https://doi.org/10.2352/EI.2025.37.4.MWSF-302

Cooking Spiders: Efficient OSINT with Chefs and Recipes

York Yannikos, Marc Leon Agel, Julian Heeger, Simon Bugert
Fraunhofer SIT / ATHENE, Darmstadt, Germany
(All authors have contributed equally)

Abstract

Social media, online forums, darknet marketplaces, and various other digital platforms are increasingly used or targeted by cybercrime. Therefore, open source intelligence (OSINT) has become an important aspect in digital forensics and cybercrime investigations: leveraging publicly available data on the Internet provides new information and offers insights into criminal behavior, patterns, and relationships. Many different tools and services exist to collect and extract data from websites for digital forensic investigations. These are often expensive and prone to errors when target websites change their structure or content.

In this paper we present MAMPF, a media acquisition and multi-processing framework for OSINT tasks. The framework is able to collect and extract data from various websites with easy extensibility and maintenance in mind. We show that our framework makes a self-hosted approach to efficient OSINT possible where a centralized core component is utilized in such a way that nodes performing crawling / scraping tasks no not require any maintenance at all. To describe our approach we use the analogy of a restaurant with chefs that prepare dishes following specific recipes.

Introduction

Open source intelligence (OSINT) has become a relevant part for digital forensics investigations dealing with cybercrime. OSINT helps understanding incidents and gaining insights from public information. However, many of the OSINT tools currently available are expensive and very often cloud-based, which can create concerns about data privacy and accessibility, especially for law enforcement agencies. New self-hosted, easily extensible tools are required that allow digital forensic investigators to customize their workflows and adapt functionality without being required to write program code, while maintaining control of the data.

As OSINT has become an integral part of cybercrime investigations and digital forensics, many different solutions are available in this area. The tools and services available for digital forensics investigators, law enforcement, or cybersecurity professionals provide functionality to collect and extract data from various sources in the Internet. However, they often come with like high purchase or running costs, very high maintenance requirements, or – especially for cloud-based services – lack of control regarding the collected (often sensitive) data.

Our objective was to develop and test a performant and efficient client-server architecture for public data collection and extraction that can run self-hosted, is easily extensible, and centralizes maintenance efforts in such a way that workers performing collection tasks rarely need code updates at all, even when target websites change.

Data Acquisition using OSINT

In past years the technology used to collect data from various sources has evolved significantly, enabling forensic investigators to extract and analyze public information efficiently. Web crawling and scraping are common techniques for collecting data from websites. Additionally, application programming interfaces (APIs) provided by platforms like X, Telegram, or Facebook facilitate the collection of structured data and often allow more specific searches for keywords, hashtags, or user mentions. These methods can be used to collect social media posts, news articles, blog posts, or other public information, in order to compile large datasets for further examination.

However, many websites do not offer public APIs and even implement counter-measures against web scraping such as CAPTCHAs. Also dynamic web content can pose an additional challenge for web scraping due to its reliance on client-side rendering. When a website utilizes JavaScript frameworks like React, Angular, or Vue.js, the initial website content presented to the user (or to a scraping tool) may only contain a minimal structure, while the relevant content is loaded asynchronously afterwards. Especially traditional web scrapers that typically download static content may not capture the dynamically loaded elements, therefore leading to incomplete data collections. Also, many websites implement *lazy loading*, where content is only loaded when the user scrolls down. Here, more advanced scraping techniques can be used that are based on headless browsers controlled by frameworks like Selenium or Playwright, which simulate user interactions and can execute JavaScript code to fully load and render all content before scraping.

Building an advanced scraper for a website typically requires a thorough analysis of the structure of the website, including its DOM and network requests made by client-side JavaScript. This analysis requires a deep understanding of web technologies and also the ability to quickly adapt the scraping logic to changes of the website over time. Both aspects are rarely part of the usual work of a forensic investigator who utilizes OSINT techniques.

Framework Overview and Architecture

In the following sections we describe MAMPF, a media acquisition and multi-processing framework for OSINT tasks. The aim of the development of MAMPF was to create a potentially long running system capable of collecting and extracting web data from various public web sources on external request. The desired data types that MAMPF should be able to process include structured data (such as product titles, object descriptions, timestamps, or prices from an online trading platform) as well as binary files (such as product images, videos, audio or document files).

We state four desired properties of such a system:

- *Resistance to changes in website structure at runtime*. When scraping web data from structured file formats such as HTML, data extraction techniques often rely on path expressions which specify the location of the desired data in the obtained files (for example XPath or CSS selector expressions for HTML or JSON Query expressions for JSON). These path expressions are highly dependent on the general structure of the files, such that data extraction processes frequently break when the website structure changes. This can happen at any time as website operators are typically unrelated third parties. Therefore, in our view it is highly desirable to enable the reconfiguration or adaption of web scraping processes at runtime in a flexible way, such that program code changes and deployments are not required in order to react on website structure changes.
- *Resilience to runtime errors*. Web data scraping takes place in complex network environments where connection or timeout errors are common, especially when extracting large sets of data in a short amount of time. Since web scraping tasks often consist of sequences of requests, where request parameters are dependent on results of previous requests, the web scraping processes should be able to recover from such intermediate request errors.
- *Flexibility*. Web data comes in various formats and different web services may offer different ways to obtain data. While certain web services offer Application Programming Interfaces (APIs) to enable access to their structured data, others might only embed the desired data in their HTML responses. Since the system should support the data collection and extraction for multiple different sources, it should be flexible enough to handle different file types and protocols.
- *Extensibility*. Since data collection tasks generally consist of a multitude of network requests followed by potential computationally expensive data transformations, it is desirable to execute them in parallel with an adjustable number of parallel data collection processes. In addition to horizontal scaling, the system should also support the extension to new data sources at runtime.

While the first two properties are considering two general challenges of web data collection and extraction, the latter two are specific design goals of MAMPF. The developed system provides an interface for the input of data collection jobs, i.e. OSINT tasks, which are formulated as predefined templates with corresponding input arguments. For example, a predefined template could be the extraction of all products, including titles, descriptions, prices, and images listed on eBay which can be found with a given search query. In this case, the search query is an additional input parameter that needs to be supplied by the requesting client. The requesting client then gains a job ID which can be used to query the status of the data collection job as well as the collected data once the job is finished.

Architecture

MAMPF is a distributed system composed of multiple worker nodes that perform data collection tasks and a centralized controller component that orchestrates the worker nodes. To explain the different components and the objects by which the components communicate, we employ the analogy of a restaurant: In this analogy, the centralized controller component can be thought of as the restaurant kitchen, which is responsible for accepting and queuing client dish orders, distributing them to kitchen chefs, and serving finished dishes back to the clients. Correspondingly, the worker nodes are analogous to the kitchen chefs who prepare the dishes and hand them back to the kitchen. The predefined templates mentioned previously can be thought of as recipes, which, together with additional input arguments, contain instructions about how to prepare a dish that is part of an order.

Restaurant Kitchen (Central Controller)

The restaurant kitchen acts as central controller in MAMPF and consists of five components, each responsible for a different part of the functionality of the controller:

The *public API* exposes an interface for clients to submit orders, check the status of orders and query extracted data of finished orders. When an order is submitted, the corresponding recipe is looked up in the *recipe book*, a database where up-to-date recipes of for several dishes are managed and stored. The ordered dishes are appended to an *order queue* from where they are distributed to the chefs acting as worker nodes. The chefs communicate their status and results back to the restaurant kitchen by using an *internal API*, which manages the life cycle of the chefs and the storage of the resulting data in a *database*.

Chefs (Worker Nodes)

A chef is a separate process that is responsible for cooking, i.e. preparing / processing *dishes*. Since each dish is defined by a recipe and additional input arguments (see below), a chef first has to specify a set of recipes which it is able to handle. When a chef connects to the restaurant kitchen, it announces its set of supported recipes such that only queued dishes suitable for the chef are routed to it.

Figure 1 shows the architecture of MAMPF with its components that are described in the following section.

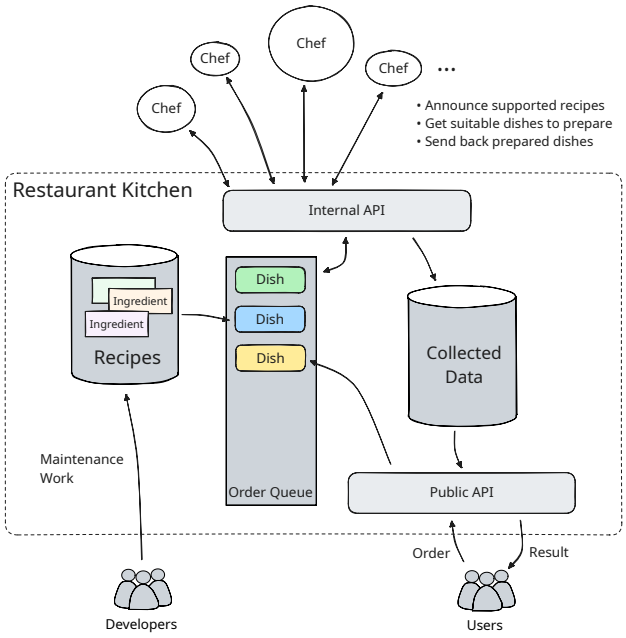


Figure 1. *Overview of the MAMPF architecture*

Kitchen Components

Besides the previously mentioned APIs and databases, MAMPF further includes components that are named using our restaurant analogy. These components are described in the following:

Recipe

A recipe is the main specification of a data collection task. It consists of one or multiple *ingredients* and defines how a dish can be prepared, e.g. how data can be collected or extracted from a target website. The ingredients a recipe consists of are structured as tree. Recipes are written in a domain-specific language.

Ingredient

An ingredient is a lower-level computation task. Each ingredient specifies a set of input parameters and include executable code that defines how to process the input parameters and return a result. For example, a "request" ingredient may define an input URL and perform an HTTP request to that URL, returning the HTML response.

Ingredients can also depend on other ingredients by specifying the required return type. When the implementation of an ingredient requests another ingredient to be computed, it may specify a context object which will be available during the execution of the other ingredient. This makes it possible to combine abstract ingredients in a useful way. For example, one could define an ingredient that requires a parsed DOM tree in the context object and an XPath expression as a parameter and returns the text node at the specified XPath. This ingredient can then be combined with another "request" ingredient to build a larger ingredient that requests an HTML page and returns the text at a specific location. The resulting tree of ingredient dependencies, up to the "root ingredient", defines a recipe (see above).

The execution of an ingredient has to be implemented in a chef (worker node). As a result, the set of recipes that a chef can handle is given by all recipes that only contain ingredients implemented in that chef. Ingredients can be implemented to handle specialized use cases, such as taking a screenshot of a website, performing a web requests using a proxy, or using the Tor network to access an onion service. Chefs can be deployed wherever they have network access to the internal API, such that they can meet the runtime requirements of those use cases.

Dish

A dish is defined as a recipe together with the input arguments of the root ingredient of the associated recipe. As such, it can be given to a chef which can start the computation by executing the root ingredient. In practice, an order queue system is used to queue up the dishes awaiting computation. When a dish is put into the order queue by the restaurant kitchen, the set of all ingredients is taken into account to find a suitable chef for this dish.

In the implementation of ingredients, chefs have access to the private API of the restaurant kitchen, such that they can save binary or structured data and even queue up new dishes. In this way, they can communicate back the results of the data collection to the kitchen. The ability to queue new dishes enables the chunking of large data collection tasks into multiple smaller dishes in a tree-like structure and a parallel execution of the dishes. Furthermore, complex data collection tasks may require multiple different ingredients for specialized use cases that are not all available on a single chef, which requires this chunking in order to supply them to suitable chefs.

Order

An order consists of one or many dishes and describes a complete OSINT task. For example, if several websites should be crawled and all found images and text data should be extracted and downloaded, the corresponding order would include a dish for each website, each consisting of the required ingredients to navigate the websites, download images, extract text data, etc.

Communication Flow

In order to process an OSINT task with MAMPF, a set of chefs must be deployed first. Chefs can be hosted and run in different locations as long as they are able to communicate with MAMPF's internal API. After starting their work, the chefs ask for the latest set of recipes, announce which kind of recipes they are able to cook, and then start watching the order queue. After the chefs are ready, a user can submit an OSINT task, i.e. put an order on the order queue. This order could include a set of websites to be crawled as well as information about which specific data should be extracted and collected – i.e. a number of dishes.

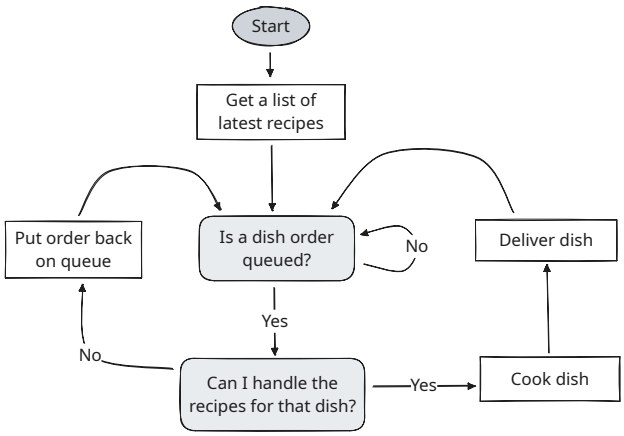


Figure 2. *Communication flow of chefs in MAMPF*

Figure 2 shows the communication flow of the chefs: They continuously watch the order queue to fetch new dishes for cooking, i.e. to perform the actual crawling / scraping tasks on a website. After a chef finished cooking, i.e. collecting and extracting the data for the dish, it delivers the dish by sending the data to the database using the internal API. In the database the collected data is deduplicated and stored, e.g. for subsequent processing or analyses.

Writing Recipes

We developed a domain-specific language (DSL) to write recipes in MAMPF. Recipes are stored in JSON format in the recipe book, i.e. the database containing all recipes available for chefs. Figures 3 and 4 show example recipes written in our DSL including different ingredients.

In order collect data from a new website we can use the DSL to write new recipes containing the relevant ingredients. This

```
{
  "id": 0,
  "$schema": "../schema.json",
  "name": "DOWNLOAD_SIMPLE",
  "version": 1,
  "body": {
    "input_params": [{
        "name": "url",
        "type": "string"
    }],
    "protocol": {
      "type": "DOWNLOAD",
      "url": "{input[url]}",
      "meta": [{
        "key": "url",
        "value": "{input[url]}"
      }]
    }
  }
}
```

Figure 3. *Example recipe to download a file from a given input URL*

```
{
  "id": 0,
  "$schema": "../../schema.json",
  "name": "TARGET_ITEM",
  "version": 1,
  "body": {
    "input_params": [
      { ... }
    ],
    "protocol": {
      "type": "CRAWL_PAGE",
      "request": {
        "type": "REQUEST",
        "url": "{input[href]}",
        "parse": "next_data"
      },
      "sub": [
        {
          "type": "DOWNLOAD_IMAGES",
          "image_links": {
            "type": "JQ",
            "q": "try(.props.pageProps.
                ↪ rawGallery.gallery[].images
                ↪ [] | last(to_entries[]).
                ↪ value.url)",
            "ret": "ALL"
          },
          "image_download": {
            "type": "INVOKE",
            "recipe_name": "DOWNLOAD_SIMPLE",
            "recipe_version": 1,
            "args": [{ "key": "url", "value":
                ↪ "{image_href}" }]
          },
          ...
        }
      ]
    }
  }
}
```

Figure 4. *Excerpt from an example recipe including ingredients to download images extracted using JSON Query*

could involve writing XPath expressions, JSON queries, or CSS selectors as well as additional website-specific content handling like pagination, or downloaders for multimedia data. To extend functionality within MAMPF, we can also utilize additional microservices, e.g. for automated CAPTCHA solving or for sending web requests via proxies.

Despite the required work to figure out how the relevant content of a new website could be extracted for data collection, this simplifies maintenance significantly within MAMPF: The chefs hardly ever need program code changes – all they have to do is fetch a list of the latest available recipes from the recipe book to continue working.

Evaluation Scenario

To test our approach we used MAMPF in the ANCHISE project (`www.anchise.eu`), where we conducted research regarding the protection of cultural heritage against looting and illicit trafficking. We defined recipes for several different platforms that sell or trade cultural goods. Using the recipes we successfully adapted the framework to crawl platforms like eBay or Catawiki (`www.catawiki.com`) searching for stolen cultural goods.

Figure 5 shows the adapted MAMPF architecture for the ANCHISE project. We added a temporary section to our database in the restaurant kitchen and built an interface to communicate with an external service. The external service included another database curated by museums and law enforcement, that included a list of stolen objects to search for.

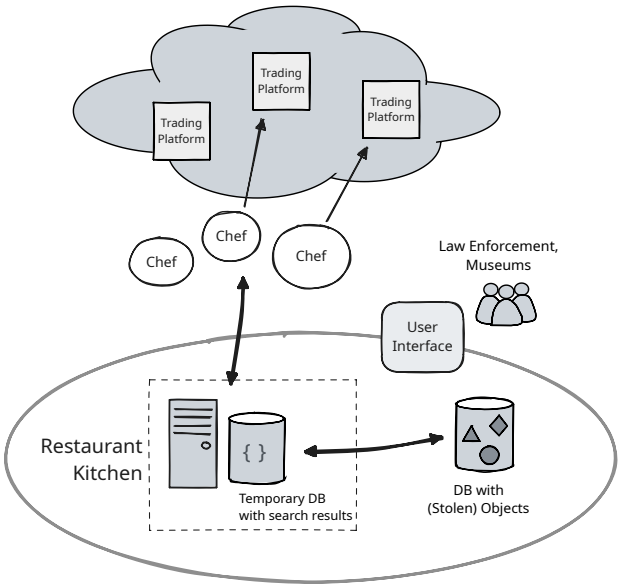


Figure 5. *Overview of the MAMPF architecture applied in the ANCHISE project for evaluation*

The communication flow between MAMPF and the parties involved in ANCHISE was as follows:

1. Museums or law enforcement agencies uploaded information about stolen objects (cultural goods), i.e. text descriptions, search keywords, and images.
2. An OSINT task was defined to search several platforms for the stolen objects.

3. The task was submitted to MAMPF's order queue.
4. MAMPF processed the order by collecting data about relevant objects from the platforms.
5. The data was then analyzed externally to determine whether or not the found objects were similar to the recorded stolen objects (based on image similarity).
6. If similar objects were found, their data was stored for further action – otherwise, the collected data was deleted.

We repeated steps 2–6 for several days to monitor the target platforms for new object postings. While we tested our framework, eBay changed their site layout slightly in such a way that our chefs would sometimes provide wrong results while crawling the website. However, we could adapt quickly (i.e. within minutes) by modifying the corresponding recipes and pushing them to the recipe book. After the chefs updated the recipes they could continue to successfully collect data from eBay.

A detailed description about the adaptions, results, and performance of our data collection within the ANCHISE project is given in [6].

Related Work

Recent efforts have been made to obtain distributed systems of web crawling agents. In [3], the authors present a distributed web crawler capable of indexing unstructured data from social platforms such as Reddit and Hacker News. While this approach is well suited for capturing unstructured data, the proposed system does not handle the extraction of structured data and the requirements for modularity at runtime resulting from changes to data structures. In another work, Khder et al. [5] provide a comprehensive introduction to the concepts of web crawling and web scraping, with a discussion of the legal and ethical implications of these practices. The authors proceed with an examination of the different applications of web crawling in fields such as data science and cybersecurity. They argue that a more clearly defined global legal framework for web scraping is required, that balances information privacy rights with business interests.

In [4] the authors provide an overview about the use of public data and free OSINT tools in computer security incident response teams (CSIRTs). Based on an online survey and interviews with 25 participants from 13 different CSIRTs, the authors find that public data is commonly collected and used in CSIRTs. They state that main challenges in using OSINT are lacking systematic processes and suitable frameworks for the validation and evaluation of free tools as well as the general availability of useful public data sources and OSINT tools. The authors conclude that more research is required for a more efficient and effective use of public data and free tools.

Regarding the combination of OSINT with AI technology, the authors of [8] review the current state of OSINT techniques, highlighting their limitations in practical applications. The paper underscores the importance of integrating AI, machine learning, and deep learning to automate intelligence gathering, enhancing OSINT applications in cyber defenses, social media analysis, and digital forensics. The authors conclude that the integration of AI improves the detection of cybercrime, helps with culprit identification, and supports counter-terrorism efforts. In another study regarding the topic, Browne et al. [1] conduct a review of current research on the combination of OSINT and AI algorithms. The research questions posed by the authors are wide-ranging, encompassing the detection of trends in the mentioned fields, the deployment of AI systems at specific stages of the OSINT process, and the data sources utilized.

Hwang et al. [2] give an introduction to OSINT, distinguishing it from other intelligence collection methods such as HUMINT and TECHINT, and outlining their respective military applications. They analyze security threats to OSINT, including data privacy breaches, where attackers exploit personal information for financial gain, and data forgery, which compromises authenticity in contexts like fake news. Finally, they examine security requirements and access control principles for managing collected OSINT data.

In previous works we presented our own findings from OSINT research in the darknet where we monitored darknet marketplaces or single vendor shops to gain insights about their activity. We scraped several marketplaces like Dream Market, Wall Street Market, or White House Market and analyzed offered goods, buyer and seller activity, or transaction amounts [11, 10]. By continuously monitoring specific platforms we could also identify a marketplace that runs an elaborated scam by faking vendor and buyer activity while offering no goods at all [7]. To access several marketplaces in the Tor network we implemented automated captcha solvers and provided an overview about used captcha types and suitable solvers in [9].

Conclusion

In this paper we introduced MAMPF, a media acquisition and multi-processing framework for OSINT tasks. Our goal was to develop a general-purpose framework that can run self-hosted, is easily extensible through the use of a domain-specific language, and uses a centralized maintenance approach with self-updating workers and high scaling flexibility. We chose to describe the rather complex framework using a restaurant analogy to illustrate the core principles and design ideas more easily.

After the implementation we evaluated MAMPF in a scenario within the ANCHISE project where we successfully collected data from various different target platforms on the Internet for a subsequent analysis. We found that our framework fulfilled all of the four desired properties we formulated: resistance, resilience, flexibility, and extensibility.

In previous publications we presented research regarding OSINT on specific target platforms, e.g. on single darknet marketplaces, using techniques to automatically evade anti-crawling mechanisms implemented by the platform operators. We are planning to integrate our research results from these projects into MAMPF to provide additional flexibility for the framework, e.g. by implementing microservices for captcha solving. This would further enable us to use MAMPF to crawl data from darknet marketplaces more efficiently. Additionally, we are currently testing and optimizing the performance of the framework when crawling and scraping data from different sources at the same time.

Acknowledgments

This research was conducted in the FROST+ML project funded by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

The evaluation has also been supported by the Horizon Europe project ANCHISE (Applying New solutions for Cultural Heritage protection by Innovative, Scientific, social and economic Engagement). This project has received funding from the European Union's Horizon Europe Framework Programme under grant agreement no. 101094824.

References

[1] Thomas Oakley Browne, Mohammad Abedin, and Mohammad Jabed Morshed Chowdhury. A systematic review on research utilising artificial intelligence for open source intelligence (OSINT) applications. *International Journal of Information Security*, 23(4):2911–2938, 2024.

[2] Yong-Woon Hwang, Im-Yeong Lee, Hwankuk Kim, Hyejung Lee, and Donghyun Kim. Current Status and Security Trend of OSINT. *Wireless Communications and Mobile Computing*, 2022(1), 2022.

[3] Donovan Jenkins, Lorie M. Liebrock, and Vince Urias. Designing a Modular and Distributed Web Crawler Focused on Unstructured Cybersecurity Intelligence. In *2021 International Carnahan Conference on Security Technology (ICCST)*, pages 1–6, 2021.

[4] Sharifah Roziah Binti Mohd Kassim, Shujun Li, and Budi Arief. How national CSIRTs leverage public data, OSINT and free tools in operational practices: An empirical study. *Cyber Security: A Peer-Reviewed Journal*, 5(3):251–276, 2022.

[5] Moaiad Ahmad Khder. Web scraping or web crawling: State of art, techniques, approaches and application. *International Journal of Advances in Soft Computing & Its Applications*, 13(3), 2021.

[6] Huajian Liu, York Yannikos, Julian Heeger, Simon Bugert, Waldemar Berchtold, and Martin Steinebach. Automated Monitoring of Stolen Cultural Artifacts on OnlineMarketplaces. In *IS&T Electronic Imaging, Mobile Devices and Multimedia: Enabling Technologies, Algorithms, and Applications 2025*. Society for Imaging Science and Technology, 2025.

[7] Florian Platzer and York Yannikos. Trust Assessment of a Darknet Marketplace. In Jia Hu, Geyong Min, Guojun Wang, and Nektarios Georgalas, editors, *22nd IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2024, Exeter, UK, November 1-3, 2023*, pages 1806–1813. IEEE, IEEE, 2023.

[8] Ashok Yadav, Atul Kumar, and Vrijendra Singh. Open-source intelligence: a comprehensive review of the current state, applications and future perspectives in cyber security. *Artificial Intelligence Review*, 56(11):12407–12438, 2023.

[9] York Yannikos and Julian Heeger. Captchas on Darknet Marketplaces: Overview and Automated Solvers. *IS&T Electronic Imaging, Media Watermarking, Security, and Forensics 2024*, 36:1–6, 2024.

[10] York Yannikos, Julian Heeger, and Martin Steinebach. Scraping and analyzing data of a large darknet marketplace. *Journal of Cyber Security and Mobility*, pages 161–186, 2023.

[11] York Yannikos, Annika Schäfer, and Martin Steinebach. Monitoring product sales in darknet shops. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES '18, page 59, New York, NY, USA, 2018. ACM, Association for Computing Machinery.

Author Biography

York Yannikos *received his Diplom (equiv. Master's degree) in computer science from the University of Rostock, Germany in 2008. Since then he has been working as research associate in the Media Security and IT Forensics department at the Fraunhofer Institute for Secure Information Technology (SIT) and at the National Research Center for Applied Cybersecurity (ATHENE) in Darmstadt, Germany. His research interests include darknet marketplaces, open source intelligence, and digital forensic tool testing.*

Marc Leon Agel *is a research associate in the Media Security and IT Forensics department at the Fraunhofer Institute for Secure Information Technology (SIT) and at the ATHENE National Research Center for Applied Cybersecurity in Darmstadt, Germany. He holds a Master's degree in physics from the Technical University of Darmstadt and his research interests include open source intelligence, digital forensics and machine learning.*

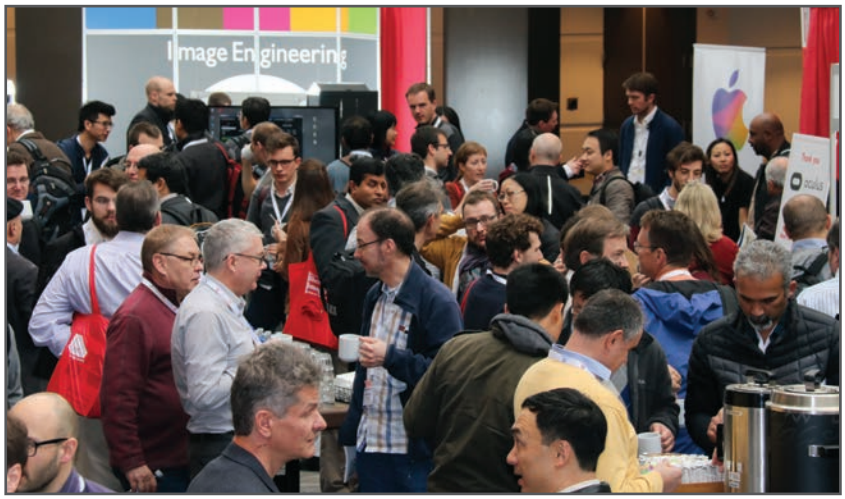
Julian Heeger *is a research associate in the Media Security and IT Forensics department at the Fraunhofer Institute for Secure Information Technology (SIT) and a researcher at the National Research Center for Applied Cybersecurity (ATHENE) in Darmstadt, Germany. He holds a Master's degree in IT security from the Technical University of Darmstadt.*

Simon Bugert *received his Master's degree in computer science from the Technical University of Darmstadt, Germany in 2021. Since then he has been a research associate in the Media Security and IT Forensics department at the Fraunhofer Institute for Secure Information Technology (SIT) and at the ATHENE National Research Center for Applied Cybersecurity.*

JOIN US AT THE NEXT EI!

electronic IMAGING

Imaging across applications . . . Where industry and academia meet!



• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

ADVANCING IMAGING SCIENCE 75 1947 IS&T 2022 imaging.org