Journal of the Brazilian Computer Society, 2026, 32:1, doi: 10.5753/jbcs.2026.5372

Identification of Services and Devices for Enhancing Vulnerability Analysis

Lucas M. Ponce [**Universidade Federal de Minas Gerais** | ***lucasmsp@dcc.ufmg.br***]
Indra Ribeiro [**Universidade Federal de Minas Gerais** | ***indramatsiendra@dcc.ufmg.br***]
Etelvina Oliveira [**Universidade Federal de Minas Gerais** | ***etelvinaoliveira@dcc.ufmg.br***]
Ítalo Cunha [**Universidade Federal de Minas Gerais** | ***cunha@dcc.ufmg.br***]
Cristine Hoepers [**Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança** | ***cristine@cert.br***]
Klaus Steding-Jessen [**Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança** | ***jessen@cert.br***]
Marcelo H. P. C. Chaves [**Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança** | ***mhp@cert.br***]
Dorgival Guedes [**Universidade Federal de Minas Gerais** | ***dorgival@dcc.ufmg.br***]
Wagner Meira Jr. [**Universidade Federal de Minas Gerais** | ***meira@dcc.ufmg.br***]

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais (UFMG). Av. Pres. Antônio Carlos, 6627, Belo Horizonte, MG, 31270-901, Brazil.

Received: 24 December 2024 • **Accepted:** 26 September 2025 • **Published:** 02 April 2026

Abstract The identification of services and devices is an important step in vulnerability analysis, responsible for listing all assets that should be scanned for vulnerabilities. In recent years, this task has become increasingly complex due to the proliferation of new types of services and Internet-connected devices, such as IoT devices. In this context, search engines like Censys and Shodan have become popular tools, often used in one or more stages of the process for scanning network-accessible vulnerabilities. However, while these tools are capable of probing numerous devices worldwide, the information they process is often incomplete, primarily due to the challenge of keeping pace with the rapid creation of new applications. This paper introduces our solution for efficient service enumeration based on fingerprint matching, which can complement existing information about scanned devices. Our solution is highly efficient as it leverages the responses from connections established by search engines during their probing as input data, eliminating the need for additional scans. Furthermore, our processing engine is optimized and capable of processing data in parallel. To validate our solution, we compared the information obtained by our framework with that provided by Shodan. Overall, we were able to identify a larger number of services and devices. For instance, our approach increased the identification of services such as operating systems by 1.6 times and hardware information by up to 14 times. Additionally, we present two use cases demonstrating how our framework can assist in vulnerability analysis by providing more accurate and detailed information.

Keywords: Fingerprint matching, Device identification, Search engine, Vulnerability analysis, Shodan

1 Introduction

Vulnerability analysis consists of identifying and assessing flaws and potential threats to the security of a system that can be exploited by compromising the integrity, availability, or confidentiality of information [Imperva, 2023]. Typically, vulnerability analysis involves four steps: (*i*) identifying and enumerating the devices to be assessed, also known as network assets; (*ii*) scanning the enumerated assets to discover services and their potential vulnerabilities; (*iii*) assessing and prioritizing the identified risks; and (*iv*) creating reports and addressing the risks. In this context, the use of search engines has become a popular approach, not only for identifying devices and services connected to the Internet but also for assisting in the steps of identifying and prioritizing vulnerabilities [Markowsky and Markowsky, 2015; Albataineh and Alsmadi, 2019].

Nowadays, several search engines that discover devices connected to the internet are available on the market, such as Censys, Shodan, and Zoomeye.[1] These tools scan the Internet for devices accessible through the network, collecting information about the identified device, services and open ports. Although tools like Shodan have been implementing logic for identifying and inferring vulnerabilities in new products over time, since search engines are generally closed and commercial, the ability to identify new types of services does not keep pace with the emergence of new applications. For instance, around 25,000 new products were added to the dictionary of Common Platform Enumeration (CPE) just in 2024.[2] Al-

[1] Censys (`https://censys.io`); Shodan (`https://shodan.io`); Zoomeye (`https://www.zoomeye.hk`)

[2] Common Platform Enumeration (CPE) is a structured naming scheme for information technology systems, software, and packages. More information at: `https://nvd.nist.gov/products/cpe/statistics`

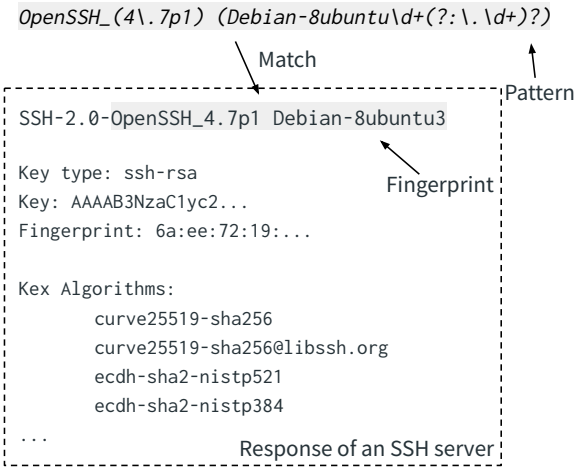

Figure 1. Example of a fingerprint match in an SSH Server response. During the inference process, we aim to identify fingerprints, portions of a device's probe response, through pattern matching, typically expressed using regular expressions.

though many of these products may not be accessible on the internet, this number still serves as a warning for the potential vulnerabilities.

In order to overcome these limitations, some studies propose combining information from multiple search engines to maximize results [Markowsky and Markowsky, 2015; Ponce *et al*., 2024]. Others, such as Genge and Enăchescu [2016], focus on creating their own mechanisms for enumerating services or vulnerabilities, usually involving two steps: (i) the identification of a service, vulnerability, or any other target of interest; and (ii) the enrichment of data through the structured addition of information related to the identified item, such as the version of a product. The main problem with these approaches is that, besides the complexity of the enrichment process, the user must manage all the complexity of handling a scalable processing architecture capable of dealing with large volumes of input data.

In this work, we propose a new approach for identifying new services and device information by enriching search engine data through data inference from fingerprints. In the context of cybersecurity, as illustrated in Figure 1, fingerprints are the part of response stored by the search engine obtained when interacting with a service, which can be used to infer information about the device or service by identifying patterns. A pattern is defined by a matching condition, such as a regular expression or even a group of tokens. Each pattern can be associated with a set of information related to the target object. For example, the fingerprint `SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu3`, generated after a request to a device, reveals not only that it is an SSH service but also the version of its protocol and of the implementation, as well as the operating system.

This article is structured as follows: Section 2 discusses the internal activities of a search engine for devices and services connected to the Internet. Section 3 presents our approach, which enables efficient processing of fingerprints from patterns defined as regular expressions, obtained from various Internet sources or manually created by experts, and applied to search engine data using our framework, TLHOP-Library [Ponce *et al*., 2024]. In Section 4, we evaluate our approach by analyzing the information inference capability, comparing it with the information provided by Shodan, and demonstrate that, in general, the level of inferred information exceeds that provided by Shodan. In Section 5, we also present two use cases that highlight the importance of this information for more accurate vulnerability analyses. For instance, a census of the diversity of SSH services available on the Brazilian Internet revealed several outdated applications, some of which are considered critical. Section 6 discusses related work and Section 7 concludes the paper.

2 Search engines and service identification

Shodan is one of the most popular search engines for Internet-connected devices [Daskevics and Nikiforova, 2021]. Launched in 2009, its logic involves connecting to random IP addresses and ports accessible via the Internet to collect information about diverse available services. Although the operation of each search engine may vary, they share a common core: hundreds of modules implement communication protocols to collect data in the form of records of applications running on the probed ports. This data is used to infer information about the services, which can range from data repositories to web pages.

There are several studies focused on the automatic identification of fingerprints for enriching service data, using information from probes conducted by search engines [Wang *et al*., 2009; Sarabi *et al*., 2023; Wang *et al*., 2022]. These approaches range from using neural networks to train models capable of identifying fingerprints [Sarabi *et al*., 2023] to employing data mining techniques to identify services from text and screenshots [Wang *et al*., 2022]. While these approaches typically outperform those provided by search engines, they face two main challenges: (i) they require a large dataset to identify the various patterns across searches, which can delay the ability to identify new services; and (ii) the techniques employed, along with the data volume, often demand significant computational resources.

Although the internal logic used in search engines is not fully disclosed, some available documentations [Durumeric *et al*., 2015] describe tools based on traditional scanning solutions like ZMap [Durumeric *et al*., 2013]. Because of that, we conjecture that most (if not all) pattern matching is performed by identifying known fingerprints registered in a catalog. Engines like Shodan are capable of scanning the entire Internet at least once a week; this frequency of data generation likely limits the use of more complex processing models. Furthermore, since engines are usually commercial tools, employing automated techniques to identify services may result in unstable or non-transparent behavior, potentially reducing tool's accuracy and usefulness.

In this context, the registration of tokens and regular expressions capable of identifying services from fingerprints is typically performed by domain experts. Once a pattern match is established between a fingerprint and a pattern, the identification of a service or device information is achieved by consulting the data associated with the registered pattern. Additionally, new fields can be created by deriving information from existing data. Examples of such fields include: (i) textual fields, such as the name or version of the operating system; (ii) numeric fields, such as the probed port; (iii)

fields with a list of values, such as a list of vulnerabilities; and (iv) complex fields (in a JSON substructure), such as the MongoDB field in Shodan, which contains information about the service's status and the list of databases. Once this process is finished, the data becomes available through the engine's web interface for queries or from dumps for local processing.

Our framework expands the usability of search engine data, such as Censys and Shodan, in vulnerability analysis by providing a new mechanism to enhance the information derived from these systems. It leverages patterns supplied by the online community, patterns manually created by experts, and patterns inferred from the analysis of the engine's own data, enabling efficient matching of thousands of regular expressions to fingerprints across millions of records. Additionally, it offers an efficient interface for handling large volumes of data in a local environment, making it applicable in diverse contexts. In the next section, we detail the architecture and components of our previous framework, demonstrating how our approach can be used to enhance vulnerability analysis.

3 Efficient enumeration of services and devices

In this section, we present the extension of our TLHOP-Library framework [Ponce *et al.*, 2024] to enable efficient enumeration of services and devices based on stored responses received during a search engine's probing campaign targeting devices accessible over the Internet. Although our research utilizes data produced by search engines, the proposed techniques are directly applicable to data from any other tools. To achieve this, we provide a unified interface for data analysis, reducing the complexity of processing large volumes of data for network operators. This process is illustrated in Figure 2: (i) data from search engines are converted into a more efficient format; (ii) patterns, in the form of regular expressions obtained from various repositories, are standardized into a common format; (iii) a parallel and distributed processing tool is used to execute a regular expression computation engine; and (iv) a system of weights is applied to rank the identified patterns based on the degree of specificity of the match. We have implemented the discussed approach in the TLHOP-Library framework and made it available as a contribution to the scientific community.[3]

3.1 Data conversion and query interface

Search engines are capable of collecting information from over 500 million devices worldwide several times per month. Processing such large volumes of data can be challenging, especially in the case of N-day vulnerabilities identification analysis, where time is a critical factor. On the other hand, data formats like the JSON structure provided by systems such as Shodan make local data processing inefficient. The approach used in this work consists of converting data from search engines into a format that maintains a good level of data compression, scalability, and flexibility for a variety of scenarios. In order to meet these requirements, we use Apache Spark[4] as our big data processing tool and the Delta Lake format for storage.[5]

The initial step of our process involves adapting the original data from search engines into this more efficient format. Our framework is based on Spark's tabular data abstraction (DataFrame), but with the extension of operators and algorithms that allow users to express complex queries on search engine data at a high level of abstraction, delegating the construction and execution of Spark commands to our framework. To achieve this, we implemented converters responsible for transforming the data, along with cleaning and standardizing operations. These include inferring the cities and Brazilian regions linked to the probed devices based on the geographic coordinates inferred by the platform, as well as creating new attributes, such as the standardized name of the organization associated with the probed IP addresses.[6] From a performance perspective, our approach enables optimization techniques for data reading, record filtering, and transactional controls, as described below:

Reading Optimization. Search engine data contain a wide variety of information, making the data reading and inference process costly if the data were kept in a textual format such as JSON. In this context, using a compressed binary columnar format like Delta Lake ensures significant performance gains, as type inference is no longer required for each read operation. Columnar formats also support additional optimizations, such as *Projection Pushdown*, which allows reading only the columns necessary for a given query.

Data Partitioning. The ability to filter records during a task, *i.e., Predicate Pushdown*, is a crucial aspect of queries targeting specific events or performing joins with external datasets. While traditional files lack features such as indexes found in databases, formats like Delta implement data partitioning. During the writing process, subfolders are created for each value of the columns used as partition keys, such as the year and day when the probes were conducted. This enables Spark to read only the files in the relevant folders when querying data based on these columns, significantly improving performance.

Schema Evolution and Transactional Control. As new data is ingested daily, schema evolution allows for the addition of new fields. Support for record updates and transactional control are also crucial aspects. For instance, it is common to encounter fields with missing information, necessitating post-processing data refinement. Updates to identified CVEs over time are also frequent. Without this support, there would be a loss of information or the need for full data reprocessing.

The data converters, the identification and enrichment of information proposed in our work, and many other useful methods are examples of how complex activities can be encapsulated by simple operators provided by the framework.[7] For instance, the proposed algorithm to enrichment of information about services and devices, which will be discussed in detail in the following sections, can be divided into three stages: (i) collection and cleaning of patterns; (ii) execution of regular expressions on search engine data; and (iii)

[3] Available at: `https://github.com/lucasmsp/tlhop-library`

[4] Apache Spark: `https://spark.apache.org/`

[5] Delta Lake: `https://delta.io/`

[6] Currently, converters for Censys and Shodan are implemented; however, converters for other engines can be extended by users.

[7] A documentation of all supported operators is available on the repository website.

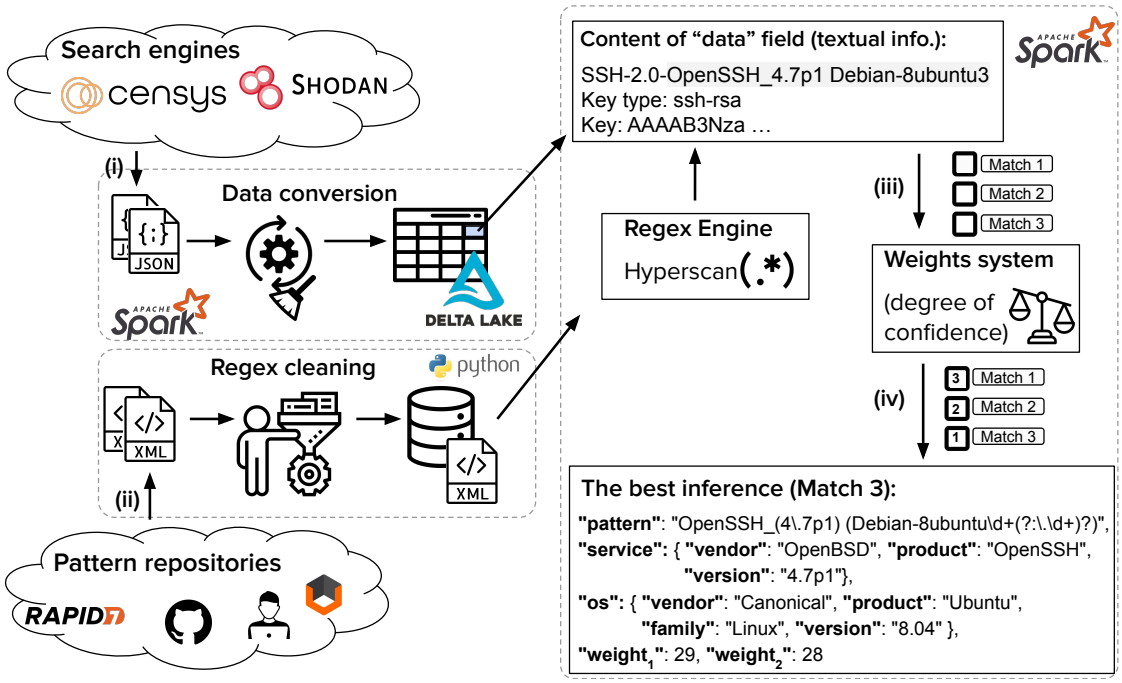

Figure 2. Architecture for service and device enumeration. Data from search engines is converted (i) into a more efficient format to enable fingerprint identification from thousands of registered patterns (ii). Each identified fingerprint (iii) has its weight calculated, which is then used to generate an ordered list of results (iv).

prioritization of inferences.

3.2 Collection and cleaning of patterns

Service identification approaches using automatic pattern discovery techniques, as mentioned in the previous section, are not only susceptible to errors but also typically address only the identification stage, without enabling the inference of related information. Even simple information derivations based on this identification, such as associating the Windows operating system upon identifying that the probed product is Microsoft Outlook, are often limited. For this reason, the use of fingerprint matching is a promising approach, as it enables us to associate all known information about each target while being less susceptible to false inferences. This is an especially important topic, particularly in the context of vulnerability assessments, where the financial or personal impact of an exploitation can be critical.

Our approach involves registering a pattern in the form of $R \rightarrow I$, where R is a regular expression that defines the item of interest, and I is the entire set of information that can be inferred from the discovery of this item. For instance, in Figure 2, information about the vendor and the product name, as well as information about the operating system, such as vendor, name, and version, can be inferred from the pattern `OpenSSH_(4\.7p1) (Debian-8ubuntu\d+(?:\.\d+)?)`. Furthermore, if we created a new pattern `SSH-[\d\.]+` or even extended the previous one, we could complement this information with the supported SSH protocol version. The information related to the patterns can be obtained directly from the fingerprint, such as the product version "`4.7p1`" or the SSH protocol version as "`2.0`". However, another part of the information is obtained indirectly, such as the vendor's name or the version of the operating system. While the first one is easy to obtain, the operating system version is more complex. In this case, the version can be estimated because Ubuntu performs OpenSSH updates via patches, not changing the version reported in the fingerprint. In other words, unless the user manually downloads the binaries of a new version of the tool, the system will continue with the same tag. Because of the string "`Debian-8ubuntu3`", we are able to known that is a version provided by the Ubuntu package repository, so we can estimate its version as 8.04.

Although creating certain patterns may require advanced knowledge of the product, our approach primarily uses the various patterns available in public repositories and other platforms shared by the community on the internet. In order to facilitate the management of these diverse patterns, our approach organizes them into `XML` files by service type. Each `XML` element in a file corresponds to a fingerprint, with its regex pattern and the information that can be derived from it. In this first version, we have cataloged 2,432 fingerprints obtained from public sources on the web; 2,366 were obtained from the open-source project Recog,[8] a Ruby framework for identifying services from fingerprints, where the rules are manually created by domain experts; 46 were obtained from the Vulners platform;[9] and another 20 were implemented by the team. However, other diverse websites or repositories on Github can be found due to the growing importance of this effort in the field of cybersecurity.

The XML files containing the fingerprints can be obtained in a semi-automated manner. Some sources, such as Recog, already provide them in XML format, requiring only periodic downloads. Others, such as the ssh-default-banners repository,[10] provide the patterns as tab-separated text files, requiring the creation of specific converters to XML format or, in simpler cases, manual creation of the XML files and entries.

[8] Recog: `https://github.com/rapid7/recog`
[9] Vulners: `https://vulners.com`
[10] SSH default banners (not used yet in our current solution): `https://github.com/richlamdev/ssh-default-banners`.

Since we use patterns defined by regular expressions from different sources, we perform preprocessing to clean and standardize the regular expressions. For example, sources like Recog assume that a record will only represent one type of service, which is not always the case for data provided by search engines. Because of this, our cleaning step standardizes the expressions by, for instance, removing case insensitivity, eliminating restrictions on regex patterns for the start and end of the pattern, and providing the option to remove incorrect or poorly formatted expressions using a blocklist file.

3.3 Execution of regular expressions

As mentioned before, we use Apache Spark to make processing of the large volume of data generated by search engines efficient and scalable. Internally, Spark distributes data processing across all available cores in its session, significantly speeding up execution. However, while Spark provides a native function for executing regular expressions, this approach has several limitations in our context: (i) the native function is restricted to removing or replacing substrings within a match, whereas our proposal uses regular expressions as a search mechanism that allows information derived from the input fingerprint to be added as a new field in the record. (ii) in our context, a textual record may contain more than one valid fingerprint, necessitating the addition of results to a list, a functionality not supported natively. (iii) the native function processes only one pattern at a time, which becomes impractical in scenarios involving hundreds or thousands of expressions. For these reasons, the approach provided by Spark's native functionality is inadequate for our needs.

In order to satisfy all these requirements, we developed a new operator for Spark that leverages a high-performance regular expression processing engine, offering not only greater usability but also improved efficiency compared to Spark's native engine. Our approach employs the Hyperscan Project [Wang *et al.*, 2019], a C-based regular expression engine designed for high performance. At the start of execution, all regular expressions are registered in the Hyperscan catalog, enabling the tool to exploit graph decomposition techniques that translate regular expression matching into a series of string and finite automata operations. This enables advanced optimizations based on Nondeterministic Finite Automata (NFA) and Deterministic Finite Automata (DFA). Additionally, during execution, Hyperscan accelerates fingerprint matching through SIMD (Single Instruction, Multiple Data) operations at the hardware level, significantly improving throughput and enabling the simultaneous execution of all regular expressions on a given input text.

Figure 3 illustrates the step-by-step process of fingerprint matching that was referenced in Figure 2 (step iii). Our framework executes Hyperscan on each record within every data partition created internally by Spark, in parallel, as a User-Defined Function (UDF). The result of this step (line 5 in Fig. 3) is a list of matches obtained in the form of dictionary (as shown in Fig. 2). For each element in this list, we compute the match's weight using our weighting system (discussed in the next section), which can be used to prioritize or assign a degree of certainty to the patterns found. The idea of this weighting system is to prioritize fingerprints whose

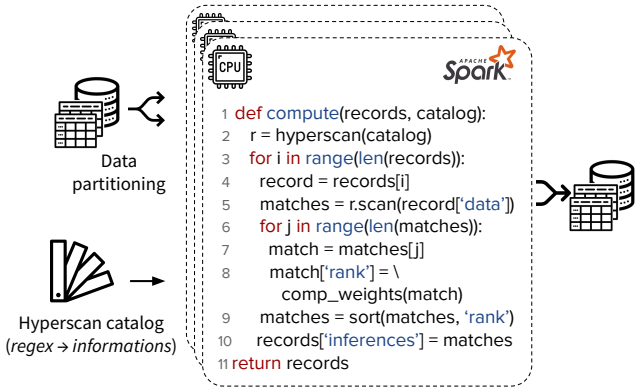

Figure 3. Exemplification of the fingerprint matching method. Spark is responsible for distributing blocks of records across the various cores available in the environment and for applying the Hyperscan catalog to each record in these blocks, with Hyperscan handling all optimization in the processing of thousands of regular expressions.

regular expressions have more constant terms than others. Our premise is that the registered expressions have already been previously tested, so a pattern that is found in the data will already have a certain confidence. However, in cases where more than one pattern is found, including for the same fingerprint, the best match can be the one that brings more specificity to the rule evaluated. For example, although the best regular expression linked to the record shown in Figure 2 was `OpenSSH_(4\.7) (Debian-8ubuntu\d+(?:\.\d+)?)`, other expressions such as `OpenSSH_[\d\.]+` would also be considered valid. For us, the difference between the two cases is that the first, by requiring a greater number of fixed terms, guarantees greater specificity in the inferences as mentioned before.

3.4 Weighting system

Our weighting system comprises two weights that can be combined to prioritize certain matches over others, as illustrated in Figure 4. The first weight is calculated based on the total number of characters in the matched fingerprint, in the example, 32. The second weight, 30, is determined by the number of characters present in the intersection of all continuous subsequences between the match and the constant terms within the pattern (*i.e*, removing terms like "\d").

Using this weighting system enables the prioritization of patterns when the same type of service is identified multi-

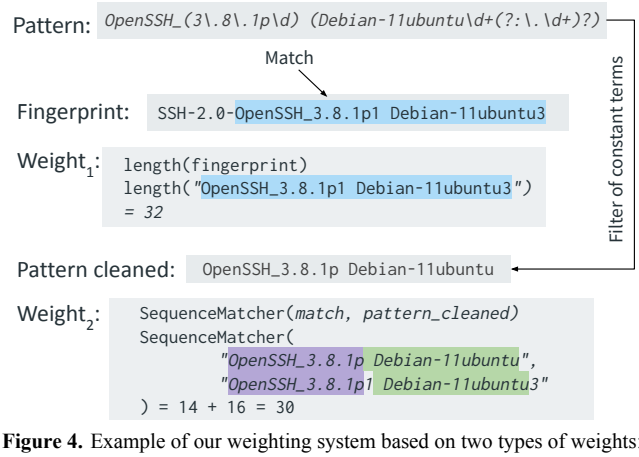

Figure 4. Example of our weighting system based on two types of weights: the first measures the size of the identified fingerprint, and the second assesses the proportion of the fingerprint that was explicitly defined in the pattern.

ple times. Patterns with zero Weight_2, for instance, indicate that the pattern consists entirely of random components, suggesting it may lack specificity. However, filtering patterns solely by a threshold is not always practical. For example, the fingerprint `nginx/2.3` for the expression `nginx/[\d\.]+` would yield a Weight_2 of only 6, which does not imply it is weak. In such cases, the ratio between weight 1 and weight 2 becomes valuable, as it demonstrates that 100% of the constant elements in the expression were matched. This flexible weighting system empowers users to tailor its application to their specific needs. For instance, they can opt to select only the pattern with the highest weight when targeting fingerprints for a single type of service or use it as a ranking system to focus on patterns of particular interest.

In our methodology, weight information is stored alongside the identified patterns. The final result, as exemplified in Figure 2 (step iv), consists of a list of matches ordered by the highest ratio between Weight_1 and Weight_2. This ordering is intended to simplify subsequent analyses of the results; however, since our system returns all inferences, users are free to develop their own prioritization schemes and access all obtained inferences. This approach is important because it provides transparency regarding the decision-making process and offers flexibility for specialists to make their own decisions.

Discussion: The use of Hyperscan within Spark is an interesting approach for enriching data retrieved from search engines by inferring information based on fingerprints. Our approach focuses on inferring services, protocols, and other characteristics that can be extracted from textual information collected by search engines during scans (specifically from the "data" field). The implementation of this method within frameworks like TLHOP-Library allows deeper analyses compared to those based solely on information provided by search engines. However, it is important to note that there is another form of inference that our current approach does not cover: inference based on the analysis of multiple types of information (fields). For example, analyses that jointly consider the probed system's response and port information (such as a probe on port 22 where the response contains the term `SSH.*`). This type of inference is currently not supported because the current fingerprint matching method (Figure 3) only considers the raw scan response field as input. Adding conditions based on other fields in a generic way would require the UDF to access all fields of a record, which could significantly impact processing time. Future work is being developed to create a hybrid two-step solution: the first step performs inference using regular expressions with Hyperscan inside the UDF; the second step, complementing the first, handles simple match validations (such as port identification or exact textual value matching) using Spark's native functionalities. In the current state, users need to perform these steps manually to obtain this type of result.

4 Experimental evaluation

In this section, we evaluate the results of the inferences obtained using our approach. Initially, we present a general characterization of the inferences based on the number of banners and IPs. In Section 4.2, we discuss in more depth the types of patterns found. In Sections 4.3 and 4.4, we evaluate the quality of the solution in inferring information about the operating system and the type of device, respectively. All experiments were conducted on a private server at the Universidade Federal de Minas Gerais using Apache Spark 3.4.1 and Delta Lake 3.0.0, in execution sessions with 10 cores and 20 GB of RAM.

4.1 Initial characterization

The evaluation of our approach uses Shodan data shared by Brazil's Center for Studies, Response, and Treatment of Security Incidents (CERT.br) as part of a partnership to map vulnerabilities within Brazil's IPv4 address space. It is noteworthy, however, that these data are publicly accessible through Shodan, and that the level of information required for the development of this work is subject to the same conditions available to any user of the platform.

The dataset corresponds to the first semesters of 2021 and 2023, 302 days in total. Table 1 provides a summary of the number of IPs and records in the sample, as well as a correlation with the number of matches obtained through the inferences of the 2,432 patterns discussed in the previous section. Our approach was able to identify fingerprints in approximately 60% of the records, distributed across roughly 13.5 million IPs within the Brazilian address space. The complete execution of the inference process for the 330 million records, including saving the results in a separate file, took approximately 175 minutes (*i.e.*, approximately 35 seconds per daily dump).

Table 1. The relationship between the number of Shodan records and IPs with the percentage of matches.

Metric	# Records	# IPs
Total	330,414,756	19,557,988
% Matches	59.8	68.9

Figure 5 presents a histogram of the number of characters in the `data` field, a field where Shodan's probes, stored in text format, are recorded and used for our inferences. Approximately 70% of the records contained between 100 and 500 characters of `data`. However, extreme cases with up to 3.9 million characters were observed.[11]

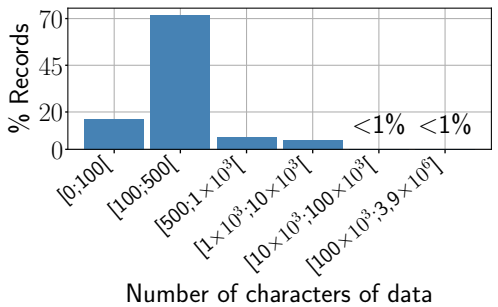

Figure 5. Number of characters evaluated in the records.

Approximately 55% of the registered patterns (1,341 out of 2,432) had at least one match in our dataset. Meanwhile,

[11] In our analysis, such cases represent probe information of data repositories like ElasticSearch, where a large volume of information is collected about available nodes and tables.

the number of IPs with at least one match was approximately 69%. The diversity of matched patterns is summarized in Table 2. A substantial portion of matches correspond to HTTP servers (represented in the first row), which amount to about 41% of matches and 36% of the IPs with matches. Overall, 402 distinct HTTP patterns match records in the dataset, representing approximately 90% of our total registered HTTP patterns. Matches based on DNS services or HTML pages (*e.g.*, from titles) ranked second and third, respectively.

Table 2. Relationship of patterns by service category.

Source	Records (%)	IPs (%)	Patterns (#)	Patterns (%)
HTTP (Servers)	41.34	36.25	402	90.33
DNS	9.15	25.01	57	81.42
HTML	8.53	19.61	195	43.33
SSH (Recog)	7.08	27.16	129	86.00
HTTP (Auth)	5.41	14.64	60	80.00
Apache (OS)	5.18	7.23	27	71.05
Telnet	4.32	18.63	57	39.58
HTTP (Cookies)	1.95	1.60	62	76.54
SNMP (SysDescr)	1.67	2.99	133	23.58
SMB (OS)	1.18	2.23	60	78.94
FTP	0.75	1.26	81	54.72
SSH (Others)	0.25	0.51	65	98.48
SNMP (SysObjId)	0.02	0.12	7	16.66
SMB (LM)	0.01	0.05	5	62.50
NTP	< 0.01	< 0.01	1	1.33

The higher number of patterns identified for HTTP servers, DNS and HTML is directly related to our analysis context: a large proportion of the services identified by Shodan are, indeed, web servers exposed to the Internet. Other services, such as SSH or Telnet, although prevalent in terms of the number of IPs, are less frequently probed by Shodan on the ports hosting these services, leading to a lower volume of corresponding records.

4.2 Evaluation of the patterns found

The relationship between the number of patterns matched in each record is illustrated in Figure 6. Typically, one or two matches were identified per record. However, in rarer instances, up to 294 matches were detected within a single record. These cases reflect scenarios where multiple applications were identified, or even multiple patterns from the same service category (as shown in column 1 of Table 2) were matched.

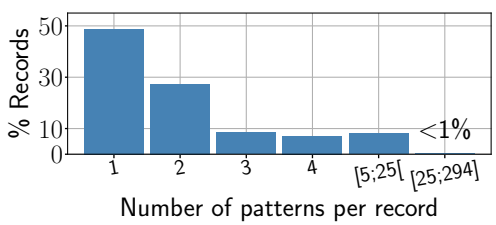

Figure 6. Histogram of the number of patterns per record.

Analyzing the distribution of matched patterns per record, approximately 75% of the records (Q3) had at most one or two inferences from the same source. However, as shown in Figure 7, which highlights the maximum number of matches from the same source per record (*i.e.,* same type of services), there were cases (notably for HTTP servers and HTML sources) exhibited significantly higher values. These correspond to the extreme cases observed in Figure 6. Specifically, records with up to 265 and 47 matches were identified for HTTP servers and HTML sources, respectively. Such occurrences, while rare, are expected and does not represent an abnormal behavior in our methodology. For example, in one instance, the Shodan response was approximately 20,000 characters long, containing traces from web services, printer servers, Windows services, and Linux services. These cases likely reflect network configurations where multiple services from distinct machines were accessible to Shodan via a single IP. Consequently, these inferences should be understood as services running on multiple physical or virtual machines within the internal network of the probed device.

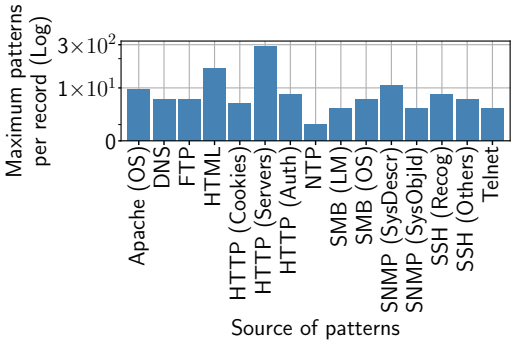

Figure 7. Maximum number of patterns from the same source per record.

The top five patterns with the highest number of matches are shown in Table 3. The list includes responses from inaccessible web pages, DropBear services (a lightweight tool providing an SSH server and client), Cisco routers, BIND DNS servers, and Huawei gateways. Identifying these services can be used for a variety of purposes. For instance, at least 21 vulnerabilities cataloged in the NIST/NVD dataset directly affect 63 versions of the DropBear SSH server, inferred using the second pattern in the table. The third pattern enables the identification of Internet-accessible devices with CWMP, a protocol exploited by the Mirai botnet in attacks targeting home routers.[12] Although these attacks were not caused by vulnerabilities in CWMP itself, attackers can exploit configuration errors and outdated versions: the original protocol employs an HTTP-based service for remote management, which is inherently insecure.

Although the degree of information gain from most patterns found in Table 3 generally allows only basic inferences, such as the type of service or simple fields like service version (*e.g.*, in the second row) or vendor (*e.g.*, in the second, third, or fourth rows), additional details can be inferred through a second stage of processing or from more specific patterns. This limitation arises because fingerprints do not always contain details for deeper insights.

Overall, out of the 1,341 patterns that matched at least once, 800 provide information about a specific service, 674 about the operating system, and 297 allow for identifying some hardware-related information. These findings are exemplified in Table 4, which highlights the five most frequent types of information inferred. The table also details the number of

[12] `https://t.ly/hiV1R`

Table 3. Top 5 frequent patterns.

Pattern	Source	# IPs
(?:404)?Not Found	HTML	3.178.775
*SSH-2.0-dropbear_.**	SSH (Others)	2.454.469
(?:Basic|Digest) realm="Cisco_CCSP_CWMP_TCPCR"	HTTP (Auth)	1.667.539
(?:BIND)?([89]\.[\d\.]+(?:[ab]\d+)? \ (?:-ESV (?:-R\d+)?)?(?:[-SPW][\d\.]+)? \ (?:-REL)?(?:-W\d+)?(?:rc\d)?)(?:-NOESW)?	DNS	1.523.558
(?:Basic|Digest) realm="HuaweiHomeGateway"	HTTP (Auth)	592.601

Table 4. Relationship of the main types of information derived from inferences.

Type	# Patterns	# Fields	Top 3 fields
Apache	35	3	variant (10)
Hardware	297	9	vendor (261), device (245), product (81)
Operating System	674	11	vendor (604), product (516), family (446)
Service	800	17	product (767), vendor (739), cpe (495)
SNMP	125	2	snmp.fpmib.oid.1 (1) e snmp.fpmib.oid.2 (1)

derived fields for each information type; for instance, operating system information is divided into up to 11 fields, with the *vendor* field being the most common, present in 604 patterns.

Our solution is capable of inferring information, such as operating system details, at a finer level of granularity than what is typically reported by search engines. For instance, while tools like Shodan usually provide basic details, such as the operating system name (*e.g.*, "`Linux`" or "`Ubuntu Linux`"), when we use inferences from fingerprints like `SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu3`, as previously illustrated, not only allows the identification of the SSH tool but also links it to the Ubuntu system and its specific version (8.04, codename Hardy). The next section explores the enrichment of operating system information in greater detail.

4.3 Identification of Operating System information

As shown in Table 4, about 50.2% (*i.e*, 674 of 1,341 total matched patterns) of the patterns identified in our dataset provide at least one type of information about the operating system that can be used to complement the information from search engines. Table 5 presents a comparison of the number of IPs and records where our approach was able to infer operating system information in relation to use only Shodan data (*i.e.*, using the TLHOP-Library framework to process Shodan data without enhancements). Our solution complements Shodan data with this type of information for approximately 2.3 million IPs (as shown in the second line, representing 11.9% of the total number of IPs), while in 16.3% (3.2 million IPs, as shown in the fourth line), both Shodan and our solution were able to identify the operating system. In total, our solution increases the availability of operating system information for IPs by a factor of 1.62×.

Table 5. Inferences about the Operating System.

Shodan	Inferred	% IPs	% Banners
✗	✗	69.08	75.96
✗	✓	11.90	18.21
✓	✗	2.70	1.77
✓	✓	16.32	4.06

By combining the inferences from Shodan and our approach (we considered only the highest-weighted match for each IP), a total of 6.0 million IPs had their operating systems identified. Table 6 presents the top 10 operating systems. In this table, whenever an IP had its OS inferred by both approaches, Shodan's result was given priority. In 14.5% of the cases, the only level of information that could be inferred was that the system was Linux-based, for example, through fingerprints such as `Linux Kernel 5.15.0-136-generic`. However, in many other cases, more detailed information was possible, including specific versions (not shown in this table). The high ranking of inferences like RouterOS (2nd place) and Windows (3rd place) may be partly due to the greater difficulty in distinguishing between different Linux-based systems.

Table 6. Top 10 Operating Systems.

S.O	# IPs	% IPs
Linux	2,833,979	14.49
RouterOS	796,890	4.07
Windows	643,290	3.29
Linux Akamai	545,435	2.79
Ubuntu	451,798	2.31
Hikvision	338,180	1.73
Debian	224,608	1.15
Red Hat	181,126	0.93
Ubiquiti AirOS	110,762	0.57
CentOS	102,312	0.52

Evaluating the conflict of information between the inferred operating system and the one provided by Shodan is a complex task. Table 7 summarizes our results. In total, there were 92 types of conflicts between the operating systems inferred by the two approaches, involving 62,406 IPs (1.95% of all IPs for which both types of inference were available). Most conflicts occur when Shodan provides a generic label, such as `Linux`, instead of specifying the distribution, like `Ubuntu` (lines 1 to 4 and line 10). These cases represent approximately 96.6% of all conflicting IPs. By mapping the relationships between systems, many of these conflicts can be resolved.

However, conflicts may still arise in cases such as the one previously illustrated, where Shodan captures information

Table 7. Top 10 Operating System inference conflicts.

Shodan	Inferred	# IPs	% IPs
Linux	Ubuntu	36,477	58,45
Linux	Debian	16,080	25,77
IOS	Cisco	6,166	9,88
Unix	Debian	1,396	2,24
Windows	Linux	384	0,62
Windows	Ubuntu	198	0,32
Windows	Unix	196	0,31
Windows	Linux Akamai	176	0,28
Windows	Compuprint	164	0,26
Ubuntu	Debian	146	0,23

from a set of services redirected from multiple machines. One common example is shown in Fig. 8, while Shodan inferred the operating system of the probed device as Windows, likely due to the presence of the fingerprint `ASP.NET`, our patterns led to the discovery of the product `Helix Server Version 9.0.4.960 (linux-2.2-libc6-i586-server)`, which have a higher weight and, therefore classified as a Linux system. In such cases, various services may be inferred, either by Shodan or by our solution, requiring post-processing to identify these instances and resolve or prioritize certain types of patterns. This highlights the importance of keeping a record of all the patterns found, rather than only retaining the highest-ranked pattern initially evaluated.

```
HTTP/1.1 302 Found
Date: Fri, 30 Jun 2023 01:13:00 GMT
X-Powered-By: ASP.NET
Server: Apache/2.2.4 (Win32) PHP/5.2.2
Apache/2.2.4 (Win32) PHP/5.2.3 Helix Server
Version 9.0.4.960 (linux-2.2-libc6-i586-server)
(RealServer compatible) Hewlett-Packard HP
Designjet T520 36in - ABC123 …
```

Figure 8. Example of a record with conflicting inferences.

4.4 Identification of hardware information

Nowadays, with the diversity of IoT devices, identifying the type of device, whether it is a router, an IP camera, or distinguishing a computer from an embedded system, is a complex task. As exemplified in Table 8, Shodan is able to infer information about the device in only 1.41% of the IPs scanned by the tool (the sum of the last two rows of the third column). However, in some cases, this identification can be made based on fingerprints similar to those previously shown. As a result, our solution is able to surpass the search engine with an inference rate of 19.8%, identifying the device type in 14.1 times more IPs than Shodan.

Table 8. Inferences about the device.

Shodan	Inferred	% IPs	% Banners
✗	✗	80.16	91.57
✗	✓	18.43	7.81
✓	✗	0.92	0.32
✓	✓	0.49	0.30

The fingerprint `Digest realm="Hikvision"` is an example of a common pattern found in our dataset that can be used to identify a DVR device, which is used to manage and record images from IP cameras, manufactured by Hikvision. The identification of such embedded devices is crucial, as they are often primary targets for attacks (*e.g.*, by creating a botnet) due to frequently having an outdated firmware or permissive security configuration [Zhu *et al*., 2021]. For example, these DVR devices may be vulnerable to CVE-2014-4880 (CVSS 7.5), which allows remote attackers to execute arbitrary code via the authorization header in a RTSP PLAY request.

4.5 Evaluation of the weights

Analyses in Sections 4.3 and 4.4 considered only the highest-weighted matches for each fingerprint. This approach works well when mapping features such as protocol name, operating system, or device type, as they are unique per IP. However, when trying to identify general services, selecting only one match, based on any criterion, would significantly reduce the number of inferences, as more than one service may be running in that IP.

Our solution allows the flexible use of our weighting system to adapt to the context of each task. The second weight, for instance, represents the portion (number of characters) in the regular expression and the obtained match. Patterns with a greater number of constant terms, such as the pattern `SSH-2\.0-OpenSSH_.+` that matches the fingerprint `SSH-2.0-OpenSSH_3.8.1p1`, will result in a higher $Weight_2$ (16), as they represent a more specific pattern compared to `SSH-[\d\.]+-OpenSSH-.+` (in this case, 13), since the first one restricts the use of the SSH protocol to version 2.0. However, a low $Weight_2$ value does not necessarily indicate that a pattern was inferred incorrectly. The validation of a pattern, which refers to its ability to capture a specific type of fingerprint, takes place during the pattern registration stage, while the current weighting stage is responsible only for filtering or selecting the best pattern.

Figure 9 shows the Cumulative Distribution Function (CDF) of the distribution of $Weight_2$ computed for all matches in our dataset. Analyzing the graph, 50% of the matches have a weight greater than 10 and up to 120 characters. Meanwhile, 25% of the matches have up to 8 characters. Patterns of this type may include services like Nginx using a pattern `nginx/[\d\.]+` on a fingerprint like `nginx/2.3`, which has a $Weight_2$ of 6. This example represents a case where, even though the weight is low, the pattern found is still valid. Due to this behavior, the two weights can complement each other: dividing $Weight_2$ by $Weight_1$ represents the ratio of the number of constant terms found in the regular expressions applied to the match. Table 9 shows the minimum, median, and maximum values for each of these metrics.

The behavior of pattern matching may vary depending on the type of target service. For instance, while HTTP (Auth) patterns required at least 11 specific characters to satisfy a match, patterns to services like HTTP (Servers) did not impose such restrictions. An example of such a pattern is `(?:IBM_HTTP_SERVER|IBM-HTTP-SERVER)`, which can be used to identify the presence of an Apache HTTP Server in an IBM server. In this case, the expression allows for the occurrence of either the term "`IBM_HTTP_SERVER`" or "`IBM-HTTP-SERVER`"; however, it does not enforce a global term requirement as seen in the previous examples. Because of

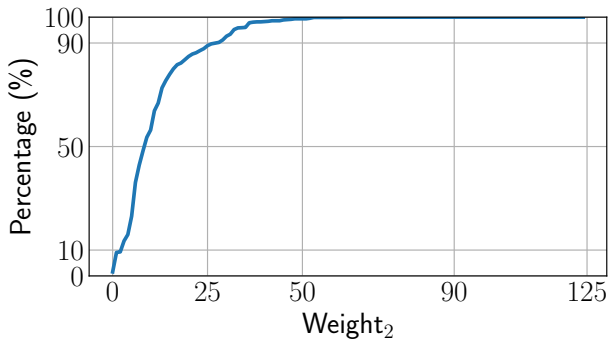

Figure 9. CDF of the Weight$_2$ distribution among the identified patterns.

that, our current solution sets the Weight$_2$ for a match of this type as zero (although it has a Weight$_1$ of 15). Note that a low Weight$_2$ does not necessarily indicate an incorrect match; rather, it reflects that our ranking approach cannot currently handle patterns with an "or" condition. Future work can extend our current solution to improve the matching validation.

Table 9. Weights distribution by service category.

Source	Weight$_2$	Weight$_2$/Weight$_1$ (%)
Apache (OS)	0/8/28	0/100/100
DNS	0/1/52	0/3.4/100
FTP	2/20/58	28.5/90.9/100
HTML	0/13/55	0.0/92.9/100
HTTP (Cookies)	1/10/30	3.3/76.9/100
HTTP (Servers)	0/8/61	0/94.4/100
HTTP (Auth)	11/32/50	12.7/84.2/100
NTP	13/13/13	100/100/100
SMB (LM)	4/7/18	100/100/100
SMB (OS)	0/0/54	0/0/100
SNMP (SysDescr)	0/9/66	0/69.2/100
SNMP (SysObjId)	22/23/29	100/100/100
SSH (Recog)	0/8/48	0/100/100
SSH (Others)	2/33/124	25.0/63.5/100
Telnet	3/12/37	42.3/100/100

5 Use cases

The information gain in discovering a running service, detailing the operating system, or identifying the device type can be applied in various contexts of vulnerability analysis. This section presents two use cases to illustrate how information obtained through our *approach* can be used: to evaluate vulnerabilities in SSH usage across the Brazilian internet (Section 5.1) and to assess the popularity of an operating system (Section 5.2) over Internet.

5.1 A Reassessment of the 2016 Census on the SSH protocol across the Internet

The study by Gasser *et al*. [2014] was one of the first to analyze the diversity of SSH services on the Internet through probing devices. Following this, other research efforts, such as the 2016 Internet Census [Popescu, 2016], further explored this critical area. The continued study of SSH services remains vital, as this protocol is foundational in numerous contexts, and vulnerabilities within it can enable attackers to gain unauthorized access to remote machines. A recent example is the CVE-2024-6387 vulnerability, dubbed "regreSSHion". This flaw represents a regression of the previously patched CVE-2006-5051 from nearly two decades ago, with the issue reintroduced in October 2020 but rediscovered only in September 2024. The vulnerability exploits an unauthenticated remote code execution flaw, granting attackers full root access to targeted machines without any user interaction.

In our present reassessment, we revisit the previous Census [Popescu, 2016] with a new mapping of information using more recent data (from 2021 and 2023), focusing specifically on IPs within the Brazilian address space. In our research, we identified 3,643,220 IPs, only in Brazil, running an SSH server, with 914,990 and 2,957,615 probed by Shodan in the first six months of 2021 and 2023, respectively. As a comparison, the previous study scanned 15,646,188 IPs worldwide with SSH servers in 2016. Approximately 75% of the IPs with an identified SSH server in our dataset of 2021 were no longer identified in 2023 (*i.e.*, only 229,385 IPs were responsive in both periods). While this could be attributed to the possibility that these devices stopped exposing SSH on the Internet, it is more likely that a significant portion of these devices use dynamic IPs, which would have changed their addresses over this period.

Although it is expected that the overall number of SSH instances has increased since 2016, it is noteworthy that the number of instances in Brazil represents about 23% of the total SSH instances probed globally during that time. Comparing our data from 2021 and 2023, there is also an increase factor of 3.23 in instances probed by Shodan. This growth may be attributed to Shodan's enhanced capacity to probe new devices, but the proliferation of Internet-connected devices could also have been a significant contributing factor.

In 2016, the authors also analyzed the distribution of SSH protocol versions. We compared some of their findings with our data, as illustrated in Figure 10: (i) approximately 95.8% of services used SSH version 2.0, a proportion similar to what we observed in our data (*e.g.*, in 2023, the ratio was 99.7%); (ii) SSH version 1.99 accounted for about 4.12%, while in our data, a reduction was observed from 2021 to 2023 (0.25%); (iii) a few instances (<0.01), the same proportion as the 2016 study, were probed as "`SH-2. 0`";[13] (iv) finally, we also identified protocol version 1.5, mostly associated with Cisco and Huawei devices. Given that SSH-1 has numerous design flaws that make it vulnerable, this version is now considered obsolete and should be avoided.

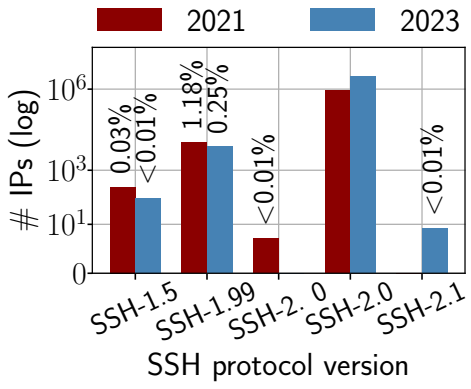

Figure 10. Diversity of the SSH protocol in Brazil.

[13]The spacing is a bug in OpenSSH version 3.8p1, released in 2004. NIST/NVD list 48 CVEs affecting this product.

The popularity of the SSH tools is perhaps one of the most significant differences between the 2016 study and our findings (illustrated in Table 10, focusing only on 2023 data). While in 2016, OpenSSH and DropBear accounted for approximately 74.4% and 13.8% of instances globally, respectively, our data reveals a clear inversion: about 77.7% of IPs are associated with DropBear while 16.8% with OpenSSH. For other tools, although their rankings shift slightly, the proportions remain similar.

Table 10. Top 5 SSH products in 2023.

Product	2023		2016	
	Top	% IPs	Top	% IPs
DropBear	1st	77.72	2nd	13.82
OpenSSH	2nd	16.86	1st	74.44
ROSSSH	3rd	2.25	4th	2.95
Missing	4th	0.71	7th	0.30
Comware	5th	0.51	12th	0.15

Table 11 shows the popularity of each DropBear version found in Brazil in 2023. That year, almost all devices running DropBear used versions from 2019, such as version 2019.78. The reason for this popularity is unclear; it is possible that this version was set as the default in certain package managers. A deeper investigation would be needed to confirm this hypothesis. However, in addition to the fact that this version was already outdated by 2023, we also found even older versions, some released in 2003 (*i.e.*, twenty-one years ago).

Table 11. IPs using DropBear in 2023 and it's release year.

Year	% IPs
≤ 2011	0.38
2012	0.05
2013	0.07
2014	0.60
2015	0.16
2016	1.96
2017	1.00
2018	0.03
2019	93.78
2020	0.03
2022	< 0.01

As mentioned in Section 4.2, there are at least 21 vulnerabilities cataloged in the NVD that directly affect 63 versions of DropBear. Some of them are critical, such as CVE-2016-7406 and CVE-2016-7407, both with a CVSS v3 score of 9.8, which affect devices running versions below 2016.74 and allow remote attackers to execute arbitrary code during authentication through manipulation of the username or host argument. Another example is CVE-2023-48795 (also known as the Terrapin attack), discovered more recently in 2023, which affects versions below 2022.83 and allows remote attackers to bypass integrity checks by omitting certain packets, consequently establishing a connection where some security features may be downgraded or disabled. Although the CVSS for CVE-2023-48795 is 5.9 (medium), it currently has an EPSS score of 95.9%,[14] representing a high likelihood of exploitation.

[14]The Exploit Prediction Scoring System (EPSS) is a data-driven effort for estimating the probability that a software vulnerability will be exploited in the wild.

Despite the large number of vulnerable devices due to these outdated versions of DropBear, no vulnerabilities were originally linked to any of the records collected by Shodan. From our assessment, when Shodan identifies an SSH service supported by its engine, its inferences focus on collecting information such as public keys, available encryption protocols, among other details. However, information such as the version or the tool name used is ignored, possibly due to the complexity of creating patterns that can adapt to the diversity of services. In this context, our approach can be used to complement Shodan's data, enriching it with such information.

5.2 Use Case 2: Identification of outdated Ubuntu versions

As in the previous use case regarding the usage of SSH protocol, search engines still fail to identify information about the operating system in many scenarios. By identifying specific versions of operating systems like Ubuntu, we can provide additional context regarding the potential vulnerabilities associated with these versions. For example, if a device is running a version of Ubuntu that no longer receives security updates (such as older versions of 16.04 or 18.04), this could serve to alert network administrators to take corrective actions, such as updating the system or mitigating risks. In this context, evaluating Ubuntu versioning is another example of how our solution can complement Shodan's data.

As mentioned earlier, the information about the operating system, when provided by Shodan, is presented in a single text field, which can range from the generic inference of "`Linux`" to more detailed descriptions like "`Ubuntu 20.04.4 LTS (Focal Fossa) (Linux 5.4.0-139-generic)`". Our inference system, on the other hand, structures this information, when available, allowing for finer granularity. For instance, throughout 2023, Shodan was able to infer the operating system for 3.7 million IPs. Of these, 2.3 million had only the term "`Linux`" associated with the field, and only 198 thousand IPs had the term "`Ubuntu`", with only 749 of them providing version information. On the other hand, our solution was able to identify 3.4 million IPs with Linux-based systems. Of these, 274 thousand were Ubuntu distributions (without a defined version), and 147 thousand had an inferred version.

Figure 11 shows the number of devices for each Ubuntu version, probed by Shodan in the year of 2023. The blue bars represent the devices where our solution was able to identify the version, while the orange bars represent the information provided by Shodan. In this figure, while Shodan was able to identify more recent distributions, such as version 22.04 (April 2022) and 23.04 (April 2023), with a bias to identify LTS versions, our approach outperforms in diversity and quantity of devices. A large portion of our inferences came from matching patterns related to SSH. Therefore, an additional effort to register new patterns for recent SSH versions could help our approach identify new Ubuntu versions. Evaluating this graph, the result is alarming: There is a wide variety of Ubuntu versions that have been unsupported for years. For instance, Canonical's extended support (Expanded Security Maintenance (ESM)) ended in 2007 for version 5.10, while the standard support for version 12.04 (LTS) ended in 2016.

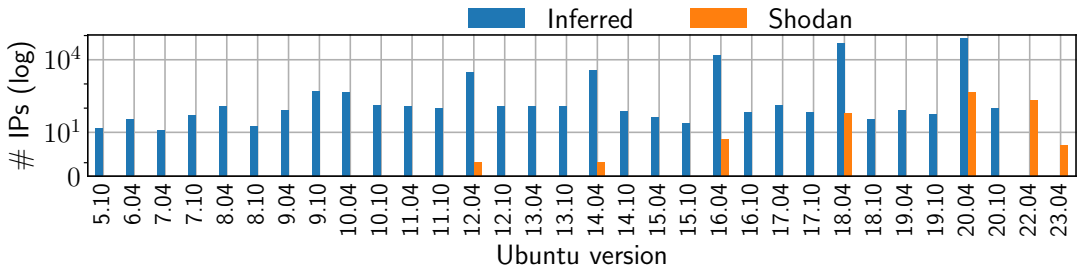

Figure 11. Number of devices with the Ubuntu system for Shodan probes in 2023.

Discussion: Although the analyses presented in this section use IP records collected from the Brazilian address space, our methodology could be applied similarly to global data, should such data be available. Furthermore, most of the fingerprints registered in our system were obtained from public Internet sources and repositories without any specific association with Brazil. Therefore, our methodology is general and applicable to any dataset. Although our findings may be biased towards the Brazilian Internet and its properties, they still provide insights and may be relevant in other contexts. Adding more country-specific patterns or repeating our analysis on other datasets would provide complementary information and provide a more complete picture.

6 Related Work

Search engines are widely used for research in various contexts, such as vulnerability analysis of specific countries [Al-Alami *et al*., 2017; Novianto *et al*., 2021; Hellberg and Vosseler, 2017], identification of medical or industrial devices [Bracciale *et al*., 2024; Samtani *et al*., 2018], or even understanding the general behavior of organizations through their devices [Nogueira *et al*., 2023; Moriot *et al*., 2022]. However, due to the large volume of data and their complexity, many of these studies are limited to analyses conducted solely through the engine's web interface. This constraint restricts the ability to post-process the data, apply more complex algorithms, or integrate with other databases, thereby limiting the scope and depth of the analyses [Al-Alami *et al*., 2017; Zaidi *et al*., 2018; Raikar and Maralappanavar, 2021].

In response to these constraints, previous approaches like the one employed by Hellberg and Vosseler [2017], which use ElasticSearch and Kibana to filter and visualize queries, or our proposed framework, leverage search engine data in independent environments. While the former approach enables advanced visualizations that Shodan does not offer, its reliance on ElasticSearch limits its ability to perform complex post-processing operations or apply updates to the data. Our framework, on the other hand, leverages the Delta Lake format to handle large volumes of data without the overhead of maintaining ElasticSearch servers. Additionally, it supports transactional operations, enabling optimized data reading and updating.

Enrichment of search engine data is often necessary in various contexts to complement information provided by these engines, which may be lacking due to the data collection mechanism. For instance, in Cheng *et al*. [2021], the authors utilized Shodan screenshots to assist in distinguishing the types of devices probed. Another example is presented in O'Hare *et al*. [2019], where the authors integrate Censys data with diverse vulnerability databases, a functionality similarly supported by our framework, to complement the vulnerability enumeration made by the engine.

Similar to our approach, Recog, initially introduced in Section 3, has been widely adopted in various projects due to its extensive fingerprint database built by the research community. In Cheng *et al*. [2021], for instance, the authors utilized Recog to develop a classification model capable of identifying critical information such as protocols, services, and vendors. In fact, during the development of our initial version, a significant portion of our patterns was derived from those created by the Recog community. However, we did not limit ourselves to this source, and there are notable distinctions between the two systems. For instance, Recog is primarily designed for direct use during penetration testing stages via a command-line interface (CLI), connecting to a service to infer information. As such, performance is not its main focus. Conversely, our approach is designed to process large volumes of data generated by search engines. It can be integrated into programming environments, supports distributed and parallel processing, and includes optimization mechanisms for pattern computation. Additionally, our framework extends matching capabilities by introducing metrics to rank and validate computed inferences, enhancing its utility in large-scale scenarios.

Our work can also benefit from approaches like that of Sarabi *et al*. [2023], mentioned in Section 2, which focus on automatic pattern generation. In this context, while the initial phase involves generating patterns from fingerprints using LLMs or machine learning models, our framework is designed to efficiently process diverse regular expressions over large volumes of data.

7 Conclusion and Future Work

This work presents an approach capable of processing large-scale patterns for recognizing applications and devices from various data sources. Using this approach on an efficient framework, we enriched data from Shodan to infer applications and devices connected to the Internet. We observed that the set of inferred information generally exceeds that provided by the search engine. Not only were we able to identify a greater number of services, such as the operating system (1.6× more IPs), and information about devices (14.1×), but the quality of the information is also more structured.

Our system is not limited to just these two types of information, but can handle any other information that can be inferred from a fingerprint pattern, such as a software product, its version, and its vendor. This type of information is important not only for security assessments but also for

other types of large-scale research, such as market studies. For example, although search engines like Shodan attempt to infer certain types of popular vulnerabilities, the ability to distinguish between a probed device being an IoT device or a general-purpose computer allows, for instance, focusing on potential targets for ransomware campaigns, even before the device has been compromised.

In Section 5, we illustrate how this type of information can be used in vulnerability analysis based on two use cases. In the first scenario, we conducted a census on the use of the SSH protocol on the Internet, comparing it with surveys from previous years. In this scenario, we identified a number of outdated tools that had published vulnerabilities; however, Shodan's lack of identification of this type of tool meant that several of these vulnerabilities could not be investigated. In the second use case, we were able to identify a number of versions of the Ubuntu operating system that had not been supported since 2007. Our approach allows us to complement the information provided by search engines by enumerating services and identifying related vulnerabilities.

Future work can be conducted on multiple fronts. From a more direct perspective, we plan to include new fingerprint patterns, potentially sourced from additional repository databases or through automated approaches using machine learning. Enhancements to the weighting system are also crucial for improving the validation of inferred information. Moreover, supporting inferences based on multiple types of information, that is, considering not only textual fingerprints but also other fields, such as the port number, could significantly enhance the refinement of the solution. From an evaluation perspective, assessments focused on risk analysis can be performed by relating fingerprints to vulnerability catalogs. Additionally, our service identification approach can be applied in other contexts, such as distinguishing devices operating under dynamic IP addresses.

Declarations

Authors' Contributions

Lucas M. Ponce conceived the presented idea, which was implemented by him along with Indra Ribeiro and Etelvina. Lucas was responsible for creating the library provided in this work, while Indra and Etelvina were responsible for creating and collecting the fingerprints and validating all the methods developed in the library. All three researchers contributed to conducting the experiments and writing the text. Ítalo Cunha, Cristine Hoepers, Klaus Steding-Jessen, Marcelo H. P. C. Chaves, Dorgival Guedes, and Wagner Meira Júnior supervised the research project and assisted in reviewing the text. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Funding

This work was partially funded by NIC.br, RNP/CTIC (2955), FAPEMIG, CNPq, CAPES, MASWEB, INCT-Cyber, and IAIA (INCT for AI).

Availability of data and materials

The developed framework is available at `https://github.com/lucasmsp/tlhop-library`. The Shodan data used in this work was provided by CERT.br as part of a partnership to map vulnerabilities within Brazil's IPv4 address space. However, this data can be collected directly by users with access to the Shodan platform by filtering for devices probed in Brazil.

References

Al-Alami, H., Hadi, A., and Al-Bahadili, H. (2017). Vulnerability scanning of IoT devices in Jordan using Shodan. In *2017 2nd International Conference on the Applications of Information Technology in Developing Renewable Energy Processes & Systems (IT-DREPS)*, pages 1–6, Jordan. IEEE. DOI: 10.1109/IT-DREPS.2017.8277814.

Albataineh, A. and Alsmadi, I. (2019). IoT and the Risk of Internet Exposure: Risk Assessment Using Shodan Queries. In *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks"*, pages 1–5, Washington, USA. IEEE. DOI: 10.1109/WoWMoM.2019.8792986.

Bracciale, L., Loreti, P., Raso, E., and Bianchi, G. (2024). In Plain Sight: A Pragmatic Exploration of the Medical Landscape (In)security. In *Proceedings of the 8th Italian Conference on Cyber Security (ITASEC 2024)*, pages 1–14. IEEE. Available at:`https://ceur-ws.org/Vol-3731/paper15.pdf`.

Cheng, H. *et al*. (2021). Identify IoT Devices through Web Interface Characteristics. In *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, pages 405–410. IEEE. DOI: 10.1109/ICCCS52626.2021.9449258.

Daskevics, A. and Nikiforova, A. (2021). ShoBeVODSDT: Shodan and Binary Edge based vulnerable open data sources detection tool or what Internet of Things Search Engines know about you. In *2021 Second International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*, pages 38–45, Estonia. IEEE. DOI: 10.1109/IDSTA53674.2021.9660818.

Durumeric, Z., Wustrow, E., and Halderman, J. A. (2013). ZMap: Fast Internet-wide Scanning and Its Security Applications. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 605–620, Washington, USA. USENIX Association. Available at:`https://dl.acm.org/doi/10.5555/2534766.2534818`.

Durumeric, Z. *et al*. (2015). A Search Engine Backed by Internet-Wide Scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 542–553, Denver, USA. ACM. DOI: 10.1145/2810103.2813703.

Gasser, O., Holz, R., and Carle, G. (2014). A deeper understanding of SSH: Results from Internet-wide scans. In *2014 IEEE Network Operations and Management Symposium*, pages 1–9, Poland. IEEE. DOI: 10.1109/NOMS.2014.6838249.

Genge, B. and Enăchescu, C. (2016). ShoVAT: Shodan-based vulnerability assessment tool for Internet-facing services. *Security and Communication Networks*, 9(15):2696–2714. DOI: 10.1002/sec.1262.

Hellberg, N. and Vosseler, R. (2017). Western European Cities Exposed: A Shodan-based Security Study on Exposed Cyber Assets in Western Europe. Technical report, Threat Research Team, Trend Micro, Texas, USA. Available at:`https://t.ly/yIswd` (visited on 2024-11-29).

Imperva (2023). Vulnerability Assessment. Available at:`https://t.ly/GY9ae` (visited on 2024-11-29).

Markowsky, L. and Markowsky, G. (2015). Scanning for vulnerable devices in the Internet of Things. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 463–467, Warsaw, Poland. IEEE. DOI: 10.1109/IDAACS.2015.7340779.

Moriot, C. *et al*. (2022). How to build socio-organizational information from remote ip addresses to enrich security analysis? In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 287–290, Canada. IEEE. DOI: 10.1109/LCN53696.2022.9843570.

Nogueira, M. *et al*. (2023). A Large Scale Characterization of Device Uptimes. *IEEE Transactions on Emerging Topics in Computing*, 11(3):553–565. DOI: 10.1109/TETC.2023.3271315.

Novianto, B., Suryanto, Y., and Ramli, K. (2021). Vulnerability Analysis of Internet Devices from Indonesia Based on Exposure Data in Shodan. In *IOP Conference Series: Materials Science and Engineering*, volume 1115, page 012045. IOP Publishing. DOI: 10.1088/1757-899X/1115/1/012045.

O'Hare, J., Macfarlane, R., and Lo, O. (2019). Identifying Vulnerabilities Using Internet-Wide Scanning Data. In *2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3)*, pages 1–10, London, UK. DOI: 10.1109/ICGS3.2019.8688018.

Ponce, L. *et al*. (2024). Arcabouço Multi-motor para Detecção de Vulnerabilidades na Internet Brasileira. In *Anais do XLII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 197–210, Niterói, Brazil. SBC. DOI: 10.5753/sbrc.2024.1302.

Popescu, M. (2016). Internet Census, 2016. Available at:`https://t.ly/aQRM2` (visited on 2024-11-29).

Raikar, M. and Maralappanavar, M. (2021). Vulnerability assessment of MQTT protocol in Internet of Things (IoT). In *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*, pages 535–540, India. IEEE. DOI: 10.1109/ICSCCC51823.2021.9478156.

Samtani, S. *et al*. (2018). Identifying SCADA Systems and Their Vulnerabilities on the Internet of Things: A Text-Mining Approach. *IEEE Intelligent Systems*, 33(2):63–73. DOI: 10.1109/MIS.2018.111145022.

Sarabi, A., Yin, T., and Liu, M. (2023). An LLM-based Framework for Fingerprinting Internet-connected Devices. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, IMC '23, pages 478–484, Montreal QC, Canada. ACM. DOI: 10.1145/3618257.3624845.

Wang, R. *et al*. (2022). WYSIWYG: IoT Device Identification Based on WebUI Login Pages. *Sensors*, 22(13). DOI: 10.3390/s22134892.

Wang, X., Hong, Y., Chang, H., Park, K., Langdale, G., Hu, J., and Zhu, H. (2019). Hyperscan: a fast multi-pattern regex matcher for modern CPUs. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, NSDI'19, pages 631—648, Boston, USA. USENIX Association. Avaialble at:`https://www.usenix.org/conference/nsdi19/presentation/wang-xiang`.

Wang, X. *et al*. (2009). Extraction of fingerprint from regular expression for efficient prefiltering. In *2009 IEEE International Conference on Communications Technology and Applications*, pages 221–226, Beijing, China. IEEE. DOI: 10.1109/ICCOMTA.2009.5349207.

Zaidi, N., Kaushik, H., Bablani, D., Bansal, R., and Kumar, P. (2018). A Study of Exposure of IoT Devices in India: Using Shodan Search Engine. In *Information Systems Design and Intelligent Applications*, pages 1044–1053, India. Springer. DOI: 10.1007/978-981-10-7512-4_105.

Zhu, R., Zhang, B., Tan, Y.-a., *et al*. (2021). Determining the Image Base of ARM Firmware by Matching Function Addresses. *Wireless Communications and Mobile Computing*, 2021(1):10. DOI: 10.1155/2021/4664882.