Katti: An Extensive and Scalable Tool for Website Analyses

Florian Nettersheim
{firstname.lastname}@bsi.bund.de
Federal Office for Information Security
Bonn, Germany

Stephan Arlt
{firstname.lastname}@bsi.bund.de
Federal Office for Information Security
Bonn, Germany

Michael Rademacher
{firstname.lastname}@fkie.fraunhofer.de
Fraunhofer FKIE
Bonn, Germany

Florian Dehling
{firstname.lastname}@h-brs.de
University of Applied Sciences Bonn-Rhein-Sieg
Sankt Augustin, Germany

ABSTRACT

Research on web security and privacy frequently relies on tools that analyze a set of websites. One major obstacle to the judicious analysis is the employment of a rock-solid and feature-rich web crawler. For example, the automated analysis of ad-malware campaigns on websites requests crawling a vast set of domains on multiple real web browsers, while simultaneously mitigating bot detections and applying user interactions on websites. Further, the ability to attach various threat analysis frameworks lacks current tooling efforts in web crawling and analyses.

In this paper we introduce Katti, which overcomes several of today's technical hurdles in web crawling. Our tool employs a distributed task queue that efficiently and reliably handles both large crawling and threat analyses requests. Katti extensively collects all available web data through an integrated person-in-the-middle proxy. Moreover, Katti is not limited to a specific use case, allowing users to easily customize our tool to their individual research intends.

CCS CONCEPTS

• **Information systems → Web mining**.

KEYWORDS

web, website, crawling, analysis, analyses

ACM Reference Format:
Florian Nettersheim, Stephan Arlt, Michael Rademacher, and Florian Dehling. 2023. Katti: An Extensive and Scalable Tool for Website Analyses. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3543873.3587351

1 INTRODUCTION

The main component of research efforts in web security and privacy typically consists of a tool that crawls and analyzes a set of websites [1, 10]. However, the automated collection of representative data from vast amounts of websites is no trivial task. For instance, dynamic content delivery results in different resources being loaded depending on the used web browser [1], and publishers' efforts to prevent automated queries of their content can lead to incomplete or incorrect data sets [7, 9]. Further, some analyses, e.g., the discovery of malicious web content such as ad-malware campaigns, require mimicking user interaction, like clicking on ad banners.

Researchers often strive to tackle these challenges by implementing highly specialized crawling tools to meet their specific requirements, which impedes the repetition of the experiments conducted or even makes it impossible in some cases [1].

In this paper, we present Katti, a tool that addresses common technical hurdles in web crawling and analyses. It is built both feature-rich, i.e., not limited to web security and privacy, and rock-solid based on industry-proven frameworks (e.g., Docker, Celery, RabbitMQ). Katti supports multiple real web browsers like Chromium/Chrome, Firefox, and Edge, allows mimicking user interactions, e.g., to circumvent bot detection measures or to crawl even deeper website content while extensively collecting almost all layer 7 data. Further, Katti employs a so-called distributed task queue, which allows performing long-running crawling requests. Besides collecting data, Katti allows the attachment of multiple threat analyses for basic Internet protocols (e.g., DNS, TLS) and crowd-based approaches (e.g., Shodan, Google Safe Browsing). Offering a robust framework that allows easy customization, Katti is not limited to a specific use case, but allows users to adapt the tool to their research and to act as a Swiss Army knife.

In the next section, we give a brief overview of Katti's architecture, outlining both the usage and the extensibility of our tool. We then present two usage scenarios in a brief evaluation in Section 3, which demonstrates how Katti scales with available computing power and how multiple real web browsers can be used. Section 4 discusses selected aspects and outlines possible future work. We provide a short conclusion in Section 5 and an overview of related work in Section 6.

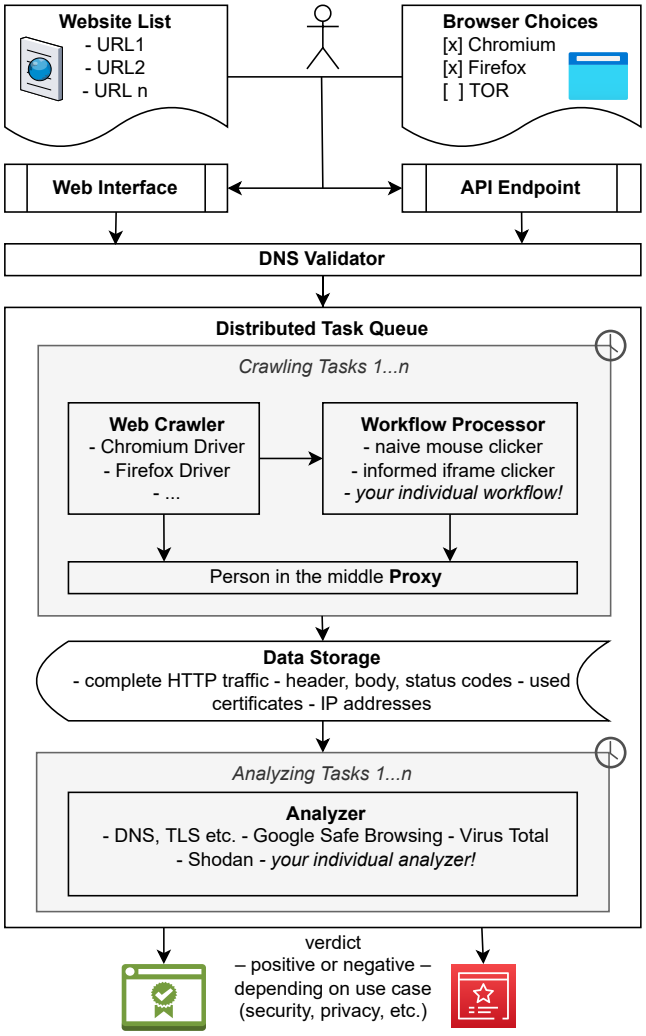
2 ARCHITECTURE

This section provides a brief overview of Katti's architecture, which is also visualized in Figure 1. We designed Katti with a strict modular concept in mind to encourage sustainable and fast development cycles. In addition, the architecture allows scaling Katti based on the user's needs by utilizing the concept of a distributed task queue.

WWW '23 Companion, April 30-May 4, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9419-2/23/04…$15.00
https://doi.org/10.1145/3543873.3587351

Figure 1: Architecture of Katti



2.1 Input and Control

A user defines a list of websites (domains, URLs) and a set of web browsers to be considered – either using a web interface or an API endpoint. The web interface is useful for an ad-hoc check of a specific URL. The API endpoint is in particular useful for automated tasks, e.g., if the user wants to check a large set of websites. Before Katti enters the main processing stage of crawling and analyzing the provided URLs, a DNS validator confirms whether A and AAAA records of the websites are resolvable. If not, i.e., the DNS resolver returns NXDOMAIN, Katti discards the corresponding URL.

2.2 Distributed Task Queue

Once the given URLs are resolved, Katti launches a crawling task for each URL and manages all tasks in a distributed task queue. Katti is able to process a large set of websites and accommodates a horizontally scalable system that allows parallel execution of different crawling requests. This is mainly achieved by using the Celery framework[1]. The basic structure of the framework corresponds to the producer-consumer pattern. First, a client places a task in a queue. Then, a worker extracts and executes this task. The communication is realized with RabbitMQ as a message broker. A Redis database is used to store the results of the individual tasks.

2.2.1 Crawling Tasks. Katti employs real web browsers for crawling websites, namely Chromium/Chrome, Firefox, Edge and even TOR. In particular, for communicating and interacting with websites, we use an adapted version of Selenium wire[2]. Selenium wire allows us to record and manipulate the entire HTTP traffic. It is implemented by a person-in-the-middle HTTP proxy, which extensively collects all available web data.

The workflow processor allows executing a predefined set of interactions. By default, Katti supports a "naive mouse clicker" in order to mitigate bot detection measures from large CDNs (content delivery networks), and an informed iframe clicker. The latter is in particular useful to analyze online advertisements, which are typically embedded in iframes.

2.2.2 Data Storage. All data recorded by the proxy is stored in a MongoDB cluster[3]. We stress that compared to other web crawling tools (cf. Section 6), Katti fetches and stores the entire HTTP traffic, including TLS certificates and IP addresses. In addition, the web browser's internal request ID, frame ID and tab ID are stored for each request. This enables us to group the recorded requests. Furthermore, a screenshot of the loaded website is saved. This heap of data makes it easy to apply custom use cases for analyzing specific elements of websites such as online advertisements.

2.2.3 Analyzing Tasks. Once all HTTP traffic is stored, Katti allows executing a predefined set of analyzing tasks. By default, Katti supports a variety of fundamental threat analyses, such as DNS checks (e.g., Google DNS, Cloudflare Security, Quad9), TLS checks (e.g., a customized version of sslythe), Google Safe Browsing, Virus Total, and Shodan. Further, Katti allows the user to add more analyzing tasks depending on the pursued use case. Finally, the analyzer can return a verdict for each analyzed website.

3 EVALUATION

This section provides a preliminary evaluation of two selected main features of Katti. First, we show that Katti is scalable to the user's needs by dynamically increasing the processing power during the run of an experiment. Second, we show that Katti is able to orchestrate multiple real-world browser by comparing the corresponding processing times.

3.1 Scalability

In order to show the scalability of Katti, we conducted an experiment based on the following scenario: A user wants to crawl and store all web data for a list of URLs for further analyzing steps. Due to an emerging security threat, this task is time critical. The user starts Katti by providing a list of URLs via its API endpoint. However, the user quickly notices that more processing power is

[1] https://docs.celeryq.dev
[2] https://github.com/wkeeling/selenium-wire
[3] https://www.mongodb.com

needed to complete the task in time. Therefore, more processing power is added dynamically without interrupting or interfering with the current state of the task.

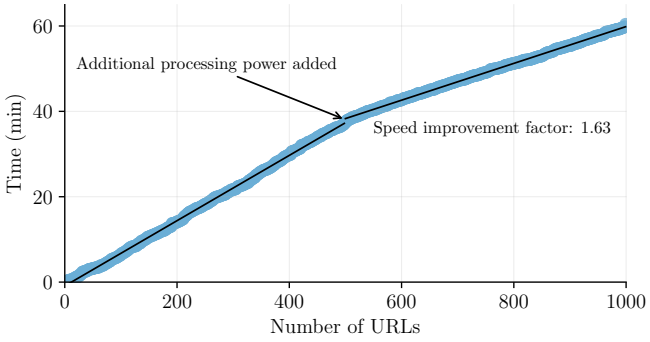
Figure 2 provides a visualization of an experiment corresponding to this scenario using a duplex Internet connection of 100 MBit. Here, the experiment starts with one virtual machine[4], which runs Katti to crawl and store the web data of the first 1,000 URLs of the Tranco list [11]. After 500 URLs, a second virtual machine is added with the same processing power by executing the following command.

```
celery -A KattiApp worker -Q crawling_crawling --
    concurrency=8
```

This decreases the time for the next 500 URLs significantly by a factor of 1.63. If one would stick to a single machine, the entire crawling process would need 75 minutes.

For brevity, in this paper we narrow down the number of domains to 1,000 and the number of machines to 2. We assume that Katti performs more efficient if three or more machines are attached. We will consider this evaluation in a follow-up study.

Figure 2: System time (crawling and data storage) for 1,000 URLs with dynamic adaption of the processing power.



3.2 Multi-Browser

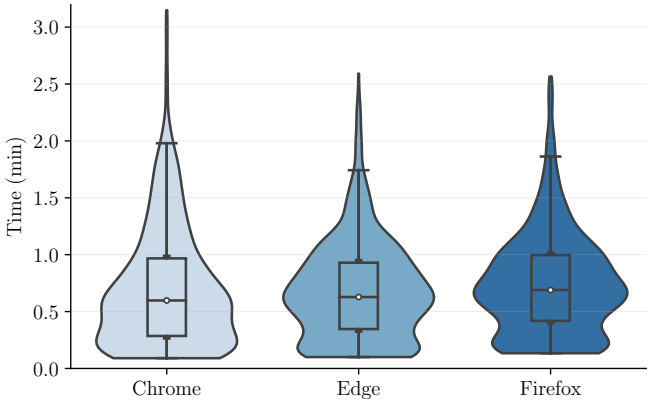
In order to show the multi-browser support of Katti we conducted a second experiment based on the following scenario. A security analyst receives an indication that a newly emerging ad-malware campaign targets the usage of specific web browsers. In addition, the analyst receives a list of 1,000 URLs which are used to deliver these malicious online advertisements. By using Katti, the analyst can crawl and store the web data using a direct execution of widely used real-world web browsers. This provides a more realistic result compared to simply changing the user-agent string. Figure 2 visualizes the metadata for such an experiment. Here, the three web browser (Chrome, Edge, Firefox all in the current version) are used to access the top 1,000 sites of the Tranco list [11]. To select a set of web browsers, the API endpoint of Katti can be called using the following legible JSON data:

```
{   'chrome': true,
    'edge': true,
    'firefox':true,
    'urls': <list[URLs]>
    'analyses': {
        'dns': true,
        'ssl': true,
        'shodan': true,
        'virus_total': true}}
```

Note that this JSON snippet also allows the user to pass a choice of built-in analyzers to Katti. Among other selected analyzers, Katti calls VirusTotal to validate whether the crawled web content is malicious. This can lead to a specific verdict of the entire process (see Figure 1).

Figure 3 shows the distribution of the system time of the conducted crawls and data storage. Note that the median of our results amounts to 0.6 minutes. One can argue that this time is relatively long for a single website crawl. However, our evaluation does not consider the pure crawling, but also the time pulling up the depending infrastructure in a clean state, e.g., storing web data and assemble screenshot from websites. We will discuss typical bottlenecks and limitations to our approach in the forthcoming section.

Figure 3: System time (crawling and data storage) for 1,000 URLs and three well-known web browsers utilized by Katti.



4 DISCUSSION AND FUTURE WORK

By analyzing the experiments presented in the previous section in depth, we identified two unexpected factors that influence the time it takes to complete and store the data of a crawl. The first factor is the time required to start an individual Docker container. We noticed that the startup takes an average of 3.84 seconds on our machines. Since we start an individual container for each crawl, this aggregates to a relevant factor and represents a known challenge[5]. Thus, part of the future work is evaluating if this time can be reduced by adapting Docker or to evaluate if alternatives such as Podman containers are better suited for Katti.

[4]Dell PowerEdge R730, 16 Cores, 16 GB RAM

[5]https://github.com/moby/moby/issues/42096

The second factor that impacts our evaluation is the time needed to create a full-page screenshot. On average, it takes 8.23 seconds to create a single screenshot. However, strong fluctuations can be observed here, which are significantly related to the structure of websites. For instance, websites incorporate so-called infinity pages, which lead to more computing time to assemble single screenshots into one screenshot. We thus plan to evaluate whether the process of creating screenshots can be accelerated.

In addition to the mentioned factors above, the current experiment is conducted on a server which uses HDDs instead of SSDs. We assume that this also influenced the average time to store the crawling data in the database. While further improving the performance of Katti is one of our main goals, we also aim to make use of the strict modular architecture and add additional features, e.g., a live-browsing feature that empowers users to interact with websites manually during analyses.

5 CONCLUSION

We introduced Katti, a novel tool that overcomes several of today's technical hurdles in web crawling and analyses. A distributed task queue handles multiple real web browsers, allows the integration of user interactions, extensively stores all available web data and enables researchers to attach various threat analyzing frameworks. Regarding scalability, Katti shows its potential on crawling large sets of domains. Katti can add more computing power in an ad-hoc style if needed. Our tool is designed to work as a Swiss Army knife. It is not limited to a single use case, but is customizable to different research domains that need robust infrastructure and an extensive pool of web data.

Artefacts (i.e., code, docs, and demos) are available on GitHub: https://github.com/BSI-Bund/Katti

6 RELATED WORK

Web crawlers are integral tools in research on web security and privacy [1, 10]. Ahmad et al. [1] proposed the classification of tools used into application-layer protocol crawlers based on Wget or cURL, browser-layer crawlers commonly based on test automation frameworks like selenium or puppeteer, and user-layer crawlers like OpenWPM [4] and TorBC [8] (deprecated), focusing on mimicking user interaction. Since modern websites are predominantly based on JavaScript and dynamic resource loading, content-focused research is usually done using browser-layer or user-layer crawlers [1].

Generating data sets that represent average web browsing impressions of users is not limited to but hardly affected by the selection of the crawling tool [10]. For example, crawling websites for scientific purposes contradicts website operators' endeavors to protect their sites from automated attacks and thus block anomalous traffic [7, 9]. Moreover, it has already been shown that some operators use cloaking strategies to hide malicious content from being detected by automated crawlers [5]. This result in data sets being obtained by a crawler that does not represent real web experience and, in the worst case, misses data that the underlying research is searching for [10]. To minimize the risk of being uncovered as an automated crawler by a website operator, the literature recommends using user-targeted tools like crawlers based on real web browsers [1, 3, 10].

Respecting this requirement, the crawling framework OpenWPM [4] has been frequently used in security research [2, 6, 13]. Even though it has been designed to be used with various web browsers, it requires a browser specific extension to provide, e.g., JavaScript instrumentation, which is only available for Firefox.

Limiting crawls to one specific web browser can also lead to incomplete data sets. Comparing Firefox and Chromium based web browsers, Ahmad et al. [1] have shown that the prominence of third-party trackers differs depending on the type of web browser used for crawling. Moreover, Li et al. [12] have observed malvertising being served only when requesting a website with a specific user-agent string, i.e., a specific type of web browser.

Recently, Cassel et al. [3] introduced the crawling framework OmniCrawl focusing on multi-browser support, including mobile platforms, while minimizing the risk of being identified as a bot. To do so, they make use of native web browsers in combination with a man-in-the-middle proxy to record traffic and inject JavaScript instrumentation. However, omitting potentially detectable web browser automation tools, such as selenium, OmniCrawl does not provide the ability to simulate user interaction like clicking on links or filling out forms.

REFERENCES

[1] Syed Suleman Ahmad, Muhammad Daniyal Dar, Muhammad Fareed Zaffar, Narseo Vallina-Rodriguez, and Rishab Nithyanand. 2020. Apophanies or Epiphanies? How Crawlers Impact Our Understanding of the Web. In *Proceedings of The Web Conference 2020*. ACM, 271–280.

[2] Stefano Calzavara, Tobias Urban, Dennis Tatang, Marius Steffens, and Ben Stock. 2021. Reining in the Web's Inconsistencies with Site Policy. In *Proceedings 2021 Network and Distributed System Security Symposium*. Internet Society.

[3] Darion Cassel, Su-Chin Lin, Alessio Buraggina, William Wang, Andrew Zhang, Lujo Bauer, Hsu-Chun Hsiao, Limin Jia, and Timothy Libert. 2022. OmniCrawl: Comprehensive Measurement of Web Tracking With Real Desktop and Mobile Browsers. *Proceedings on Privacy Enhancing Technologies* 2022, 1 (Jan. 2022).

[4] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Vienna Austria, 1388–1401.

[5] Luca Invernizzi, Kurt Thomas, Alexandros Kapravelos, Oxana Comanescu, Jean-Michel Picod, and Elie Bursztein. 2016. Cloak of Visibility: Detecting When Machines Browse a Different Web. In *2016 IEEE Symposium on Security and Privacy (SP)*. 743–758. ISSN: 2375-1207.

[6] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2021. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE.

[7] Gregoire Jacob, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. 2012. PUBCRAWL: protecting users and businesses from CRAWLers. In *Proc. of the 21st USENIX conference on Security symposium (Security'12)*. 25.

[8] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. Association for Computing Machinery, 263–274.

[9] Jordan Jueckstock and Alexandros Kapravelos. 2019. VisibleV8: In-browser Monitoring of JavaScript in the Wild. In *Proceedings of the Internet Measurement Conference*. ACM, Amsterdam Netherlands, 393–405.

[10] Jordan Jueckstock, Shaown Sarker, Peter Snyder, Aidan Beggs, Panagiotis Papadopoulos, Matteo Varvello, Benjamin Livshits, and Alexandros Kapravelos. 2021. Towards Realistic and ReproducibleWeb Crawl Measurements. In *Proceedings of the Web Conference 2021*. ACM, 80–91.

[11] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *26th Annual Network and Distributed System Security Symposium (NDSS '19)*.

[12] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. 2012. Knowing your enemy: understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security (CCS '12)*. Association for Computing Machinery, New York, NY, USA, 674–686.

[13] Valentino Rizzo, Stefano Traverso, and Marco Mellia. 2021. Unveiling Web Fingerprinting in the Wild Via Code Mining and Machine Learning. *Proceedings on Privacy Enhancing Technologies* (2021).