
    dictionary(nation : group_name_list) = extract_nation_and_names(source)
    // returns a dictionary, with nations as keys and list of group names as values
    for each nation do
        group_name_list = standardize(group_name_list)
        // removes punctuation and converts to lowercase
        group_name_list = remove_duplicates(group_name_list)
        // removes any duplicates from the list of group names
    end
    for name in group_name_list do
        if (nation, name) not in final_list then
            | final_list.append((nation, name))
        end
    end
end
final_list = group_nations(final_list)
// joins together groups from the same nations
return final_list

```

**return**

---

### 5.1.2 Gathering open-source threat intelligence

We performed an extensive search on GitHub for trustworthy repositories containing any OSINT information. In particular, we focused on repositories storing (i) reports (typically PDF files), (ii) indicator of compromises (IoC), and (iii) YARA rules. We chose GitHub as the majority of OSINT is shared on the platform from other researchers collecting their own repositories of intelligence. We wanted to collate as many files as possible to ensure we maximized the number of malware hashes.

### 5.1.3 Extracting hashes and labels

We are aware of many OSINT parsers, however, these just extract the indicators of compromise [12, 53] or try to gather tactics and techniques for groups [68] without extracting the most important piece of information for our own purpose: APT labels and hashes. Thus, we required a new approach to gather a likely label for the malware hash.

APTClass provides a fine-grained approach to extracting the label. We set out this technique in Figure 1 and describe the process below:

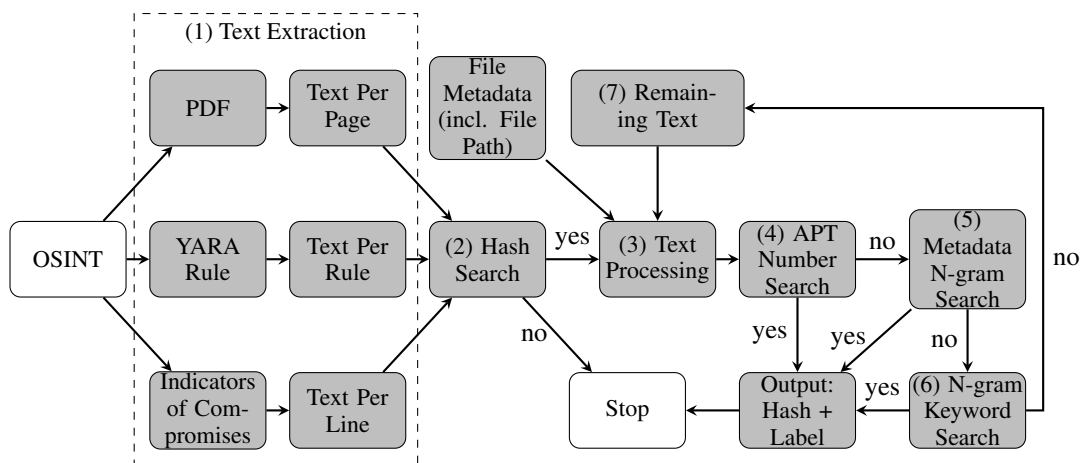


Figure 1: A high-level view of the extraction process for APTClass.

1. **Text Extraction:** APTClass extracts the text per page of PDF reports, text per YARA rule and text per line of IoC files. This allows APTClass to try and identify the best possible label closest to the hash.
2. **Hash Search:** We perform a regular expression search on the extracted text for any MD5, SHA1, SHA256 or SHA512 hashes.
3. **Text Processing:** APTClass removes punctuation, stop words and hashes from the text. The stop words consist of stop words from NLTK [17], spaCy [55] and gensim [98] as well as any cyber words in the dictionary created by Bishop Fox [18] and words previously determined “noise” from running APTClass multiple times.
4. **APT Number Search:** APTClass performs an extensive search against the APT label list looking for a match with either:
  - APT<number>,
  - APT-C-<number>,
  - ATK<number>,
  - SIG<number> or
  - FIN<number>

We do this as these labels tend to be extremely popular labels among analysts. APTClass only uses this as a label if there is a clear majority within the matches. APTClass also designates this match as the label when there is no further match against the APT label list created in Section 5.1.1, *i.e.* steps (5-7) all fail.

5. **Metadata N-gram Search:** APTClass considers a n-gram word search on the metadata. Due to the likelihood of duplication within the OSINT, APTClass also includes any metadata of the same file. APTClass considers all possible word n-grams of the metadata. The logic for this is the metadata is likely to include the original filename and any keywords attached by the author of the report. We also include the file path as part of the metadata as the analyst is likely to store the reports in the most relevant folder and therefore using previous file paths increases our chance of matching the right label.
6. **N-gram Keyword Search:** APTClass extends the n-gram search to additionally include the extracted text. APTClass performs the match based on all possible n-grams of the top five keywords extracted. We empirically verified in most cases the correct label lies among the top words. APTClass uses the top five keywords but this can be increased until APTClass achieves a exact match with a corresponding linear increase in processing time.
7. **Remaining Text:** If APTClass fails to identify a label for a hash in steps (4-6) then it stores the text and repeats steps (3-7) using this remaining text. If it fails again then the label will be a dictionary consisting of the top five keywords from the full text and the keywords from the metadata.

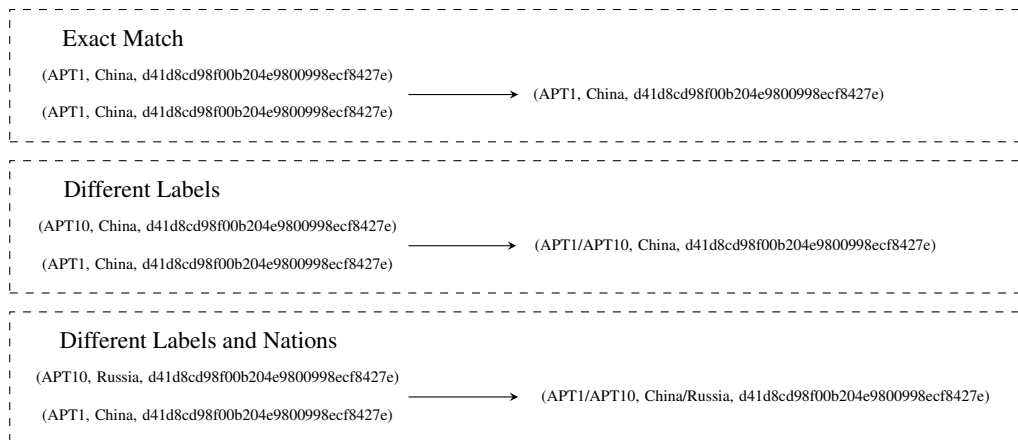


Figure 2: Example of APTClass cleaning process.

### 5.1.4 Cleaning, verifying and filtering

Before checking the hashes discovered from the extraction process, APTClass cleans the data by joining identical hashes and collates any information which suggests mis-attribution. APTClass joins any samples with a exact match (*i.e* the hash, the group name and group nation are identical). Next APTClass joins any samples where the hash and the nation are identical but the labels differ; in this case APTClass concatenates the labels. Finally, APTClass joins any remaining samples with identical hashes but nations and labels differ; in this case APTClass concatenates the labels and concatenates the nations. We provide an example of this cleaning process in Figure 2. Once this step is complete, APTClass submits each MD5, SHA1 and SHA256 sample to VirusTotal to check the malware legitimacy, the file type and the corresponding hash values. After this, APTClass repeats the cleaning step above and joins together any labels for identical hashes. Finally, APTClass filters for executable samples.

## 5.2 Results

We run APTClass using the sources listed in Table 8, including 373 report files, 504 IoCs and 19 Yara Rules. The analysis takes approximately 116 hours<sup>16</sup> on a Ubuntu 16.04 Virtual Machine equipped with 16 vCPU and 16GB RAM. At the end of this process, APTClass returns a list of 15,660 labeled samples. The results are shown in Table 9, together with a comparison of existing APT datasets. As we see from Table 9, APTClass is comfortably larger than both [64] and [33]. Unfortunately, there lacks the availability of the OSINT used within [64] and [33] and so we cannot run APTClass on the same reports to see if there is any comparison. However, we believe the issues discussed in Section 3.2.2 and slight difference in goals of the three systems makes it very difficult to compare datasets in terms of the granularity within Table 10.

Table 9: Comparison of our dataset against both [64] and [33].

	[64]	[33]	APTClass
Total labeled Samples	8,927 <sup>a</sup>	3,594 <sup>b</sup>	<b>15,660</b>
Number of groups	88	12	<b>164</b>
Number of threat intelligence files processed	821	33	<b>896</b>
Total unknown samples	N/A	N/A	<b>7,485</b>
Number of groups with 50+ samples	N/A	11	<b>37</b>
Number of groups with 25+ samples	N/A	12	<b>54</b>

<sup>a</sup> This includes file types other than ELF, Windows 32 EXE, Windows 16 EXE, Windows Installation file and Windows DLL.

<sup>b</sup> cyber-research [33] include information on a further 855 samples which are not on VirusTotal.

APTClass creates an overall diverse dataset with 164 APT groups from which we can create a concentrated subset consisting of 37 groups with 50 or more samples. In Table 10, we provide a breakdown of the results by the 13 nations

<sup>16</sup> Approximately 80% of the time taken is accounted by the *cleaning, verifying and filtering* process, this is determined in the *verifying process* by the rate limit of the VirusTotal API.

Table 10: The number of SHA256 hashes per Nation and APT Group.

Nation	APT Group	APT Group	APT Group
China	5,548	apt10	548
India	417	apt17	2462
Iran	637	apt27	85
Israel	5,000	apt28	500
Italy	6	apt29	93
Lebanon	26	apt33	83
Libyan Arab Jamahiriya	1	apt37	77
DPRK	1,236	apt40	103
Pakistan	8	be2	110
Russia	1,658	black vine	316
Turkey	89	blackgear	270
United States	74	blacktech	333
Vietnam	679	cleaver	112
		comment crew	260
		confucius	87
		darkhotel	94
		fin7	181
		gamaredon group	159
		higaisa	53
		icefog	90
		infy	189
		kimsuky	77
		lazarus	1046
		mirage	75
		muddywater	63
		oceanlotus	679
		patchwork	282
		promethium	89
		rtm	88
		scarlet mimic	61
		sig17	4,992
		silence	65
		ta505	171
		thrip	105
		tick	70
		tropic trooper	59
		turla	86

(without potential mis-attribution) and the 37 groups with 50 or more samples. Although there exists a clear disparity among the nations, this reflects the information sources and publicly known attacks. Similarly among APT groups there are certain groups where there are considerably more samples linked to them (e.g. *APT17 - China* and *SIG17 - Israel*), which reflects the samples by nation with both *China* and *Israel* linked to the most amount of samples. Overall, Table 10 mirrors the observations made in Section 2.1 and those seen in Table 1. In fact, this further highlights the bias towards non-Western nation sponsored APT groups. Interestingly, only two APT groups (*Oil Rig* and *Emissary Panda*) of the 2020 top ten are not included in Table 1. Additionally, the group *kimsuky* is linked to 77 samples compared to zero in Table 1. In general, the number of samples vary considerably to Table 1 which is most likely because not every threat intelligence company shares their intelligence with MITRE.

### 5.3 Discussion

Even though we focus purely on extracting malware hashes from OSINT, APTClass can be enriched by extracting other indicators or relevant information from OSINT such as *Tactics, Techniques and Procedures* and *malware families* to build further datasets for wider research into malware analysis. APTClass also allows the user to select the sources used for the creation of the APT label list (Section 5.1.1) and OSINT collection (Section 5.1.2).

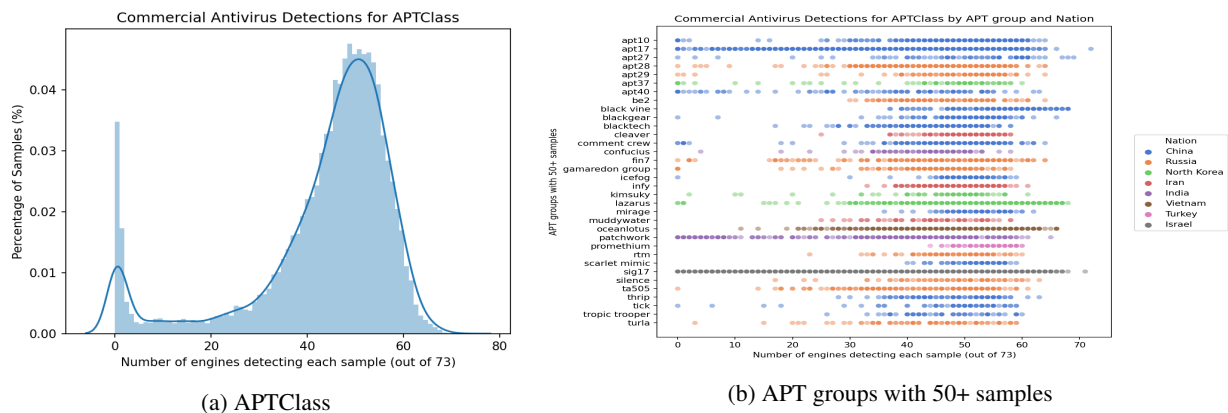


Figure 3: The detection results against up to 73 commercial anti-virus engines.



One additional use of APTClass is it can help produce new methods for malware detection. APTClass offers a different perspective on traditional detection methods as well as testing them on the most sophisticated malicious techniques. We show this by including the detection results of APTClass against up to 73 commercial anti-virus engines from VirusTotal in Figure 3. In Figure 3a, we see there exists a small proportionate of APT malware which no anti-virus engines detect. Interestingly, this issues is not specific to one APT group or Nation (Figure 3b). These graphs highlight an unsolved problem within malware detection. Furthermore, APTClass offers a unique niche dataset for testing data modeling techniques used in the malware domain. Specifically, we can see APTClass being used to further develop and understand sophisticated adversarial attacks.

Due to the cross-domain benefits APTClass provides, we publish the code and dataset for this joint project at <https://s3lab.isg.rhul.ac.uk/aptclass>. We additionally welcome contributions towards evolving APTClass to continually support the research community.

## 6 Conclusion

We presented a comprehensive survey of the Malware Authorship Attribution problem by focusing on threat actor style and adversarial techniques to the current state-of-the-art systems. We specifically examine the current data modeling techniques, datasets and features used for malware authorship style. We compared the results of eighteen binary attribution systems and identified the current limitations of state-of-the-art techniques. Surprisingly, we found most of these limitations apply to all of the eighteen systems, which shows a lack of progression. Therefore, we envision our work as a source of stimulation for future research, especially for new practitioners. Furthermore, we mitigated the issue of lack of author labeled malware dataset by creating a verified dataset containing 15,660 APT samples linked to 164 APT group names and 13 nations. This is the largest dataset of this type publicly available, and can be used by researchers and practitioners as a common ground to test and compare their approaches.

## References

- [1] S. Afroz, A. C. Islam, A. Stolerman, R. Greenstadt, and D. McCoy. Doppelgänger finder: Taking stylometry to the underground. In *2014 IEEE Symposium on Security and Privacy*, pages 212–226, May 2014. doi: 10.1109/SP.2014.21.
- [2] I. Albluwi. Plagiarism in programming assessments: A systematic review. *TOCE*, 20(1):6:1–6:28, 2020. doi: 10.1145/3371156. URL <https://doi.org/10.1145/3371156>.
- [3] S. Alrabaee, N. Saleem, S. Preda, L. Wang, and M. Debbabi. Oba2: An onion approach to binary code authorship attribution. *Digital Investigation*, 11:S94 – S103, 2014. ISSN 1742-2876. doi: <https://doi.org/10.1016/j.diin.2014.03.012>. URL <http://www.sciencedirect.com/science/article/pii/S1742287614000176>. Proceedings of the First Annual DFRWS Europe.
- [4] S. Alrabaee, P. Shirani, M. Debbabi, and L. Wang. On the feasibility of malware authorship attribution. In F. Cuppens, L. Wang, N. Cuppens-Bouahia, N. Tawbi, and J. Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 256–272, Cham, 2017. Springer International Publishing. ISBN 978-3-319-51966-1.
- [5] S. Alrabaee, P. Shirani, L. Wang, and M. Debbabi. Fossil: A resilient and efficient system for identifying foss functions in malware binaries. *ACM Trans. Priv. Secur.*, 21(2):8:1–8:34, Jan. 2018. ISSN 2471-2566. doi: 10.1145/3175492. URL <http://doi.acm.org/10.1145/3175492>.
- [6] S. Alrabaee, P. Shirani, L. Wang, M. Debbabi, and A. Hanna. On leveraging coding habits for effective binary authorship attribution. In J. Lopez, J. Zhou, and M. Soriano, editors, *Computer Security*, pages 26–47, Cham, 2018. Springer International Publishing. ISBN 978-3-319-99073-6.
- [7] S. Alrabaee, M. Debbabi, and L. Wang. On the feasibility of binary authorship characterization. *Digital Investigation*, 28(Supplement):S3–S11, 2019. doi: 10.1016/j.diin.2019.01.028. URL <https://doi.org/10.1016/j.diin.2019.01.028>.
- [8] S. Alrabaee, E. B. Karbab, L. Wang, and M. Debbabi. Bineye: Towards efficient binary authorship characterization using deep learning. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part II*, pages 47–67, 2019. doi: 10.1007/978-3-030-29962-0\_3. URL [https://doi.org/10.1007/978-3-030-29962-0\\_3](https://doi.org/10.1007/978-3-030-29962-0_3).
- [9] V. M. Alvarez. YARA, 2020. URL <https://virustotal.github.io/yara/>.
- [10] D. Andriess, A. Slowinska, and H. Bos. Compiler-agnostic function detection in binaries. pages 177–189, 04 2017. doi: 10.1109/EuroSP.2017.11.

- [11] Apple. Xcode, 2020. URL <https://developer.apple.com/xcode/>.
- [12] armbues. ioc\_parser, 2015. URL [https://github.com/armbues/ioc\\_parser](https://github.com/armbues/ioc_parser).
- [13] AT&T Cybersecurity. OTX trends 2018 Q1 and Q2, 2018. URL <https://cybersecurity.att.com/resource-center/white-papers/2018-open-threat-exchange-trends>.
- [14] B. Bartholomew and J. A. Guerrero-Saade. Wave your false flags! deception tactics muddying attribution in targeted attacks. 2016. URL <https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2017/10/20114955/Bartholomew-GuerreroSaade-VB2016.pdf>.
- [15] O. B. Bassat and I. Cohen. Mapping the connections inside russia’s apt ecosystem, 2019. URL <https://www.intezer.com/blog-russian-apt-ecosystem/>.
- [16] B. Bencsáth, G. Pék, L. Buttyán, and M. Félegyházi. The cousins of stuxnet: Duqu, flame, and gauss. *Future Internet*, 4(4):971–1003, 2012. ISSN 1999-5903. doi: 10.3390/fi4040971. URL <http://www.mdpi.com/1999-5903/4/4/971>.
- [17] E. L. Bird, Steven and E. Klein. Natural language processing with python, 2009.
- [18] Bishop Fox. cyber.dic, 2019. URL <https://github.com/BishopFox/cyberdic>.
- [19] X. Bouwman, H. Griffioen, J. Egbers, C. Doerr, B. Klievink, and M. van Eeten. A different cup of TI? the added value of commercial threat intelligence. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 433–450. USENIX Association, Aug 2020. ISBN 978-1-939133-17-5. URL <https://www.usenix.org/conference/usenixsecurity20/presentation/bouwman>.
- [20] M. Brennan, S. Afroz, and R. Greenstadt. Adversarial stylometry: Circumventing authorship recognition to preserve privacy and anonymity. *ACM Trans. Inf. Syst. Secur.*, 15(3), nov 2012. ISSN 1094-9224. doi: 10.1145/2382448.2382450. URL <https://doi.org/10.1145/2382448.2382450>.
- [21] S. Burrows, A. L. Uitdenbogerd, and A. Turpin. Comparing techniques for authorship attribution of source code. *Softw., Pract. Exper.*, 44(1):1–32, 2014. doi: 10.1002/spe.2146. URL <https://doi.org/10.1002/spe.2146>.
- [22] A. Caliskan, F. Yamaguchi, E. Dauber, R. E. Harang, K. Rieck, R. Greenstadt, and A. Narayanan. When coding style survives compilation: De-anonymizing programmers from executable binaries. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, 2018. URL [https://www2.seas.gwu.edu/~aylin/papers/caliskan\\_when.pdf](https://www2.seas.gwu.edu/~aylin/papers/caliskan_when.pdf).
- [23] A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt. De-anonymizing programmers via code stylometry. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 255–270, Washington, D.C., 2015. USENIX Association. ISBN 978-1-931971-232. URL <https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/caliskan-islam>.
- [24] A. Calleja, J. Tapiador, and J. Caballero. A look into 30 years of malware development from a software metrics perspective. In F. Monrose, M. Dacier, G. Blanc, and J. Garcia-Alfaro, editors, *Research in Attacks, Intrusions, and Defenses*, pages 325–345, Cham, 2016. Springer International Publishing. ISBN 978-3-319-45719-2.
- [25] A. Calleja, J. Tapiador, and J. Caballero. The malsource dataset: Quantifying complexity and code reuse in malware development. *IEEE Transactions on Information Forensics and Security*, 14(12):3175–3190, Dec 2019. ISSN 1556-6021. doi: 10.1109/TIFS.2018.2885512.
- [26] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, May 2017. doi: 10.1109/SP.2017.49.
- [27] E. Carrera. pefile, 2020. URL <https://github.com/erocarrera/pefile>.
- [28] C. C. N. (CCN-CERT). Ciberamenazas Y Tendencias, 2020. URL <https://www.ccn-cert.cni.es/informes/informes-ccn-cert-publicos/5377-ccn-cert-ia-13-20-ciberamenazas-y-tendencias-edicion-2020/file.html>.
- [29] Chronicle. VirusTotal, 2004. URL [www.virustotal.com](http://www.virustotal.com).
- [30] Clang. Compiler, 2020. URL <https://clang.llvm.org/index.html>.
- [31] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC ’71*, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. ISBN 9781450374644. doi: 10.1145/800157.805047. URL <https://doi.org/10.1145/800157.805047>.

- [32] Council on Foreign Relations. Cyber operations tracker, 2020. URL <https://www.cfr.org/interactive/cyber-operations>.
- [33] cyber-research. APTMalware, 2019. URL <https://github.com/cyber-research/APTMalware>.
- [34] E. Dauber, A. Caliskan, R. E. Harang, G. Shearer, M. Weisman, F. Nelson, and R. Greenstadt. Git blame who?: Stylistic authorship attribution of small, incomplete source code fragments. *PoPETs*, 2019(3):389–408, 2019. doi: 10.2478/popets-2019-0053. URL <https://doi.org/10.2478/popets-2019-0053>.
- [35] T. N. development team. Netwide assembler, 2015. URL <https://www.nasm.us/>.
- [36] M. V. Emmerik and T. Waddington. Using a decompiler for real-world source recovery. In *11th Working Conference on Reverse Engineering*, pages 27–36, Nov 2004. doi: 10.1109/WCRE.2004.42.
- [37] M. R. Farhadi, B. C. Fung, Y. B. Fung, P. Charland, S. Preda, and M. Debbabi. Scalable code clone search for malware analysis. *Digital Investigation*, 15:46 – 60, 2015. ISSN 1742-2876. doi: <https://doi.org/10.1016/j.diin.2015.06.001>. URL <http://www.sciencedirect.com/science/article/pii/S1742287615000705>. Special Issue: Big Data and Intelligent Data Analysis.
- [38] FireEye. FLOSS, 2017. URL <https://github.com/fireeye/flare-floss>.
- [39] T. Foltýnek, N. Meuschke, and B. Gipp. Academic plagiarism detection: A systematic literature review. *ACM Comput. Surv.*, 52(6):112:1–112:42, 2020. doi: 10.1145/3345317. URL <https://doi.org/10.1145/3345317>.
- [40] G. Frantzeskou, E. Stamatatos, S. Gritzalis, C. E. Chaski, and B. S. Howald. Identifying authorship by byte-level n-grams: The source code author profile (SCAP) method. *IJDE*, 6(1), 2007. URL <http://www.utica.edu/academic/institutes/ecii/publications/articles/B41158D1-C829-0387-009D214D2170C321.pdf>.
- [41] N. Gamer. The problem with open source malware, 2016. URL <https://blog.trendmicro.com/the-problem-with-open-source-malware/>.
- [42] GitHub. Github repositories, 2020. URL <https://github.com>.
- [43] GNU. Compiler, 2020. URL <https://gcc.gnu.org/>.
- [44] H. Gonzalez, N. Stakhonova, and A. A. Ghorbani. Authorship attribution of android apps. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY ’18, pages 277–286. ACM, 2018. ISBN 978-1-4503-5632-9. doi: 10.1145/3176258.3176322. URL <http://doi.acm.org/10.1145/3176258.3176322>.
- [45] Google. Google code jam, 2008-2020. URL <https://codingcompetitions.withgoogle.com/codejam/>.
- [46] C. Guarnieri. Cuckoo sandbox, 2019. URL <https://cuckoosandbox.org/>.
- [47] H. Haddadpajouh, A. Azmoodeh, A. Dehghantanha, and R. M. Parizi. Mvfcc: A multi-view fuzzy consensus clustering model for malware threat attribution. *IEEE Access*, 8:139188–139198, 2020.
- [48] I. U. Haq and J. Caballero. A survey of binary code similarity. *CoRR*, abs/1909.11424, 2019. URL <http://arxiv.org/abs/1909.11424>.
- [49] A. Henderson, L. Yan, X. Hu, A. Prakash, H. Yin, and S. McCamant. Decaf: A platform-neutral whole-system dynamic binary analysis platform. *IEEE Transactions on Software Engineering*, 43(2):164–184, 2 2017. ISSN 0098-5589. doi: 10.1109/TSE.2016.2589242.
- [50] S. Hendrikse. *The Effect of Code Obfuscation on Authorship Attribution of Binary Computer Files*. PhD thesis, 2017. URL [https://nsuworks.nova.edu/gscis\\_etd/1009](https://nsuworks.nova.edu/gscis_etd/1009).
- [51] B. Herzog. The gandcrab ransomware mindset, 2018. URL <https://research.checkpoint.com/2018/gandcrab-ransomware-mindset/>.
- [52] Hex-Rays. Ida, 2020. URL <https://www.hex-rays.com/products/ida/>.
- [53] F. Hightower. Observable finder, 2017. URL <https://github.com/fhightower/ioc-finder>.
- [54] J. Hong, S. Park, S.-W. Kim, D. Kim, and W. Kim. Classifying malwares for identification of author groups. *Concurrency and Computation: Practice and Experience*, 30(3):e4197, 2018. doi: 10.1002/cpe.4197. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4197>. e4197 cpe.4197.
- [55] M. Honnibal and I. Montani. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 2017.

- [56] M. Hurier, G. Suarez-Tangil, S. K. Dash, T. F. Bissyandé, Y. L. Traon, J. Klein, and L. Cavallaro. Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 425–435, 2017. doi: 10.1109/MSR.2017.57.
- [57] Intel. C++ compiler, 2020. URL <https://software.intel.com/en-us/c-compilers>.
- [58] V. Kalgutkar, N. Stakhanova, P. Cook, and A. Matyukhina. Android authorship attribution through string analysis. In S. Doerr, M. Fischer, S. Schrittwieser, and D. Herrmann, editors, *Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, Hamburg, Germany, August 27-30, 2018*, pages 4:1–4:10. ACM, 2018. doi: 10.1145/3230833.3230849. URL <https://doi.org/10.1145/3230833.3230849>.
- [59] V. Kalgutkar, R. Kaur, H. Gonzalez, N. Stakhanova, and A. Matyukhina. Code authorship attribution: Methods and challenges. *ACM Comput. Surv.*, 52(1):3:1–3:36, Feb. 2019. ISSN 0360-0300. doi: 10.1145/3292577. URL <http://doi.acm.org/10.1145/3292577>.
- [60] Kaspersky. The power of threat attribution, 2020. URL <https://media.kaspersky.com/en/business-security/enterprise/threat-attribution-engine-whitepaper.pdf>.
- [61] J. Kinder. Jakstab, 2013. URL <https://github.com/jkinder/jakstab>.
- [62] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 533–537, 2018. doi: 10.23919/EUSIPCO.2018.8553214.
- [63] I. Krsul and E. H. Spafford. Authorship analysis: identifying the author of a program. *Comput. Secur.*, 16(3):233–257, 1997. doi: 10.1016/S0167-4048(97)00005-9. URL [https://doi.org/10.1016/S0167-4048\(97\)00005-9](https://doi.org/10.1016/S0167-4048(97)00005-9).
- [64] G. Laurenza and R. Lazzaretti. daptaset: A comprehensive mapping of apt-related data. In A. P. Fournaris, M. Athanatos, K. Lampropoulos, S. Ioannidis, G. Hatzivasilis, E. Damiani, H. Abie, S. Ranise, L. Verderame, A. Siena, and J. Garcia-Alfaro, editors, *Computer Security*, pages 217–225, Cham, 2020. Springer International Publishing. ISBN 978-3-030-42051-2.
- [65] G. Laurenza, L. Aniello, R. Lazzaretti, and R. Baldoni. Malware triage based on static features and public apt reports. In S. Dolev and S. Lodha, editors, *Cyber Security Cryptography and Machine Learning*, pages 288–305, Cham, 2017. Springer International Publishing. ISBN 978-3-319-60080-2.
- [66] G. Laurenza, R. Lazzaretti, and L. Mazzotti. Malware triage for early identification of advanced persistent threat activities. *CoRR*, abs/1810.07321, 2018. URL <http://arxiv.org/abs/1810.07321>.
- [67] R. Layton, P. A. Watters, and R. Dazeley. Automatically determining phishing campaigns using the USCAP methodology. In *2010 eCrime Researchers Summit, eCrime 2010, Dallas, TX, USA, October 18-20, 2010*, pages 1–8. IEEE, 2010. doi: 10.1109/ecrime.2010.5706698. URL <https://doi.org/10.1109/ecrime.2010.5706698>.
- [68] V. Legoy, M. Caselli, C. Seifert, and A. Peter. Automated retrieval of att&ck tactics and techniques for cyber threat reports, 2020.
- [69] A. Lemay, J. Calvet, F. Menet, and J. M. Fernandez. Survey of publicly available reports on advanced persistent threat actors. *Computers and Security*, 72:26 – 59, 2018. ISSN 0167-4048. doi: <https://doi.org/10.1016/j.cose.2017.08.005>. URL <http://www.sciencedirect.com/science/article/pii/S0167404817301608>.
- [70] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah. Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, page 755–766, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341394. doi: 10.1145/2976749.2978315. URL <https://doi.org/10.1145/2976749.2978315>.
- [71] LLVM. Compiler, 2020. URL <http://llvm.org/>.
- [72] Lockheed-Martin. Gaining the advantage applying cyber kill chain® methodology to network defense, 2015. URL [https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/Gaining\\_the\\_Advantage\\_Cyber\\_Kill\\_Chain.pdf](https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/Gaining_the_Advantage_Cyber_Kill_Chain.pdf).
- [73] M. Lottmann and F. Yamaguchi. bjoern, 2016. URL <https://github.com/octopus-platform/bjoern>.
- [74] M. Marquis-Boire, M. Marschalek, and C. Guarnieri. Big game hunting: The peculiarities in nation-state malware research. 2015. URL <https://www.blackhat.com/docs/us-15/materials/us-15-MarquisBoire-Big-Game-Hunting-The-Peculiarities-Of-Nation-State-Malware-Research.pdf>.

- [75] Masrepus, vfsrfs, and garanews. Un{i}packer, 2019. URL <https://github.com/unipacker/unipacker>.
- [76] A. Matyukhina, N. Stakhanova, M. Dalla Preda, and C. Perley. Adversarial authorship attribution in open-source projects. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY '19*, pages 291–302, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6099-9. doi: 10.1145/3292006.3300032. URL <http://doi.acm.org/10.1145/3292006.3300032>.
- [77] X. Meng. Fine-grained binary code authorship identification. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016*, pages 1097–1099. ACM, 2016. ISBN 978-1-4503-4218-6. doi: 10.1145/2950290.2983962. URL <http://doi.acm.org/10.1145/2950290.2983962>.
- [78] X. Meng and B. P. Miller. Binary code multi-author identification in multi-toolchain scenarios. *Under Submission*, 2018. URL <http://ftp.cs.wisc.edu/paradyn/papers/Meng17MultiToolchain.pdf>.
- [79] X. Meng, B. P. Miller, W. R. Williams, and A. R. Bernat. Mining software repositories for accurate authorship. In *2013 IEEE International Conference on Software Maintenance (ICSM)*, pages 250–259, Los Alamitos, CA, USA, 2013. IEEE Computer Society. doi: 10.1109/ICSM.2013.36. URL <https://doi.ieeecomputersociety.org/10.1109/ICSM.2013.36>.
- [80] X. Meng, B. P. Miller, and K.-S. Jun. Identifying multiple authors in a binary program. In S. N. Foley, D. Gollmann, and E. Sneekenes, editors, *Computer Security – ESORICS 2017*, pages 286–304, Cham, 2017. Springer International Publishing. ISBN 978-3-319-66399-9.
- [81] X. Meng, B. P. Miller, and S. Jha. Adversarial binaries for authorship identification. *CoRR*, abs/1809.08316, 2018. URL <http://arxiv.org/abs/1809.08316>.
- [82] Microsoft. Visual studio, 2019. URL <https://visualstudio.microsoft.com/>.
- [83] MISP: Open Source Threat Intelligence Platform. List of threat actors, 2020. URL <https://raw.githubusercontent.com/MISP/misp-galaxy/main/clusters/threat-actor.json>.
- [84] Mitre. ATT&CK, 2020. URL <https://attack.mitre.org/>.
- [85] P. Moore and H. V. Pham. On context and the open world assumption. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pages 387–392, 2015. doi: 10.1109/WAINA.2015.7.
- [86] National Institute of Standards and Technology. Managing information security risk, 2011. URL <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-39.pdf>.
- [87] T. J. Neal, K. Sundararajan, A. Fatima, Y. Yan, Y. Xiang, and D. L. Woodard. Surveying stylometry techniques and applications. *ACM Comput. Surv.*, 50(6):86:1–86:36, 2018. doi: 10.1145/3132039. URL <https://doi.org/10.1145/3132039>.
- [88] OASIS Cyber Threat Intelligence. STIX 2.0, 2020. URL <https://oasis-open.github.io/cti-documentation/stix/intro>.
- [89] OASIS Cyber Threat Intelligence. TAXII 2.0, 2020. URL <https://oasis-open.github.io/cti-documentation/taxii/intro.html>.
- [90] Office of the Director of National Intelligence. A guide to cyber attribution, 2018. URL [https://www.dni.gov/files/CTIIC/documents/ODNI\\_A\\_Guide\\_to\\_Cyber\\_Attribution.pdf](https://www.dni.gov/files/CTIIC/documents/ODNI_A_Guide_to_Cyber_Attribution.pdf).
- [91] Open Source Software. UPX (Ultimate Packer for Executables), 2020. URL <https://upx.github.io/>.
- [92] Paradyn-Project. Dyninst: Putting the performance in high performance computing, 2019. URL <http://www.dyninst.org>.
- [93] Planet-Source-Code. Planet source code repositories, 2020. URL <https://www.planet-source-code.com>.
- [94] E. Quiring, A. Maier, and K. Rieck. Misleading authorship attribution of source code using adversarial learning. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 479–496, Santa Clara, CA, Aug 2019. USENIX Association. ISBN 978-1-939133-06-9. URL <https://www.usenix.org/conference/usenixsecurity19/presentation/quiring>.
- [95] radare.org. radare2, 2020. URL <https://www.radare.org/n/radare2.html>.
- [96] E. Raff, R. Zak, G. L. Munoz, W. Fleming, H. S. Anderson, B. Filar, C. Nicholas, and J. Holt. Automatic Yara Rule Generation Using Biclustering. In *13th ACM Workshop on Artificial Intelligence and Security (AISec'20)*, 2020. doi: 10.1145/3411508.3421372. URL <http://arxiv.org/abs/2009.03779>.

- [97] A. Rahimian, P. Shirani, S. Alrbaee, L. Wang, and M. Debbabi. Bincomp: A stratified approach to compiler provenance attribution. *Digital Investigation*, 14:S146 – S155, 2015. ISSN 1742-2876. doi: <https://doi.org/10.1016/j.diin.2015.05.015>. URL <http://www.sciencedirect.com/science/article/pii/S1742287615000602>. The Proceedings of the Fifteenth Annual DFRWS Conference.
- [98] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. <http://is.muni.cz/publication/884893/en>.
- [99] R. Reynolds. The four biggest malware threats to uk businesses. *Network Security*, 2020(3):6 – 8, 2020. ISSN 1353-4858. doi: [https://doi.org/10.1016/S1353-4858\(20\)30029-5](https://doi.org/10.1016/S1353-4858(20)30029-5). URL <http://www.sciencedirect.com/science/article/pii/S1353485820300295>.
- [100] T. Rid and B. Buchanan. Attributing cyber attacks. *Journal of Strategic Studies*, 38(1-2):4–37, 2015. doi: 10.1080/01402390.2014.977382. URL <https://doi.org/10.1080/01402390.2014.977382>.
- [101] E. Robbins. *Solvers for Type Recovery and Decompilation of Binaries*. PhD thesis, University of Kent., January 2017. URL <https://kar.kent.ac.uk/61349/>.
- [102] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi. Microsoft malware classification challenge. *CoRR*, abs/1802.10135, 2018. URL <http://arxiv.org/abs/1802.10135>.
- [103] I. Rosenberg, G. Sicard, and E. O. David. Deepapt: Nation-state apt attribution using end-to-end deep neural networks. In A. Lintas, S. Rovetta, P. F. Verschure, and A. E. Villa, editors, *Artificial Neural Networks and Machine Learning – ICANN 2017*, pages 91–99, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68612-7.
- [104] I. Rosenberg, G. Sicard, and E. O. David. End-to-end deep neural networks and transfer learning for automatic analysis of nation-state malware. volume 20, 2018. doi: 10.3390/e20050390. URL <http://www.mdpi.com/1099-4300/20/5/390>.
- [105] J. Rosenberg and C. Beek. Examining code reuse reveals undiscovered links among north korea’s malware families, 2018. URL <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/examining-code-reuse-reveals-undiscovered-links-among-north-koreas-malware-families/>.
- [106] N. Rosenblum, X. Zhu, and B. P. Miller. Who wrote this code? identifying the authors of program binaries. In V. Atluri and C. Diaz, editors, *Computer Security – ESORICS 2011*, pages 172–189, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-23822-2.
- [107] N. E. Rosenblum, B. P. Miller, and X. Zhu. Extracting compiler provenance from program binaries. In *Proceedings of the 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, PASTE ’10*, pages 21–28. ACM, 2010. ISBN 978-1-4503-0082-7. doi: 10.1145/1806672.1806678. URL <http://doi.acm.org/10.1145/1806672.1806678>.
- [108] sapphires00. APTs and OPs table guide, 2018. URL [https://github.com/sapphires00/Threat-Hunting/raw/master/apts\\_and\\_ops\\_tableguide.xlsx](https://github.com/sapphires00/Threat-Hunting/raw/master/apts_and_ops_tableguide.xlsx).
- [109] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero. Avclass: A tool for massive malware labeling. In *Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings*, pages 230–253, 2016. doi: 10.1007/978-3-319-45719-2\_11. URL [https://doi.org/10.1007/978-3-319-45719-2\\_11](https://doi.org/10.1007/978-3-319-45719-2_11).
- [110] P. Shirani, L. Wang, and M. Debbabi. Binshape: Scalable and robust binary library function identification using function shape. In M. Polychronakis and M. Meier, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 301–324, Cham, 2017. Springer International Publishing. ISBN 978-3-319-60876-1.
- [111] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna. Sok: (state of) the art of war: Offensive techniques in binary analysis. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 138–157, 2016. doi: 10.1109/SP.2016.17.
- [112] L. Simko, L. Zettlemoyer, and T. Kohno. Recognizing and imitating programmer style: Adversaries in program authorship attribution. *Proceedings on Privacy Enhancing Technologies*, 2018(1):127 – 144, 2018. URL <https://content.sciendo.com/view/journals/popets/2018/1/article-p127.xml>.
- [113] P. Stirparo, D. Bizeul, B. Bell, Z. Chang, J. Esler, K. Bleich, M. Moreno, M. K. A. J. Capmany, P. Hutchinson, B. Ivanov, A. Girona, D. Ackerman, C. Fragoso, E. Sela, and F. Egloff. Apt groups and operations, 2015. URL <https://apt.threattracking.com>.

- [114] Symantec. Internet security threat report 2019, 2019. URL <https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf>.
- [115] Thailand Computer Emergency Response Team. Threat group cards: A threat actor encyclopedia, 2020. URL <https://apt.thaicert.or.th/>.
- [116] TIOBE - The Software Quality Company. TIOBE Index, 2018. URL <https://www.tiobe.com>.
- [117] G. van Rossum, B. Warsaw, and N. Coghlan. Pep 8 style guide for python code, 2001. URL <https://www.python.org/dev/peps/pep-0008/>.
- [118] N. Virvilis and D. Gritzalis. The big four - what we did wrong in advanced persistent threat detection? In *2013 International Conference on Availability, Reliability and Security (ARES)*, volume 00, pages 248–254, 2013. doi: 10.1109/ARES.2013.32. URL [doi.ieeecomputersociety.org/10.1109/ARES.2013.32](http://doi.ieeecomputersociety.org/10.1109/ARES.2013.32).
- [119] H. Xue, S. Sun, G. Venkataramani, and T. Lan. Machine learning-based analysis of program binaries: A comprehensive study. *IEEE Access*, 7:65889–65912, 2019.
- [120] J. Zhao, Q. Yan, X. Liu, B. Li, and G. Zuo. Cyber threat intelligence modeling based on heterogeneous graph convolutional network. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 241–256, San Sebastian, Oct. 2020. USENIX Association. ISBN 978-1-939133-18-2. URL <https://www.usenix.org/conference/raid2020/presentation/zhao>.