Article title: ShoVAT: Shodan-based vulnerability assessment tool for Internet-facing services

B. Genge and C. Enăchescu
Petru Maior' University of Tg. Mureș, Department of Informatics, Tg. Mureș, Romania
Email: bela.genge@ing.upm.ro, ecalin@upm.ro

Abstract – Shodan has been acknowledged as one of the most popular search engines available today, designed to crawl the Internet and to index discovered services. This paper expands the features exposed by Shodan with advanced vulnerability assessment capabilities embedded into a novel tool called ShoVAT. ShoVAT takes the output of traditional Shodan queries and performs an in-depth analysis of service-specific data, i.e., service banners. It embodies specially crafted algorithms which rely on novel in-memory data structures to automatically reconstruct Common Platform Enumeration (CPE) names and to proficiently extract vulnerabilities from National Vulnerability Database (NVD). Compared to the state-of-the-art, ShoVAT brings several novel and significant contributions, since it encompasses automated vulnerability identification techniques, it can return highly accurate results with customized and even purposefully modified service banners, and it supports historical service vulnerability analysis without the need to deploy additional monitoring infrastructures. Experiments performed on 1501 services in twelve different institutions across different sectors revealed high accuracy of results and a total of 3922 known vulnerabilities.

SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2014; **00**:1–18
DOI: 10.1002/sec

RESEARCH ARTICLE

ShoVAT: Shodan-based vulnerability assessment tool for Internet-facing services

B. Genge*and C. Enăchescu

“Petru Maior” University of Tg. Mureş, Department of Informatics, Tg. Mureş, Romania
Email: bela.genge@ing.upm.ro, ecalin@upm.ro

ABSTRACT

Shodan has been acknowledged as one of the most popular search engines available today, designed to crawl the Internet and to index discovered services. This paper expands the features exposed by Shodan with advanced vulnerability assessment capabilities embedded into a novel tool called ShoVAT. ShoVAT takes the output of traditional Shodan queries and performs an in-depth analysis of service-specific data, i.e., service banners. It embodies specially crafted algorithms which rely on novel in-memory data structures to automatically reconstruct Common Platform Enumeration (CPE) names and to proficiently extract vulnerabilities from National Vulnerability Database (NVD). Compared to the state-of-the-art, ShoVAT brings several novel and significant contributions, since it encompasses automated vulnerability identification techniques, it can return highly accurate results with customized and even purposefully modified service banners, and it supports historical service vulnerability analysis without the need to deploy additional monitoring infrastructures. Experiments performed on 1501 services in twelve different institutions across different sectors revealed high accuracy of results and a total of 3922 known vulnerabilities. Copyright © 2014 John Wiley & Sons, Ltd.

KEYWORDS
vulnerability assessment, Shodan, internet of things (IoT), common platform enumeration (CPE), common vulnerability and exposure (CVE), national vulnerability database (NVD)

***Correspondence**
N. Iorga Street, No. 1, Tg. Mureş, Mureş, Romania, 540088

Received . . .

1. INTRODUCTION

Advancements in software, hardware and network connectivity in general led to an exponential growth in the number of devices connected to the Internet. In fact, predictions indicate that more than 25 billion devices will be connected to the Internet by 2015 and approximately 50 billion devices by 2020 [1]. At the same time each device will expose a rich palette of remotely accessible services ranging from typical Web and Email servers to customized services providing, for instance, access to control devices installed in a “Smart House” programme [2].

Nonetheless, this trend also brings new opportunities to malicious actors by increasing their chances to identify vulnerable Internet-facing services [3, 4]. These concerns have been augmented by the development of search engines capable to identify devices and services directly connected to the Internet [5, 6, 7]. One of the most popular search engines within this class is Shodan [7], designed to crawl the Internet and to index all common services found within a predefined list (currently reaching more than 80). Its ability to search and index devices ranging from Web cameras to industrial automation hardware [8], has made from Shodan a powerful tool, but also “the scariest search engine on the Internet”, as reported by a CNN article [9].

Shodan was launched in 2009 by programmer John Matterly [7]. It is a computer search engine equipped with a graphical user interface that can identify Internet-facing devices and services. Shodan crawls the Internet for available devices and services and stores the collected data, namely IP address, port and service banner in a database accessible via `http://www.shodanhq.com` or via Shodan Application Programming Interface (Shodan API). For each discovered service Shodan scans and stores results repeatedly over time. This yields a time series of results available for each service and accessible to security experts for further processing and analysis.

Shodan is mainly intended for legitimate security specialists to aid the identification of Internet-facing services and devices and to improve security by limiting the degree of exposure. In light of these issues, in this paper we propose *Shodan-based Vulnerability Assessment Tool* (*ShoVAT*) to complement Shodan's features with advanced capabilities that enable identification of known vulnerabilities in discovered services.

Essentially, ShoVAT takes the output of traditional Shodan queries and performs an in-depth analysis of service-specific data, e.g., service banners, in order to identify software version numbers, product vendor, product name, etc. By leveraging specially crafted heuristics ShoVAT rebuilds Common Platform Enumeration (CPE) names for each identified product and it extracts the list of Common Vulnerability Entries (CVE) from National Vulnerability Database (NVD). Since Shodan also provides historical data on indexed services, ShoVAT can explore historically the degree of service exposure, i.e., the number and severity of discovered vulnerabilities.

In particular, ShoVAT exposes two main algorithms implemented in two distinct modules. The first algorithm automatically reconstructs CPE names from service banners returned by Shodan. The procedure employs an in-memory mapping of all CPE names extracted from NVD and structured in a multi-tree-like fashion. The second algorithm fuses two mathematical constructions into a *bipartite-hyper graph* model in order to ensure accurate and memory-efficient representation of CVE entries. The algorithm takes the CPE names discovered in previous steps and returns a list of CVE entries and vulnerability scores given in Common Vulnerability Scoring System (CVSS) format.

We analyze ShoVAT's features from several perspectives. We demonstrate the linear complexity of the two aforementioned algorithms with respect to the number of version strings and the number of services. Subsequently, we highlight the efficiency of the two data structures used to store CPE names and CVE entries. Most importantly, we demonstrate ShoVAT's ability to perform realistic service analysis by randomly selecting IP address classes associated to twelve institutions from seven different countries of European Union (EU). The analysis shows intrinsic service lifetime patterns and reveals 3922 known vulnerabilities. Finally, we evaluate the accuracy of results returned by ShoVAT (false positives and false negatives) and we compare ShoVAT's output against the reports generated by *Nessus* [10] vulnerability scanner.

Throughout this paper we make the following major contributions: (i) to the best of our knowledge ShoVAT is the first documented approach that extends Shodan's capabilities with automated CPE and CVE identification; (ii) while the examined commercial and open-source tools such as *Nessus* [10], *p0f* [11] and *PRADS* [12] rely on manually defined search patterns for processing service banners, ShoVAT tackles the problem differently by automatically compiling a list of all possible products available in NVD, which eliminates the need of source code updates and plug-in development; (iii) ShoVAT accurately reconstructs CPE names from modified banners, a feature that is not available in other approaches; (iv) ShoVAT does not require direct interaction with the inspected services, which ensures that reports can be delivered in few minutes, as opposed to existing techniques that necessitate repeated scanning and probing; (v) ShoVAT provides a novel approach to dynamically reconstruct CPE names from service banners by leveraging a multi-tree view of CPE version numbers and several heuristics for the identification of product names; (vi) ShoVAT embraces an in-memory representation of the entire NVD through a novel data model built on bipartite-hyper graphs; and (vii) ShoVAT has built-in capabilities (owed to Shodan) to generate historical reports that do not require further installations of additional monitoring software/hardware.

The remaining of this paper is organized as follows. An overview of related work is given in Section 2. Then, Section 3 presents ShoVAT's building blocks and algorithms for reconstructing CPE names and extracting CVE entries. Experimental results are presented in Section 4 and the paper concludes in Section 5.

2. RELATED WORK

In the field of vulnerability assessment, or more generally in the field of network asset discovery we find a variety of methods where classification is performed at various network levels starting from IP/UDP/TCP headers to application-layer packets. Popular network scanning tools such as *Nmap* [13] and more recently *ZMap* [14] can provide valuable information on discovered services to vulnerability assessment tools. As such, ShoVAT uses Shodan API to acquire a list of available services and service-specific information in a target IP address range. This data is then used by subsequent processing phases in order to reconstruct CPE names and to identify possible vulnerabilities.

In the field of network asset discovery we can find several tools such as *p0f* [11] and *PRADS* [12], which rely on user-specified signatures to distinguish between specific products and version numbers. These tools generate a list of discovered assets from network traffic capture files. Compared with such approaches, ShoVAT goes further and automatically reconstructs exact CPE names and extracts CVE entries from NVD, providing a comprehensive set of reports to security analysts.

Nessus is an “all-in-one” vulnerability assessment tool [10]. It actively probes services in order to test for known vulnerabilities and possible service configuration weaknesses. It relies on plug-ins which are specifically written to test for the presence of particular vulnerabilities. Although tools such as *Nessus* can accurately identify vulnerabilities, they require constant development of plug-ins which rely on predefined service patterns. Therefore, minor changes to service description strings, i.e., banners, can render such tools ineffective. Another limitation of

Nessus is that it does not monitor service vulnerability changes over time. However, this is an important requirement in modern computer networks, where changes may lead to security breaches and significant exposure of important assets. Conversely, since Shodan stores historical data on discovered services, ShoVAT supports historical vulnerability assessments and it can generate detailed analysis reports on vulnerabilities which appeared or which have been eliminated throughout a service's lifetime. This constitutes a significant advancement of the security monitoring field, which is owed to the combination of features provided by Shodan and ShoVAT.

NetGlean [15] provides real-time and historical view on the analyzed system. It continuously monitors the underlying network infrastructure and constructs machine fingerprints based on a variety of system features such as installed services, Operating System (OS) name and version, and so on. *NetGlean* relies on distributed network "sensors" to acquire network packets and a database system to store results in a time-ordered series. Although from one point of view this approach can provide historical data on the analyzed services, it suffers from several shortcomings. First, it requires the deployment of complex monitoring infrastructure. Second, the generated fingerprints are not suitable for vulnerability assessment since they are targeted towards asset discovery, instead of the construction of well-formatted identifiers, e.g., CPE names. Lastly, since *NetGlean* falls into the category of active scanners, it requires interaction with all target hosts, while ShoVAT does not need such measures since it relies on information provided by Shodan scanners.

Although not in the main scope of this paper, we mention that besides service discovery, vulnerability assessment in general also embraces OS fingerprinting. Besides classical tools such as *Nmap* [13] and *ZMap* [14], a variety of existing techniques explore the use of different network packet fields to extract valuable OS-specific information [16, 17, 18, 19, 20, 21, 22].

In this category the work of Auffret [16] combines the advantages of signature-based passive fingerprinting techniques with active probing in order to accurately identify OS in a restrictive scenario in which services are located behind filtering devices and have only one TCP port open. The proposed tool, called *SinFP* is suitable for fingerprinting over IPv4 as well as IPv6 and is available as open-source software [23]. More recently, Shamsi, *et al.* [24], proposed *Hershel*, a single packet probing for OS fingerprinting. The tool pioneers a novel approach to OS fingerprinting where a variety of fields from SYN packets are used to ensure accurate OS fingerprinting.

Since ShoVAT falls into the category of passive vulnerability assessment tools, it relies on the accuracy of data acquired from Shodan search engine. Compared to the aforementioned techniques, ShoVAT provides complementary features which expand the applicability of tools such as *Nmap*, *ZMap*, *Nessus*, *SinFP*, and *Hershel*. More specifically, ShoVAT embodies intelligent banner processing techniques which are not found in existing techniques and automated CPE name reconstruction as well as CVE entry extraction from the entire NVD. Subsequently, we have found that only *NetGlean* provides full support for time-based analysis, but *NetGlean* is mainly targeted towards fingerprinting, while ShoVAT provides historical assessment for changes in service vulnerabilities.

3. PROPOSED APPROACH

In this section we present the building blocks of ShoVAT and an analysis of the algorithms providing its rich set of features.

3.1. Vulnerability reports

Vulnerabilities constitute software flaws that enable attackers to perform malicious operations such as altering data, take control of underlying Operating System, or to expose and destroy valuable/sensitive information. Dealing with vulnerabilities is a challenging task mainly because every vendor and institution running a vulnerability database may have its own naming convention. In order to facilitate the sharing of vulnerability-related information, the Common Vulnerability and Exposure (CVE) was introduced in 1999.

One of the most well-established vulnerability databases, the National Vulnerability Database (NVD), builds on the information provided by CVE. NVD is meant to "enable(s) automation of vulnerability management, security measurement, and compliance" and it is often viewed as the "ground truth" for software vulnerability assessment [25]. The CVE entries available in NVD include a variety of fields such as a brief overview of the vulnerability, external references to advisories, impact rating, and a list of vulnerable software.

At the heart of every CVE entry lies the Common Platform Enumeration (CPE), "a standardized method of describing and identifying classes of applications, operating systems, and hardware devices present among an enterprise's computing assets" [26]. CPE names provide information on software vendor, name, version, language, edition, etc. A list of CPE names enclosed in 'AND'/'OR' schemas in each CVE entry provide the list of vulnerable software products. An example CPE is the following:

```
cpe:/a:microsoft:ie:7.0:beta1,
```

where *microsoft* denotes the vendor, *ie* denotes the product, *7.0* denotes the version number, and *beta1* denotes the software update.

An important feature of each CVE entry is its associated vulnerability score provided in the Common Vulnerability Scoring System (CVSS) format. CVSS scores range from 0 to 10, 0 being the lowest (low severity), and 10 being the highest (highest severity) vulnerability score that can

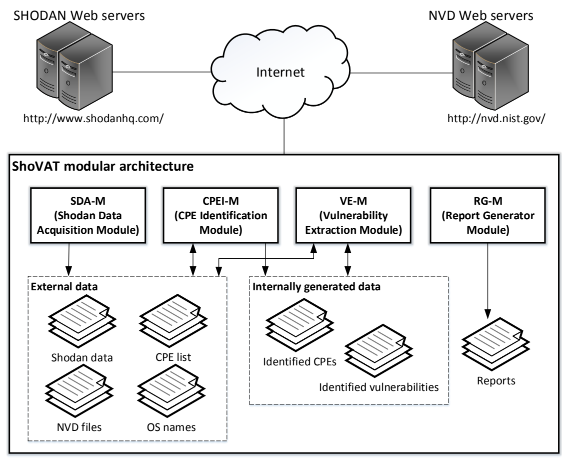

SHODAN Web servers
NVD Web servers
Internet
http://www.shodanhq.com/
http://nvd.nist.gov/
ShoVAT modular architecture
SDA-M (Shodan Data Acquisition Module)
CPEI-M (CPE Identification Module)
VE-M (Vulnerability Extraction Module)
RG-M (Report Generator Module)
External data
Shodan data
CPE list
NVD files
OS names
Internally generated data
Identified CPEs
Identified vulnerabilities
Reports

Figure 1. ShoVAT modular architecture.

Example A

```
http/1.0 302 found
date: tue, 20 aug 2013 12:41:10 gmt
server: apache/2.2.21 (win32) mod_ssl/2.2.21 openssl/1.0.0e
php/5.3.8 mod_perl/2.0.4 perl/v5.10.1
x-powered-by: php/5.3.8
location: http://[redacted]/xampp/
content-length: 0
content-type: text/html
```

Example B

```
cisco_ios software, c3750 software (c3750-ipbase-m), version
12.2(53)se, release software (fc2)
technical support: http://www.cisco.com/techsupport
copyright (c) 1986-2009 by cisco systems, inc.
compiled sun 13-dec-09 16:25 by prod_rel_team
```

Figure 2. Example of two service banners returned by Shodan.

be assigned to a specific CVE entry. Additional details on CVE can be found in [25, 26].

3.2. Architectural overview

ShoVAT was mainly designed to complement the data provided by traditional Shodan queries. It uses Shodan API to issue queries on the analyzed network block to `http://www.shodanhq.com` and it processes the received service banners in the search for known CPE names and CVE entries. Before launching its analysis modules, ShoVAT compiles a list of all known CPE names into a single input file by processing NVD's XML data files. Since NVD is updated once every day, it suffices to download NVD's XML files and run this sequence once per day. ShoVAT's internal modules build an in-memory representation of the entire NVD using an efficient graph structure presented in the following sub-section. By doing so, CVE entries are proficiently extracted and processed in order to produce meaningful reports. Owed to Shodan's ability to index past device configurations, one of ShoVAT's key features is that it can generate reports on past service vulnerabilities. This paves the way toward visualizing an institute's ability to reduce its security exposure over time, but also to identify critical dates which have led to significant security changes.

3.3. ShoVAT's modular architecture

ShoVAT's architecture embodies four main modules responsible for querying external Shodan and NVD databases, processing and storing intermediate results, and generating reports. Namely, the four modules depicted in Figure 1 are the following:

- *Shodan Data Acquisition Module (SDA-M)*. By using Shodan API SDA-M queries Shodan's database for the requested IP addresses and stores the results in local `json` files. For each IP address the results include the following main details: the timestamp at which the scan was performed, open ports, banner, and identified OS.

- *CPE Identification Module (CPEI-M)*. Once data is fetched from Shodan by SDA-M, this module processes the results in order to identify possible CPE names. First, it processes the local NVD copy in order to extract the list of all known CPE names. Since NVD database is updated once per day, this procedure is run once per day, when a new version of NVD is made available. The preparation of CPE lists is followed by the identification of CPE names in the results returned from Shodan's database. The output of this procedure is then stored in local files.

- *Vulnerability Extraction Module (VE-M)*. Its main purpose is to identify specific vulnerabilities, i.e., CVE entries, based on CPE names provided by CPEI-M. For this purpose, as documented later in this paper, it builds an in-memory mapping of CVE entries and CPE names from the local copy of NVD. Specific vulnerabilities together with their associated metrics are then extracted through in-memory look-up operations and are stored in local files. The module also serves as update interface for downloading new NVD versions.

- *Report Generator Module (RG-M)*. Finally, this module generates reports in the form of XML and `png` image files. Reports produce meaningful summaries by grouping together the discovered services, vulnerabilities and metrics.

At this point it is also important to underline that besides their enclosed features, modules can be selected to run independently of each other given that the required input data is already available. This is an essential characteristic of ShoVAT, which enables running specific modules without the need to re-execute previous time-consuming procedures. In the remaining of this section we describe the algorithmic constructions behind the core features supported by ShoVAT: the identification of CPE names and of CVE vulnerability entries.

3.4. CPE name reconstruction

The exact identification of CPE names is a fundamental requirement in ShoVAT's architecture. This is mainly owed to the fact that ShoVAT uses NVD as vulnerability database, where well-formatted CPE names are the core building blocks of CVE entries. However, this procedure is not trivial since product vendors might use a variety of formats for product-version strings.

In order to illustrate the main difficulties arising from exact CPE identification let us consider the two examples outlined in Figure 2. *Example A* illustrates the simplest case in which the banner contains several vendor, product and version numbers (underlined with continuous lines) in a well-structured and intuitive format. Contrarily, *Example B* exposes a banner where information on one single product is spread across the entire service string (underlined with dashed lines). Faced with these challenges, ShoVAT provides an automated approach to extract product information spread across a given string, e.g., service banner, in the attempt to find exact CPE names. The approach builds on hash tables for storing version numbers and CPE names as well as specially crafted heuristics embedded in an algorithm capable to automatically reconstruct CPE names.

The fundamental observation on which CPE name identification is built in ShoVAT is that typical version numbers consist of a sequence of numbers and '.' symbols. Therefore, instead of searching for sub-string matches for the entire CPE database consisting of more than 164 thousand entries, ShoVAT processes the input service string in order to identify version numbers that match the regular expression pattern V_{pat}="$I.(I.)^*I$", where I is an integer and '*' denotes zero or more repetitions of 'I.'. This way the search space is dramatically reduced to the number of CPE names that contain the identified pattern. The search continues with the identification of known vendor and product names, sub-version numbers and update numbers.

Based on this observation we adopted a hash table to store CPE names associated to version numbers. As such, given the set of all CPE names C extracted from the NVD database, and the set of all version numbers V extracted from the CPE file list, we define the hash-mapping function $h_v : V \to \mathcal{P}(C)$ to return the sub-set of CPE names for a given version number, where $\mathcal{P}(C)$ is the powerset of C. However, by using only h_v, in case of partially identified version numbers, e.g., for v ="`10.3.1`" extracted from "`10.3.1_build_476`" $\in V$, $h_v(v)$ will yield $\emptyset$. Consequently, a second data structure is defined to organize version numbers in multiple hierarchical tree structures where each intermediate node denotes a sub-version number and each leaf node belongs to V. For instance, in case of the previous version number, "`10.3`" will become the root of one such tree, "`10.3.1`" will represent its child node and finally "`10.3.1_build_476`" will become one of the leaf nodes. This way search operations can continue with partially identified version strings, which will eventually point to the correct version number available in CPE.

By taking each CPE version number and recursively building sub-version numbers we generate a multi-tree data structure. The elements of this tree are version numbers in set V', where $V' = V \cup S$ and S denotes the set of all sub-version numbers generated from each element in V. To ensure fast access to nodes ShoVAT's implementation of this data structure uses a hash-map h'_v : $V' \to \mathcal{P}(V' \times flag)$, where $flag \in \{0, 1\}$ and $\mathcal{P}(V' \times flag)$ is the powerset of $V' \times flag$. h'_v returns a set of version-flag pairs $(v, flag)$, where if $flag = 0$, then v is a leaf node and if $flag = 1$, then v is an intermediary node. A graphical view of the main data structures used in the process of CPE number identification is provided in Figure 3.

Next, we present the description of the CPE name reconstruction algorithm implemented in ShoVAT, which uses the data structures described above. The algorithm relies on the data received from Shodan. Recall that for each host and port/service discovered, Shodan API returns multiple timestamped entries corresponding to different Shodan scan times. At the same time Shodan returns other fields as well including CPE, version, product, and so on. Therefore, each entry returned by Shodan can be characterized by the tuple $\sigma =< host, port, t, banner, os, cpe, product, version >$. Nevertheless, since os, cpe, $product$, and $version$ are most of the times empty or inaccurate, ShoVAT builds its knowledge independently, but uses these values to complete its view on the analyzed services.

Continuing with the algorithm's description, from each Shodan entry σ the algorithm extracts the set of version numbers corresponding to V_{pat} pattern from $banner$ field. This operation is performed by $get\_pat(\mathsf{V}_{pat}, banner)$ helper function which returns the set of version numbers, denoted by ϑ. Then, for each $v \in \vartheta$ and $\varsigma = h_v(v)$, if $|\varsigma| > 0$ we search for best CPE matches for which at least vendor and product names are found in the CPE name. For this purpose ShoVAT uses a weighted matching algorithm implemented as $weighted\_match(cpe, banner)$ helper function. From each cpe the function extracts the list of continuous but distinct sequences of letters and numbers, denoted by τ, other than the version number which was already identified. For example, for CPE "`cpe:/o:cisco:cisco_ios:12.2:se`", $\tau = [cisco, ios, se]$. Then, for each $s \in \tau$ that is also found in $banner$ as a distinct word, the function increases a weight counter's value by 1. However, if the pattern is found next to a version number the weight is incremented by 2. By doing so, we reduce the weight of words which might lead to the selection of invalid CPE names, e.g., words from English language which are not associated to CPE names. Such an example is provided in Figure 4, where it is shown that 'by' word would lead to the improper selection of "`cpe:/o:cisco:cisco_ios:12.2:by`" and

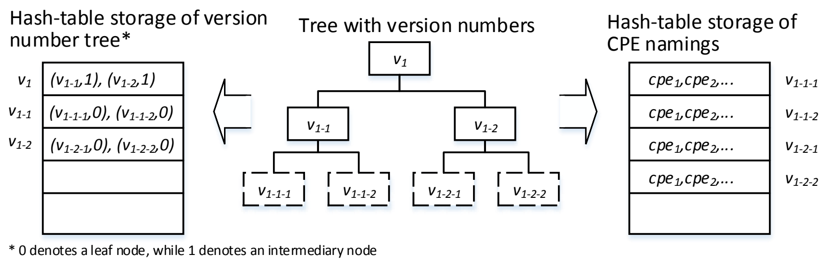
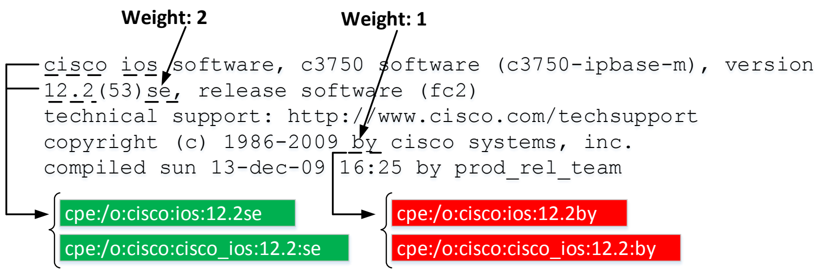

Hash-table storage of version number tree*
Tree with version numbers
Hash-table storage of CPE namings
v_1 | $(v_{1\text{-}1},1), (v_{1\text{-}2},1)$
$v_{1\text{-}1}$ | $(v_{1\text{-}1\text{-}1},0), (v_{1\text{-}1\text{-}2},0)$
$v_{1\text{-}2}$ | $(v_{1\text{-}2\text{-}1},0), (v_{1\text{-}2\text{-}2},0)$
v_1
$v_{1\text{-}1}$
$v_{1\text{-}2}$
$v_{1\text{-}1\text{-}1}$
$v_{1\text{-}1\text{-}2}$
$v_{1\text{-}2\text{-}1}$
$v_{1\text{-}2\text{-}2}$
$cpe_1, cpe_2, \ldots$ | $v_{1\text{-}1\text{-}1}$
$cpe_1, cpe_2, \ldots$ | $v_{1\text{-}1\text{-}2}$
$cpe_1, cpe_2, \ldots$ | $v_{1\text{-}2\text{-}1}$
$cpe_1, cpe_2, \ldots$ | $v_{1\text{-}2\text{-}2}$
* 0 denotes a leaf node, while 1 denotes an intermediary node

Figure 3. Data structures used to store CPE names.



```
cisco ios software, c3750 software (c3750-ipbase-m), version
12.2(53)se, release software (fc2)
technical support: http://www.cisco.com/techsupport
copyright (c) 1986-2009 by cisco systems, inc.
compiled sun 13-dec-09 16:25 by prod_rel_team
```

Figure 4. Weighted word matching in banner returned by Shodan. Green color denotes the selected (correct) CPE names.

Algorithm 1: Reconstruction of CPE names from data retrieved from Shodan.

```
input  : Shodan entry $< host, port, t, banner, os, cpe, product, version >$
output: A set of CPE names $\mathcal{R}_{cpe}$ and a set of possible OS names $\mathcal{R}_{os}$

Let $\mathsf{V}_{pat}$="$I.(I.)^*I$";
$\vartheta = get\_pat(\mathsf{V}_{pat}, banner)$;
Let $\mathcal{R}_{cpe} = \{cpe\}$;
Let $\mathcal{R}_{os} = \{(os, version)\}$;
foreach $v$ in $\vartheta$ do
    Let $\varsigma = h_v(v)$;
    if $|\varsigma| > 0$ then
        $\mathcal{R}_{cpe} = \mathcal{R}_{cpe} \cup \{(cpe_i, weighted\_match(cpe_i, banner)) | cpe_i \in \varsigma\}$;
    else
        Let $\vartheta' = get\_leaves(h'_v(get\_exact\_pattern(\mathsf{V}_{pat}, v)))$;
        Let $\varsigma' = \bigcup\{h_v(v'')|v'' \in \vartheta'\}$;

        $\mathcal{R}_{cpe} = \mathcal{R}_{cpe} \cup \{(cpe_i, weighted\_match(cpe_i, banner)) | cpe_i \in \varsigma'\}$;
    end
end
$\mathcal{R}_{os} = \mathcal{R}_{os} \cup \{match\_os\_names(OS_{names}, banner)\}$;
```

"`cpe:/o:cisco:ios:12.2by`" as CPE names. Nevertheless, in the same example 'se' is weighted with a superior value than 'by' since 'se' is in the direct proximity of 12.2 version number, which leads to the correct identification of CPE names.

Next, the algorithm continues with $|\varsigma| = 0$, that is, with the case in which the version number extracted from *banner* was not found in V. The algorithm derives the sub-string v' from each $v \in \vartheta$ such that v' follows exactly the V_{pat} pattern. This operation is performed by the helper function $get\_exact\_pattern()$ and is aimed at identifying the starting point for further CPE name searches in a sub-tree containing version numbers starting with v'. Since CPE names are associated to leaf nodes, the $get\_leaves()$ helper function is used to return the set ϑ' of version numbers (leaf nodes) for each v'. Similarly to the previous case, CPE names associated to elements in ϑ' are processed by $weighted\_match(cpe, banner)$, where $cpe \in h_v(v'')$ and $v'' \in \vartheta'$.

Finally, the algorithm looks for possible OS names in each *banner* based on a set of fixed (known) OS names OS_{names}. Once a possible OS name is identified by helper function $match\_os\_names()$, it is added to the return set $\mathcal{R}_{os}$, which is finally returned by the algorithm together with the possible CPE names found in the previous steps, denoted by $\mathcal{R}_{cpe}$. Note that the algorithm does not attempt to identify OS version numbers at this stage of the process since the experiments we performed with Shodan showed that generally, OS version numbers are missing or are poorly defined not only in banners, but in CPE names as well. Therefore, it is important to note that ShoVAT can only approximate possible OS version numbers based on available information.

Nevertheless, since several CVE entries also define OS-dependent vulnerabilities, ShoVAT uses this approximate information in the vulnerability identification phase and it reports back entries in which case exact OS matching was not possible. It is also important to note, however, that since ShoVAT is a passive vulnerability assessment tool it's body of knowledge is derived from third-party tools such as Shodan. More precise matching may be delivered by active probing techniques such as the ones described in [14, 18, 19, 24, 27]. However, this is not a trivial task and as confirmed by [28], today it still combines manual and automated techniques to ensure high OS fingerprinting precisions. Therefore, although we mainly consider this direction of work to be out of ShoVAT's scope, previous research on active OS fingerprinting may be embodied into ShoVAT, which would transform ShoVAT into a hybrid passive-active vulnerability detection tool. The CPE reconstruction algorithm is summarized as Algorithm 1.

3.5. CVE entry extraction

CVE entries are extracted from a local copy of NVD files, based on CPE names identified by Algorithm 1. However, the procedure is not as trivial as one would believe, since each CPE name requires a complete search through the entire NVD consisting of approximately 450MB of data structured in several XML files. Therefore, we propose an efficient in-memory mapping of NVD entries which reduces significantly the complexity of search operations.

Fundamentally, as already discussed at the beginning of this paper, each CVE entry embodies amongst others, several 'OR' sections which may be placed within one 'AND' section. Each section includes one or more CPE names denoting vulnerable products which might depend on the presence of other products in case 'AND' sections are defined. Conceptually, we may also view each CVE entry as being associated to several CPE names through 'OR' and 'AND' links, while each CPE name may be associated to at least one CVE entry.

Based on these observations we propose a novel approach to model the NVD database into a memory data structure. The proposed model fuses two mathematical constructions: *bipartite graphs* and *hyper graphs*.

Bipartite graphs are typically used to model concepts and data structures where vertices can be divided into two disjoint sets `U` and `V` such that each vertex in `U` is linked to at least one vertex in `V`. By applying this principle to

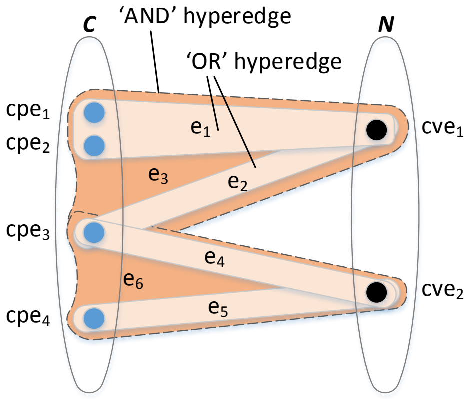


Figure 5. Example graphical representation of the bipartite-hyper graph model used to store data from NVD.

NVD, '`U`' is equivalent to the set of CPE names C and '`V`' is equivalent to the set of all CVE entries from NVD, denoted by N.

In the context of *hyper graphs* on the other hand, each edge can connect any number of vertices. Formally, a *hyper graph* is modeled as a pair (`V`,`E`), where `V` is the set of all vertices and `E` is the set of edges such that $\mathtt{E} \subseteq \mathcal{P}(\mathtt{V}) \setminus \{\emptyset\}$, where $\mathcal{P}(\mathtt{V})$ is the powerset of `V`. By assuming that each CVE entry defines only 'OR' and/or 'AND' links to several CPE names, the concept of *hyper graphs* can be naturally mapped to modeling NVD. The main advantage of using *hyper graphs* is that CPE names can be grouped together in order to ensure fast identification of related CPE names linked to a specific CVE entry. This also provides a precise mirroring of the original XML representation of NVD and reduces the amount of memory required to store all CVE entries.

By leveraging the previous constructions we model NVD using a *bipartite-hyper graph* defined as $B^H = (C, N, E, L)$. In this definition C is the set of CPE names, N is the set of CVE entries, and E is the set of edges such that $E \subseteq \mathcal{P}(C \cup N) \setminus \{\emptyset\}$. Additionally, the model includes L, denoting the set of labels attached to each edge. Each element of L is a pair (l, ϵ), where l is the label given as a string, and ϵ is a subset of E. In our case, however, L has only two elements, namely one where $l =$*"OR"* and another one where $l =$*"AND"*. A graphical visualization of the *bipartite-hyper graph* model for two CVE entries and four CPE names is given in Figure 5.

Since the majority of operations built into ShoVAT concerning specific B^H instances constitute searches on CVE entries, our implementation of *bipartite-hyper graph* adopts mappings based on several hash tables. As such, we define $h_{cpe} : C \rightarrow \mathcal{P}(N)$ to return the set of CVE entries for a given CPE name, and its inverse $h_{cpe}^{-1} : N \rightarrow \mathcal{P}(C)$ to return the set of CPE names for a given CVE entry. Additionally, we define $h_e : C \cup N \rightarrow \mathcal{P}(E)$ to return the set of edges connected to a CVE entry or a CPE name. In the proposed CVE entry extraction algorithm we also employ two helper functions: the $label(e), e \in E$ helper function is used to return the label string for a given edge, and the $count\_os\_words(cpe, OS), cpe \in C$ function used to count the number of words from OS set (possible OS string names) that appear in cpe string.

The CVE entry extraction algorithm takes for each host and timestamp the set of CPE names ς and the set of possible OS names OS identified by Algorithm 1 and returns a set of possible CVE entries $\mathcal{R}_{cve}$. Each element in $\mathcal{R}_{cve}$ is a pair $(cve, match)$, where $cve \in N$ is a CVE entry, while $match$ is a real number with several possible values:

- '0' denotes positive CVE match without the need to check additional dependencies, i.e., no 'AND' section present.
- '1' denotes positive CVE match with the additional matching of CPE dependencies.
- '2' denotes positive CVE match with the additional matching of OS names.
- '3' denotes negative CVE match owed to missing matching dependencies.
- $>$'3' (a number greater than 3) denotes the percentage of positive match based on the number of OS words matching specific CPEs in CVE entry.

The algorithm takes each CPE name for the given host, i.e., $cpe \in \varsigma$, and each CVE entry connected to cpe, i.e., $cve \in h_{cpe}(cpe)$, and performs the following steps. First, it identifies the edges connecting cpe and cve by intersecting the output of $h_e(cpe)$ and $h_e(cve)$. The result of this intersection is stored in ϵ_{all} which is then used to find if any 'AND' edges are present between cpe and cve. If so, then the algorithm finds the dependencies, i.e., CPE names, which are associated to each 'AND' edge $e_{and} \in \bigcup\{e \in \epsilon_{all} | label(e) =$"AND"$\}$. For this purpose it builds the set of 'OR' edges linked to cpe, i.e., $\epsilon_{or} = \bigcup\{e \in \epsilon_{all} | label(e) =$"OR"$\}$, and builds the set of dependencies ς_{dep} by looking for CPE names $cpe' \in e_{and}$, which do not have 'OR' edges in ϵ_{or}.

Once the algorithm finds dependencies for a specific cpe name it verifies if dependencies from ς_{dep} are within its initial set of CPE names ς. If so, it stores the positive match $(cve, 1)$ in $\mathcal{R}_{cve}$ and it runs the same sequence for the next CVE entry, denoted in the algorithm by *@StepNextCVE*. Otherwise, a deeper inspection is performed on CPE dependencies in the search for possible OS name-based dependencies. The $count\_os\_words()$ helper function is used for this purpose which returns '2' for an exact word count match and '3' plus a match percentage, otherwise. The match value together with the CVE entry are then stored in $\mathcal{R}_{cve}$.

Finally, the algorithm runs the sequence corresponding to the case in which no dependencies have been identified, i.e., there are no 'AND' sections in CVE entry. This is a simpler scenario in which $(cve, 0)$ is stored in $\mathcal{R}_{cve}$ and the execution steps to the next CVE entry. A summary of the algorithm is given in Algorithm 2.

3.6. Implementation details

ShoVAT has been implemented in `Python` language mainly because Shodan provides `Python` API for querying its database. As already stated, ShoVAT's architecture is highly modular, each of the four modules is implemented as a separate `Python` module accessible from the console by running `shovat.py` main script and selecting different options.

In its present form ShoVAT is a console-based application and it operates on files and generates results in different output files. This approach is well-suited for security analysis since it allows the automation of vulnerability assessment and the coupling of ShoVAT with additional security assessment tools. ShoVAT has been mainly tested on Ubuntu and Debian distributions, but since it relies on traditional `Python` libraries, it should run on other distributions as well. Finally, we mention that ShoVAT's current code is available in `GitHub` as a private repository, but future versions might be publicly released.

4. ASSESSMENT AND EXPERIMENTAL RESULTS

We conducted several experiments and tests in order to validate the proposed approach and to identify possible limitations of ShoVAT's built-in algorithms. In this section we present the results of these experiments as well as data confirming high performance of ShoVAT in terms of execution time as well as of false/true positive rates. Since the most significant parts of ShoVAT are concentrated in CPEI-M and VE-M modules, we focus on the operations implemented within these two modules.

4.1. Complexity of operations

As already discussed in the previous sections, results from Shodan are structured according to host, port and timestamp at which the scan was performed. Therefore, Algorithm 1 will be executed for each host, port and timestamp entries as well. We use n to denote the number of hosts, p to denote the number of discovered ports, and t to denote the number of timestamped scans performed on each host. For each such entry the algorithm extracts all version number strings from service banner, denoted by v. Then, for each version number and each associated CPE name the algorithm finds the best possible match. The number of CPE names for each version number is denoted by c.

The complexity of Algorithm 1 is linear with respect to n, p, t, v, and the number of CPE names c. The complexity C_1 of the first algorithm can therefore be estimated as:

$$C_1 = \mathcal{O}(nptvc). \quad (1)$$

Algorithm 2: Extracting CVE entries.

```
input  : A set of CPE names $\varsigma$ and a set of possible OS names $OS$
output: A set of CVE entries $\mathcal{R}_{cve}$

foreach $(cpe, cve)$, where $cpe \in \varsigma$ and $cve \in h_{cpe}(cpe)$ do
    Let $\epsilon_{all} = h_e(cpe) \cap h_e(cve)$;
    if $\exists e \in \epsilon : label(e) =$ "AND" then
        Let $\epsilon_{or} = \bigcup\{e \in \epsilon_{all} | label(e) =$ "OR"$\}$;
        foreach $e_{and} \in \bigcup\{e \in \epsilon_{all} | label(e) =$ "AND"$\}$ do
            $\varsigma_{dep} = \bigcup\{cpe' \in C | cpe' \in e_{and}$ and $\forall e \in h_e(cpe')$ if $label(e) =$ "OR", then $e \notin \epsilon_{or}\}$;
            foreach $cpe' \in \varsigma_{dep}$ do
                if $cpe' \in \varsigma$ then
                    $\mathcal{R}_{cve} = \mathcal{R}_{cve} \cup \{(cve, 1)\}$;
                    @StepNextCVE
                end
                $match = count\_os\_words(cpe', OS)$;
                $\mathcal{R}_{cve} = \mathcal{R}_{cve} \cup \{(cve, match)\}$;
            end
        end
    else
        $\mathcal{R}_{cve} = \mathcal{R}_{cve} \cup \{(cve, 0)\}$;
        @StepNextCVE
    end
end
```

It should be noted, however, that this represents a relatively pessimistic upper bound since throughout our experiments we noticed in many cases identical host/port entries at different scan times. These entries are filtered by ShoVAT and Algorithm 1 is only executed for new entries, which means that in many cases $t = 1$. At the same time, the number of version numbers discovered in each banner is usually one. Nevertheless, in the typical case of Web services for example, banners may include additional information on sub-product versions as well, e.g., `openssl` and `php` versions. Therefore, a rough estimation of v places its values in the $[1, 6]$ interval. The number of CVE names c can however range from 1 to the worst case scenario of approximately 4000.

It is also important to realize that hash table implementations embodied in Algorithm 1 have led to a complexity of approximately $\mathcal{O}(1)$ for search operations. This way C_1 is free of such search operations on relatively large tables with more than 164 thousand entries.

The execution of Algorithm 2 is performed on each set of CPE names r returned by Algorithm 1 for each host, port, and scan time. The algorithm extracts from NVD loaded into memory the associated CVE entry and verifies if dependencies are present and if so, are satisfied. The number of CVE entries associated to each CPE name is denoted by e, and the number of dependencies, i.e., CPE names which need to be verified, is denoted by d. The complexity C_2 of the second algorithm can therefore roughly be estimated as:

$$C_2 = \mathcal{O}(nptred). \qquad (2)$$

Similarly to the previous algorithm, ShoVAT verifies if identical CPE names from the same host have already been processed and returns the previous set of results without re-executing the algorithm sequence, i.e., in several cases $r = 1$. The number of CVE entries depends on the number of discovered CPE names, while the available dependencies may also vary according to the identified CVE entries. Nonetheless, the algorithm has a linear complexity with respect to n, p, t, r, e, and the number of dependencies denoted by d.

Lastly, it should be noted that both algorithms can operate independently of each other, making them well suited for parallel processing. Nonetheless, in its present form ShoVAT was destined to run sequentially, although its modular architecture is not constrained to such executions.

4.2. Memory data structures

As already discussed, CPEI-M uses two memory-resident hash tables. The first one stores CPE names loaded from NVD, while the second one builds several version number trees for quick access to sub-version elements.

In Figure 6 (a) we depicted the number of trees and the number of nodes (version numbers) contained by each tree. As shown in this figure, there are few trees with more than 10 nodes; there is in fact only one tree with 1000 nodes. This distribution is more visible in Figure 6 (b), where it can be seen that approximately 90% of trees have less than 10 nodes. This constitutes an important finding since smaller trees yield better processing performance and

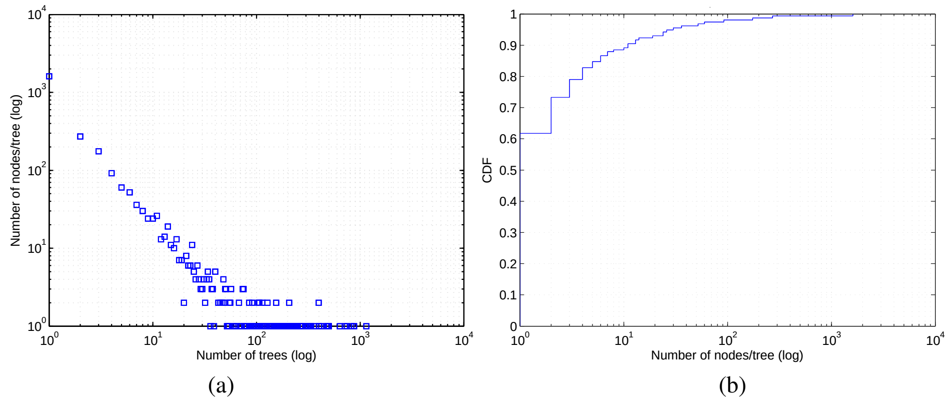


Figure 6. The number of trees and nodes: (a) the number of nodes and trees; and (b) CDF of the number of nodes/tree.

Table I. Institute types used in the use case assessment process (from seven different countries across the EU).

Type of institution	Count	Notation
University	5	$Univ_i$
Telecom operator	3	$Telco_i$
Railway system	2	$Rail_i$
Bank	1	$Bank_1$
Power company	1	$Power_1$

finally to less CPE names that need to be processed for each version number that is found in service banners.

The proposed bipartite-hyper graph model for in-memory CVE representation provides an efficient approach for reducing the number of edges associated to each CVE. To illustrate the valuable properties of the proposed CVE model, in Figure 7 (a) the Cumulative Distribution Function (CDF) of the number of CPE names associated to each CVE is shown, as originally available in NVD. Here it is shown that approximately 30% of CVE entries have more than 10 CPE names associated, while 10% have more than 70 CPE names associated and 5% have more than 100. It should be further noted that we have also found 352 CVE entries with more than 1000 CPE names associated, which constitute an important motivation for the construction of efficient data structures for in-memory representation. Two particular CVE entries in this list are `CVE-2007-2586` and `CVE-2007-2587`, which include each 1345 CPE names. Both vulnerabilities refer to the same Cisco IOS versions (from 11.3 through 12.4), which is also the reason why they have the same number of CPEs associated.

Conversely, the bipartite-hyper graph model proposed in this paper can associate more than one CPE name to the same CVE through a single edge. The dramatic reduction in the number of required edges to associate CPE names with CVE entries in the proposed model is depicted in Figure 7 (b). Here it can be seen that 98% of CVE entries require one "AND" edge and five "OR" edges in the bipartite-hyper graph structure. Furthermore, only less than 1% of CVE entries require more than five "OR" edges and in only one case we have found the need of 47 "OR" edges (see `CVE-2007-1093`). These last cases are owed to CVE entries that include more than 10 "AND" edges, where each "AND" edge is associated to a specific vulnerable software configuration.

4.3. Use case assessment

In the next phase of the assessment we selected twelve 24-bit IP blocks, i.e., class 'C' sub-nets, allocated to different institutions across seven countries from EU. IP addresses were carefully verified using Shodan API and they were cross-referenced with `https://www.maxmind.com/` in order to ensure that they were in fact assigned to the specific institutions. More specifically, we selected five IP blocks assigned to five different universities, three IP blocks assigned to three different telecommunications operators, two IP blocks assigned to two different railway transportation operators, one IP block assigned to one banking institution and one IP block assigned to one power company. A summary of the selected institution types and the notation used throughout this section for each institution are given in Table I.

In the next phase we launched ShoVAT's acquisition module (SDA-M) with the selected list of IP blocks. By leveraging Shodan's API, the module scanned the entire class of IP addresses for each institution's IP block in the search for available historical data. For each institution Shodan API returned historical data on all scans performed since the service was first detected.

The results revealed a total of 886 hosts and 1501 services spread across the twelve institutions included in this study. The most common services discovered were HTTP and HTTPS, however, we have also discovered FTP services, DNS, SSH, and SNMP. In total, we discovered 41 different type of services, the largest variety of services was found in University-type institutions. Nonetheless, by counting the total number of services, a closer inspection of $Rail_1$ revealed 228 different services, which was slightly lower than the maximum number of 265 services

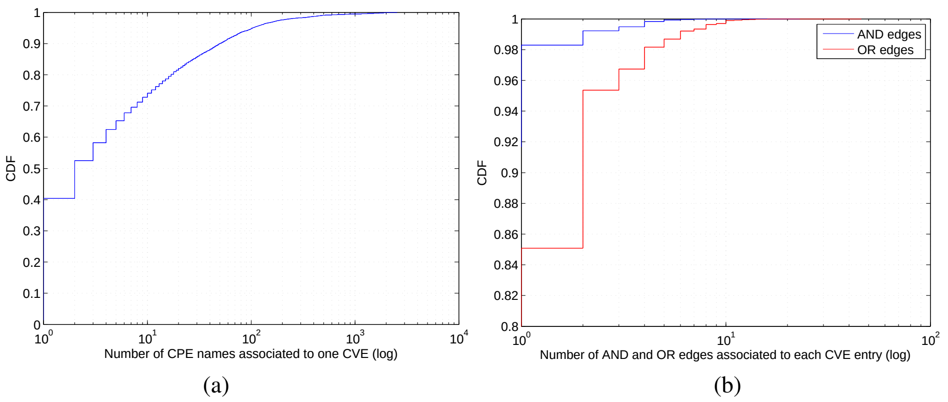


Figure 7. The number of edges associated to CVE entries: (a) CDF of the total number of CPE names associated to each CVE as they appear in NVD; and (b) the number of edges associated to each CVE in the bipartite-hyper graph representation.

Table II. Discovered service types and hosts across the twelve selected institutions.

	Universities					Telecommunications			Railway		Banking & Power	
Service	*Univ*$_1$	*Univ*$_2$	*Univ*$_3$	*Univ*$_4$	*Univ*$_5$	*Telco*$_1$	*Telco*$_2$	*Telco*$_3$	*Rail*$_1$	*Rail*$_2$	*Bank*$_1$	*Power*$_1$
DNS	4	–	1	10	2	–	1	2	–	2	–	2
FTP	9	–	4	20	6	2	1	6	–	4	–	6
HTTP	75	34	5	54	39	48	14	53	145	47	15	80
HTTPS	26	38	5	20	31	19	13	33	83	17	17	52
IMAP	1	–	–	4	2	0	1	2	–	–	–	–
IMAPS	7	1	–	4	4	–	2	–	–	1	–	–
POP3S	6	1	–	4	2	–	1	2	–	1	–	–
RDP	15	–	1	22	1	5	2	5	–	1	–	–
SMTP	7	–	–	12	–	2	7	10	–	3	–	2
SSH	27	3	5	19	28	0	1	5	–	2	–	–
Total serv. type	26	8	7	30	13	8	17	17	2	14	2	9
Total serv. count	265	80	23	244	127	80	55	134	228	84	32	149
Host count	103	46	10	91	72	55	29	81	207	60	25	107

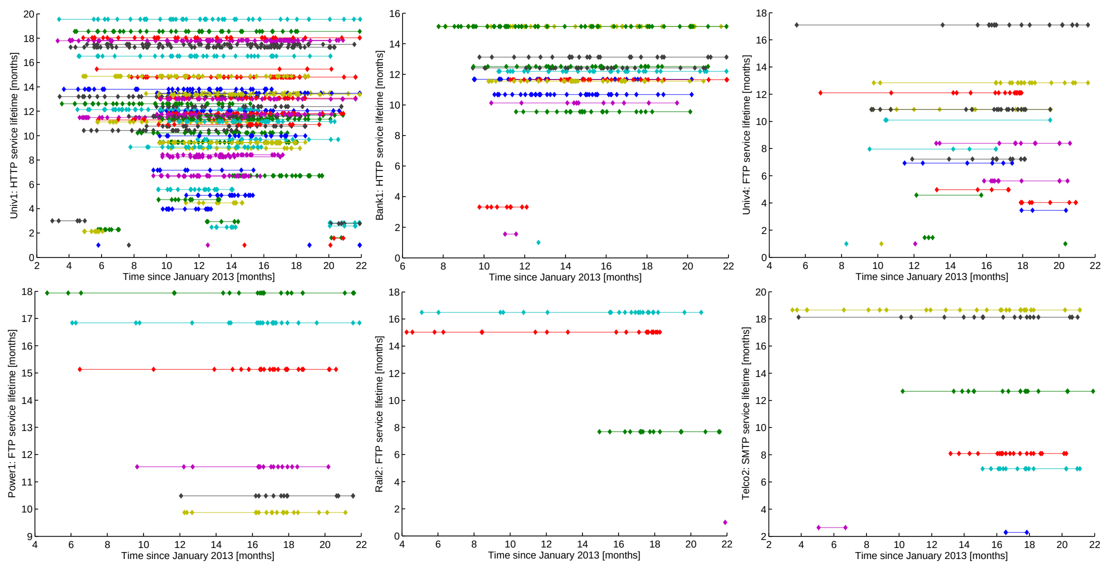
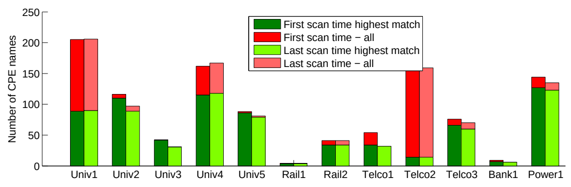
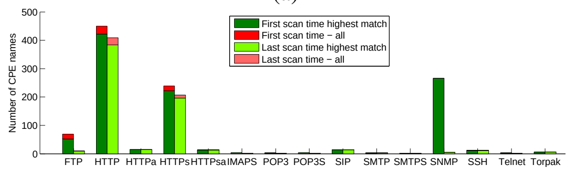


Figure 8. Example of ShoVAT's ability to return historical data on different services. The figures also illustrate possible lifespan differences across different sectors.

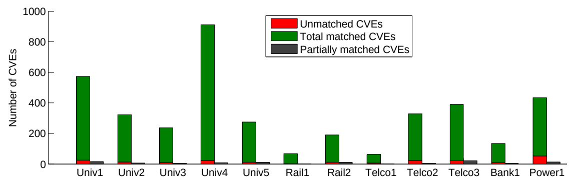


(a)

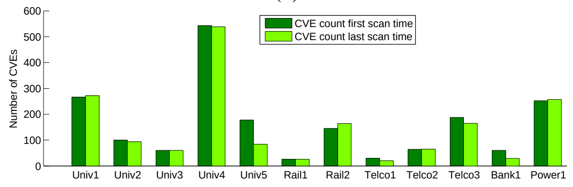


(b)

Figure 9. CPE names discovered by ShoVAT: (a) CPE names across different institutions; and (b) aggregated service-based report.



(a)



(b)

Figure 10. The number of unique CVEs discovered by ShoVAT: (a) the total number of unique CVEs across different institutions from all service scans reported by Shodan; and (b) the total number of unique CVEs across different institutions for the first and last scan of each service.

discovered in $Univ_1$. A summary of this report is provided in Table II.

The results returned by SDA-M also revealed the lifetime of each service where we noticed a fundamental difference between the different types of institutions. By taking the most common type of service, HTTP, for instance, the lifetime of services detected by ShoVAT varies greatly with the type of institution. In the case of Universities included in this study we have found a large number of services with a lifetime of at most 6 months, while in the banking system services seemed to have a longer lifespan (at least for the services included in this study). This pattern is also depicted in Figure 8, where we have included examples on FTP and SMTP services as well. Although in the present assessment we do not provide an in-depth analysis on service lifetime patterns, we emphasize the fact that these results constitute valuable testimonies on the important features delivered by ShoVAT, which in this case are also owed to Shodan's service indexing capabilities.

In the next phase of the assessment we launched ShoVAT's CPE name identification module (CPEI-M) on the results returned by SDA-M. In order to highlight the time-dimension capabilities of ShoVAT, in the following analysis we included the first and last scan reports on each service. For the first time a service was reported, the highest number of CPE names, 205, have been identified for $Univ_1$. Out of these, by selecting only the ones that were assigned the highest match, the number of CPE names was reduced to 89. At the same time, we have found a high number of CPE names for most of the other universities as well. Surprisingly, the analysis revealed 127 CPE names for the power company included in the study and 34 CPE names for $Rail_2$. Conversely, we identified only seven CPE names for $Bank_1$ and only four CPE names for $Rail_1$.

By taking into account the last time a service was reported, we noticed a slight decrease in the number of CPE names for most institutions. The most significant reduction was reported for $Univ_2$, where the number of CPE names decreased from 110 for the first time a service was scanned to 89, for the last scan time. A deeper investigation of these changes revealed software upgrades and configuration changes which limited Shodan's capability to retrieve service banners. A summary of results is provided in Figure 9 (a).

The classification of results based on different service types showed that the largest number of CPE names have been identified for HTTP and HTTPS (see Figure 9 (b)). The time-based analysis also revealed that in case of SNMP there is a significant difference between the number of CPE names found the first time these services were scanned and the last time they were reported. A closer look at the results returned by Shodan showed that these SNMP services are still available, but recent changes in settings, possibly to community access rights, do not permit "public" interrogations.

With this set of discovered CPE names we launched ShoVAT's vulnerability extraction module (VE-M). Taking into account all CPE names from all Shodan scans, we identified a total number of 3922 known vulnerabilities. The distribution across the twelve analyzed institutions is shown in Figure 10 (a). Here it is shown that out of the 3922 vulnerabilities 205 have not been fully matched due to the absence of CPE names in "AND" conditions and 96 were only partially matched by the discovered OS names.

Subsequently, we analyzed the changes in the number of CVEs from the first to the last time a specific service was scanned by Shodan. As expected from the number of discovered CPE names, we have found only minor changes (see Figure 10 (b)). These are mainly attributed to the changes in SNMP configurations, which have already been discussed earlier in this section.

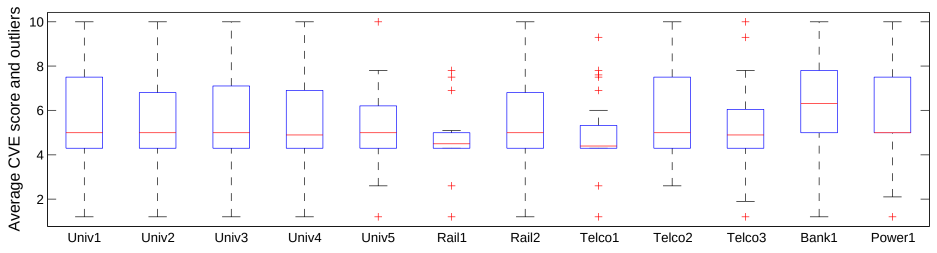


Figure 11. CVE average scores and outliers across different institutions.

Finally, ShoVAT's VE-M module calculated the average CVE vulnerability score of each institution by using the Common Vulnerability Scoring System (CVSS) included in each CVE entry. As depicted in Figure 11, average CVSS scores revolve around 5 and in most cases CVSS scores are spread across the 0-10 spectrum. However, in few cases such as $Rail_1$, $Telco_1$ and $Telco_3$ we find that CVSS scores are concentrated around 5 and exhibit several outliers. More surprisingly, CVSS scores calculated for $Bank_1$ are also spread across the 0-10 scoring spectrum. In this particular case it should be noted that although we have found only two service types, HTTP and HTTPS, $Bank_1$ exposes several vulnerable services with maximum CVSS scores, which might allow attackers to overtake such services and underlying hosts.

The use case assessment presented in this section showed that passive vulnerability assessment based on tools such as Shodan, can be a powerful approach to historically analyze the dynamics of vulnerabilities for Internet-facing services. Nevertheless, it is important to realize that since ShoVAT does not directly interact with the analyzed systems, conclusions based on ShoVAT's results should be carefully formulated. It is also important to realize the limitations of ShoVAT and the fact that it relies on correct and up-to-date scans returned by Shodan. Therefore, ShoVAT can only assess vulnerabilities based on banner information and the service types detected by Shodan. Nevertheless, vulnerability assessment in general encompasses a series of complex steps, each requiring a variety of tools and techniques in order to achieve highly accurate results. ShoVAT should therefore be seen as an important asset and part of a large palette of vulnerability tool-chains which can aid in the passive vulnerability assessment of Internet-facing services.

4.4. Precision analysis

Considering that CVE entry matching relies on correctly identified CPE names, in this section we analyze ShoVAT's ability to correctly identify CPE names. For this purpose we randomly selected 296 service banners ranging from Apache servers to FTP and SSH servers from the twelve different institutions included in the previous study. Then, we manually analyzed each CPE name reconstructed by ShoVAT from the service banners.

The results of this analysis showed that in most cases ShoVAT can accurately reconstruct CPE names. Nonetheless, we have found 23 false positives as well, i.e., CPE names with incorrect highest matching score. The presence of these false positives is explained by banners which include more than one version number and valid keywords from other CPE names. For example, the following banner was included in this analysis (for obvious security reasons, the "HIDDEN-STRING" was used to hide the real name of the server):

```
* ok [capability imap4 imap4rev1 literal+
id auth=login auth=gssapi auth=plain
sasl-ir] HIDDEN-STRING cyrus imap4
v2.3.7-invoca-rpm-2.3.7-12.HIDDEN-STRING5_7.2
server ready
```

In this particular banner for the 2.3.7 version number ShoVAT correctly identified two possible CPE names: cpe_1 =`cpe:/a:rpm:rpm:2.3.7` and cpe_2 =`cpe:/a:cmu:cyrus_imap_server:2.3.7`. However, since the "rpm" keyword is located next to the version number, it receives a higher matching score than the "cyrus" string, which is located further away. Consequently, ShoVAT reported cpe_1 with the highest score, which may be interpreted as a *false positive* result. A total of 23 similar cases have been identified as false positives from the overall 296 analyzed banners, leading to approximately 7.77% false positive rate. It should be noted, however, that ShoVAT's reports include all CPE names and their rankings, which allows security experts to choose between different levels throughout the assessment process.

The false negative rate, i.e., CPE names with incorrect low matching score, attributed to ShoVAT is actually identical to the rate of false positives since for each incorrectly ranked CPE name ShoVAT also reports the correct CPE name (with a lower ranking). Therefore, as already discussed in the previous example, this leads to 23 false negatives, that is 7.77%. Results have been summarized in Table III.

Next, we compared the accuracy of ShoVAT's CPE name reconstruction algorithm to the results returned by the *Nessus* vulnerability scanner tool. With the permission of our University's IT security staff we performed a full

Table III. Manual assessment: CPE reconstruction algorithm's false positive and false negative rates.

Analysis	Number of banners analyzed	Number of false results	Percentage
False positives	296	23	7.77%
False negatives	296	23	7.77%

Table IV. Automated assessment: CPE reconstruction algorithm's false positive and false negative rates.

Nessus CPE	ShoVAT CPE	False positives	False negatives
67	65	6 (8.95%)	8 (11.94%)

scan of 40 Internet-facing hosts within the premises of Petru Maior University. The results of this scan returned 67 CPEs derived from service banners and more than 80 CPEs derived from all service-specific traffic generated by *Nessus*' advanced probing modules. Within these 80 CPEs we find a variety of software configurations and vulnerabilities ranging from SSL/TLS vulnerabilities to DNS server recursive query cache poisoning weaknesses which are out of ShoVAT's scope and capabilities. In this respect the analysis that follows focuses on the 67 CPEs returned by *Nessus* and compares these results with the output of ShoVAT. As tabulated in Table IV, ShoVAT identified 65 CPEs, out of which 6, i.e., 8.95% represent false positive results. As previously discussed, these are a direct result of complex banners that incorporate several tightly coupled version numbers and software names, which are not trivial to analyze even for a human expert. On the other hand, as already mentioned, the number of false negatives should be equal to the number of false positives. However, in this particular scenario we have found two instances of `nginx` service version 1.4.4, which does not have any known vulnerabilities up to date. Nevertheless, *Nessus*'s CPE reconstruction module calculates a possible CPE name and includes this information in the final report. This aspect is illustrated in Table IV where we have shown a possible increase in the number of false negatives from 6 to 8.

Finally, we evaluated the sensitivity of the CPE identification algorithm to randomly selected sub-strings injected into service banners. Recall that this algorithm processes service banners in the search for keywords available in CPEs loaded from NVD. In this respect the algorithm is sensitive to the presence of CPE sub-strings, and more specifically to the number of CPE sub-strings present in service banners. To evaluate the sensitivity of the approach we randomly selected 5000 CPEs from NVD and for each CPE we constructed a custom service banner encompassing the sub-strings of each CPE, delimited by white-space. Then, for each custom banner we randomly selected a CPE with the same version number and we added 50%, 75%, and 100% of its sub-strings into the service banner. With each banner we

Table V. Sensitivity of the CPE identification algorithm to randomly injected sub-strings for 5000 different CPEs.

Sub-strings injected (% of chosen CPE)	Correct CPE identifications	Incorrect CPE identifications
50%	4962 (99.24%)	38 (0.76%)
75%	4856 (97.12%)	144 (2.88%)
100%	4688 (93.76%)	312 (6.65%)

ran the CPE identification algorithm and we recorded the impact on the selected CPE. Obviously, the algorithm is highly sensitive to the number of sub-strings associated to a specific CPE. In fact, as depicted in Table V, in case we inject only 50% of sub-strings, there is an approximate rate of 0.76% of incorrect (false) CPE identifications. However, by increasing the injection rate to 100% the incorrect identification rate also increases to 6.65%. This means that despite the availability of complete sub-strings for two distinct CPEs, the algorithm will weigh differently each CPE according to the number of sub-strings and to their position in the service banner. Nevertheless, it should be noted that the algorithm returns the full set of CPE names, each with a different weight, which are then later used in subsequent NVD vulnerability searches.

Obviously, the outcome of the aforementioned experiments may be affected by software updates, configuration changes, and newly discovered services. At the same time, since *Nessus* is continuously updating its plug-ins, and since Shodan is repeatedly scanning the Internet for new services, the execution of the same set of tests at a different point in time may yield different results, as documented in the previous sub-sections. Nevertheless, the presented experiments rely on statistically significant data sets chosen from a variety of real Internet-facing services. Therefore, we believe that the results presented in this section provide an accurate view on the capabilities and on the precision of the proposed tool.

4.5. Detailed comparison to existing tools

In order to better understand the performances of ShoVAT, its possible limitations and advantages over existing tools, we compared its output with the results returned by the well-known *Nessus* vulnerability scanner tool. More specifically, we installed on a Windows 7 system an Apache Web server, SSH server, FileZilla FTP server, and TightVNC server. Additionally, we also used a Cisco 7200 series router as a target system.

At first, we launched *Nessus* scanner from another station, we performed a scan of the target systems and we generated a *Nessus* report. Then, we extracted the banners from *Nessus* report and we used ShoVAT to identify CPE names and CVE entries within these banners. As depicted in Table VI, in most of the cases the number of CPE names and CVE entries are the same for both tools. ShoVAT accurately identified the CPE name in PHP 5.3.6 module of Apache server, which was also identified by *Nessus*.

Table VI. Nessus versus ShoVAT.

Service	Nessus CPE	Nessus CVE	ShoVAT CPE	ShoVAT CVE
PHP 5.3.6 (Apache Server)	1	37	1	37
PHP 5.3.7 (Apache Server)	1	11	1	11
WeOnlyDo 2.1.3 (SSH Server)	0	0	1!	4!
Cisco 7200	2	37	2(90)	37
FTPd-SSL 0.17	1	2	1	2
FileZilla 0.9.44 (FTP Server)	0	0	0	0
TightVNC server viewer 2.7.2	0	0	0	0

Following this identification, ShoVAT correctly extracted from NVD all 37 CVE entries associated to this CPE. Similarly, we recorded the same accuracy in case of an updated PHP module, version 5.3.7. Nevertheless, in case of the SSH server (WeOnlyDo 2.1.3) ShoVAT reported a CPE name which was not included in the *Nessus* report. A closer analysis revealed the following banner:

```
SSH version : SSH-2.0-WeOnlyDo 2.1.3 SSH supported
authentication : password,publickey
```

We contacted a local security expert and we have been informed that ShoVAT correctly identified `cpe:/a:ssh:ssh2:2.0` as a valid CPE name for the above banner. However, since *Nessus* does not include a plug-in to analyze this particular SSH server's banner, it is not able to identify the CPE and its associated CVE entries. For the remaining services ShoVAT accurately reported the same number of CPE names as *Nessus*. In the particular case of Cisco 7200, ShoVAT returned more than 90 results, however, only two CPE names were ranked with highest scores and were used in subsequent CVE entry explorations.

Next, we experimented with ShoVAT's ability to extract CPE names from customized, i.e., modified, banners. For instance, we have made minor changes to the Cisco 7200 banner by moving the version number from the beginning to the end of the service banner. This had a significant impact on *Nessus*, which was unable to identify CPE names and was unable to report CVE entries. Conversely, ShoVAT's dynamic CPE reconstruction algorithm returned the same CPE names as in the case of the unaltered banner and all 37 associated CVE entries.

Despite the superior precision reported by *Nessus*, as documented in the previous sub-section, we underline once again that ShoVAT belongs to a different class of tool-chains: that of passive vulnerability assessment. In this respect, however, the major advantage of ShoVAT over *Nessus* is best illustrated by its execution time. Figure 12 compares the execution time of ShoVAT and *Nessus* for 40 Internet-facing hosts. *Nessus's* execution time corresponds to full host scans, while ShoVAT's execution time includes full communication and processing times over historical data as well. It can be seen that there is a significant difference between *Nessus* and ShoVAT execution time.

Table VII. Feature-based comparison. We used '●●●' to denote a strong support, '●●' to denote a moderate support, and '●' for a weak support (i.e., unavailable) of a specific feature.

Tool	Active assess.	Passive assess.	Automated CPE, CVE	Custom banner	History assess.
nmap[13]	●●●	●	●	●	●
ZMap[14]	●●●	●	●	●	●
p0f[11]	●	●●●	●	●	●
PRADS[12]	●	●●●	●	●	●
Nessus[10]	●●●	●	●●	●●	●
NetGlean[15]	●●●	●	●●	●	●●●
SinFP[16]	●●●	●	●●	●	●
Hershel[24]	●●●	●	●●	●	●
ShoVAT	●	●●●	●●●	●●●	●●●

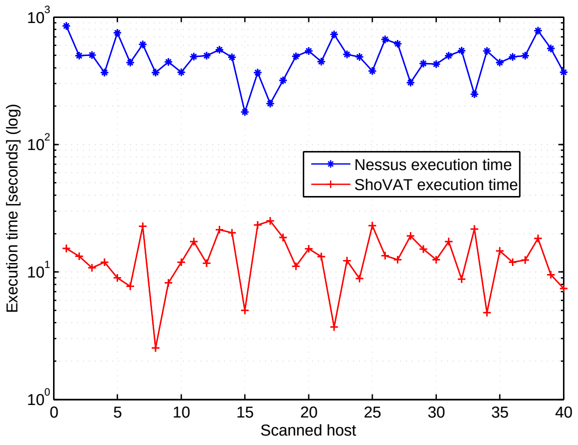


Figure 12. Comparison of Nessus and ShoVAT execution time.

More specifically, *Nessus* requires several minutes (in several cases more than 13 minutes) to fully scan a specific host, while the maximum recorded execution time for ShoVAT is less than 35 seconds. It is noteworthy that in this specific time frame ShoVAT is able to provide full historical reports on the scanned hosts, while *Nessus* provides only the most recent scan results.

Finally, we performed a comparison between the features delivered by other tools and the capabilities offered by ShoVAT. As depicted in Table VII ShoVAT falls into the category of passive vulnerability assessment tools together with *p0f* [11] and *PRADS* [12]. However, compared to these, ShoVAT embodies automated CPE and CVE identification algorithms together with custom banner processing and built-in historical assessment capabilities. Conversely, compared to active vulnerability assessment tools such as *Nessus* [10], ShoVAT does not require the development of plug-ins for the recognition of new CPEs and CVEs, while its built-in features enable accurate results even with purposefully modified banners. At last, compared to *NetGlean* [15], ShoVAT focuses on the passive identification of known CPEs and CVEs, a key feature that is not supported by *NetGlean*.

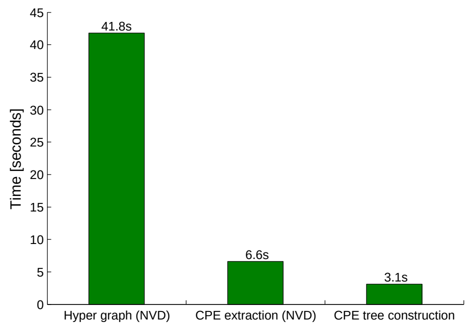


Figure 13. Execution time of pre-processing operations: the construction of bipartite-hyper graph, the extraction of CPE names from the entire NVD, and the construction of CPE trees.

4.6. Execution time

The reconstruction of CPE names and the correct identification of CVE entries are based on properly pre-processed data entries from NVD. ShoVAT assumes that the entire NVD containing all CPE names and CVE entries is available locally in the form of XML files. These are processed at an early stage in order to extract the necessary information and to build ShoVAT's data structures described in the previous sections. The measurements that follow were performed on a Debian 32-bit OS with 4GB RAM and Intel Dual Core 3.0 GHz processor.

Figure 13 depicts the execution time of ShoVAT's pre-processing operations. Here it can be seen that ShoVAT takes approximately 41.8 seconds to build an in-memory representation of NVD, which is then used by VE-M to ensure fast access to NVD's CVE entries. Since the CPE identification (CPEI-M) module operates independently of VE-M, it extracts all CPE names from NVD by using a line-by-line processing of all NVD's XML files. The extraction phase is executed in approximately 6.6 seconds, while the construction of CPE trees takes around 3.1 seconds to execute.

As already stated in the previous sections, the algorithms for the reconstruction of CPE names and for the extraction of CVE entries have a linear complexity. Figure 14 (a) and (b) illustrate a linear behavior for each of the two algorithms. More specifically, Figure 14 (a) shows the execution time of the CPE name reconstruction algorithm based on the number of version strings. It can be seen that for 23 version strings the algorithm runs in 3.07 seconds, while for 200 version strings the algorithm takes 20.7 seconds to execute. At the same time we measured the execution time of the CVE entry identification algorithm, based on the number of CPE names identified in the previous step. As depicted in Figure 14 (b), the algorithm runs in 8.7 seconds for 11 CPE names and in 24 seconds for 119 CPE names. The full memory consumption of ShoVAT, with all modules running, is of approximately 245 MB, which means that only of 6% memory (out of 4GB) is required by ShoVAT to run and perform complex operations on the entire NVD.

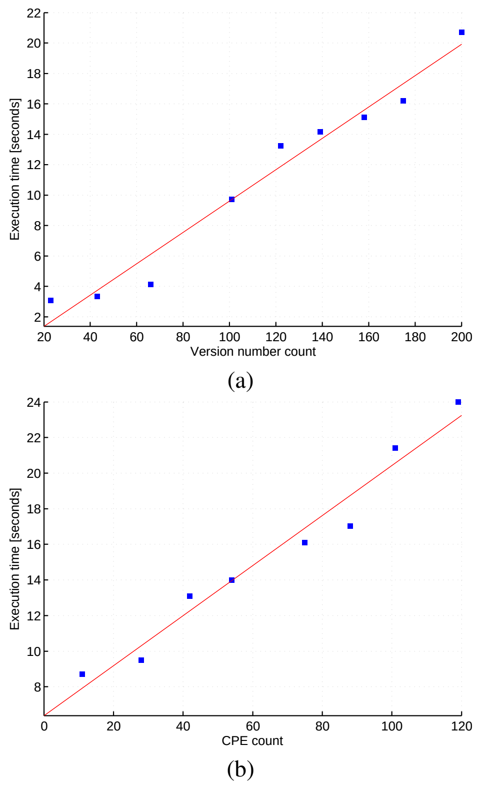


Figure 14. CPE and CVE algorithm execution time: (a) CPE reconstruction based on the number of version strings identified in banner; and (b) CVE entry extraction based on the number of CPE names.

Finally, we mention that the average execution time of communications with Shodan for the acquisition of historical scan results is of 1.4 seconds for each host (see Figure 15). Subsequently, we note that ShoVAT can perform a full host scan in approximately 20 seconds, which includes the reconstruction of CPE names and the extraction of CVE entries for all historical data. At the same time, an entire 24-bit IP network block is scanned for vulnerabilities in approximately 1 hour (depending on the number of discovered hosts).

5. CONCLUSION

In this paper we presented ShoVAT, a unique tool aimed at the automated and passive vulnerability assessment of Internet-facing services. ShoVAT leverages Shodan search engine's indexing capabilities to discover services. It implements specially crafted algorithms to process Shodan results in the attempt to reconstruct key vulnerability identifiers. Vulnerabilities are derived from an efficient in-memory model of National Vulnerability Database (NVD), built on bipartite and hyper graphs.

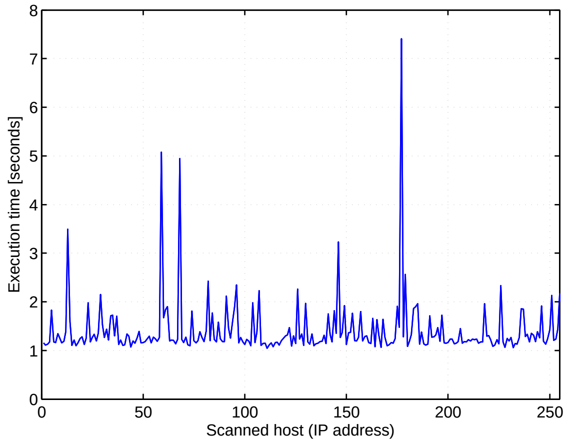


Figure 15. Execution time of communications with Shodan.

Experimental results confirmed the unique features of ShoVAT. Namely, ShoVAT returned the full set of vulnerabilities for all hosts and all historical data in a 24-bit network address block in approximately 1 hour. At the same time, cross-sectoral experiments performed on 1501 services exposed by twelve different institutions revealed a total of 3922 known vulnerabilities, most of which are attributed to HTTP services. Moreover, ShoVAT's capabilities to analyze historical data highlighted minor reductions on the number of discovered vulnerabilities over time, but major configuration changes for specific services, e.g., SNMP.

The results presented in this paper constitute important testimonies on the advanced capabilities brought by ShoVAT to the field. Recall, however, that ShoVAT falls into the category of passive vulnerability assessment tools, which are generally known to have limited capabilities and to depend on the quality of supplied data. Nonetheless, we believe that ShoVAT represents an important asset in the field of vulnerability assessment since: ShoVAT supports the identification of vulnerabilities at early security assessment phases and does not require implementation of active and possibly disrupting probing techniques; ShoVAT integrates historical service analysis capabilities, which do not require the deployment of monitoring infrastructures; and finally, ShoVAT builds on independent modules, which can provide new opportunities to analyze data originating from other sources as well, e.g., network scans.

As future work we intend to expand ShoVAT's capabilities with a graphical front-end which will provide real-time feedback to security experts on the progress of the assessment. At the same time, this interface will ensure a user-friendly and a visually structured environment to display reports.

6. ACKNOWLEDGMENT

The research presented in this paper was supported by the European Social Fund under the responsibility of the Managing Authority for the Sectoral Operational Programme for Human Resources Development, as part of the grant POSDRU/159/1.5/S/133652. The authors thank Flavius Graur for his assistance throughout the development of ShoVAT's code and for his help regarding experimentation with the *Nessus* vulnerability scanner.

REFERENCES

1. Cisco. The internet of things. `http://share.cisco.com/internet-of-things.html` 2014. [Online; accessed December 2014].
2. Genge B, Haller P, Gligor A, Beres A. An approach for cyber security experimentation supporting sensei/iot for smart grid. *Second International Symposium on Digital Forensics and Security*, 2014; 37–42.
3. Ziegeldorf JH, Morchon OG, Wehrle K. Privacy in the internet of things: threats and challenges. *Security and Communication Networks* 2014; **7**(12):2728–2742, doi:10.1002/sec.795. URL `http://dx.doi.org/10.1002/sec.795`.
4. Ghani H, Khelil A, Suri N, Csertan G, Gonczy L, Urbanics G, Clarke J. Assessing the security of internet-connected critical infrastructures. *Security and Communication Networks* 2014; **7**(12):2713–2725, doi:10.1002/sec.399. URL `http://dx.doi.org/10.1002/sec.399`.
5. Leonard D, Loguinov D. Demystifying service discovery: Implementing an internet-wide scanner. *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, ACM: New York, NY, USA, 2010; 109–122, doi:10.1145/1879141.1879156. URL `http://doi.acm.org/10.1145/1879141.1879156`.
6. Leonard D, Loguinov D. Demystifying internet-wide service discovery. *IEEE/ACM Trans. Netw.* Dec 2013; **21**(6):1760–1773, doi:10.1109/TNET.2012.2231434. URL `http://dx.doi.org/10.1109/TNET.2012.2231434`.
7. Matterly J. Shodan. `http://www.shodanhq.com` 2014. [Online; accessed December 2014].
8. Bodenheim R, Butts J, Dunlap S, Mullins B. Evaluation of the ability of the shodan search engine to identify internet-facing industrial control devices. *International Journal of Critical Infrastructure Protection* 2014; **7**(2):114 – 123, doi:http://dx.doi.org/10.1016/j.ijcip.2014.03.001. URL `http://www.sciencedirect.com/science/article/pii/S1874548214000213`.
9. Goldman D. Shodan: The scariest search engine on the internet. `http://money.cnn.com/2013/04/08/technology/security/shodan/`

2013. [Online; accessed December 2014].

10. Rogers R. *Nessus Network Auditing*. Syngress publishing, 2008.
11. Zalewski M. p0f v3: Passive fingerprinter. `http://lcamtuf.coredump.cx/p0f3/` 2012. [Online; accessed December 2014].
12. Fjellskal E. Passive real-time asset detection system. `http://gamelinux.github.io/prads/` 2009. [Online; accessed December 2014].
13. Nmap. `http://nmap.org/` 2014. [Online; accessed December 2014].
14. Durumeric Z, Wustrow E, Halderman JA. Zmap: Fast internet-wide scanning and its security applications. *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, USENIX Association: Berkeley, CA, USA, 2013; 605–620. URL `http://dl.acm.org/citation.cfm?id=2534766.2534818`.
15. Manes G, Schulte D, Guenther S, Shenoi S. Netglean: A methodology for distributed network security scanning. *Journal of Network and Systems Management* 2005; **13**(3):329–344, doi:10.1007/s10922-005-6263-2. URL `http://dx.doi.org/10.1007/s10922-005-6263-2`.
16. Auffret P. Sinfp, unification of active and passive operating system fingerprinting. *Journal in Computer Virology* 2010; **6**(3):197–205, doi:10.1007/s11416-008-0107-z. URL `http://dx.doi.org/10.1007/s11416-008-0107-z`.
17. Beck R. Passive-aggressive resistance: Os fingerprint evasion. *Linux J.* Sep 2001; **2001**(89):1–. URL `http://dl.acm.org/citation.cfm?id=509824.509825`.
18. Shu G, Lee D. Network protocol system fingerprinting - a formal approach. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006; 1–12, doi:10.1109/INFOCOM.2006.157.
19. Medeiros J, Brito J AgostinhoM, Motta Pires P. An effective tcp/ip fingerprinting technique based on strange attractors classification. *Data Privacy Management and Autonomous Spontaneous Security*, *Lecture Notes in Computer Science*, vol. 5939, Garcia-Alfaro J, Navarro-Arribas G, Cuppens-Boulahia N, Roudier Y (eds.). Springer Berlin Heidelberg, 2010; 208–221, doi:10.1007/978-3-642-11207-2_16. URL `http://dx.doi.org/10.1007/978-3-642-11207-2_16`.
20. Caballero J, Kang MG, Venkataraman S, Song D, Poosankam P, Blum A. Fig: Automatic fingerprint generation. *In 14th Annual Network and Distributed System Security Conference (NDSS)*, 2007.
21. Sarraute C, Burroni J. Using neural networks to improve classical operating system fingerprinting techniques. *Electronic Journal of SADIO* 2008; **8**(1):35–47.
22. Yarochkin F, Arkin O, Kydyraliev M, Dai SY, Huang Y, Kuo SY. Xprobe2++: Low volume remote network information gathering tool. *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, 2009; 205–210, doi:10.1109/DSN.2009.5270338.
23. Auffret P. Sinfp. `http://search.cpan.org/~gomor/Net-SinFP/` 2010. [Online; accessed December 2014].
24. Shamsi Z, Nandwani A, Leonard D, Loguinov D. Hershel: Single-packet os fingerprinting. *SIGMETRICS Perform. Eval. Rev.* Jun 2014; **42**(1):195–206, doi:10.1145/2637364.2591972. URL `http://doi.acm.org/10.1145/2637364.2591972`.
25. Nannen V. The Edit History of the National Vulnerability Database. Master's Thesis, ETH Zurich, Switzerland 2012. URL `ftp://ftp.tik.ee.ethz.ch/pub/students/2012-FS/MA-2012-06.pdf`.
26. Cheikes B, Waltermire D, Scarfone K. Common platform enumeration: Naming specification version 2.3. *Technical Report NIST Inter-agency Report 7695*, NIST August 2011. URL `http://csrc.nist.gov/publications/nistir/ir7695/NISTIR-7695-CPE-Naming.pdf`.
27. Prigent G, Vichot F, Harrouet F. Ipmorph: fingerprinting spoofing unification. *Journal in Computer Virology* 2010; **6**(4):329–342, doi:10.1007/s11416-009-0134-4. URL `http://dx.doi.org/10.1007/s11416-009-0134-4`.
28. Richardson DW, Gribble SD, Kohno T. The limits of automatic os fingerprint generation. *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*, AISec '10, ACM: New York, NY, USA, 2010; 24–34, doi:10.1145/1866423.1866430. URL `http://doi.acm.org/10.1145/1866423.1866430`.