The Rise of GoodFATR: A Novel Accuracy Comparison Methodology for Indicator Extraction Tools

Juan Caballero[a], Gibran Gomez[a,b], Srdjan Matic[a], Gustavo Sánchez[c], Silvia Sebastián[a,b], Arturo Villacañas[a,b]

[a]*IMDEA Software Institute, Campus de Montegancedo, s/n, 28223, Madrid, Spain*
[b]*Universidad Politecnica de Madrid, Campus de Montegancedo, s/n, 28223, Madrid, Spain*
[c]*Karlsruhe Institute of Technology, Am Fasanengarten 5, 76131, Karlsruhe, Germany*

Abstract

To adapt to a constantly evolving landscape of cyber threats, organizations actively need to collect Indicators of Compromise (IOCs), i.e., forensic artifacts that signal that a host or network might have been compromised. IOCs can be collected through open-source and commercial structured IOC feeds. But, they can also be extracted from a myriad of unstructured threat reports written in natural language and distributed using a wide array of sources such as blogs and social media. There exist multiple indicator extraction tools that can identify IOCs in natural language reports. But, it is hard to compare their accuracy due to the difficulty of building large ground truth datasets. This work presents a novel majority vote methodology for comparing the accuracy of indicator extraction tools, which does not require a manually-built ground truth. We implement our methodology into GoodFATR, an automated platform for collecting threat reports from a wealth of sources, extracting IOCs from the collected reports using multiple tools, and comparing their accuracy.

GoodFATR supports 6 threat report sources: RSS, Twitter, Telegram, Malpedia, APTnotes, and ChainSmith. GoodFATR continuously monitors the sources, downloads new threat reports, extracts 41 indicator types from the collected reports, and filters non-malicious indicators to output the IOCs. We run GoodFATR over 15 months to collect 472,891 reports from the 6 sources; extract 978,151 indicators from the reports; and identify 618,217 IOCs. We analyze the collected data to identify the top IOC contributors and the IOC class distribution. We apply GoodFATR to compare the IOC extraction accuracy of 7 popular open-source tools with GoodFATR's own indicator extraction module.

Keywords: Indicators of Compromise, IOC, Cyber Threat Intelligence, RSS, Twitter, Telegram

1. Introduction

Cyber Threat Intelligence (CTI) provides information on attacker behavior that allows gaining visibility into the fast-evolving threat landscape; to understand the techniques, tactics, and procedures (TTPs) attackers use; and to timely identify and contain attacks. CTI is a multi-billion dollar industry, expected to keep growing to more than 16 billion USD by 2026 [1]. An essential piece of CTI is extracting and sharing *indicators of compromise* (IOCs), forensic artifacts that when observed on a device or network indicate it may have been compromised, e.g., malicious IPs, domains, and file hashes. IOCs are an actionable piece of CTI, as they can be fed to security systems (e.g., NIDS, firewall, HIDS, blocklists) to detect and block attacks.

IOCs can be distributed through open and commercial feeds [2, 3], which provide structured IOC data following standardized formats (e.g., STIX [4], OpenIOC [5]). However, much CTI is distributed through unstructured threat reports, written in natural language and published through a wealth of security blogs and social media platforms. For example, Twitter has become widely used for exchanging and spreading cybersecurity information, not only by cybersecurity companies but also by experts that sometimes rush to share their discoveries [6, 7]. Natural language reports often contain IOCs and typically provide more detailed contextual descriptions about the IOCs compared to IOC feeds. In addition, while IOC feeds usually focus on a small set of indicator types (i.e., IP addresses, domain names, file hashes), unstructured threat reports frequently provide more varied indicator types such as vulnerability identifiers, email addresses, blockchain addresses, Tor onion addresses, fuzzy hashes (e.g., SSDeep [8]), social handles, and target countries.

There exist multiple indicator extraction tools that can identify IOCs from unstructured threat reports in natural language using regular expressions [9, 10, 11, 12, 13, 14]. Even recent works that apply natural language processing (NLP) techniques to analyze threat reports still use regular expressions for identifying indicators as part of their pipeline [15, 16, 17, 18]. Regular expressions for extracting indicators can easily become complex, making it difficult to understand what they match. Also, small differences between regular expressions for the same indicator type may significantly affect the extraction results. Furthermore, regular expressions can be affected

Email addresses: `juan.caballero@imdea.org` (Juan Caballero), `gibran.gomez@imdea.org` (Gibran Gomez), `srdjan.matic@imdea.org` (Srdjan Matic), `sanchez@kit.edu` (Gustavo Sánchez), `silvia.sebastian@imdea.org` (Silvia Sebastián), `arturo.villacanas@imdea.org` (Arturo Villacañas)

by catastrophic backtracking introducing ReDoS vulnerabilities [19]. Despite the popularity of regular expression based indicator extraction tools [9, 10, 11, 12, 13, 14] no previous work has systematically evaluated them to understand which tool is more accurate for each indicator type and how accurate their extraction is. The main challenge to evaluate indicator extraction accuracy is the difficulty of building large-scale ground truth datasets from real threat reports. To address this challenge, this work presents a novel majority vote methodology for comparing the accuracy of indicator extraction tools, which does not require a manually-built ground truth.

Another challenge is that in a large and fast-evolving landscape of threats, coverage is difficult to achieve by any single entity, as shown by different sources having little IOC overlap [2, 3]. Thus, it is not sufficient to rely on a single source, or a small set of sources, to build an accurate, comprehensive, and up-to-date IOC list. To overcome this limitation, organizations and security analysts can resort to extracting IOCs from the threat reports collected from multiple sources. However, there exists a myriad of sources through which threat reports are disseminated including hundreds of blogs, Twitter accounts, and Telegram channels. To address this challenge, we present a modular threat report collection pipeline that can collect reports from a wealth of *sources* in a unified manner.

Our collection pipeline currently supports 6 diverse sources: RSS, Twitter, Telegram, and three report datasets Malpedia [20], APTnotes [21], and ChainSmith [17]. For RSS, Twitter, and Telegram it takes as input a list of *origins* (RSS feeds, Twitter accounts, Telegram channels) to monitor. It periodically visits those origins to identify new *entries* (blog posts, tweets, messages). For entries that contain a URL to a report, it downloads the report's *document*. The selected sources are complementary. RSS allows collecting reports from hundreds of blogs from cybersecurity companies and individual experts, while Twitter and Telegram cover social media distribution. In addition, crowd-sourced datasets such as Malpedia and APTnotes allow identifying previously unknown blogs and Twitter accounts that should be monitored, as well as collecting reports from blogs without an RSS feed.

We have implemented our novel methodology for comparing indicator extraction tools and our collection pipeline into an automated platform called GoodFATR. GoodFATR supports a variety of open-source indicator extraction tools but also provides its own IOCSEARCHER tool, which takes as input a document in HTML, PDF, or plain text format and applies regular expressions to extract 41 indicator types. When processing reports, indicator extraction tools may identify malicious indicators (e.g., C&C domains), as well as benign ones (e.g., URL references or contact emails for security companies). We call malicious indicators IOCs and benign indicators *generic*. GoodFATR provides a filtering module that removes generic indicators extracted from the documents to output only the IOCs. Its filtering module builds a dynamic blocklist from the monitored sources and origins. This approach avoids hard-coded blocklists used by prior tools, which cannot adapt to the different sources and origins each user may monitor.

We use GoodFATR to compare the accuracy of 7 popular open-source indicator extraction tools [9, 10, 11, 12, 22, 13, 14], as well as GoodFATR's own IOCSEARCHER tool, on 4,420 reports. The results show IOCSEARCHER being the most accurate tool on 11 of the 13 indicator types supported by multiple tools. We also identify ReDos vulnerabilities in two of the tools, one new and one previously reported. We also evaluate the filtering module provided by GoodFATR against the filtering rules used in two prior tools. The filtering by GoodFATR achieves an F1 score of 0.91 compared to 0.47 (IOC_PARSER) and 0.46 (CACADOR).

We have used GoodFATR to collect 472,891 reports over 15 months from 3,226 origins distributed through 6 sources; extract 978,151 indicators from the reports; and identify 618,217 IOCs. We analyze the collected data to identify the top IOC contributors and the IOC class distribution.

This work provides the following contributions:

- We present a novel majority-vote methodology for evaluating the accuracy of indicator extraction tools, which does not require a manually-built ground truth. We apply our methodology to compare the accuracy of 7 popular tools and our own IOCSEARCHER over 4,420 reports. The results show IOCSEARCHER is the most accurate tool on 11 of the 13 indicator types supported by multiple tools.
- We present GoodFATR, an automated platform to collect threat reports from a variety of sources in a unified manner, extract the indicators in the collected reports, and filter generic indicators to output only the IOCs.
- We use GoodFATR to collect 472,891 reports over 15 months from the 6 sources; extract 978,151 indicators from the reports; and identify 618,217 IOCs.
- We release the source code of the IOCSEARCHER tool at `https://github.com/malicialab/iocsearcher`

The remainder of this paper is organized as follows. Section 2 presents prior related work. Section 3 provides an overview of our platform. Section 4 details the sources and approach used in the threat report collection. Section 5 performs a systematization and comparative study of 8 indicator extraction tools. Section 6 analyzes the reports collected over 15 months, the IOCs extracted from those reports, and the top contributing sources and origins. Section 7 presents our novel accuracy comparison methodology and applies it to compare the 8 indicator extraction tools on 4,420 threat reports. Section 8 discusses limitations and future improvements. Finally, Section 9 concludes.

2. Related Work

Extracting IOCs from natural language documents is an important CTI task that comprises two problems: threat report collection and IOC extraction. Prior work can be split into two groups. First, there exist open source tools that focus on extracting IOCs from a given threat report using regular expressions (*e.g.,* [9, 10, 11, 12, 13, 14]). These tools do not address the problem of threat report collection. We summarize the most popular open-source IOC extraction tools in Table 3 and detail how they work in Section 5. Second, prior academic works

Table 1: Summary of prior academic works on threat report collection and IOC extraction. Open-source IOC extraction tools are summarized in Table 3.

		Collection Sources				Extraction	
Work	**Year**	**Blogs - Crawlers**	**Blogs - RSS**	**Telegram**	**Twitter**	**Regexp**	**NLP**
Liao et al. [15]	2016	✓	✗	✗	✗	✓	✓
TTPDrill [16]	2017	✗	✗	✗	✗	✓	✓
Malpedia [20]	2017	✓	✗	✗	✗	✓	✗
ChainSmith [17]	2018	✓	✗	✗	✗	✓	✓
IOCMiner [23]	2019	✗	✗	✗	✓	✓	✗
TIMiner [24]	2020	✓	✗	✗	✗	✓	✓
Alves et al. [7]	2020	✗	✗	✗	✓	✓	✗
Twiti [25]	2021	✗	✗	✗	✓	✓	✓
GoodFATR	2022	✗	✓	✓	✓	✓	✗

typically address both the threat report collection and the IOC extraction problems. We detail prior academic work in the remainder of this section.

Threat report collection. Table 1 shows that prior academic work has collected reports from one source, either from cybersecurity blogs [17, 15, 24, 20] or from Twitter [23, 7, 25]. There is also one work that does not address threat report collection, focusing exclusively on IOC extraction [16]. In contrast, our work proposes a modular threat report collection platform that can collect reports from multiple sources, namely cybersecurity blogs, Telegram, Twitter, and datasets provided by prior works such as ChainSmith [17], Malpedia [20], and APTnotes [21].

Previous work that collected threat reports from cybersecurity blogs built dedicated crawlers for each blog of interest (e.g., [17, 15, 24]). The advantage of dedicated crawlers is that they can be designed to collect the historical archive of reports available on a blog's website. One disadvantage is that a crawler needs to be developed for each blog, which limits the number of monitored blogs. For example, Liao et al. [15] collected reports from 45 blogs, Zhu and Dumitras from 10 [17], and Zhao et al. [24] from 75. Moreover, as websites evolve, the crawlers may break and need to be updated. For example, ChainSmith originally collected reports from 10 blogs in 2018, but the supported blogs gradually decreased over time, possibly due to crawlers breaking due to website updates. In mid-2020 only two blogs were still being crawled, and since mid-2021 ChainSmith does not collect any new blog posts.

Rather than building dedicated crawlers, GoodFATR collects threat reports from cybersecurity blogs that offer an RSS feed. The use of RSS feeds allows us to build a generic collection module for cybersecurity blogs, enabling us to scale the collection to 300 blogs, 6 times larger than prior works. Furthermore, our RSS module still collects entries from the same set of blogs supported by ChainSmith, showing that those origins are still active. It is important to note that our RSS feed collection may also require updates (*e.g.,* if a feed's XML URL changes), but updating a feed's URL is significantly easier than building a new dedicated crawler. Furthermore, if a blog does not offer an RSS feed, it will still need a dedicated crawler.

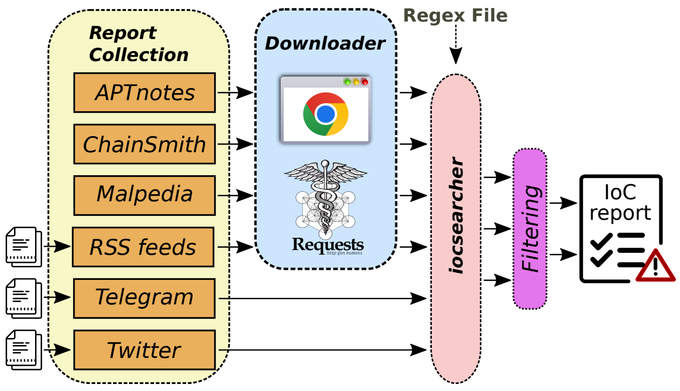

Figure 1: The architecture of GoodFATR.

IOC extraction. All open-source IOC extraction tools in Table 3 and three academic works in Table 1 [20, 23, 7] extract IOCs from the collected reports using regular expressions. Some academic papers have instead proposed natural language processing (NLP) approaches to improve the extraction [15, 17, 16, 24]. However, NLP approaches also rely on regular expressions to extract indicators in their pipelines. Despite the popularity of regular expression-based IOC extraction, the large number of tools a user can choose for this task, and the impact of regular expression differences in the extraction results, we are not aware of any prior work that systematically analyzes which tool works for each indicator type. Such a comparison should include the regular expression used for the extraction, but also other important steps such as how to filter benign indicators that are not IOCs. The main difficulty for such evaluation is building large-scale datasets from real documents. To address this challenge, in this work we propose a novel methodology to compare indicator extraction tools, which does not require a manually-built ground truth.

Also related are metrics to evaluate the quality of IOC feeds [2]. We apply such metrics to compare different sources and origins to identify the best and worst IOC contributors.

Extended version. This paper is an extended version of a previous 4-page work-in-progress paper that appeared in JNIC 2022 [26]. Compared to that work, this paper provides four new contributions. First, we double the number of sources from which we collect reports by including three curated sources specialized in threat reports (Malpedia, APTnotes, ChainSmith). We also expand the collection period to 15 months (Section 4). Second, we provide an in-depth comparative analysis of 7 popular indicator extraction tools, as well as our own IOCSEARCHER tool (Section 5). Third, we redo our analysis of the collected data including the new sources and the extended observation period, and add a new analysis of the distribution of IOCs and the top IOC contributors (Section 6). Fourth, we develop a novel majority-vote methodology to automatically compare the accuracy of indicator extraction tools. We apply this methodology to the 8 indicator extraction tools (Section 7).

3. Overview

An *indicator* represents an artifact such as an IP address, a domain name, a URL, a file, a registry key, or a blockchain address. An indicator is typically represented as a pair of a type (e.g., *email*) and a string value (e.g., `contact@test.com`). An *indicator of compromise* (IOC) is a malicious indicator whose presence on a device or network indicates the device might have been compromised.

This paper presents GoodFATR, a platform for collecting threat reports from a wealth of sources and extracting IOCs from the collected reports. Figure 1 depicts the architecture of GoodFATR. It consists of four modules: *Report Collection*, *Downloader*, IOCSEARCHER, and *Filtering*. The Report Collection gathers documents from a wealth of sources including RSS feeds, Telegram channels, Twitter accounts, and threat report datasets like Malpedia [20], APTnotes [21], and ChainSmith [17]. The collection outputs a list of *entries* each corresponding to the observation of a report in a source. Each entry contains the URL of the report or the content of a tweet or Telegram message. In addition, it may contain optional metadata about the report (e.g., publication date, author, language) and the source where it was observed (e.g., RSS feed, Twitter account). The Downloader leverages an instrumented Chrome browser (and Python's requests library as backup) to fetch the document pointed by an entry's URL and store it to file together with download metadata (e.g., download date, HTTP status code, redirection chain followed). For Twitter and Telegram, the Downloader is not used, as their entries already contain the content of the posts. IOCSEARCHER takes as input a document in HTML, PDF, or plain text format, extracts its text (for HTML and PDF documents), and applies regular expressions to identify 41 indicators. We choose regular expressions because they are an efficient technique for identifying, in a given string, indicators with some intrinsic structure such as email addresses or URLs. To extract indicators, IOCSEARCHER applies a match-and-validate approach. Matching applies each of the regular expressions to identify candidate indicators in the input string. Validation uses a function specific to each indicator type to ensure the candidate is indeed an indicator. Separating matching from validation allows the validation to perform computations that are not possible within regular expressions such as checking a checksum embedded in an indicator value (e.g., blockchain addresses, bank account numbers). It also prevents regular expressions from becoming too complex. While validation minimizes the set of incorrect indicators output, it is possible that indicators present in the text are not IOCs, but rather correspond to benign indicators (e.g., the email of the report's author). Thus, before generating the final list of IOCs, the Filtering removes generic indicators that are not IOCs.

Throughout its pipeline, GoodFATR maintains traceability. From an IOC an analyst can identify the documents that contained it, from a document the entries from where it was collected, and from an entry the sources where it was observed.

4. Threat Report Collection

Our threat report collection module has been designed to support a variety of threat report sources. We collect information from RSS, Twitter, Telegram, and publicly available report datasets such as the Malpedia malware encyclopedia [20], the APTnotes repository [21], and the ChainSmith database [17]. To handle diverse sources in a unified manner, the collection is structured around three concepts: *origin*, *entry*, and *document*.

An origin captures the distribution vector through which a report is disseminated, at a finer granularity than a source. Each source typically has many origins; for RSS, an origin corresponds to the feed from a specific blog, for Twitter to a user account, and for Telegram to a channel. For report datasets (Malpedia, APTnotes, ChainSmith) we use as origin the organization that authored the report (e.g., Norton, TrendMicro). The same threat report might be distributed through multiple origins within the same source (e.g., through different blogs), as well as through different sources (e.g., a blog and a Twitter account). We use the origins to produce fine-grained statistics about the report distribution, e.g., to measure which Twitter accounts, Telegram channels, and RSS feeds provide more threat reports. We format the origin to also include the source in the form *source:origin*. For example, *rss:nakedsecurity* and *twitter:nakedsecurity* correspond to the RSS feed of the Naked Security blog by Sophos, and its Twitter account `@NakedSecurity`, respectively.

An entry captures a specific mention of a threat report through an origin. It corresponds to a post in a Twitter account, Telegram channel, or RSS feed. For report datasets, an entry is a record in the report metadata, namely a line in the report index file of APTnotes, a row in the database of ChainSmith, and a BibTex entry in the Malpedia bibliography. Each entry has an origin and contains either a download URL pointing to the report or in the case of social media, a string with the post's text. An entry may also contain additional report metadata as provided by the source: the report author, its original publication timestamp, and the last update timestamp.

A document is an HTML, PDF, or text file that has been collected by GoodFATR. A document is identified by the file's SHA256 hash. A document is generated every time the Downloader downloads a URL and also from the text content of each Twitter and Telegram entry. A document corresponds to an instance of a threat report. The distinction between *threat report* and *document* is important to understand how GoodFATR handles report distribution and changes to the report over time. Multiple documents collected by GoodFATR may correspond to the same report. There are three main reasons for this. First, a threat report may be distributed in different formats. For example, an APT report originally distributed as a webpage in a blog will be stored by APTnotes as a PDF document. When GoodFATR collects the report from the blog's RSS feed it will obtain an HTML document. When GoodFATR collects the report from APTnotes, it will obtain a PDF document. Both documents are different instances of the same threat report. Second, for reports distributed through URLs, if GoodFATR downloads the URL at different points in time, it might collect a different

Table 2: Sources used for collecting threat reports.

Source	Origin	Input Origins	Cumulative	Download URL	Content	IOCs
aptnotes	organization	✗	✓	✓	✓	✗
chainsmith	organization	✗	✓	✓	✗	✓
malpedia	organization	✗	✓	✓	✗	✗
rss	feed	✓	✗	✓	✗	✗
telegram	channel	✓	✓	✗	✓	✗
twitter	account	✓	✓	✗	✓	✗

HTML document (i.e., different SHA256) each time. This may happen if the webpage contains dynamic content that changes over time. Each of those HTML documents is an instance of the same threat report. Third, a threat report may have multiple versions. For example, the original blog entry for the report may be updated a day later to fix an errata. If GoodFATR downloads the URL of the report on both days, it will collect two different HTML documents, which correspond to different versions of the same threat report.

Identifying which collected documents correspond to the same threat report is not necessary for the operation of GoodFATR. All collected documents go through IOC extraction and the extracted IOCs are aggregated and deduplicated. If two documents are different versions of the same report, GoodFATR will extract the same (or very similar) IOCs from those documents and will remove the duplicated IOCs. GoodFATR provides information that can help identify documents that correspond to the same report, e.g., by querying for all documents downloaded from the same URL or by searching for all documents with the same title (extracted from the PDF and HTML document metadata). But, this problem is not addressed in this paper. We further discuss this issue in Section 8.

4.1. Threat Report Sources

Table 2 summarizes the six report sources currently supported by GoodFATR. Each source has a dedicated collection submodule. The RSS, Twitter, and Telegram submodules take as input a file that specifies the origins (i.e., feeds, accounts, channels) that should be monitored. The report datasets do not require an input origin list as the whole dataset is downloaded. We manually created our initial origin lists for RSS, Twitter, and Telegram by including resources from prominent companies, cybersecurity news websites, and well-known security experts. We continuously add new origins as we identify interesting RSS feeds, Twitter accounts, and Telegram channels. We discuss the update process in Section 8. Table 5 captures the total number of origins in each source at the time of writing: 300 RSS feeds, 383 Twitter accounts, 9 Telegram channels, 2,368 organizations in Malpedia, 155 organizations in APTnotes, and 11 blogs in ChainSmith.

For each source, Table 2 shows how the origin is defined and 5 properties: whether it requires an input list of origins; whether the reports are cumulative (i.e., old entries are always available) or entries expire after some time; whether it provides a URL for each entry from where the report needs to be downloaded; whether it includes the actual report or only the URL to the report; and whether it includes IOCs extracted from the report.

RSS. We use RSS feeds to collect reports from 300 security blogs. Each RSS feed is uniquely identified by a URL pointing to the feed's XML file. The owner of an RSS feed configures it to provide a maximum number of the latest entries, *e.g.,* the 5, 10, or 100 most recent ones. To avoid missing entries, the RSS module needs to visit a feed frequently enough that the feed has not posted more entries than the maximum available since the RSS module last visited it. For example, the *Hexacorn* blog only provides the last 5 posts in its feed, but they only post twice per month. Thus, visiting its feed every two months would be enough to avoid missing entries. For each feed, the RSS module monitors the maximum number of entries it has obtained in a visit (*i.e.,* the configured maximum unless the feed has posted less than the maximum since its creation), as well as the maximum daily rate at which posts are added. While we could configure a different visit frequency for each feed, we have empirically observed that no feed posts more than their configured maximum in a single day. Thus, for simplicity, the RSS module queries each feed on a daily basis. The module is configured to throw a warning if any feed publishes more than its maximum number of available entries in a single day. So far, the warning has not been observed, so no entries have been missed.

Since the RSS module is configured not to miss feed entries, two consecutive visits to the same feed will provide some overlapping entries. The RSS module has an incremental approach where feed entries that were already collected (*i.e.,* older than the last feed visit) are ignored. Only the new entries are passed to the Downloader to collect their content. The incremental approach avoids downloading the same content multiple times.

Telegram & Twitter. The Telegram and Twitter modules leverage the official APIs to query each service. Using these APIs they can fetch not just the most recent messages, but any message that was ever posted on an account or channel. Each module takes as input a list of origins that should be monitored, and connects to each origin on a daily basis to fetch all new posts. Using the API, our modules collect each post from an account, or channel, only once. However, it is possible to obtain posts with the same content from multiple origins, e.g., if different users re-tweet the same original message.

Malpedia. Malpedia offers a manually-curated BibTex file with reports related to malware. BibTex entries for new reports are added on a nearly daily basis [20]. The BibTex file is generated dynamically upon request and includes all reports in the database in a cumulative manner. Each bibliography entry contains the following report fields: author, title, publication date, organization (which we use as origin), URL, language, and the date when the report was added to Malpedia. The reports' documents are not provided, so the URLs are passed to the Downloader to collect them.

APTnotes. APTnotes is a GitHub repository with manually-curated APT reports [21]. It provides an index file with report metadata containing the report's title, source (i.e., author organization which we use as origin), publication date, the filename and file hash of the report document, and the URL where the document can be obtained. Reports in webpages are converted from HTML to PDF. All reports are stored in the `box.com` service as PDF files.

ChainSmith. The ChainSmith project [17] makes available an SQLite3 database with the reports it has downloaded, their metadata, and the IOCs it has extracted from the reports. The database has been updated on a weekly basis since 2018. It contains one table where each row corresponds to an IOC extracted from a report including the report URL, the source (i.e., the blog identifier we use as the origin), the report title, and the publication date. The reports' documents are not provided, so we extract the document URLs and pass them to Downloader.

4.2. Downloader

The Downloader takes as input an entry and tries to download the content pointed to by the entry's URL. It uses Selenium, a popular framework used for testing Web applications [27]. We instrument Selenium to render URLs using a fully-fledged instance of Google Chrome. The Downloader is able to follow redirects, it supports dynamic content executed with JavaScript, and, in addition to HTML pages, it can also download plain text documents as well as other MIME types such as PDFs. In case our instrumented browser did not succeed in retrieving the content, the Downloader makes an additional attempt with Python's *requests* library [28]. For each successfully downloaded URL, the Downloader stores the document to a file and it updates the entry with the download information including the download timestamp, the document hash, the HTTP status code, and the redirection chain followed. In the final step, the downloaded documents are filtered to remove resources with HTTP status code errors and HTML content where the title states the webpage was not found.

5. Indicator Extraction: A Comparative Study

In this section, we perform a survey and functionality comparison of indicator extraction tools including our own. We further quantitatively evaluate the tools in Section 7.

To identify the tools, we search for open-source projects for extracting indicators. For this, we query GitHub for projects related to the *ioc* keyword. Then, we manually examine each matching project to select those that correspond to tools that extract indicators. We keep only popular tools with at least 30 stars, which helps avoid the forks of more popular projects. If an identified tool references any other indicator extraction tools, we also include those in our search. This process identifies 7 popular open-source tools for extracting indicators. We also include our own IOCSEARCHER tool, for a total of 8 tools in Table 3.

The 8 tools follow the same model comprising three steps, two of which are optional. The first optional step is *text extraction*, which given an HTML or PDF document, extracts its text. The second step, present in all tools, is *indicator extraction*, which extracts the indicators present in an input string by applying a number of regular expressions, grammars, and rules. Finally, some tools apply an optional *filtering* step whose goal is to remove indicators that are benign (e.g., domains of security vendors), and thus cannot be considered IOCs. The dominant language for implementing the tools is Python used by 6 tools including the two most popular tools (IOC_PARSER and IOCEXTRACT) and our own IOCSEARCHER tool. CACADOR is written in Go and IOC-EXTRACTOR in JavaScript.

In Table 3, a solid circle (●) indicates full support, and a half-filled circle (◐) partial or optional support, and an empty circle (○) no support. Next, we detail each of the three steps.

5.1. Text Extraction

Since most threat reports are distributed as HTML and PDF documents, text extraction is an important step in the complete indicator extraction process. However, it can be performed as a separate pre-processing step that takes as input an HTML or PDF document and outputs its content as text. Content extraction is an independent process from identifying indicators, and this is likely the reason why 5 of the tools do not support it and assume that the input is a string, e.g., containing the full text of a report. In summary, all 8 tools can take an input string, read text from standard input or from a file, and extract indicators in the string. In addition, three tools including our own, accept directly input HTML and PDF files, including the text extracted from the input document, before extracting indicators. This is convenient for the user since it does not need to write its own text extraction code and allows the tool developer to customize the text extraction.

Text extraction is a common step in many Natural Language Processing (NLP) pipelines such as those used to analyze privacy policies (e.g., [30, 31, 32, 33, 34]). Text extraction can have a significant impact on the extracted indicators. For example, the text extraction process may pre-pend or append text to the indicator value, making the extraction miss the indicator or output an incorrect indicator with extraneous characters. It is also possible for the extracted text to split indicators across multiple lines, causing them to be missed. For example, PDF text extraction libraries oftentimes split long URLs appearing in footnotes across multiple lines in the output text (e.g., URLs that do not fit the length of a column).

The only three tools that support all three input types are JAGER, IOC_PARSER, and IOCSEARCHER. These three tools use the pdfminer.six [35] library for PDF text extraction. JAGER and IOC_PARSER use the BeautifulSoup [36] library for HTML text extraction, while our IOCSEARCHER tool supports both BeautifulSoup and Readability.js [37]. Recently, Hosseini et al. [38] analyzed 7 HTML text extraction approaches used in privacy policy analysis showing wide variability among the produced text. They concluded that the best-performing HTML text extraction libraries for privacy policies were Boilerpipe [39] and Readability.js [37], while the worst-performing one was BeatifulSoup [36]. BoilerPipe is written in Java and Readability.js in JavaScript, but both have Python wrappers available.

Table 3: Comparison of indicator extraction tools.

			Inputs			Indicator Extraction				
Tool	**Language**	**Stars**	**Text**	**HTML**	**PDF**	**Rearm**	**Validate**	**Dedup.**	**IOCs**	**Filtering**
JAGER [9]	Python	69	●	●	●	○	○	●	11	○
IOC_PARSER [10]	Python	378	●	●	●	◐	○	◐	11	●
CACADOR [11]	Go	119	●	○	○	◐	○	●	12	◐
CYOBSTRACT [12]	Python	66	●	○	○	●	◐	○	25	○
IOC-FINDER [22]	Python	96	●	○	○	●	○	◐	25	○
IOCEXTRACT [13]	Python	356	●	○	○	●	○	○	8	○
IOC-EXTRACTOR [14]	JavaScript	35	●	○	○	●	○	●	18	○
IOCSEARCHER [29]	Python	-	●	●	●	●	●	●	41	●

5.2. Indicator Extraction

Given an input string, indicator extraction identifies indicators present in the string. Seven of the tools use regular expressions for the extraction, while IOC-FINDER uses grammars instead. The tools using regular expressions process the text in a loop, each time applying one regular expression at a time to the whole text. Each regular expression is associated with one indicator type that is assigned to each of the regular expression matches. Most tools have one regular expression for each supported indicator type, although some tools (e.g., IOCEXTRACT, IOCSEARCHER) support multiple regular expressions for the same indicator.

The middle part of Table 3 captures four properties of the indicator extraction step: whether defanged indicators are supported and rearmed, whether tools split the extraction process into matching and validation, whether the indicators are deduplicated, and the supported indicator types. We describe each property next.

Defanged and rearmed indicators. It is common for threat reports to *defang* malicious indicators, (i.e., IOCs) in case a user inadvertently clicks on them in a navigation tool like a browser. For example, IP address 9.9.9.9 may appear in a threat report as 9[.]9[.]9[.]9, while URL `http://example.com/badfile` may appear as `hxxp://example(.)com/badfile`. Such indicators are often called *defanged* to indicate that they have been converted into less harmful ones, and are not dangerous anymore. These transformations are similar to the ones applied to email addresses to limit automated collection by spammers, e.g., "contact_at_somewhere[.]com". Of the 8 tools, only JAGER does not identify defanged indicators. The other tools support a subset of the following defang transformations: (I) replacing the dot in IPv4 addresses, domain names, URLs, emails, and filenames, (II) replacing the @ sign in emails, (III) replacing the scheme in URLs (e.g., using hxxp:// instead of http://), and (IV) replacing the backslash in URLs. Two tools are marked as having partial support: IOC_PARSER only supports the dot in URLs and domain names, while CACADOR only supports the dot in IPv4 addresses. The other tools are marked as full support. However, they may not exactly support the same transformations, e.g., only IOC-EXTRACTOR supports the replacement of the backslash in URLs. The space of defang transformations that users can apply is very large and the extraction tools only support the limited set presented above, which is clearly incomplete. Intuitively, the defang transformations most important to support are those most often used by threat report authors (or by users when analyzing other sources like webpages). However, it is hard to know the most popular transformations a priori, so it is typical to add support for new defang transformations as they are observed in the wild.

The tools use two approaches to handle defanged indicators. The most popular approach, used by six tools, is broadening the regular expressions used to identify the indicators to cover common defang operations. For example, the regular expression should support that an IP address optionally contains brackets or parenthesis around the dots. Once the defanged indicators have been matched by the regular expression, they can be *rearmed* (or *refanged*) to output the original indicator values. Four tools rearm the defanged indicators by default, while another two (IOC_PARSER, IOCEXTRACT) allow the user to choose if defanged or rearmed values should be returned. The alternative approach used by IOC-FINDER and IOC-EXTRACTOR is to first apply a rearm transformation to the raw text before applying the regular expressions. This approach does not require broadening the regular expressions to handle different defang transformations. On the other hand, blindly rearming the text before applying the regular expressions could incorrectly modify the text (e.g., rearming some text that is not part of an indicator). It also prevents returning the defanged value as it appears in the text since it has been rewritten.

Match and validate. One goal of indicator extraction is to minimize false positives (FPs), i.e., avoid outputting indicators that are not real indicators, such as two tokens concatenated with a period, that is not a fully qualified domain name. This requires making the regular expressions as narrow as possible, without introducing false negatives. For example, it is typical that a regular expression for domain names will check that the top-level domain (TLD) is one of the IANA-approved TLDs. For this, it is common for tools to include a long list of valid TLDs inside the regular expression. Unfortunately, the IANA list already contains over 1,500 TLDs and may continue to grow over time making the regular expression cumbersome. An alternative approach is to split the indicator extraction process into two steps: regular expression matching and validation. In this model, the regular expression can be a bit wider (i.e., produce more matches) because the validation, which is specific to each indicator type, will discard incorrect matches. This match and validate process was first used by CYOBSTRACT, with basic checks such as validating that ASN numbers are in hard-coded ranges or that the length of a domain name is below 160 characters (which is not entirely correct as the maximum length is 255 octets).

Our IOCSEARCHER tool implements a more complete match-and-validate process where each indicator type has an optional validation function that checks the returned regular expression matches. For example, in IOCSEARCHER the *fqdn* regular expression does not need to include the list of valid TLDs because there is a *fqdn* validation function that ensures the matched value indeed contains a valid TLD. The advantage is that the list of IANA-approved TLDs can be kept in a separate file, which can be updated without modifying the regular expression. More importantly, the match and validate approach allows to perform Turing-complete processing on the matches such as validating an embedded checksum in a bank account number or a Bitcoin address. Such validation is not possible using only a regular expression, and it can significantly reduce FPs. For example, it is common for MD5 hashes to match a Bitcoin regular expression, but it is highly unlikely those spurious matches will pass the checksum validation. It is worth mentioning, that the implementation in our IOCSEARCHER tool performs validation after the indicator has been rearmed so that the validation does not need to handle defang transformations.

Deduplication. It is possible for the same indicator to appear multiple times in the same string at different locations. We say that a tool *deduplicates* if it removes duplicated indicators from its output. For example, if the same URL appears twice in the input string, the output only contains the *url* indicator once. Tools can be classified into those that always deduplicate (●), those that do not deduplicate (○), and those where deduplication is optional (◐). None of the prior tools return the position (i.e., start offset) at which the indicator is matched. Thus, the main value of not deduplicating is to know how many times an indicator appeared in the input string. The most flexible approach is to make deduplication optional, as done by IOC_PARSER and IOC-FINDER.

Our IOCSEARCHER tool offers two different APIs. The raw API does not deduplicate. It returns all indicators identified, with their indicator type, starting offset, raw value, and rearmed value. In contrast, the deduplicated API first invokes the raw API, and then it deduplicates the received values by removing the starting offset and the raw values, i.e., returns only deduplicated rearmed indicators. Providing all matches with their starting offset allows the raw API to be used in additional scenarios. For example, Gao et al. [40] propose IOC protection to handle IOCs that contain dots (e.g., URLs, domains, IPs) in NLP pipelines. Such indicators negatively impact sentence tokenization in NLP libraries that leverage dots to identify the end of sentences. IOC protection first identifies the indicators (e.g., using regular expressions), replaces their value in the text with a keyword that does not contain dots, tokenizes the text into sentences, and finally replaces back the keyword with the original indicator value. IOC protection requires the starting offset and the raw value of the indicator in the input string, precisely what our raw API provides.

Indicators. One key difference between indicator extraction tools is the set of indicator types they support, i.e., the set of indicators for which they have regular expressions, grammars, or extraction rules. Overall, we have identified 63 indicators that the tools extract, as summarized in Table 4. We group indicator types into 16 classes: network (13 indicators), social (12), blockchain (9), cryptographic hashes (6), analytics (3), attack (3), contact (2), file (2), intellectual property (2), organization (2), payment (2), vulnerability (2), fuzzy hash (1), information sharing (1), location (1), registry (1), and yara (1).

The largest class corresponds to network-related artifacts such as domain names (*fqdn*), URLs (*url*), IP addresses (*ip4*, *ip6*), IP subnets in CIDR form (*ip4cidr*, *ip6cidr*), IP address ranges (*ipv4range*, *ipv6range*), AS numbers (*asn*), MAC addresses (*macaddress*), Tor onion addresses used to access hid-

Table 4: Indicators extracted by different tools.

Indicator	**JAGER**	**IOC_PARSER**	**CACADOR**	**CYOBSTRACT**	**IOC-FINDER**	**IOCEXTRACT**	**IOC-EXTRACTOR**	**IOCSEARCHER**	**NumTools**
asn				✓	✓		✓		3
asnOwner				✓					1
attackType				✓					1
attCk					✓				1
authentihash					✓				1
avLabel				✓					1
bitcoin					✓		✓	✓	3
bitcoincash								✓	1
copyright								✓	1
country				✓					1
cve	✓	✓	✓	✓	✓		✓	✓	7
dashcoin								✓	1
dogecoin								✓	1
email	✓	✓	✓	✓	✓	✓	✓	✓	8
ethereum							✓	✓	2
facebookHandle								✓	1
filename	✓	✓	✓	✓					4
filepath		✓		✓	✓				3
fqdn	✓	✓	✓	✓	✓		✓	✓	7
githubHandle								✓	1
googleAdsense					✓		✓	✓	3
googleAnalytics					✓		✓	✓	3
googleTagManager								✓	1
iban								✓	1
icp								✓	1
importHash					✓				1
incident				✓					1
instagramHandle								✓	1
ip4	✓	✓	✓	✓	✓	✓	✓	✓	8
ip4cidr				✓	✓			✓	3
ip4range				✓					1
ip6			✓	✓	✓	✓	✓		5
ip6cidr				✓					1
ip6range				✓					1
isp				✓					1
linkedinHandle								✓	1
litecoin								✓	1
macAddress					✓		✓		2
md5	✓	✓	✓	✓	✓	✓	✓	✓	8
monero					✓		✓	✓	3
onionAddress								✓	1
phoneNumber					✓			✓	2
pinterestHandle								✓	1
regKey		✓		✓	✓				3
sha1	✓	✓	✓	✓	✓	✓	✓	✓	8
sha256	✓	✓	✓	✓	✓	✓	✓	✓	8
sha512	✓		✓			✓	✓		4
ssdeep	✓		✓	✓	✓		✓		5
telegramHandle								✓	1
tezos								✓	1
tlpLabel					✓				1
trademark								✓	1
twitterHandle								✓	1
url	✓	✓	✓	✓	✓	✓	✓	✓	8
userAgent				✓	✓				2
webmoney								✓	1
whatsappHandle								✓	1
xmppHandle					✓				1
yara						✓			1
youTubeChannel								✓	1
youtubeHandle								✓	1
zcash								✓	1

den services (*onionAddress*), Internet Content Provider numbers that uniquely identify the owner of a Chinese website (*icp*), and HTTP User-Agent strings (*userAgent*). The social category comprises of user handles for social networks (Facebook, Instagram, LinkedIn, Pinterest, Twitter), open source repositories (GitHub), Instant Messaging tools (Jabber, Telegram, WhatsApp), as well as YouTube usernames and channel identifiers. The blockchain category comprises addresses for 9 popular blockchains (Bitcoin, BitcoinCash, DashCoin, DogeCoin, Ethereum, LiteCoin, Monero, Tezos, ZCash). Cryptographic hashes include MD5, SHA1, SHA256, and SHA512, as well as their application on specific parts of a Windows executable such as the import table (*importHash*) and the whole executable without Authenticode code-signing fields (*authentihash*). There is also a fuzzy hash (*ssdeep*) used to identify files with similar content [8]. The analytics category includes three Google identifiers: Google Adsense, Google Analytics, and Google Tag Manager. The attack category contains MITRE ATT&CK techniques, tactics, and procedures, a list of attack-related keywords (e.g., DoS, spam), and antivirus labels (*avLabel*). The payment category includes IBAN bank account numbers and WebMoney addresses [41]. The vulnerability category includes CVE identifiers and also a variety of identifiers for other vulnerability sources (e.g., BugTrack, Microsoft Bulletins) grouped into a generic *incident* indicator by CYOBSTRACT. The information-sharing category comprises a single indicator for the traffic light protocol (*tlpLabel*) that controls the dissemination of information [42]. Indicators in the remaining categories are straightforward and capture: contact (*email*, *phoneNumber*), files (*filename*, *filepath*), intellectual property (*copyright*, *trademark*), organization names (*asnOwner*, *isp*), locations (*country*), Windows registry keys (*registryKey*), and Yara rules (*yara*).

There are only 6 indicators that are extracted by all 8 tools: IPv4 addresses (*ip4*), emails, hashes (*md5*, *sha1*, *sha256*), and URLs. Another two indicators are extracted by 7 tools: CVE vulnerability identifiers (*cve*) and domain names (*fqdn*). Next come two indicators extracted by 5 tools: IPv6 addresses (*ip6*) and SSDeep fuzzy hashes. However, the majority corresponds to 38 (61%) indicators extracted by a single tool.

Among the indicators in Table 4 there are some that deserve discussion. There are three indicators (*country*, *tlpLabel*, *attackType*) that are identified using regular expressions that are a disjunction of keywords. For example, TLP labels can have only four values: *TLP:white*, *TLP:green*, *TLP:amber*, *TLP:red*. Since the set of possible TLP values is finite, the regular expression can identify all possible indicator values. Similarly, there is only a finite number of recognized countries. However, *attackType* includes a number of attack-related keywords that CYOBSTRACT identifies. Such a list only includes attack topics of interest for the tool authors and may not include attack topics other users are interested in. The approach of building a regular expression from a set of keywords can be applied to any taxonomy of terms (e.g., family names, exploit kit names). CYOBSTRACT provides support for building such regular expressions. The main limitation of this technique is that it cannot identify new terms (e.g., new family names or exploit kits).

CYOBSTRACT extracts two indicators that aim to capture organization names (*asnOwner*, *isp*). However, organization names do not have a clear structure save when using company-related suffixes (e.g., Ltd., Inc.) and thus are typically extracted using Named Entity Recognition (NER) techniques based on machine learning classifiers. Furthermore, trying to separate whether an organization name corresponds to an ISP or ASN owner is a challenging problem that is better suited for NLP techniques. There are also some indicators that one could argue should be split into multiple indicators. For example, the regular expression for the *incident* indicator extracted by CYOBSTRACT captures a disjunction of regular expressions, each for a different vulnerability report identifier (e.g., BugTrack, Microsoft Bulletins). Such disjunction is more efficient than having a separate regular expression for each identifier but does not allow users to extract only those identifiers they are interested in. Another interesting case is cryptographic hashes like *authentihash* and *importHash* which are really *subtypes* of other cryptographic hashes. For example, *importHash* is the MD5 of the import table of a PE executable [43], while *authentihash* is the SHA256 of a PE executable excluding Authenticode code signing fields. Thus, it is not possible for a regular expression to differentiate an *importHash* from a *md5* or an *authentihash* from a *sha256* except if the subtype indicators appear with some specific keywords before or after. Such regular expressions are very specific, they rarely produce FPs but they will introduce FNs when the expected keywords do not appear before or following the hashes. On the other hand, regular expressions for the parent types will also identify indicators subtypes (e.g., the *md5* regular expression will also identify *importHash* indicators), but classifying an MD5 into what the MD5 captures (e.g., a file hash, a certificate hash, an import table hash) is a challenging problem that is may be easier to tackle through NLP. There are also two indicators that correspond to ranges of IP addresses (*ip4range*, *ip6range*). Similar range indicators could be defined for any integer indicators, e.g., ASN numbers. The last interesting case is Yara rules extracted by IOCEXTRACT. Yara rules can be long and have an internal structure with mandatory and optional fields. Thus, some readers may not consider them indicators.

5.3. *Filtering.*

While most tools in Table 3 label themselves as IOC extraction tools, in reality, what they extract are indicators. Not all indicators are IOCs; an indicator is an IOC only if it represents a malicious artifact. For example, threat reports often contain hyperlinks to prior reports by other security vendors. Those references may be extracted as *url* indicators, but they can hardly be considered IOCs. We use the term *generic indicators* to (emphatically) refer to indicators that are not IOCs, *i.e.,* not malicious.

Determining whether an indicator is benign or malicious, *i.e.,* whether it is an IOC, is challenging. Most tools in Table 3 do not try to perform such determination and simply output all indicators they find. But, three tools (IOC_PARSER, CACADOR, and our IOCSEARCHER) implement a per-indicator filtering step, whose goal is to remove generic indicators leaving

Table 5: Data collection summary.

Source	Orig.	Start Date	Entries	Indicators Extracted	IOCs
aptnotes	155	2022/07/18	629	44,540	41,031
chainsmith	11	2022/07/18	3,792	48,046	41,246
malpedia	2,368	2022/07/18	11,363	271,495	237,011
rss	300	2021/04/01	97,882	314,351	210,149
telegram	9	2021/09/09	61,892	44,534	31,445
twitter	383	2021/10/28	397,333	338,967	126,081
ALL	*3,226*	2021/04/01	*472,891*	*978,151*	*618,217*

only the IOCs. IOC_PARSER and CACADOR both make use of hard-coded blocklists (or regular expressions) for benign indicators. Overall, the blocklists capture domains (or emails and URLs that contain those domains) from cybersecurity companies, news outlets, law enforcement agencies, and a few other trusted sites such as `github.com`. CACADOR uses a single blocklist with 15 domains, 12 of them from cybersecurity companies. IOC_PARSER uses 11 blocklists, one per each indicator type it supports. However, only 4 of those blocklists (*email*, *fqdn*, *ip4*, *url*) have entries. The *email* blocklist contains 6 domains of cybersecurity companies. The *fqdn* blocklist contains 143 domains of cybersecurity companies, news outlets, and law enforcement agencies. The *url* blocklist contains 79 root URLs for popular cybersecurity blogs. Finally, the *ip4* blocklist contains 6 reserved IP address ranges.

The main difference in our filtering module is that its blocklist is dynamically generated, *i.e.,* created from the collected documents. Indicators are added to the blocklist if they satisfy at least one of the following 5 rules: (*i*) the indicator is a *fqdn*, *url*, or *email*, whose domain is in the list of domains from where a document was collected; (*ii*) the indicator was extracted from at least 20 documents from the same origin; (*iii*) the indicator is a *fqdn* or *url* whose domain (excluding the "www." prefix if present) is in the top-100k of the Tranco domain popularity list [44]; (*iv*) the indicator appears in more than 90% of all the collected documents; or (*v*) the indicator is a private IP address. The first two rules aim to filter indicators that belong to the origins from where documents are collected, *e.g.,* email contact addresses for security companies such as *contact@trendmicro.com*. Instead of hard-coding domains of cybersecurity companies and blogs, we infer them from the origins of the collected documents. This allows each user to personalize the filtering to the list of origins he chooses to monitor. The next two rules target popular indicators that appear in most documents or in a list of popular domains. The frequently updated Tranco list is supplemented with a dynamically generated list of popular indicators appearing in the collected documents, which again can be personalized for each user.

Section 6 provides an evaluation of our filtering module and the filtering from CACADOR and IOC_PARSER on a manually generated ground truth.

6. Evaluation

We have applied GoodFATR to collect reports from six sources over the last 15 months. We started our collection with RSS on April 1st, 2021. We then added Telegram on September 9th 2021, and Twitter on October 10th, 2021. On July 18th, 2022, we added the three report datasets (APTnotes, ChainSmith, Malpedia), but these datasets are cumulative so GoodFATR can also process their past entries.

Table 5 summarizes the data collection and IOC extraction results for each source. It captures the number of origins in the source at the end of the collection, the collection start date, the number of collected entries, as well as the number of extracted indicators (before filtering) and IOCs (after filtering). The most diverse source by the number of origins is Malpedia with reports from 2,368 organizations, followed by Twitter with 383 accounts, RSS with 300 feeds, and APTnotes with reports from 155 organizations. In contrast, ChainSmith includes reports from only 11 blogs, and GoodFATR tracks only 9 Telegram channels.

Twitter is the source with the largest number of collected entries despite being introduced later than RSS and Telegram, but its entries are tweets and thus short compared to full reports collected from RSS and the three datasets. RSS is the most stable contributor with an average of 6.1K new report URLs per month. Among the three report datasets, Malpedia provides the most reports (11,363), with ChainSmith providing one-third (3,792), and APTnotes only 629 reports. The relatively small size of APTnotes is due to its more focused goal of only collecting reports for APTs.

IOC extraction. The last two columns in Table 5 show the indicators extracted by IOCSEARCHER (before filtering) and the final IOCs (after filtering). Overall, GoodFATR extracted 978k indicators of which 618K (63%) are IOCs. Thus, filtering is extremely important for discarding over one-third of indicators that are generic and thus are not IOCs. Filtering affects each source differently. In the extreme case of Twitter, two-thirds of the indicators are filtered. Across all sources, domain names and IPv4 addresses account for 99% of the generic indicators. Filtered domain names are those matching the domain from where the report was collected and those included in the Tranco top-100K list. Filtered IPv4 addresses are those of private and local networks,

The average number of IOCs per report is highest for the report datasets: APTnotes (66), Malpedia (22), and ChainSmith (11). Of those, APTnotes and Malpedia are manually curated, which prevents the collection of non-technical reports. The collection in ChainSmith is automated, but it leverages a small number of blogs from large security companies known for their high-quality technical content. In contrast, reports from RSS have a lower ratio of IOCs (2.2), likely due to the variety of feeds and reports they disseminate. For example, some feeds may focus on technology news for the wider public (and thus provide fewer IOCs), while blogs from security companies may mix technical reports with less technical reports that focus on the virtues of their products. Still, the large number of RSS feeds GoodFATR monitors compensates for the lower ratio, making RSS a consistent IOC contributor. Finally, Telegram (0.5) and Twitter (0.3) have the lowest ratios, which is expected due to the limited content in each entry.

IOC class distribution. To understand what type of indica-

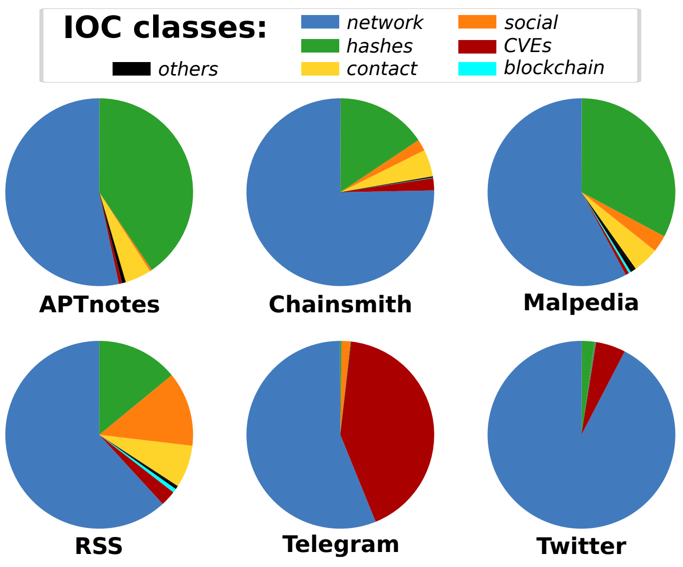

Figure 2: Distribution of IOC classes grouped per source.

tors each source distributes, we group indicators using the categories introduced in Section 5.2. Figure 2 reports the filtered IOCs split by class and source. The plots include only the six most popular classes, the rest are aggregated in the *other* category. Across all sources, *network* indicators dominate. Malpedia, Telegram, and Twitter are the biggest contributors to the *network* category, with more than 110K IOCs of this category per source. The *hash* category ranks second with most indicators in this category coming from Malpedia (82K file hash IOCs). In third place we find *contact* and *social* indicators, both having similar distributions with the vast majority coming from RSS and Malpedia. These two sources are also the highest contributors of IOCs that survive the filtering and exhibit the highest variety in terms of indicator types. For example, RSS and Malpedia provide 94% of the *blockchain* IOCs and over 61K email addresses, phone numbers and social network handles. Telegram and Twitter show a significantly smaller variety of IOC types. For example, the union of the *contact* and *social* classes represents only 1.6% of the IOCs in Telegram, and they only account for 92 of the IOCs extracted from Twitter. One possible reason is that Twitter accounts are more specialized with some accounts focusing only on certain indicator types such as malicious file hashes or vulnerability identifiers.

Top origins. Table 6 shows the top-10 origins for each source, ranked by the percentage of IOCs the origin contributes to the source. Overall, we notice a long tail distribution, with the top-10 origins generating 50% of all IOCs. This pattern is consistent across all sources and IOC classes. All three report datasets are dominated by large security companies, with the exception of the Citizen Lab at the University of Toronto (rank 8 in APTnotes) and three personal blogs (ranks 8–10 in ChainSmith). A surprising contributor is Price Waterhouse Coopers (rank 10 in APTnotes), one of the largest financial accounting companies, but less known for their security services. In general, personal blogs exhibit less activity compared to company blogs with 7% of all IOCs coming from personal blogs. RSS has more variety in the origins. Surprisingly, the top contributor is the personal blog from Dancho Danchev, followed by two Medium blogs that aggregate blockchain and cybersecurity news. The RSS top-10 is rounded by the research labs of three large companies (Cisco, F5, Malwarebytes), two other personal blogs (Bruce Schneier, contagiodump), and two magazines (Cointelegraph.com and BleepingComputer). Interestingly, two of the RSS top 10 origins focus on blockchain. We added blockchain-specific sources motivated by recent work that requires tagged Bitcoin addresses [45]. Telegram is largely dominated by the `cybsecurity` channel, which provides 76% of all Telegram IOCs. Similarly, Twitter also has one dominant account `@ecarlesi` with more than 140K tweets, 76% of all Twitter IOCs. Second, but far behind with 4.7% of IOCs, comes `@threatmeter`, an automated bot that publicizes vulnerabilities and accounts for 90% of *vulnerability* IOCs.

7. Evaluation of IOC Extraction Tools

To evaluate the indicator extraction tools, we design a majority-vote methodology that runs the 8 tools in Table 3 on the text of the same set of threat reports. Then, we compare the indicators extracted by the different tools on the same report, assuming that the correct indicators are those extracted by a *majority* of tools. The advantage of such evaluation is that it can be run on a large set of reports bypassing the challenge of building a ground truth that is representative of the wealth of indicators the tools extract. The disadvantage is that this approach works only when indicators are extracted by multiple tools, and in some occasions it is possible that the *minority* of tools is actually correct. We detail our methodology and the obtained results next.

We first build a document dataset by extracting the text using GoodFATR from all reports in APTnotes and ChainSmith. We choose these two sources because they cover different report file types (APTnotes has PDF reports and ChainSmith mostly HTML reports) and because both sources have a high average of IOCs per report, as shown in Section 6. In four PDF reports the text extraction failed: two were Excel spreadsheets that our text extraction does not support and the other two were corrupted. We evaluate the tools using the 4,420 successfully extracted text documents.

Methodology. Algorithm 1 details our accuracy comparison methodology. We run all tools on each document, saving the indicators each tool extracts to a separate file. To compare all tools in a similar setting, we disable filtering for tools that support it (IOC_PARSER, CACADOR, IOCSEARCHER) and deduplicate indicators. Since each tool may assign slightly different names to indicator types (e.g. ipv4addr, ip, and ipv4 for IPv4 addresses), we normalize them to match the names in Table 4. Additionally, some tools transform case-insensitive indicator values (e.g., domain names, email addresses) to lowercase or uppercase, while others output them as they appear in the text. To address such differences, we also normalize indicator values. In particular, we lowercase the following indicator values: hashes (*md5*, *sha1*, *sha256*, *sha512*, *ssdeep*), *regkey*, *ip6*, *fqdn*,

Table 6: Top-10 origins that contribute with the highest number of IOCs, grouped per source. In brackets we report the percentage of IOCs associated to each origin.

Rank	APTnotes	ChainSmith	Malpedia	RSS	Telegram	Twitter
#1	Kaspersky (13.11%)	Webroot (27.92%)	Palo Alto Networks (4.16%)	Dancho Danchev's Blog (26.21%)	cybsecurity (76.70%)	ecarlesi (76.42%)
#2	Palo Alto Networks (7.18%)	Sucuri (22.07%)	Trend Micro (4.11%)	Blockchain on Medium (9.06%)	VulnerabilityNews (8.28%)	threatmeter (4.73%)
#3	Norman (6.11%)	Malwarebytes (15.01%)	Kaspersky (2.92%)	Cybersecurity on Medium (8.63%)	Cyber_Security_Channel (7.33%)	malwrhunterteam (2.54%)
#4	Symantec (5.24%)	Virus Bulletin (14.12%)	ESET (2.82%)	Cisco Talos (6.57%)	cKure (4.01%)	bgpstream (2.02%)
#5	ClearSky (4.74%)	Sophos (4.88%)	Proofpoint (1.68%)	BleepingComputer News (2.44%)	malwr (2.18%)	YourAnonRiots (1.54%)
#6	FireEye (4.24%)	Forcepoint (4.60%)	Cisco Talos (1.66%)	Cointelegraph.com News (2.33%)	androidMalware (0.84%)	cryptolaemus1 (1.44%)
#7	ESET (3.72%)	ESET (4.34%)	FireEye (1.65%)	F5 Labs (1.28%)	ckuRED (0.48%)	ActorExpose (1.32%)
#8	Trend Micro (2.67%)	TaoSecurity (4.15%)	BitDefender (1.44%)	Schneier on Security (1.26%)	canyoupwnme (0.16%)	MalwarePatrol (0.98%)
#9	Citizen Lab (2.40%)	Hexacorn (1.70%)	360netlab (1.39%)	Malwarebytes Labs (1.12%)	itsecalert (0.03%)	1zrr4h (0.97%)
#10	PwC UK Blogs (1.88%)	Roger McClinton (0.90%)	Microsoft (1.34%)	contagiodump (1.08%)	-	dubstard (0.74%)

Table 7: Precision (P), recall (R), and F1 score achieved by each indicator extraction tool on the comparative evaluation over 4,420 documents. Only indicator types extracted by at least two tools are included.

		JAGER			IOC_PARSER			CACADOR			CYOBSTRACT			IOC-FINDER			IOCEXTRACT			IOC-EXTRACTOR			IOCSEARCHER		
Indicator	**Count**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
asn	202	-	-	-	-	-	-	-	-	-	0.93	0.95	0.94	**0.99**	**1.00**	**0.99**	-	-	-	0.69	0.97	0.81	-	-	-
bitcoin	2,698	-	-	-	-	-	-	-	-	-	-	-	-	0.77	1.00	0.87	-	-	-	**1.00**	**1.00**	**1.00**	1.00	0.01	0.03
cve	2,135	0.96	0.99	0.98	**1.00**	**1.00**	**1.00**	-	-	-	0.98	0.97	0.97	0.98	1.00	0.99	-	-	-	0.96	1.00	0.98	**1.00**	**1.00**	**1.00**
email	1,947	0.60	0.85	0.70	0.89	0.75	0.81	0.60	0.73	0.66	0.99	0.98	0.98	0.93	1.00	0.96	0.75	0.97	0.85	0.97	0.99	0.98	**0.99**	**1.00**	**1.00**
filename	17,082	**0.98**	**0.97**	**0.98**	0.97	0.71	0.82	0.87	0.97	0.92	0.78	0.87	0.82	-	-	-	-	-	-	-	-	-	-	-	-
filepath	1,551	-	-	-	**0.73**	**0.66**	**0.69**	-	-	-	0.29	0.97	0.45	0.25	0.76	0.37	-	-	-	-	-	-	-	-	-
fqdn	41,360	0.48	0.06	0.10	0.99	0.91	0.94	0.56	0.39	0.46	0.97	0.96	0.97	0.95	0.99	0.97	-	-	-	0.92	0.99	0.95	**0.98**	**1.00**	**0.99**
googleAdsense	4	-	-	-	-	-	-	-	-	-	-	-	-	**1.00**	**1.00**	**1.00**	-	-	-	**1.00**	**1.00**	**1.00**	1.00	0.75	0.86
googleAnalytics	3	-	-	-	-	-	-	-	-	-	-	-	-	**1.00**	**1.00**	**1.00**	-	-	-	0.75	1.00	0.86	**1.00**	**1.00**	**1.00**
ip4	9,479	0.99	0.87	0.92	0.98	0.92	0.95	0.97	1.00	0.98	1.00	0.98	0.99	0.99	0.99	0.99	0.98	0.97	0.98	0.98	1.00	0.99	**1.00**	**1.00**	**1.00**
ip4cidr	287	-	-	-	-	-	-	-	-	-	**1.00**	**0.99**	**0.99**	0.97	1.00	0.99	-	-	-	-	-	-	**1.00**	**0.99**	**0.99**
ip6	967	-	-	-	-	-	-	0.50	0.91	0.65	**0.87**	**0.66**	**0.75**	0.91	0.12	0.21	0.15	0.80	0.25	0.51	0.97	0.66	-	-	-
macAddress	64	-	-	-	-	-	-	-	-	-	-	-	-	**1.00**	**1.00**	**1.00**	-	-	-	**1.00**	**1.00**	**1.00**	-	-	-
md5	14,635	1.00	0.98	0.99	1.00	1.00	1.00	0.45	1.00	0.62	1.00	0.97	0.98	**1.00**	**1.00**	**1.00**	1.00	1.00	1.00	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
monero	2	-	-	-	-	-	-	-	-	-	-	-	-	**1.00**	**1.00**	**1.00**	-	-	-	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
regkey	608	-	-	-	0.96	0.72	0.82	-	-	-	**0.80**	**0.90**	**0.85**	0.69	0.73	0.71	-	-	-	-	-	-	-	-	-
sha1	4,150	**1.00**	**1.00**	**1.00**	1.00	0.99	0.99	0.41	1.00	0.58	1.00	0.99	0.99	**1.00**	**1.00**	**1.00**	1.00	1.00	1.00	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
sha256	5,336	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.97	1.00	0.99	1.00	0.94	0.97	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
sha512	1	**1.00**	**1.00**	**1.00**	-	-	-	0.07	1.00	0.12	-	-	-	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	-	-	-
ssdeep	74	1.00	0.28	0.44	-	-	-	0.16	0.30	0.21	**0.81**	**0.88**	**0.84**	0.55	0.91	0.69	-	-	-	0.47	1.00	0.64	-	-	-
url	14,818	0.61	0.56	0.59	0.53	0.56	0.54	0.51	0.52	0.52	0.76	0.96	0.85	0.62	0.83	0.71	0.60	0.99	0.75	0.70	0.80	0.74	**0.77**	**1.00**	**0.87**
All	117,403	0.87	0.57	0.69	0.91	0.85	0.88	0.61	0.70	0.65	0.92	0.96	0.94	0.87	0.96	0.91	0.77	0.99	0.87	0.90	0.96	0.93	**0.95**	**0.97**	**0.96**

and *email*. Finally, we normalize AS numbers to the format *AS1234* (e.g. from asn1234 to AS1234) and URLs by prepending "http://" when no scheme is present.

For each tool, the evaluation keeps counters for true positives (TPs), false positives (FPs), false negatives (FNs), and true negatives (TNs). The evaluation processes one document at a time, updating the counters with the document results. For each document, it examines all indicators extracted from the document by at least one tool. For each indicator, it generates three sets: the *found* set captures the tools that identified the indicator in the document, the *missed* set captures the tools that support the indicator type but did not identify it, and the *unsupported* set captures the tools that do not support the indicator type. If the size of the found set is larger than the size of the missed set, then the evaluation assumes the majority is correct and therefore the indicator was indeed present in the document. Thus, it adds a TP for each tool in the found set and a FN for each tool in the missed set. If the size of the missed set is larger than the size of the found set, then the evaluation assumes the majority is correct and therefore the indicator was not present in the document. Thus, it adds a FP for each tool in the found set and a TN for each tool in the missed set. If the size of the found and missed sets is the same, then there is no majority. In such cases, we skip the indicator and do not update the counters. In future work, we plan to investigate how to break such ties. After processing all documents, we compute the precision, recall, and F1 score for each tool across all indicators, as well as separately for each indicator type.

Handling errors. When running all tools on the 4,420 documents we observed that IOCEXTRACT, the second most popular tool, needed hours to process some documents, compared to seconds or a few minutes for other tools. We suspected that one of their regular expressions had a ReDoS vulnerability, which caused a worst case that got triggered only in some documents [19]. We found an open issue in the IOCEXTRACT repository regarding catastrophic backtracking in the regular expression used to identify defanged URLs that modify the backslash [46]. To be able to complete the evaluation in a reasonable time, we configured IOCEXTRACT to avoid using the problematic regular expression. This change does not affect the accuracy results, as IOCEXTRACT is the only tool supporting that defang transformation. Similarly, there are two documents where IOC-EXTRACTOR does not terminate. We identify the root of this issue in the regular expressions used to extract domain names. Of the four domain regular expressions used, only one avoids the problem, while the others require hours to process the document. For these two documents, we added IOC-EXTRACTOR to the missed set for all indicators in the document. These results show that ReDoS vulnerabilities are a serious problem for indicator extraction tools, but oftentimes these worst cases appear only when examining a large number of documents. We examine whether ReDoS vulnerabilities exist in the regular expressions used by IOCSEARCHER with the *rat* Re-

Algorithm 1 Accuracy comparison methodology.

```
 1: procedure COMPARE($tools$, $docs$)
 2:     $accuracy \leftarrow \{\}$
 3:     for tool in tools do
 4:         $accuracy[tool] \leftarrow (0,0,0,0)$
 5:     end for
 6:     for doc in docs do
 7:         $extracted \leftarrow \{\}$
 8:         $all \leftarrow set()$
 9:         for tool in tools do
10:             $iocs \leftarrow Run(tool, doc)$
11:             $iocs \leftarrow Normalize(iocs)$
12:             $all \leftarrow all \cup iocs$
13:             for ioc in iocs do
14:                 $extracted[ioc].add(tool)$
15:             end for
16:         end for
17:         for ioc in all do
18:             $found \leftarrow extracted[ioc]$
19:             $supported \leftarrow canExtract(ioc.type)$
20:             $missed \leftarrow supported - found$
21:             if len(found) $>$ len(missed) then
22:                 for tool in found do
23:                     $accuracy[tool].tp$ += 1
24:                 end for
25:                 for tool in missed do
26:                     $accuracy[tool].fn$ += 1
27:                 end for
28:             end if
29:             if len(missed) $>$ len(found) then
30:                 for tool in found do
31:                     $accuracy[tool].fp$ += 1
32:                 end for
33:                 for tool in missed do
34:                     $accuracy[tool].tn$ += 1
35:                 end for
36:             end if
37:         end for
38:     end for
39:     return $accuracy$
40: end procedure
```

DoS checker [47]. The tool implements a sound static analysis (*i.e.,* no false positives) that identifies regular expressions with exponential time worst case [48]. Except for 3 regular expressions the *rat* tool could not handle due to negative lookbehinds, all other IOCSEARCHER regular expressions are free from exponential time worst case.

There are also a few documents where a tool throws an exception and thus extracts no indicators. This happens for CYOBSTRACT in 41 documents and for JAGER in 20. For CYOBSTRACT, only the indicators of the type causing the exception are missed. We included JAGER in the missed set for all indicators in those documents.

Results. Table 7 summarizes the results for the 21 indicator types that are extracted by more than one tool. The columns show the indicator type, the number of indicators of that type considered TPs, as well as the precision (P), recall (R), and F1 score for each tool. We do not include indicators extracted only by a single tool, as there is no concept of majority for those.

Overall, IOCSEARCHER is the tool that achieves both the best precision (0.95) and F1 score (0.96), while IOCEXTRACT achieves the best recall (0.99). The highest overall F1 scores are for IOCSEARCHER (0.96), CYOBSTRACT (0.94), IOC-EXTRACTOR (0.93), and IOC-FINDER (0.91). The lowest F1 scores are for CACADOR (0.65) and JAGER (0.69). These results seem to indicate that recent tools (IOCSEARCHER, IOC-EXTRACTOR, IOC-FINDER) perform better than those released earlier (JAGER, CACADOR), possibly because newer tools could use older ones for comparison.

Table 8: Top-10 indicator types extracted by all tools.

Indicator	Count	Tools
fqdn	41,360	7
attackType	26,109	1
country	17,654	1
filename	17,082	4
url	14,818	8
md5	14,635	8
ip4	9,479	8
sha256	5,336	8
avLabel	4,660	1
sha1	4,150	8

The table also presents the accuracy results for each indicator type. We observe perfect agreement for MAC address extraction (although only 64 MAC addresses are found by two tools) and nearly perfect agreement on the hashes, with the exception of CACADOR which has low F1 scores for *sha512* (0.12), *sha1* (0.58), and *md5* (0.62). IOCSEARCHER performs best in 11 of the 13 (85%) indicator types it supports in the table, IOC-EXTRACTOR in 8 out of 17 (47%), and IOC-FINDER in 9 out of 20 (45%). CYOBSTRACT is the best tool for extracting registry keys (0.85) and SSDeep hashes (0.84); JAGER for extracting filenames (0.98), IOC_PARSER for filepaths (0.69), and IOCSEARCHER for emails (1.0), domain names (0.99), IPv4 addresses (1.0), and URLs (0.87). These results are useful for future work in this area, to identify which prior tool may have the best regular expression for an indicator type. For example, if we were to add registry key support to IOCSEARCHER we would start by looking at CYOBSTRACT and for filenames to JAGER.

The indicator types with the lowest agreement are IPv6 addresses with F1 score ranging 0.21–0.75; filepaths (0.37–0.69); URLs (0.52–0.87); and registry keys (0.71–0.85). These are arguably the indicator types for which regular expressions are harder to build. We observe that most IPv6 addresses extracted are actually FPs caused by a regular expression that matches serial numbers in certificates and certificate fingerprints.

One surprising result is that IOCSEARCHER has an F1 score of 0.03 on Bitcoin addresses, while it achieves the best F1 score on most other indicators it extracts. We manually checked the extracted Bitcoin addresses and observed that IOCSEARCHER is actually always correct in identifying Bitcoin addresses. Both IOC-EXTRACTOR and IOC-FINDER return hashes that are erroneously included as Bitcoin addresses. IOCSEARCHER avoids those FPs by examining the checksum in the regular expression matches, which do not validate. This is an example of the minority of tools being correct and the majority being wrong. We discuss this issue in Section 8.

Table 8 shows the Top 10 most popular indicator types across the 4,420 documents. The count column represents the number of TPs, and the last column reports the number of tools that ex-

Table 9: Post-hoc t-tests between groups summary.

A	B	p-unc	p-corr	eta-square
CACADOR	CYOBSTRACT	7.6527e-03	1.0203e-02	3.6639e-04
CACADOR	IOCEXTRACT	3.1698e-21	1.4792e-20	1.2091e-03
CACADOR	IOC-EXTRACTOR	2.9591e-18	1.1836e-17	8.6447e-04
CACADOR	IOC-FINDER	1.12366-17	3.9328e-17	8.48226-04
CACADOR	IOC_PARSER	7.11888-05	1.4237e-04	4.7131e-05
CACADOR	JAGER	1.0000e+00	1.0000e+00	0.0000e+00
CACADOR	IOCSEARCHER	9.5401e-23	8.9041e-22	1.3045e-03
CYOBSTRACT	IOCEXTRACT	1.0013e-02	1.2744e-02	2.4565e-04
CYOBSTRACT	IOC-EXTRACTOR	1.1071e-01	1.2916e-01	1.0536e-04
CYOBSTRACT	IOC-FINDER	1.1717e-01	1.3123e-01	9.9363e-05
CYOBSTRACT	IOC_PARSER	4.2949e-04	7.5162e-04	6.7494e-04
CYOBSTRACT	JAGER	2.1251e-02	2.5871e-02	3.6767e-04
CYOBSTRACT	IOCSEARCHER	5.4571e-03	8.0421e-03	2.8976e-04
IOCEXTRACT	IOC-EXTRACTOR	5.9633e-03	8.3486e-03	2.9454e-05
IOCEXTRACT	IOC-FINDER	3.5693e-03	5.5523e-03	3.2923e-05
IOCEXTRACT	IOC_PARSER	3.6201e-25	5.0681e-24	1.7284e-03
IOCEXTRACT	JAGER	2.6479e-10	7.4141e-10	1.2132e-03
IOCEXTRACT	IOCSEARCHER	2.4354e-08	5.2456e-08	1.8189e-06
IOC-EXTRACTOR	IOC-FINDER	6.2505e-01	6.4820e-01	9.3784e-08
IOC-EXTRACTOR	IOC_PARSER	3.8698e-22	2.7089e-21	1.3122e-03
IOC-EXTRACTOR	JAGER	1.1041e-08	2.8105e-08	8.6746e-04
IOC-EXTRACTOR	IOCSEARCHER	6.1777e-04	1.0175e-03	4.5934e-05
IOC-FINDER	IOC_PARSER	1.4453e-21	8.0938e-21	1.2927e-03
IOC-FINDER	JAGER	1.3778e-08	3.2149e-08	8.5116e-04
IOC-FINDER	IOCSEARCHER	3.6823e-04	6.8737e-04	5.0259e-05
IOC_PARSER	JAGER	1.3860e-01	1.4927e-01	4.7302e-05
IOC_PARSER	IOCSEARCHER	1.0825e-26	3.0310e-25	1.8419e-03
JAGER	IOCSEARCHER	5.4592e-11	1.6984e-10	1.3090e-03

tract each indicator. Among those, 7 correspond to indicators supported by most tools, namely domain names, file names, URLs, IPv4 addresses, and hashes (*md5*, *sha256*, *sha1*). On the other hand, the other three indicators are extracted only by CYOBSTRACT and correspond to attack-type keywords, countries, and AV labels. A total of 16 indicator types are not found in the 4,420 reports. These include *attribution*, *groupName*, *authentihash*, *ip6range*, *useragent*, *iban*, *icp*, 7 blockchain addresses (*dashcoin*, *dogecoin*, *ethereum*, *litecoin*, *tezos*, *webmoney*, *zcash*), and two social handles (*telegramHandle*, *whatsappHandle*). These correspond to less popular indicators in APTnotes and ChainSmith reports. However, other sources may differ. For example, RSS collection includes blockchain-related blogs, which leads to IOCSEARCHER extracting indicators for all blockchain addresses in Section 6 (e.g., 726 *ethereum* addresses).

Statistical significance. We evaluate whether differences in the extracted indicators across tools are statistically significant by first performing an analysis of variance test and then measuring the strength of the differences using pairwise t-tests between the tools. The independent variable is the tool used to produce each set of IOC detections. There are eight groups accounting for the source of variability, one per tool. The dependent variable is the number of detections (TPs) produced by each tool. We perform a Repeated Measures One-Way ANOVA test because all tools are evaluated on the same samples (*i.e.,* documents). We focus on the 6 indicator types extracted by all tools (*email*, *ip4*, *md5*, *sha1*, *sha256*, *url*).

The test results indicate a significant statistical difference between the detections produced by the tools, with a p-value lower than 10^{-5}. We apply pairwise t-tests among groups to identify which pairs of tools exhibit significant differences. We use the Benjamini/Hochberg FDR correction method for the p-values because of the multiple statistical tests produced. Table 9 shows the results of this test. A statistically significant difference between detections is found among every pair of tools (shown by the *p-corr* column), except for 5 pairs (CACADOR–JAGER, CYOBSTRACT–IOC-EXTRACTOR, CYOBSTRACT–IOC-FINDER, IOC-EXTRACTOR–IOC-FINDER, and IOC_PARSER–JAGER). Figure 3 shows the strength of the differences between tools, measured by the *eta-square* value resulting from the tests. The results show that detections produced by IOCSEARCHER are statistically significant compared to all other tools.

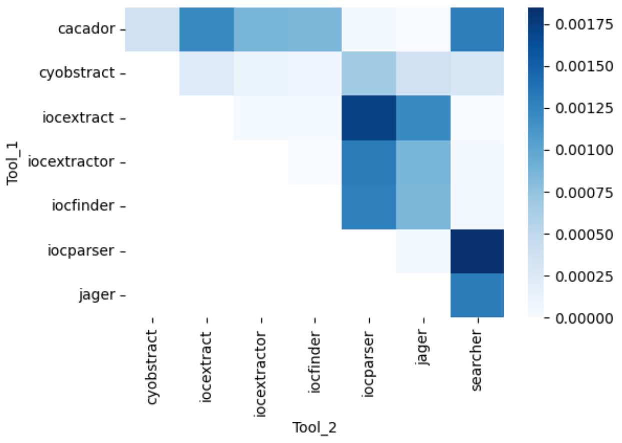

Figure 3: Eta-square values resulting from pairwise comparison of tools.

Table 10: Indicator extraction runtime (in seconds) across all APTnotes and average by indicator type.

Tool	Runtime (sec)	Ind.	Avg. per Ind. (sec)
CACADOR	489.23	12	40.76
IOCEXTRACT	1350.69	8	168.83
IOC_PARSER	1563.33	11	142.12
IOC-EXTRACTOR	1978.51	18	109.91
JAGER	2360.02	11	214.54
IOCSEARCHER	2847.98	41	69.46
CYOBSTRACT	6945.03	25	277.80
IOC-FINDER	96596.23	25	3863.84

Runtime. Table 10 shows the total time in seconds each tool took to extract indicators on the 4,420 text documents, the number of indicator types supported, and the average time by indicator type. In our tests, the fastest tool is CACADOR, likely due to being implemented in Go and compiled to native code. The slowest tool is IOC-FINDER, which is 14 times slower than the second slowest tool (CYOBSTRACT). This is probably because IOC-FINDER is the only tool that uses grammars for the extraction, highlighting the efficiency of using regular expressions to extract indicators. All tools perform one pass on the text for each regular expression and they typically use one regular expression for each indicator type. Thus, the runtime is highly influenced by the number of indicator types extracted. When normalizing the total runtime by the number of extracted indicator types, CACADOR is still the fastest with 40.8 seconds, followed by IOCSEARCHER (69.5), and IOC-EXTRACTOR (109.9).

Social handle validation. Our majority-based evaluation cannot be applied to the 38 indicators only extracted by one tool.

Table 11: Social indicators validation.

Indicator	Total	Validated
facebookHandle	150	124
gitHubHandle	258	242
instagramHandle	10	8
pinterestHandle	1	1
youTubeHandle	24	24
youTubeChannel	6	6
twitterHandle	516	431
All	965	836

Table 12: Accuracy of filtering approaches on a manually-built ground truth.

Tool	TP	FP	FN	TN	Prec.	Recall	F1
CACADOR	29	69	0	8	0.30	1.00	0.46
IOC_PARSER	29	66	0	11	0.31	1.00	0.47
IOCSEARCHER	29	6	0	71	0.83	1.00	0.91

Among those, there are 11 social indicators only extracted by IOCSEARCHER, of which 9 appear in APTnotes and ChainSmith reports (no Telegram and WhatsApp handles are found in these sources). For social network handles we perform an alternative validation, which checks whether there *currently* exists an account in the social network for that handle. For this, we use the Blackbird [49] open-source tool, which given a username checks if there currently exists an account with that handle on 143 different social networks. Blackbird supports 6 social networks IOCSEARCHER extracts handles for (Facebook, GitHub, Instagram, Pinterest, Telegram, and YouTube). In addition, we expanded Blackbird to also support *youtubeChannel* indicators. For Twitter handles, we leverage the official Twitter API for the same purpose [50]. LinkedIn does not allow searching for account names.

Table 11 summarizes the results. Our automated approach successfully validated 87% of the social indicators. We manually inspect the 102 indicators that we could not validate with Blackbird, using a web browser to search for a particular handle on each social network. For Twitter, 15 of of the 85 usernames that did not validate, were linked to suspended accounts (e.g., `@MalwareSigs`). In all of the remaining cases, the social networks returned a page to inform us that the particular account does not exist. For 75 of the 87 accounts for which we could not find the user profile page, the identifier contained a meaningful sequence of characters (e.g., "sucuri_security", "avast_antivirus"), suggesting that the identifier could belong to an old account that was closed. Overall, this evaluation suggests that over 90% of the social identifiers IOCSEARCHER extracted are likely true positives. For the rest, we are not able to determine if they are false positives or correspond to accounts that have been closed since they appeared in the sources.

Filtering. To evaluate the filtering, we manually built a ground truth (GT) of 106 indicators. We randomly selected those indicators among those extracted by IOCSEARCHER across all datasets and prior to the filtering. Then, an analyst manually reviewed the documents from where the indicators were extracted to determine whether each indicator was benign or malicious. The analyst made a determination based on the indicator context in the document, i.e., by reading the parts of the document that mentioned the indicator.

To measure the filtering accuracy, we first apply the filtering approach of CACADOR, IOC_PARSER, and our platform to the 106 indicators. Then, for each approach, we compare the indicators that survive the filtering with the GT. We consider a true positive (TP) a malicious indicator in the GT that correctly survives the filtering and a TN is a benign indicator in the GT that was correctly filtered out. A false positive (FP) is a benign indicator in the GT that incorrectly passes the filtering, while a false negative (FN) is a malicious indicator in the GT incorrectly removed by the filtering. From those counts, we compute the precision, recall, and F1 score. Note that it does not matter that indicators in the GT were selected among those extracted by IOCSEARCHER, as we do not extract indicators using CACADOR and IOC_PARSER in this experiment. We only run their filtering rules on the GT indicators.

Table 12 summarizes the results. All three filtering approaches have zero FNs and thus perfect recall, i.e., they do not remove any truly malicious indicators. On the other hand, precision significantly differs; our filtering achieves more than twice higher precision (0.83) than the filtering used by IOC_PARSER (0.31) and CACADOR (0.30). The low precision by CACADOR and IOC_PARSER is due to its filtering being fairly incomplete, allowing many benign indicators to be output as malicious IOCs. The final F1 score is 0.91 for our filtering, followed by IOC_PARSER (0.47) and CACADOR (0.46). The results indicate that the static blocklists used by IOC_PARSER and CACADOR fail to filter many of the benign indicators in the documents. Furthermore, despite the IOC_PARSER blocklist being an order of magnitude larger than the CACADOR blocklist, the recall improvement is marginal. This shows that it is hard to predict the sources from where a user will collect reports, making static blocklists largely incomplete. The use of a dynamic blocklist in our filtering module significantly improves the recall by adjusting the filtering to the collected sources. For example, it is very common for tweets to contain URLs to external references, including links to documents from other sources (e.g, an already monitored blog or RSS feed). The blocklists of CACADOR and IOC_PARSER would assume that such a URL is an IOC, possibly because the tool developers did not consider Twitter as a source. IOCSEARCHER correctly flags those generic indicators as FPs, since it relies on a blocklist that is dynamically expanded each time the user adds a new source that will be monitored. In summary, the dynamic blocklist allows filtering a larger amount of benign indicators than static blocklists without losing any malicious IOCs.

8. Discussion

This section discusses limitations and future improvements.

Threats to validity. Our methodology for comparing indicator extraction tools assumes that the majority of tools will be correct. However, in some cases, a minority of tools may be correct instead, as illustrated by the *bitcoin* results where IOCSEARCHER is wrongly assigned FNs due to FPs from two other tools. In general, the more tools support an indicator, the higher confidence we have in the majority of results. An alternative approach for evaluation tools is to use a ground truth (GT) dataset.

However, the manual effort required to build a GT with thousands of documents would be very large, and it would be difficult to cover different indicator types. For this reason, regular expression evaluations tend to be built leveraging synthetic examples (e.g., [51]). Unfortunately, synthetic examples may not represent results on real documents.

Our evaluation could be biased due to document selection, unknown normalizations, expired social accounts, and the size of the ground truth used for evaluating the filtering. For example, we evaluate tools exclusively on English documents. While regular expressions should largely be language-agnostic, it is possible that some language-specific feature (e.g., special characters) affects the results. It is also possible for our comparison methodology to negatively affect a tool if we miss some normalization it performs on its outputs. The validation of social handles may underestimate the IOCSEARCHER accuracy since extracted handles may correspond to accounts since deleted. And, our filtering evaluation is performed on a small ground truth of 106 indicators. A larger ground truth may uncover harder cases where accuracy drops.

Adding origins. Adding new origins is currently the only step where GoodFATR requires human involvement. GoodFATR can automatically identify new origins to be considered for inclusion. For this, it examines the report URLs in the crowdsourced datasets (Malpedia, APTnotes) ranking domains in the report URLs by the number of report URLs where they appear. Then, it filters domains already in the list of RSS origins. Finally, the top-ranked domains are flagged as potential cybersecurity blogs, so that an analyst can examine them and identify their RSS URL. To identify candidate new Twitter accounts to monitor, GoodFATR examines the re-tweets in the monitored accounts, building a ranking from the original tweeting account to the number of collected re-tweets from the account. The top re-tweeted accounts, not yet monitored, are output so that a human can analyze them for inclusion. In future work, we would like to explore integrating techniques to identify valuable Twitter accounts by analyzing their posts [23].

Report variants. If the same document (i.e., same SHA256 hash) is collected from different origins, GoodFATR deduplicates it to store only one copy of the document and updates the document's traceability information to capture all origins from where the document was collected. As introduced in Section 4, it is possible that GoodFATR collects multiple documents corresponding to different instances of the same threat report. This may happen if a report is distributed in different formats, in case GoodFATR downloads multiple times the URL of a report that points to a webpage with dynamic content, and if a report generates multiple versions over time (e.g., by fixing some errata). Currently, the user can identify multiple instances of the same report by querying for all documents downloaded from the same URL or for all documents with the same title. However, this may miss some cases, e.g., when the title has changed. To address this issue, GoodFATR could store a similarity hash of the text content (extracted from the PDF or HTML document), which could be used to identify small variations of the same content.

In this work, GoodFATR was configured to download a URL only once (unless an error happened). This configuration reduces the number of needed downloads and removes some of the above cases. On the other hand, it may fail to collect different versions of the same report that appear over time. If the user wants to collect multiple versions of a report, he can easily change this configuration, at the cost of increased network bandwidth and runtime. In future work, we would like to explore leveraging the ETag header to keep track of the document pointed by a URL and use an HTTP HEAD request to check if the document was updated since the last time we retrieved it. With this approach, the report would only be re-downloaded if a new version exists. However, not all websites provide the ETag header or support HTTP HEAD requests.

9. Conclusions

This paper presents GoodFATR, an automated platform for collecting threat reports from 6 sources (RSS, Twitter, Telegram, Malpedia, APTnotes, ChainSmith) and comparing the accuracy of indicator extraction tools on the collected reports. GoodFATR implements a novel majority vote methodology for comparing the accuracy of indicator extraction tools, which does not require a manually-built ground truth. GoodFATR continuously monitors the sources, downloads new threat reports, extracts indicators from the reports, and filters generic indicators to produce a list of IOCs. GoodFATR includes the IOCSEARCHER tool for extracting 41 indicator types from HTML, PDF, and text files using regular expressions. We run GoodFATR for over 15 months to collect 472,891 reports from the 6 sources; extract 978,151 indicators from the reports using IOCSEARCHER; and identify 618,217 IOCs. We analyze the collected data to identify the top IOC contributors and the IOC class distribution. Then, we applied GoodFATR for comparing 7 popular indicator extraction tools with IOCSEARCHER and assess their accuracy using our novel majority-vote methodology.

Acknowledgments

We are grateful to Pierre Ganty for his help with the rat ReDoS checker tool. This work has been partially supported by the Madrid regional government through program S2018/TCS-4339 (BLOQUES-CM) and Atracción de Talento grant 2020-T2/TIC-20184. This work has also been partially supported by MCIN/AEI/10.13039/501100011033/ through grants RTI2018-102043-B-I00 (SCUM), TED2021-132464B-I00 (PRODIGY), and PRE2019-088472. Partial funding was also provided by Ministerio de Ciencia, Innovación y Universidades grant FPU18/06416. The above programs are co-funded by the European Union EIE, ESF, and NextGeneration EU/PRTR Funds. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators, and do not necessarily reflect the views of the sponsors.

References

[1] MarketWatch, Threat intelligence market 2022 report, https://www.marketwatch.com/press-release/threat-intelligence-market-2022-report-examines-latest-trends-and-key-drivers-supporting-growth-till-2030-2022-07-27 (2022). 1

[2] V. G. Li, M. Dunn, P. Pearce, D. McCoy, G. M. Voelker, S. Savage, Reading the Tea leaves: A Comparative Analysis of Threat Intelligence, in: USENIX Security, 2019. 1, 2, 3

[3] X. Bouwman, H. Griffioen, J. Egbers, C. Doerr, B. Klievink, M. Van Eeten, A different cup of TI? The added value of commercial threat intelligence, in: USENIX Security Symposium, 2020. 1, 2

[4] O. Open, Stix: A structured language for cyber threat intelligence, https://oasis-open.github.io/cti-documentation/ (2022). 1

[5] W. Gibb, D. Kerr, Openioc: Back to the basics, https://www.mandiant.com/resources/openioc-basics (October 2013). 1

[6] C. Sabottke, O. Suciu, T. Dumitras, Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits, in: USENIX Security, 2015. 1

[7] F. Alves, A. Andongabo, I. Gashi, P. M. Ferreira, A. Bessani, Follow the Blue Bird: A Study on Threat Data Published on Twitter, in: ESORICS, 2020. 1, 3

[8] J. Kornblum, ssdeep Project, https://ssdeep-project.github.io/ssdeep/index.html (2016). 1, 9

[9] S. J. Roberts, jager, https://github.com/sroberts/jager (2015). 1, 2, 7

[10] A. Buescher, ioc_parser, https://github.com/armbues/ioc_parser/ (2017). 1, 2, 7

[11] S. J. Roberts, Cacador, https://github.com/sroberts/cacador (2016). 1, 2, 7

[12] M. Sisk, R. Ruefle, S. Perl, Harvesting Artifacts: Improving Useful Data Extraction from Cybersecurity Incident Reports, https://github.com/cmu-sei/cyobstract (2018). 1, 2, 7

[13] InQuest, iocextract, https://github.com/InQuest/python-iocextract (2019). 1, 2, 7

[14] M. Niseki, IoC extractor, https://github.com/ninoseki/ioc-extractor (2019). 1, 2, 7

[15] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, R. Beyah, Acing the IOC Game: Toward Automatic Discovery and Analysis of Open-Source Cyber Threat Intelligence, in: CCS, 2016. 1, 3

[16] G. Husari, E. Al-Shaer, M. Ahmed, B. Chu, X. Niu, TTPDrill: Automatic and Accurate Extraction of Threat Actionsfrom Unstructured Text of CTI Sources, in: ACSAC, 2017. 1, 3

[17] Z. Zhu, T. Dumitras, ChainSmith: Automatically Learning the Semantics of Malicious Campaigns by Mining Threat Intelligence Reports, in: Euro S&P, 2018. 1, 2, 3, 4, 6

[18] K. Satvat, R. Gjomemo, V. Venkatakrishnan, Extractor: Extracting Attack Behavior from Threat Reports, in: Euro S&P, 2021. 1

[19] J. C. Davis, C. A. Coghlan, F. Servant, D. Lee, The Impact of Regular Expression Denial of Service (ReDoS) in Practice: An Empirical Study at the Ecosystem Scale, in: ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018. 2, 12

[20] D. Plohmann, M. Clauß, S. Enders, E. Padilla, Malpedia: A Collaborative Effort to Inventorize the Malware Landscape, The Journal on Cybercrime & Digital Investigations 3 (1) (2017). 2, 3, 4, 5

[21] K. Bandla, S. Castro, APTnotes, https://github.com/aptnotes/data (2022). 2, 3, 4, 6

[22] F. Hightower, IOC Finder, https://github.com/fhightower/ioc-finder (2018). 2, 7

[23] A. Niakanlahiji, L. Safarnejad, R. Harper, B.-T. Chu, IoCMiner: Automatic Extraction of Indicators of Compromise from Twitter, in: IEEE Big Data, 2019. 3, 16

[24] J. Zhao, Q. Yan, J. Li, M. Shao, Z. He, B. Li, TIMiner: Automatically extracting and analyzing categorized cyber threat intelligence from social data, Computers & Security (2020). 3

[25] H. Shin, W. Shim, S. Kim, S. Lee, Y. G. Kang, Y. H. Hwang, Twiti: Social Listening for Threat Intelligence, in: WWW, 2021. 3

[26] J. Caballero, G. Gomez, S. Matic, G. Sánchez, S. Sebastián, A. Villacañas, (Work-in-progress) FATR: a Framework for Automated Analysis of Threat Reports, in: JNIC, 2022. 3

[27] Software Freedom Conservancy, Selenium, https://www.selenium.dev/ (2022). 6

[28] Python Software Foundation, Requests, https://github.com/psf/requests (2022). 6

[29] MaliciaLab, iocsearcher, https://github.com/malicialab/iocsearcher (2019). 7

[30] S. Zimmeck, S. M. Bellovin, Privee: An Architecture for Automatically Analyzing Web Privacy Policies, in: USENIX Security, 2014. 6

[31] R. Slavin, X. Wang, M. B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T. D. Breaux, J. Niu, Toward a Framework for Detecting Privacy Policy Violations in Android Application Code, in: International Conference on Software Engineering, 2016. 6

[32] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. M. S. M, S. M. Bellovin, J. R. Reidenberg, Automated Analysis of Privacy Requirements for Mobile Apps, in: NDSS, 2017. 6

[33] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, K. Aberer, Polisis: Automated Analysis and Presentation of Privacy Policies Using Deep Learning, in: USENIX Security, 2018. 6

[34] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, T. Xie, PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play, in: USENIX Security, 2019. 6

[35] Y. Shinyama, P. Guglielmetti, P. Marsman, pdminer.six, https://github.com/pdfminer/pdfminer.six (2019). 6

[36] L. Richardson, Beautiful Soup, https://beautiful-soup-4.readthedocs.io/en/latest/ (2022). 6

[37] Mozilla, Mozilla Readability.js library, https://github.com/mozilla/readability (2022). 6

[38] H. Hosseini, M. Degeling, C. Utz, T. Hupperich, Unifying Privacy Policy Detection, in: PoPETs, 2021. 6

[39] C. Kohlschütter, boilerpipe: Boilerplate Removal and Fulltext Extraction from HTML pages, https://github.com/kohlschutter/boilerpipe (2022). 6

[40] P. Gao, F. Shao, X. Liu, X. Xiao, Z. Qin, F. Xu, P. Mittal, S. R. Kulkarni, D. Song, Enabling Efficient Cyber Threat Hunting With Cyber Threat Intelligence, in: IEEE International Conference on Data Engineering, 2021. 8

[41] WebMoney, WebMoney - Universal Payment System, https://www.wmtransfer.com/ (2022). 9

[42] FIRST, Traffic Light Protocol (TLP). FIRST Standards Definitions and Usage Guidance - Version 1.0, https://www.first.org/tlp/docs/tlp-v1.pdf (2016). 9

[43] Mandiant, Tracking Malware with Import Hashing, https://www.mandiant.com/resources/tracking-malware-import-hashing (2014). 9

[44] V. L. Pochat, T. V. Goethem, S. Tajalizadehkhoob, M. Korczyński, W. Joosen, Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation, in: NDSS, 2019. 10

[45] G. Gomez, P. Moreno-Sanchez, J. Caballero, Watch Your Back: Identifying Cybercrime Financial Relationships in Bitcoin through Back-and-Forth Exploration, in: CCS, 2022. 11

[46] DaveCrim, Catastrophic backtracking in BACKSLASH_URL_RE, https://github.com/InQuest/python-iocextract/issues/52 (2021). 12

[47] F. Parolini, A. Miné, rat - redos abstract tester, https://github.com/parof/rat (April 2022). 13

[48] F. Parolini, A. Miné, Sound Static Analysis of Regular Expressions for Vulnerabilities to Denial of Service Attacks, in: International Symposium on Theoretical Aspects of Software Engineering, 2022. 13

[49] p1ngul1n0, Blackbird, https://github.com/p1ngul1n0/blackbird (2022). 15

[50] Twitter, Twitter API, https://developer.twitter.com/en/docs/twitter-api (2022). 15

[51] Mathias Bynens, In search of the perfect URL validation regex, https://mathiasbynens.be/demo/url-regex (2021). 16