Array 3-4 (2019) 100011

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/journals/array/2590-0056/open-access-journal

A survey on vulnerability assessment tools and databases for cloud-based web applications

Kyriakos Kritikos [*], Kostas Magoutis, Manos Papoutsakis, Sotiris Ioannidis

ICS-FORTH, Heraklion, Crete, Greece

A R T I C L E I N F O

Keywords:
Vulnerability
Scanning
Assessment
Survey
Database
Tools
Evaluation
Accuracy
Performance

A B S T R A C T

Due to its various offered benefits, an ever increasing number of applications are migrated to the cloud. However, such a migration should be carefully performed due to the cloud's public nature. Further, due to the agile development cycle that applications follow, their security level might not be the best possible, exhibiting various sorts of vulnerability. As such, to better support application migration and runtime provisioning, this article supplies three main contributions. First, it attempts to connect vulnerability management to the application lifecycle so as to highlight the exact moments where application vulnerability assessment must be performed. Second, it analyses the state-of-the-art open-source tools and databases so as to enable developers to make an informed decision about which ones to select. In this sense, discovering such vulnerabilities will enable to better secure applications before or after migrating them to the cloud. The analysis conducted is quite rich, covering various aspects and a rich sets of criteria. Third, it explores the claim that vulnerability scanning tools need to be orchestrated to reach the highest possible vulnerability coverage, both in terms of extend and breadth. Finally, this article concludes with some challenges that current vulnerability tools and databases need to face to increase their added-value and applicability level.

1. Introduction

Cloud computing is a novel computing paradigm that has revolutionized the way applications are designed and provisioned. It promotes the use of computational and networked resources on demand to support application provisioning. As such, resource management is outsourced to cloud providers enabling to reduce or diminish operational and management costs.

The cloud computing success has also led to raising the abstraction level via the supply of extra service types above the infrastructural ones. In particular, platform services are offered to reduce the burden in creating and managing the execution environments within the leashed resources. Further, the scope of such services has been also extended to facilitate cloud application design and development.

The result from the use of both infrastructure as a service (IaaS) and platform as a service (PaaS) services was that existing applications were migrated to the cloud while new ones popped up, exhibiting the benefit that they can infinitely scale to handle the ever-increasing workloads. The plethora of services at the software level (SaaS) also enables composing such services to create added-value, advanced functionality. As such, nowadays software houses can acquire a plethora of tools and services to rapidly build their applications and offer them in an ever-competing market.

While migration of applications to the cloud continues with a fast pace, there are two prohibitive factors that make organisations hesitant to perform it. The first factor is the lack of standardisation, as each cloud provider offers its own APIs and tools to support application development and provisioning. Thus, adopting one cloud provider leads to a lock-in effect as the application owner becomes bound to that provider and cannot easily migrate to a new one, when, e.g., new business opportunities arise. Fortunately, various frameworks have been proposed to confront this lock-in issue, which enable applications to move from one cloud to another, thus becoming multi-cloud. Avoiding this lock-in effect introduces greater flexibility as there is plenty of space to optimise applications by exploiting in a combined way those cloud services that more optimally satisfy the application requirements.

The second migration factor is security. By giving more control on third-parties, which might not offer services with suitable security guarantees, makes organisations to be reluctant to move to the cloud. Many applications already handle private or sensitive data which will never be moved to the cloud as long as these guarantees are not satisfied. Further, privacy laws might also forbid data movement from certain areas. Yet, a

[*] Corresponding author.
E-mail address: kritikos@ics.forth.gr (K. Kritikos).

https://doi.org/10.1016/j.array.2019.100011
Received 27 June 2019; Received in revised form 3 September 2019; Accepted 11 October 2019
Available online 15 October 2019

cloud provider might not possess datacentres in these areas or guarantee that data might not be stored outside of them.

Virtualisation also impacts security as physical hosts are shared by users in the form of virtual machines (VMs). This can raise issues where malicious users can see the data stored in a VM or take control over all VMs by attacking the hypervisor. Further, as applications are distributed over public clouds, networked attacks can be performed over them which can, e.g., cause crucial information to be leaked during transmission or application overload based on denial-of-service attacks.

While all these are valid security issues, research has led to particular achievements in the cloud security area such that we could even reason that public clouds might be even more secure than private ones. A new breed of application development methodologies has been also recently worked out, enabling to secure an application by design, enabling to adopt security services that best match the designed application's nature and structure.

Security services can be of a different nature, while they be also reactive or preventive. Reactive services can detect an attack and possibly address it. For example, intrusion detection and protection systems can be used to protect applications from network attacks by, e.g., detecting denial-of-service attacks and blocking their origin IPs. Preventive services prevent a security incident from happening by detecting application areas that are vulnerable to certain attacks. The latter service kind can take different forms.

Vulnerability scanning tools enable to detect vulnerabilities in different application parts and kinds. Static (code) analysis tools can find code bugs exploitable by security attackers. Audit tools can be utilised to find well-known rootkits, Trojans and backdoors as well as to unveil hidden processes and sockets. Finally, antivirus tools can detect viruses which either attempt to infect or have already infected the underlying operating system (OS).

This article focuses on vulnerability scanning tools and databases. The reason for this is that, while malware and antivirus software is well-known and adopted by both organisations and individuals, vulnerability scanning tools are not widely used in practice. Further, it is hard for a practitioner to choose the right vulnerability scanning tool due to the great tool diversity and varied coverage. As such, this article aims to guide the practitioner in making an informed decision about which vulnerability scanning tool and database to select for his/her application. Such a selection relates also to particular challenges which have not been currently confronted in research, thus supplying an invaluable contribution to the academic community. In addition, this article contributes towards the identification of those places in the application lifecycle where vulnerability scanning can be performed and supplies valuable insights towards better securing cloud applications. Finally, this article investigates and proves that vulnerability scanning tool orchestration can be the answer towards higher vulnerability detection coverage.

The rest of the article is structured as follows. The next section formally defines what is a vulnerability scanning tool while explicates its architecture and the vulnerability management lifecycle. Section 3 attempts to set up and explain the article's main research goals and methodology. Sections 4 and 5 evaluate the identified vulnerability scanning tools and databases, accordingly, based on a certain evaluation framework and analyse the main results produced. Finally, Section 6 explains the main research challenges involved in the quest for protecting a cloud application from security vulnerabilities. Finally, the last section concludes the paper.

2. Vulnerability management lifecycle and architecture

A vulnerability scanning tool aims at finding problematic parts of an application or its system that make it vulnerable to security attacks. In this sense, a vulnerability is a kind of malfunction, misconfiguration or security gap in general that might affect the whole system hosting the application, including the VMs, OSs, application code or any other system component. For instance, an OS might suffer from well-known security issues, while the application might use a third-party software suffering from security bugs.

Internally, a scanning tool employs a set of rules to perform the scanning which indicate how the scanning can be performed and when a certain vulnerability is raised. A scanning tool can either offer a rule set extensible by users, or it can cater for higher modularity by formulating different rule sets to match different application kinds or scanning situations.

The scanning via a tool can be performed in two different modes. In the internal mode, the tool is installed inside the application and attempts to scan its components, hosted in VMs or containers. Such a mode can address any application component kind due to the potential use of both static and dynamic scanning methods. Thus, its scanning accuracy is increased. Further, it leads to reduced communication overhead as most of the communication occurs within the same hosting environment. However, it can have the greatest impact in terms of interfering with the application performance as it can steal precious resources from the same hosting environment. In the external mode, the scanning is performed externally to the application and relies on the endpoints on which the application web-based components reside. As such, only the latter component kind can be confronted. This scanning mode has the advantage of conducting the scanning without interfering internally with the application hosting environment. However, it might not be able to attain the highest possible scanning accuracy as it cannot exploit static scanning methods while it increases the communication overhead in the application system.

Fig. 1 depicts a reference architecture for a vulnerability scanning tool, showcasing both its main components and different operation modes. The architectural variability that can be observed across all vulnerability scanning tools (in terms of two architecture classes) is also depicted, which is is explained in the next paragraph.

In a client-server architecture, a server remotely orchestrates the execution of different agents, either residing on the application VMs/containers to perform the scanning internally or on different VMs to conduct an external scanning. Such an architecture can more easily scale while it can accelerate the scanning by orchestrating the parallel agent execution. In a standalone architecture, the scanning tool is installed in one place, either internal or external to an application, and then initiated to perform the scanning. Such an architecture variant/class has the main deficit that the orchestration of the installation, execution and report merging of the respective vulnerability reports burdens the user.

In any case, as also shown in Fig. 1, each architecture alternative can map to a different scanning mode which means that the two main

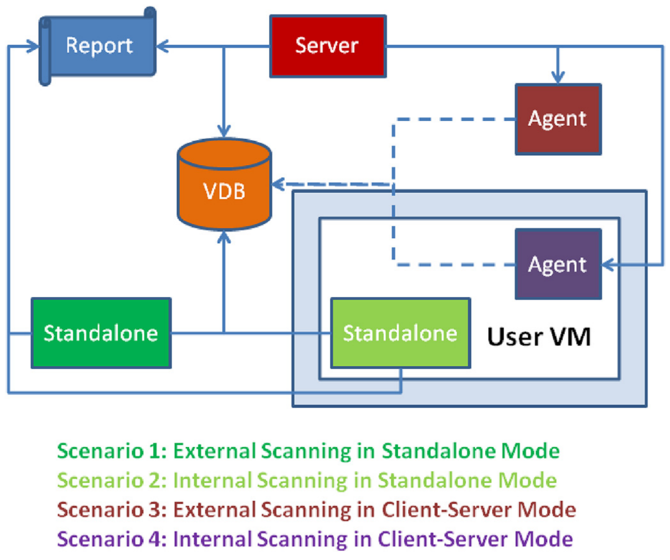
Report
Server
Agent
VDB
Agent
Standalone
Standalone
User VM
Scenario 1: External Scanning in Standalone Mode
Scenario 2: Internal Scanning in Standalone Mode
Scenario 3: External Scanning in Client-Server Mode
Scenario 4: Internal Scanning in Client-Server Mode

Fig. 1. The different vulnerability scanning architecture alternatives.

architecture classes, i.e., client-server and standalone, are orthogonal to this mode. This is depicted through the use of different colours in the respective components involved mapping to the four possible architecture scenarios (associated with all possible combinations between the two scanning modes and architecture classes).

A scanning tool might also exhibit the functionality to assess the relevant application exposure risk with respect to the kind of vulnerabilities discovered and their criticality. Such a risk can not only indicate the need to modify the application system but also those parts that need immediate handling. Fig. 1 depicts this tool capability with the sole architectural difference being that in the standalone mode, the report is produced directly while in the client-server mode, the report is compiled by the server after merging the vulnerability results produced by the agents/clients.

As such, vulnerability databases (VDBs) are usually employed to cover the information to be included in a vulnerability scanning report. VDBs usually include some common attributes dedicated to properly identifying and explaining vulnerabilities. They might also include extra information, like which artefact is affected and how the vulnerability could be addressed. Many VDBs also conform to standards that support a kind of standardised identification or determination of the information to be covered.

Fig. 1 highlights two different connection kinds between a VDB and a scanning component. In case of a standalone scanner or server, the continuous lines indicate that the VDB is mainly used to produce the final scanning report. In case of a standalone scanner, the VDB could be also used to define how the vulnerability can be detected. On the other hand, the dash lines between the scanning agents and VDB indicate that the VDB might not always be used by them. In some cases, two main optional usage scenarios can hold: (a) the agent can obtain the scanning rules from the VDB; (b) it might use the VDB to construct its partial report to be sent to the server.

Fig. 2 showcases the lifecycle of vulnerability management which involves the following activities:

1. *Scanning tool installation*: in this activity a system admin also chooses the right scanning architecture that more properly matches the application system architecture and the kind of application to be scanned.
2. *Scanning configuration*: the admin then attempts to configure appropriately the scanning tool. She might use only a subset of all rules or a certain rule set in case of modular tools. She can also point the exact places where scanning needs to be performed.
3. *Scanning execution*: the admin executes here vulnerability scanning.
4. *Vulnerability report inspection*: the vulnerability report produced by the scanning tool is first inspected by the application owner to evaluate the overall application risk and determine the strategy for reducing it.
5. *Vulnerability handling*: this strategy is then materialised by the system admin who takes an action first on the most critical vulnerabilities. Such an action either takes the form of applying recommended remedies from the report, if they exist, on the affected application system component, or first searching to find ways to confront the vulnerability and then applying the most suitable one. The action executor needs not be solely the system admin. The admin is mostly responsible for application deployment and provisioning, e.g., on vulnerabilities covering the OS level or the application execution environment in general. However, the application devop can also be involved, when issues regarding the application code must be confronted.

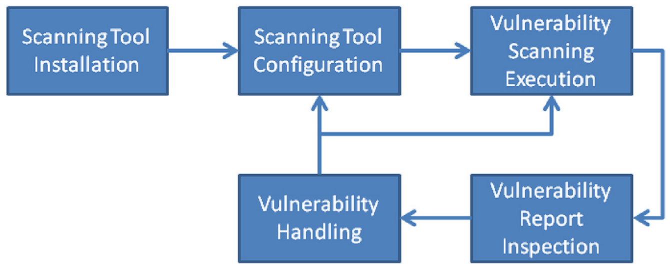
Scanning Tool Installation
Scanning Tool Configuration
Vulnerability Scanning Execution
Vulnerability Handling
Vulnerability Report Inspection

Fig. 2. The vulnerability management lifecycle.

Once the handling is over, scanning can be re-executed to check that the vulnerabilities in focus have been corrected. Further, scanning might also need reconfiguration. For instance, it might be possible that one vulnerability is too coarse-grained and a more focused scanning needs to be performed to enable detecting more concrete vulnerabilities and subsequently handling them. It might be also possible that the overall handling strategy prescribes that the scanning frequency needs to be aligned with, e.g., the application modification pace.

The above lifecycle analysis highlights that various actors or roles in the user application management are also involved in the vulnerability handling and management. Further, the application management lifecycle is connected to the vulnerability management lifecycle, a logical consequence of the fact that vulnerabilities are not only related to an application but also affect its development and production process. Fig. 3 depicts such a connection, where vulnerability management is part of two application lifecycle activities:

- *Testing* in the sense that vulnerability scanning can be considered as part of application security testing. Such a testing is performed once an application is deployed and executed in an integration environment such that it can be fully tested according to multiple aspects apart from the security one, such as the (functional) integration aspect.
- *Application Monitoring & Adaptation* Once an application has been deployed and executed in a production environment, it must be monitored and adapted according to the *current* problematic situation. As such, scanning must be continuously performed as there can be configuration changes, OS updates or even security incidents that might have taken place such that they must be checked for vulnerabilities to be then properly handled. In overall, an application is a living organisation that can be affected by various changes that occur within its system. As such, we could consider scanning as part of dynamic application testing at runtime, already promoted as a kind of preventive application monitoring and adaptation measure in literature.

In overall, we foresee two different paths via which vulnerabilities can be handled, connected to changes on the application based on its management lifecycle. The first and simpler path indicates that

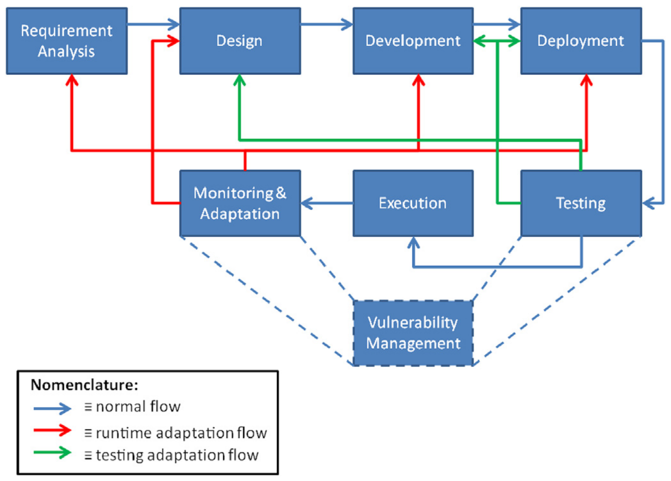

Fig. 3. The application management lifecycle.

vulnerabilities only affect the application component code. As such, only certain application components will need to be modified, thus going back to the *Development* lifecycle activity. Then, the physical management flow will be followed leading to application redeployment and testing to check whether the vulnerabilities have been properly addressed without breaking the application integration.

The second path is followed when vulnerabilities affect multiple components in the application system. In this case, we need to both change application components as well as the way the application is packaged, deployed and configured. This then will affect both *Development* and *Deployment* activities. The changes can be so critical that might affect the application design while they can lead to producing a new application version. This is not so exceptional as we have seen in the past holistic changes on applications that attempt to secure them more than before.

3. Survey goals & tool selection process

3.1. Survey goals

As Section 1 indicated, there is currently no guidance in properly selecting the right tools to identify application vulnerabilities. Such a selection is usually performed via a manual process that can involve searching the web to find the right tool, inspecting the tool features and possibly getting in contact with the tool provider to find the most suitable pricing model matching the application risk requirements.

Based on this manual and time-consuming practice, this paper aims at evaluating vulnerability scanning tools against a criteria-based framework. Such an evaluation can unveil which tools prevail and according to which aspects. We put special focus in the latter case on the trade-off between scanning accuracy and time to which we add the vulnerability coverage dimension.

Apart from the scanning tools, the selection of a database is crucial for obtaining the right information about a vulnerability which not only conforms to standards but is also frequently updated, by relying on the fact that new vulnerabilities are constantly identified each year. Such information can then constitute the basis for formulating the most suitable mitigation strategy.

We concentrate mainly on open-source vulnerability scanning tools and databases. Our rationale is twofold: (a) open-source model is well adopted in the market where more than 90% of software is available either in this or a dual open-source and proprietary form. This model is critical for both maintaining the software's sustainability and extending it based on community contributions, especially for small or medium enterprises; (b) for practical reasons, it is impossible to experimentally evaluate a proprietary tool without purchasing it, while such an evaluation was required as Section 5 shows.

In practice, each tool might come with a different performance over the three aforementioned dimensions. It might be also specialised over different application or component types. As such, it may be possible that not just one but multiple tools might be required to achieve a certain vulnerability detection goal. Thus, this article also aims at investigating both the tools differentiation and complementarity to assist in their possible orchestration.

Based on the above analysis, the main research questions that this article attempts to answer are the following:

- Q_1: Which are the top open-source vulnerability databases and according to which aspects?
- Q_2: Similarly, which are the top open-source vulnerability scanning tools and according to which aspects?
- Q_3: How complementary are the open-source vulnerability scanning tools and is their orchestration meaningful?

3.2. Survey methodology

To answer the above questions, a survey over the state-of-the-art was conducted through a methodology comprising the following steps:

- *tool search*: search procedure for discovering and selecting the right vulnerability scanning tools and databases
- *evaluation method & criteria determination*: procedure for identifying the right evaluation methods and criteria
- *evaluation & reporting*: evaluation, analysis & result reporting

In the following subsection, we focus more on the first two steps. The last step is the subject of the next two sections of this article.

3.2.1. Tool search & selection

A certain procedure for tool search and selection was followed, comprising two main steps: (a) actual search of the tools; (b) selection of the right tools from those discovered based on a set of inclusion criteria.

The search relied on a two-front approach. In both fronts, we have relied on a set of keywords deemed as most relevant for the search. This set was slightly differentiated depending on the concerned artefact (database or tool).

For vulnerability databases, the keywords used and combined in a logical form were the following: *vulnerability AND* (*database OR repository OR store*). Similarly, for the scanning tools the keywords were the following: *vulnerability AND* (*scanning OR detection OR discovery OR search*) *AND* (*tool OR software OR prototype OR framework*.

As it can be seen, the keywords were split into two or three categories depending on the artefact concerned. The first category is related to the research subject so it maps to the sole term *vulnerability*. The second category concerned the characterisation of the searched artefact, i.e., the research object. In this case, different alternative terms were used to capture all expression possibilities. The third category was only employed in the context of tool search to explicate the artefact's specialisation, i.e., what it can perform in terms of the research subject. In this case, we focused on providing alternative terms that would explicate the discovery of a vulnerability.

The first search front was the wide Internet. This front simulated how an application administrator would search for a vulnerability scanning tool or database. Thus, we have employed Google as the well-known and most efficient web search engine to apply the keyword search for each artefact. This was proceeded by snowball search [1] by following the cross-references in a tool's page to other related tools. While inspecting each tool web page, we had already in mind recording information for those criteria that will regulate the tool/database selection.

The second search front was formulated to cover prototype tools or databases from academia that cannot be found easily from a web search. Such tools could be increasingly improved and extended due to the relative research and development tasks conducted by the research organisations producing them. To discover such tools, we employed well known academic data sources, i.e., Web of Science and Scopus, enabling us to have a full literature coverage.

Once all tools and databases were discovered, they were filtered by employing a set of both inclusion and exclusion criteria. Such criteria kinds were split based on the two followed fronts due to the nature of their findings.

The inclusion criteria from the web front were the following:

- A tool should perform at least vulnerability scanning
- A database should cover at least the basic description of vulnerabilities
- The artefact (tool/database) should be open-source
- The artefact has some sort of community behind, even if it is quite small. Thus, we do not favour tools developed by individuals as such tools will tend to have limited capabilities and coverage.

- The artefact last development must have been performed in last 8 years.

While the first four criteria cover quite logically the aforementioned research scope, the last criterion guarantees that only a non-outdated tool is selected by considering the rapid development and changes in the vulnerability area.

On the other hand, the academic front inclusion criteria were as follows:

- Only peer-reviewed articles must be considered
- The article should have been published since 8 years from now.
- The article should deal with the proposal of a vulnerability database or scanning tool
- Quality assessment: we have maintained only articles that were easy to understand and had a suitable quality and contribution level.

The exclusion criteria for the web front were the following:

- The web page is not in English
- Artefacts with suspicious web pages
- Artefacts having web pages with no documentation
- Artefacts for which the source code is not available or does not compile any more or has been tampered

Finally, the exclusion criteria for the academic front were the following:

- Inaccessible articles: An email was sent to their authors to obtain a private copy of them. In case of no answer, the articles were rejected.
- Articles written in a different language than English
- Very short articles (e.g., posters) with a limited contribution

3.2.2. Evaluation criteria & method determination

For the survey on vulnerability databases, we focused on a comparative evaluation based on the very nature of research question Q_1. Thus, we created a set of evaluation criteria which were either devised by us or drawn from the literature. All criteria were produced by considering two aspects: *information coverage*, *capabilities &* support. More details are supplied in Section 4.

To cover research question Q_2, a similar approach was followed for the vulnerability scanning tools, resulting in producing another set of evaluation criteria. However, it was impossible to evaluate the selected tools with respect to criteria like vulnerability coverage, scanning time and accuracy based only on their documentation. To this end, it was decided to select a benchmark-based evaluation approach for these criteria, concentrating on a small subset of tools, with the rationale that: (a) we just need to prove the need for the tools' orchestration according to research question *Q3* as well as their complementarity; (b) it was impossible to evaluate all tools according to the benchmark as this would require a huge effort, involving: reading the technical documentation of the tools, downloading and installing them, running them over the benchmark, and translating their results to a form processable by the benchmark to evaluate the respective criteria.

To summarise, two comparative evaluations were performed for the first two research questions plus a benchmark-based evaluation for the last, thus adopting one evaluation method for each of the research questions posed.

4. Vulnerability database analysis

4.1. Evaluation criteria

As the previous section indicated, we focused on devising a set of criteria concerning the aspects of *capabilities &* support and *information coverage*.

Information coverage criteria attempt to assess whether all necessary information aspects are covered for a vulnerability. They include the following:

- *scope* – what kinds of vulnerabilities are covered by a VDB
- *impact & risk* – what is the impact and risk of the vulnerability
- *resolution* – how this vulnerability can be resolved
- *vendor* – what is the vendor with products affected by the vulnerability
- *products* – what are the main products affected by the vulnerability
- *exploit* – information related to how the vulnerability can be exploited
- *categorisation* – ability to provide an hierarchical vulnerability categorisation to support a more user-intuitive browsing.
- *relations* – relationships with other vulnerabilities; they could be used to, e.g., understand how vulnerabilities can be chained such that they can be more effectively addressed by the respective defence system.

The second aspect concerns what are the main VDB capabilities in terms of interfacing, standards support, information freshness plus user support, which comes via not only VDB documentation but also an active community, able to answer any enquiry related to the VDB and its usage. As such, the following criteria have been considered:

- *standards* – the supported standards. The higher is the number of standards supported, the better is the VDB suitability.
- *community* – who is behind and updates the VDB. Maintaining a big community ensures VDB sustainability and extensive coverage while caters for high-quality entries due to cooperative curation by experts.
- *interfacing* – the offered VDB interfaces which should enable the suitable search of the information provided plus information pushing mechanisms for retrieving up-to-date vulnerability information, catering for better securing a system against even the most recent vulnerabilities.
- *freshness* – the up-to-dateness of the supplied vulnerability information.

4.2. Analysis of evaluation results

The evaluation results are summarised in the form of two tables, explicating how well a certain VDB satisfies the two considered aspects, respectively. In the first from these two tables, the following symbols have been utilised: (a) '✓': indicates full satisfaction of a criterion; (b) '~': signifies partial coverage of a criterion; (c) '-': indicates that the criterion is not supported at all.

The results of Table 1, mapping to the first criteria partition, indicate that the best approach in information coverage is Vulcan followed in 2nd place by 2 other approaches: HPI-VDB and vFeed. Vulcan needs to better handle only the *exploit* criterion which is partially satisfied. On the other hand, the other 2 approaches need to cover completely 2 criteria. In overall, we can see that Vulcan is very close to an ideal approach for this criteria category, quite normal if we consider that it is ontology-based and thus designed to well integrate and correlate different information pieces.

To unveil some patterns from the results, we can indicate that most approaches focus on addressing the first 5 criteria while the rest are rarely covered. These 5 criteria can be regarded as the most essential for well characterising vulnerabilities. The rest have an added-value focus on clarifying how a vulnerability can be exploited as well as in categorising and relating it with other vulnerabilities. The former is an excellent information source, leading to equipping a certain tool with extra vulnerability detection capabilities. The latter criteria enable to nicely categorise vulnerabilities for more user-intuitive browsing and exploration while also unveil relations between vulnerabilities, to be exploited to conduct more advanced exploit forms. This relationship knowledge can also assist in detecting a vulnerability's root cause by, e.g., identifying the first vulnerability in the detected vulnerability chain.

Table 1
Vulnerability databases evaluation summary according to information coverage.

DB	Scope	Impact & Risk	Res.	Vendor	Products	Exploit	Cat.	Relations
CVE [2]	Any	–	–	–	–	–	–	–
Seven Pernicious Kingdoms [3]	SW security errors	–	–	–	–	–	✓	–
OWASP Top Ten [4]	App risks	Risk	✓	–	–	–	–	–
NVD [5]	Any	CVSS [6] & version 3 metrics	✓	–	✓	–	✓	–
CWE [7]	SW security weaknesses	Consequence effect incl. technical impact	✓	–	~	~	✓	✓
OSVDB [8]	Any	–	✓	–	–	–	–	–
HPI-VDB [9,10]	SW	CVSS	✓	✓	✓	–	–	✓
Vulcan [11]	Any	CVSS	✓	✓	✓	~	✓	✓
Vulnerability & Exploit DB [12]	Any	Severity & CVSS	✓	–	–	–	–	–
vFeed [13]	Any	CVSS	✓	–	✓	✓	✓	–
Embedded Vulnerability Detector [14]	jar & class	–	–	✓	✓	–	–	–
SecurityFocus Vulnerability DB [15]	SW	–	✓	–	✓	~	–	–
Vuldb [16]	Any	CVSS	✓	–	✓	✓	–	–
SecurityTracker [17]	Any	Impact	✓	✓	✓	–	–	–
ZeroDayInitiative [18]	Any	CVSS	✓	✓	✓	–	–	–
ExploitDatabase [19]	Broad[a]	Severity	–	–	✓	✓	–	✓
ICS-CERT [20]	Any	CVSS	✓	✓	✓	–	–	–
Japan Vulnerability Notes [21]	Any	CVSS	✓	✓	✓	–	–	–
AusCERT Bulletins [22]	Any	Impact	✓	✓	✓	–	–	–
CERT-EU Security Advisories [23]	Any	Impact & CVSS	–	✓	✓	–	–	–

[a] Remote, web, application, local and privilege escalation, DDoS & PoC.

Most VDBs focus on any kind of vulnerability; very few have a more restrained scope. For risk assessment, many VDBs adopt Common Vulnerability Scoring System (CVSS) while very few just provide an impact measure. Some VDBs supply both impact and risk information for each vulnerability.

Finally, only Common Weakness Enumeration (CWE) offers a very good vulnerability categorisation in form of a deep hierarchy. This contrasts the other approaches that focus mainly on supplying concrete vulnerabilities at the leaf level of such hierarchy. For instance, Vuldb advertises covering around 114198 leaf vulnerabilities. However, in our view, all hierarchy levels are important. As such, it is imperative that a VDB integrates its content with CWE to enable producing such a rich hierarchy. Such an approach is followed only by very few efforts, including Vulcan, as Table 2 shows.

Table 2
Vulnerability databases evaluation summary according to the remaining aspects.

DB	Community	Interfaces	Freshness	Standards
CVE	CVE Numbering Authorities, CVE Board, CVE Sponsors, ITU-T, FIRST, NIST, DISA	File, Twitter Feed, Change log, NVD, Web Search	Daily	No
Seven Pernicious Kingdoms	Fortify Software	Web search and hierarchical browsing	Static	No
OWASP Top Ten	46000 + participants, > 65 corporate organisations, many academic supporters	github, wiki page	Every 3 years	No
NVD	NIST	web search, file, data feeds, hierarchical browsing, change log	Daily	CVSS, CPE [24], CVE, CWE
CWE	researchers & academic representatives, industry, US Department of Homeland security	web search, hierarchical browsing, file	Continuous updates	No
OSVDB	OpenSecurityFoundation, High-Tech Bridge, RiskBasedSecurity Black Duck Software, RTS Labs, ISECOM, DVI	open-source relational DB, alterting & logging	Shutdown in 2016	No
HPI-VDB	Hasso-Plattner Institute	API, web search	Daily	CVE, CWE, CVSS, CPE
Vulcan	University of North Texas, NIST	Text-based search over constructed KB via SNLP techniques	On-demand via NVD, XML feed	CVE, CWE, CPE, CVSS
Vulnerability & Exploit DB	Rapid 7	web search, web list	Daily	CVSS
vFeed	vFeed.io	API, SQLite DB	Daily	Several[a]
Embedded Vulnerability Detector	redhat	Canonical DB, clients (maven, ant, java jenkins, eclipse), API	Not freq. updated	CVE
SecurityFocus Vulnerability DB	SecurityFocus	Web search	Daily	CVE
Vuldb	crowd-based	web search, API, RSS	Daily	CVE,CPE,CVSS
SecurityTracker	SecurityTracker	Web search, RSS	Daily	CVE
ZeroDayInitiative	TippingPoint, DVLabs, researchers	Web search, RSS	Freq. within a month	CVE, CVSS
ExploitDatabase	Offensive Security	Web search, DB repository with linux client, RSS	Daily	CVE
ICS-CERT	US Department of Homeland Security	Web search, RSS	Almost daily	CVE, CVSS
Japan Vulnerability Notes	JPCERT Coordination Centre, Information-technology Promotion Agency	Web search, RSS	Almost daily	CWE, CVE, CVSS
AusCERT Bulletins	AusCERT	Web search, RSS	Daily	CVE
CERT-EU Security Advisories	CERT-EU	Web search, RSS, social media	Daily	CVE, CVSS

[a] CVE, CWE, CPE, CVSS, CAPEC [25], OVAL [26], WASC [27].

Table 3
Evaluation results on support aspect.

Tool/ Approach	Standards	Community	Freshness
Grabber [29]	–	github	2015 last commit
GoLismero [30]	CWE, CVE	github	6 months
Nikto2 [31]	–	github	Very frequently
Vega [32]	–	Subgraph, github	6 months
Wapiti [33]	–	sourceforge	2013 last commit
OWASP Xenotic XSS [34]	–	OWASP, github	4 months
OWASP ZAP [35]	CWE, WASC	OWASP, github	Very frequently
OpenVAS [36]	OVAL, CVE, CVSS	Greenbone, German Federal Office for Information Security (BSI), DFN-CERT, Github	Very frequently
Arachni [37]	CWE	Github	Very frequently
IronWASP [38]	CWE	Github	2015 last commit
w3af [39]	–	Github	Very frequently
OpenSCAP [40,41]	SCAP [42], OVAL, CVE CPE, CCE [43], XCCDF [44]	Github, Red Hat, NIST	Very frequently
Clair [45]	CVE, CVSS	Gitnub, CoreOS	Very frequently
Vulcan [11]	SCAP, CVE, CWE, CPE, CVSS	University of North Texas, NIST	Community-driven
HPI-Vuln [46]	CVSS, CWE, CVE, CPE	Hasso-Plattner Institute	Community-driven
UC-OPS [47]	CVSS, CVE, CWE, CPE, OVAL	University of Marburg	Community-driven
AWS-Vuln [48]	CVSS, CVE, CWE, CPE	EURECOM, Northeastern University	Community-driven

Table 2 depicts the evaluation results for the *capabilities & support* criteria partition. These results indicate that there is no clear winner. NVD can be distinguished due to its community support and wide adoption. vFeed.io can be discerned due to its extensive support to standards and its rich interfaces (API & database). Further, it supports rapid threat response development which comes via pinpointing security scripts of vulnerability scan tools, referencing exploit information utilisable for automated testing via penetration tools like metasploit, applying effective defence rules (e.g., via Snort) and discovering relevant patches and hot fixes via online web crawling.

In the following, we analyse the results per each criterion in separate paragraphs while the final paragraph attempts to nominate the best approach according to all criteria partitions.

Community. The best possible community support seems to come from CVE while the second best is OWASP Ten. In rest of the efforts, the community is smaller and sometimes confined in the form of a single organisation.

Interfaces. The most common interface is web-based search. In some cases, we can see feed mechanisms and APIs. A database is scarcely supplied.

Freshness. Community support aligns with this criterion as VDBs with excellent or good community support, also provide frequent updates to their content. However, there is also correlation with the interface mechanism provided as a feed mechanism unveils frequent VDB updating. In overall, the need to provide fresh vulnerability content is recognised by all VDB efforts.

Standards. We investigate both the most widely supported standard and the VDB that adopts most vulnerability assessment standards. As such, we can observe that the Common Vulnerability Enumeration (CVE) is a clear winner followed by CVSS. Such results are well expected as these two standards concern major information aspects for vulnerabilities spanning their identification and risk assessment. Common Platform Enumeration (CPE) and CWE seem to come third. CPE is a standard supporting platform enumeration. Correlating CPE with vulnerability information enables mapping vulnerabilities to their affected entities. While very few VDBs support CPE, many VDBs establish a correlation of vulnerabilites to IT products.

vFeed is the VDB supporting most standards; not only the aforementioned ones but also others like: (a) OVAL, the open vulnerability and assessment language; (b) CAPEC, the Common Attack Pattern Enumeration and Classification; (c) the Web Application Security Consortium (WASC) threat classification. vFeed is followed by NVD, HPI-VDB and Vulcan, supporting the 4 most common standards: CPE, CWE, CPE & CVSS.

Best Vulnerability Database Nomination. By considering both criteria partitions, Vulcan, the first partition winner, is not the best possible VDB. On the contrary, vFeed is nominated as the best as it offers a very good information coverage and top support for the second partition criteria. Vulcan is positioned second due to its excellent information coverage and its good standards support. HPI-VBD and NVD are finally positioned as third. The first is an academic effort with good information coverage by focusing on correlating information via exploiting academic expertise and knowledge, it supports a good number of standards and offers a good interface support with extra services on top, including a programmatic API or self-diagnosis and attack graph creation. NVD, on the other hand, is widely adopted, has the same standards coverage, a good interface level and vast community support. Thus, while a plethora of interesting VDBs exists, only 4 of them can be distinguished as the most promising.

5. Vulnerability assessment tool analysis

5.1. Comparative evaluation

5.1.1. Criteria framework

For comparatively evaluating the discovered tools, a set of criteria were devised, clustered in 3 categories: support, functionality and configuration.

The support category includes the following set of criteria concerning the level of support and updating of the vulnerability detection tool.

- *standards*: signifies the standards supported by the tool. Such a support enables not to develop everything from scratch plus interoperability with external tools/software. It also relates to community creation/maintenance as usually some communities evolve around standards.
- *community*: the community behind a tool can support it by identifying bugs, supplying fixes or extensions, and developing complementary tools, thus catering for its proper evolution.
- *freshness*: it indicates how frequently the tool is updated, either in terms of improved versions, fixing detected bugs, or extensions. In the context of vulnerability assessment, all these aspects are critical. Bugs can stop a vulnerability assessment tool from functioning. While the reduced or non-existing ability to evolve can mean that the tool is not able to rapidly provide support for detecting new vulnerabilities.

The functionality category includes the following set of criteria, related to certain important and critical features that a tool must exhibit.

- *categorisation*: One or more categories that characterise a tool's functionality, derived from the OWASP taxonomy [28], detailed in the appendix. Depending on its functionality, a tool might belong to multiple categories. This can occur not only for single tools but also tool agglomerations, where each component tool might focus on delivering a different kind of vulnerability detection functionality.

Such tool agglomerations are further inspected by the *Tool Coverage* criterion. Finally, as all tools can perform vulnerability detection, they trivially map to the *vulnerability scanning* category which is omitted from the analysis.

- *Object*: this criterion investigates the kind of object/component scanned by a tool. A tool might focus on one component kind or multiple. Thus, the higher is the number of component kinds, the better is the tool.
- *Vulnerability Coverage*: attempts to evaluate a tool's with respect to the vulnerability kinds it can detect. In other words, it assesses the percentage of all possible vulnerability kinds covered. To derive this percentage, we have followed an approach, detailed in Appendix B, comprising (3) main steps: (a) identification of all possible web application vulnerability kinds by relying on SecToolMarket.com; (b) assessment of the percentage of vulnerability kinds covered by a tool; (c) mapping of the derived coverage percentages into a qualitative scale with values from "Very Low" to "Very High" going from a very low coverage below 16.66% up to a perfect coverage close to 100%.
- *Inferencing*: the ability to support inferencing that could, for instance, enable a tool to derive new vulnerabilities or correlate vulnerabilities to each other, possibly allowing it to support root-cause analysis.
- *Counter-Measures*: the ability to associate vulnerabilities to counter-measures. This could enable a security system to check the alternatives in addressing the current security situation and select the best possible.
- *Risk Assessment*: the capability to evaluate the individual and overall risk across all vulnerabilities that have been detected.
- *Tool coverage*: it determines the sub-tools of a tool, in case it maps to a tool agglomeration. This criterion is related to the *categorisation* one as a tool that utilises multiple tools can also cover multiple categories.

The last category comprises the next set of criteria related to the tool configuration, i.e., how a tool can be configured to work properly as desired, including architectural and modularity aspects, plus its resource requirements.

- *architecture*: identifies the tool architecture. For instance, tools can follow a client-server architecture, where the client is installed in places requiring scanning, while the logic and vulnerability definitions are taken from the server side. Each architecture kind can have specific pros and cons. For instance, a tool working in standalone mode does not spend network bandwidth and resources to communicate with a server. However, it cannot receive updates on logic or vulnerability definitions.
- *usage level*: indicates whether a tool can be executed internally or externally to the scanning place. An external execution is non-intrusive, not spending precious resources from the running application component. However, it might not detect all possible vulnerabilities, especially those related to a component's internal execution environment. On the other hand, an internal execution can detect more vulnerabilities but can also spend some precious resources, which might prevent the application from delivering a suitable service level. So, tools flexibly operating in different levels are more preferable as they allow to detect only the most relevant vulnerability kinds and thus achieve a better trade-off between vulnerability coverage and resource consumption.
- *Vulnerability DB*: ability to connect to a VDB from which fresh vulnerability definitions and other crucial information, like counter-measures, can be drawn. While this enables to address properly new vulnerabilities, it can also assist in suitably producing risk assessment reports with a detail level also correlated to the kind of risk assessment that can be supported (see risk assessment criterion).
- *Modularity*: capability to operate under different modules where each module could support detecting certain vulnerability kinds. This enables supporting a more focused, resource-efficient, dynamic vulnerability assessment due to the one-demand scanning scope adjustment via proper module selection. This can be quite beneficial for rapid security issue detection and addressing (via a more focused vulnerability assessment) and for conducting periodic full scans correlated to periods of low resource usage in the application.
- *OS support*: A tool operating across different OSs should be preferred as it can cover all applications executing under these OSs. OS support comes into two flavours: (a) *operational support* meaning that the tool can operate on that OS; (b) *functional support* meaning that the tool can also detect vulnerabilities at the OS on which it is run.
- *Resource requirements*: Each tool comes with its own minimum resource requirements that must be satisfied to work properly. However, such requirements can affect the application's resource usage. Thus, in case of resource-intensive applications, it might be better to use tools either lightweight or supporting a non-intrusive, external usage level. Otherwise, especially in case of critical applications, more heavy tools might be required to operate in full mode such that the detection of a high number of different kinds of vulnerability issues can be achieved.
- *Access control mode*: A tool might need to be executed based on certain access control rights to be operationally successful. Thus, if the goal is to find OS issues, the tool should have privileged access to check the OS-specific part of the hosting component (e.g., VM). This criterion also correlates to the *usage level* as when that level is non-intrusive, the access control mode can be the least critical.

5.1.2. Analysis of comparative evaluation results

5.1.2.1. Support category. The evaluation results for the support category can be seen in Table 3. In the following, we analyse these results per each criterion involved in this category.

Standards. Both CVE and CWE seem to be mostly supported. This is logical as these standards enable to better characterise a certain vulnerability and classify it. Next comes CVSS followed by CPE which have slightly more than one third of support. This indicates that the need to associate vulnerabilities to the components concerned has not been well recognised yet while the capability to assess the risk related to a vulnerability seems to be neglected.

OVAL and SCAP are not very well supported. However, we believe in these standards as they will enable to automate vulnerability detection and addressing, thus allowing scanning tools to move forward by providing a new breed of functionalities and extending their vulnerability coverage.

It must be noted that one third of the tools do not support any standard. Such tools do not also employ a VDB (see Table 5). This seems a logical correlation as the need to store information in a VDB would have created the requirement to supply a structured way to perform this via standards. In our opinion, support to standards has been recognised by most tool providers as: (a) it can allow any kind of integration or cooperation between their tools; (b) it facilitates a better identification and assessment of vulnerabilities, thus enabling tool users to really benefit from the extra information retrieved.

We must underline the good performance of academic approaches in this criterion, indicating that researchers have well understood the power of standards such that they tend to adopt them more easily than other tool providers. The sole exception is OpenSCAP which has considered the support to standards as one of its major design requirements. This well justifies the support of this tool to 8 security standards.

Community. We have found that most tools have their own communities, usually maintained by the tool provider. Such communities can interact with the tool provider via emailing lists and blogs, while they can also participate in the tool development. The latter is the cornerstone of open-source communities, usually built around interesting or innovative tools that become sustainable via their participation. As such, security software providers follow faithfully this model which is widely adopted in the software world.

Table 4
Evaluation results on functional aspect.

Tool/ Approach	Categorisation	Object	Vuln. Coverage	Inf.	Counter-Measures	Risk Assessment	Tool Coverage
Grabber	Data Validation, Info. gathering, Source code analysis	Web Application	Very Low	No	No	No	Javascript Lint [50] PHP- SAT [51]
GoLismero	Data validation, Info. gathering,	Web Application, Network, DB	Very High	No	Yes	No	OpenVAS, nmap [52], ssl-scan [53]
Nikto2	Data validation	Web Server	Low	No	No	No	–
Vega	Data validation	Web Application	Low	No	No	No	–
Wapiti	Data validation	Web Application	Medium	No	No	No	–
OWASP Xenotic XSS	Data validation, Info. gathering,	Web Application, Network	Very Low	No	No	No	–
OWASP ZAP	Data validation, Info. gathering	Web Application, Network	Good	No	No	No	DirBuster [54], JBroFuzz [55]
OpenVAS	Data validation Info. gathering	Web Server, DB, OS, Web Application, Network, VM, Container	Very High	No	Yes	Yes	Several[a]
Arachni	Data validation, Info. gathering	Web Application	Good	No	Yes	Yes	–
IronWASP	Data validation	Web Application	Medium	No	No	Yes	Several[b]
w3af	Data validation, Info. gathering	Web Application	High	No	No	No	–
OpenSCAP	Data validation	Web Server, DB, OS, Web Application, Network, VM, Container	Very High	No	Yes	Yes	–
Clair	Data validation	Image	Low	No	No	Yes	–
Vulcan	Data validation	Web Application	Very Low	Yes	Yes	Yes	–
HPI-Vuln	See OpenVAS	See OpenVAS	Very High	No	Yes	Yes	OpenVAS
UC-OPS	See OpenVAS	See OpenVAS	Very High	No	Yes	Yes	OpenVAS, Nessus
AWS-Vuln	Data validation, Info. gathering, Malware detection	See OpenVAS	Very High	No	Yes	Yes	Several[c]

[a] Greenbone Security Assistant [56], Nikto, NMap, ike-scan [57], snmpwalk [58], amap [59], ldapsearch [60], slad [61], ovaldi [62], pnscan [63], portbunny [64], w3af.
[b] WiHawk [65], XmlChor [66], IronSAP [67], SSL Security Checker [68], OWASP Skanda [69], HAWAS [70], CSRF PoC Generator [71].
[c] Nessus, Clam AV [72], Chkrootkit [73], RootkitHunter [74], Rootkit Revealer [75], NMap, John the Ripper [76], extundelete [77], WhatWeb [78].

Table 5
Evaluation results on configuration aspect.

Tool/Approach	Architecture	Usage Level	VDB	Modularity	OS Support	Resource Reqs.	Access Control Mode
Grabber	Standalone	External	No	No	Linux	Low	Normal
GoLismero	Client-Server	Both	Yes	No	Any	See OpenVAS	Admin
Nikto2	Standalone	External	No	No	Any	Low	Normal
Vega	Standalone	External	No	Yes	Linux, OS-X, Windows	Low	Normal
Wapiti	Standalone	External	No	Yes	OS with python support	Low	Normal
OWASP Xenotic XSS	Standalone, API	External	No	Yes	Windows	Low	Normal
OWASP ZAP	Standalone, API	External	No	Yes	Any	(4 GB RAM)	Normal
OpenVAS	Client-Server	Both	Yes	Yes	Any	(2vCPUs, 2–4 GB)	Both
Arachni	Standalone, API, Client-Server	External	No	Yes	Any	(2BG RAM, 2–4 GB Disk)	Normal
IronWASP	Standalone	External	No	Yes	Windows, Linux, OS-X	Low	Normal
w3af	Standalone	External	Yes	Yes	Linux, OS-X, *BSD	Low	Normal
OpenSCAP	Standalone	Both	Yes	Yes	Windows, Linux	Low	Both
Clair	Standalone, Client-Server	External	Yes	No	Debian, Ubuntu, CentOS	Low	Normal
Vulcan	Standalone	External	Yes	Yes	Android	Low	Normal
HPI-Vuln	Client-Server	External	Yes	Yes	Any	(2vCPUs, 2–4 GB)	Both
UC-OPS	Client-Server	Both	Yes	Yes	Linux	(2vCPUs, 2–4 GB)	Both
AWS-Vuln	Client-Server	Both	Yes	Yes	Windows, Linux	(1 dual-core 2 GHz CPU, 2–4 GB RAM, 30 GB disk)	Both

We must appraise the existence of certain organisations, i.e., NIST and OWASP, that promote in the best possible way security software. NIST focuses more on developing standards aiming to bridge interoperability issues, while OWASP focuses on promoting innovative tools, creating tool benchmarks plus classifying both tools and vulnerabilities. Both organisations appear twice as community members in two respective scanning tools. This also indicates that these organisations have a great belief in the dynamics and capabilities of the tools they support.

We have also discovered that by following new software development practices along with the open-source model, the tool providers invest on github as a wide source tank for potential community members while still offering the right facilities for agile software development. We believe that this is a trend which will be followed by most scanning tools in the near future.

We must also highlight that OpenVAS and OpenSCAP have more than 2 communities targeted. This is not accidental by considering their vulnerability coverage (see analysis below in functional category). OpenSCAP is supported by both NIST and RedHat while it has already engaged multiple OSproviders as these providers have already specified their vulnerabilities and policies in a format acceptable by OpenSCAP. On the other hand, OpenVAS is a fork of the well-known Nessus [49] vulnerability scanner, which has moved from an open-source to a closed-source model. Fortunately, with the support from 3 organisations, OpenVAS has been quite successful as it can be witnessed by the great amount of tools which exploit internally this scanner.

Finally, we must stress that as the last 4 approaches are academic ones, their community is more restricted, usually in form of one or two organisations, where one is affiliated to the other. This restriction, then, impacts the tools' update frequency which is quite scarce. This underlines the need for the involved organisations to have a more aggressive move towards the open-source community to better support their tools sustainability and evolution.

Freshness. More than half of the tools are updated frequently (every some months) or very frequently (every some days) while the rest are either not updated any more (tool provider has stopped investing on them) or the update is infrequent and community-driven (esp. for academic tools). This is a nice result, signifying that the need to update the tools to address the continuously increasing vast set of vulnerabilities has been well recognised.

This result can be correlated with the previous one by considering that frequently-updated open-source tools either rely on more than one community or are pushed by their providers as they see great value in them. In the first case, the update frequency can reach even higher levels depending on the tool community size. For instance, if we consider OWASP, this organisation is both big and well-known for its security expertise. As such, it does not only use its own resources to evolve its tools but it also incorporates an enormous in size member list supporting both the tools updating and maintenance via two alternative feedback mechanisms (merge requests & bug reporting).

Best Tool Nomination for the Support *Category.* By considering the evaluation results across all criteria of this category, the best performers in the *Community* criterion, OpenVAS and OpenSCAP, can be discerned based on their community support and update frequency. We further discern OpenSCAP as the topmost performer, due to the great amount of standards it supports. Finally, academic approaches, especially Vulcan and UC-OPS, are placed third, with a good standards coverage.

5.1.2.2. *Functionality.*

The evaluation results for the functionality category can be seen in Table 4. In the following, we analyse these results per each criterion involved in this category.

Categories. All tools go beyond pure vulnerability scanning and offer extra functionality mapping to other OWASP categories from OWASP taxonomy, spanning *data validation*, *information gathering*, and *application protection.*

All tools exhibit the functionality of *data validation* testing. This is well expected as many vulnerabilities originate from improper or non-existent data input validation. Thus, a tool must check against the target object various kinds of data validation penetration scenarios. Three ways exist to support this functionality: (a) develop it internally in the tool; (b) rely on the definitions of vulnerabilities which include the way to be detected; (b) use another tool, like w3af, exhibiting this functionality.

Similarly, more than half of the tools support *information gathering*. This is also logical as before any kind of vulnerability assessment is performed, the target object needs to be scanned to find the right places where vulnerability assessment will focus, collected in an information gathering step. Depending on the object of focus, information gathering can take different forms. For pure web application scanners, spider software is exploited to find all actual web pages from which the scanning can be performed. For more sophisticated scanners, information gathering involves checking the whole host, finding open ports and attempting to infer the components that run on such ports.

Finally, we must remark that only one tool exhibits application protection capabilities, taking the form of antivirus/anti-malware detection. Such functionality can be considered as complementary to vulnerability detection in the way it is employed. It has the added-value that it is actively enforced, enabling the system to immediately detect and react on a vulnerability that comes in form of a virus/malware. This is essential as continuous security assessment is highly required for most applications/systems, which constantly evolve over time. Further, we must stress that in contrast to pure vulnerability detection, focusing on discovering but not addressing vulnerabilities, antivirus detection can react on the vulnerability by, e.g., putting infected assets into quarantine or deleting them. Based on this argument, it is recommended that either scanning tools are extended with application protection capabilities or are complemented with tools offering such capabilities. Further, by considering application evolution, it is also advocated that vulnerability scanning is not an one-shot but a continuous process with a frequency conforming to the evolution frequency of the respective application.

Object. Almost all vulnerability scanners can operate over a web application. However, as the next paragraph and Section 5.2 will show, each scanner might have a different focus on the web application vulnerabilities by, e.g., concentrating only on specific kinds, like the most usual ones.

Most old scanners focus solely on web applications as the target scanning object. The main rationale is that application developers would care most how web applications are vulnerable in their operation space (e.g., set of web pages) from the outside. However, as an application might comprise multiple components and might run on vulnerable media, it is apparent that the focus should now move to holistically cover the whole application system. This is actually grabbed by both recent scanners and academic approaches. A good example for the former case is OpenSCAP that covers not only pure web application but also cross-level system vulnerabilities. In the latter case, academic approaches extend an almost complete vulnerability scanner, like OpenVAS, to make it perfect.

Vulnerability Coverage. By considering all tools, it seems that only OpenVAS and OpenSCAP have the highest vulnerability coverage along with tools that re-use them. By considering that: (a) the percentage of these tools is less than half of the overall tools; (b) OpenVAS and OpenSCAP do not focus on scanning the whole web application operational space, it is easy to understand that such a result marks the need to improve most existing scanning tools.

In fact, this might signify that security experts must now move to a new direction: as there exists a sophisticated state-of-the-art tool, the community must focus on both improving and evolving it over time to also detect new vulnerabilities via the production of respective plugins. In fact, this already occurs for the 2 aforementioned tools. In the context of Network Vulnerability Tests (NVTs) or OVAL definitions, OpenVAS and OpenSCAP, respectively, rely on a huge community effort, involving a great amount of OS and software providers contributing to the definition and detection of vulnerabilities.

On the other hand, the academic community, while taking the same direction on exploiting such state-of-the-art scanners, has a different focus and attempts to improve or orchestrate them with other tools to achieve both a better coverage extent and breadth plus capture a greater set of vulnerability detection scenarios. For instance, as indicated in the *Categorisation* criterion, one academic approach, AWS-Vuln, focuses on combining vulnerability scanning with application protection to sustain a smooth, vulnerability- and malware-free application lifecycle. This approach signifies that due to the application system's complexity, it is not always possible to cover all possible vulnerability kinds and different tools need to be employed to achieve a perfect coverage. Thus, it surely moves to the right direction and paves the way via which vulnerability scanning tools should be utilised in the near future.

Inferencing. Only one tool, Vulcan, supports some kind of inferencing. This academic tool employs ontology-based reasoning to support inferencing. Restricted to the current tool functionality, this inferencing takes the form of discovering vulnerabilities of component agglomerations apart from those of individual components. However, this ontology-based reasoning approach is quite promising as it could be extended to cover the derivation of extra kinds of vulnerability-related knowledge; these kinds are explained in Section 6.

Counter-Measures. Also correlated to mitigation plan production, this criterion highlights the need to associate vulnerabilities with the ways they can be addressed known as counter-measures. In contrast to the previous criterion, though, the situation is much better as almost half of the tools support this criterion. This means that the need to report this correlation kind to users has been well recognised and realised. Further, most tools exhibiting this feature support standards like CVE or CWE which do provide counter-measure information. This is another benefit related to the support of standards.

OpenSCAP seems to be the sole tool going beyond counter-measure reporting to counter-measure enforcement. In particular, through supporting SCAP, this tool can mitigate the vulnerabilities found in some cases via automatic path application. This is the right direction to be followed by all other scanners to really advance their usage and increase their added-value.

Risk Assessment. More than half of the tools offer some risk assessment form. So, they have well recognised this information's importance, which, by accompanying the vulnerability detection one, enables users to assess their application's overall risk level and thus supports them in producing a mitigation plan that prioritises more the vulnerabilities with the highest risk level. Similarly to the previous criterion, many tools supporting this criterion, also conform to CVSS, the most prevailing and de-facto standard.

Note that 3 tools claim to support some risk assessment form but it is not clear how. We suspect that they map vulnerability categories to certain risk levels to address this. However, such an approach is too coarse-grained and does not consider a vulnerability's severity with respect to the consequences it might have and the component(s) on which it applies.

The 4 academic approaches support both this and the previous criterion which signifies that they have well recognised the requirement to support the supply of the respective information. In most of the cases, this support comes from the adoption and possible extension of other scanning tools.

Tool Coverage. Scanning tools re-use a great variety of sub-tools, spanning many OWASP taxonomy categories, including antivirus protection, information gathering, network scanning, and data validation testing. From these categories, we can discern Nessus and OpenVAS, as those sub-tools mostly re-used due to their almost perfect coverage extent, as well as nmap, as the most popular and well-adopted information gathering/network scanning tool.

However, more than half of the tools do not re-use any sub-tool. This can be due to various factors: (a) they just do not report such a re-use – this then needs to be explored in the tool's source code; (b) no other sub-tool is re-used – this means that the scanning tool was developed from scratch while its coverage level is influenced by its maturity level, developer base and level of support from its community.

In fact, the second factor seems to apply to most (6 out of 9) pure web application scanning tools. This means that such tools have a focus that can be independently realised. This also signifies that possibly these tools' providers do not want to invest on adopting other tools with the main rationale that this could restrict them technologically in the tool implementation.

Best Tool Nomination for the Functionality Category. In this criteria category, there is not a clear winner. We can actually see 2 main tool partitions: (a) top approaches which do not support inferencing; (b) the Vulcan academic approach which supports inferencing but has a very low coverage and focuses only on web applications as target objects. The latter also indicates that if Vulcan was able to use Nessus or OpenVAS, we would surely nominate it as the best in this category, as this would immediately improve in an ultimate manner its two main deficits.

In the first partition, the academic tool AWS-Vuln is the clear winner. It not only has the best possible categorisation and vulnerability coverage but also re-uses a great sub-tools set, which constitutes another proof of its completeness. From the pure tool world, we can discern OpenVAS, which, however, does not cover additional security scanning categories.

From the 2 best result partitions, academic tools can be discerned. This interesting result signifies that the academia not only pioneers research but also attempts to be more innovative. As such, this is a good paradigm for tool providers which need to either re-use academic tools or follow their orchestration approach to improve their own tools and increase their uptaking.

5.1.2.3. *Configuration.*
The evaluation results for the configuration category can be seen in Table 5. In the following, we analyse these results per each criterion involved in this category.

Architecture. A tool's architecture along with its mode of operation impacts both the vulnerability coverage and the flexibility in the tool runtime administration and execution. In particular, a tool that works in a standalone manner and scans externally an application has the benefit of not interfering with the application's normal operation, thus taking a non-intrusive approach. On the other hand, this comes with the disadvantage that vulnerability coverage is low. A tool working in client-server mode that is executed internally to an application is more intrusive but has the benefit of better vulnerability coverage even for the whole application via the well configured concurrent deployment of tool clients that scan all application components in parallel.

Table 5 signifies that most tools can be run in a standalone mode. This is a natural, well-expected result as: (a) this is the desired operation mode for simple, non-expert users who prefer to have a simplified way of launching and interacting with the tool; (b) some tools have low vulnerability coverage and can work only in non-intrusive mode. As such, it is not rationale to make these tools distributed and complicate their administration as this will affect their main competitive advantage: their simplified usage and administration.

Tools that employ a client-server architecture are about one third in number. They usually exhibit a better vulnerability coverage and focus on more advanced, IT-security expert users. As such, in contrast to the previous paragraph argumentation, these tools should employ a client-server architecture as the most natural way to completely cover vulnerabilities across the whole application plus the most flexible and capable way in terms of administering the scanning. Due to these advantages, it is worth sacrificing slightly the simplicity to better attract the main target users, i.e., IT-security experts.

There is also the case of tools (especially Arachni) which enable the alternative use of both architectures. This might be a very good move to extend a tool's applicability towards all kinds of users and not only the novice ones. As such, the adoption and market share of such tool could be increased.

In very limited cases, an API is offered by tools which either operate in standalone mode or in a client-server architecture. Such a mechanism targets developers of more extensive tools requiring a standardised way to interact with the encompassing tool. An API supply can be also preferable in case that the user does not need to be burdened with the tool installation and deployment. In particular, a security organisation can make the API available as a service such that the user can immediately use it without spending resources to deploy, install and maintain it. However, an external API's supply comes with a maintenance cost for its offering organisation which needs to be accompanied with a certain business model to enable this organisation to gain from it. This business model could take the form of an usage fee or the supply of API usage support or the paid subscription to more advanced API features. However, none of the tools seems to be offered in a SaaS mode. This is natural as all tools considered are open-source so only open-source-based business models would make sense in this case.

Usage Level. The usage level directly impacts a tool's ability to cover a suitable coverage level. In particular, a tool operating in external mode cannot access an application component's internal environment. As such, it might discover vulnerabilities that might jeopardise the component's operation from the outside but it will not detect other vulnerability kinds that might concern the component (e.g., bugs or modelling mistakes in the source code) and its environment's elements. On the other hand, for web application kinds with a simplified architecture, this external scanning mode might be more suitable as more focused and less intrusive. As such, the operation mode could rely on the type of application targeted, its architecture and its environment.

The evaluation results unveil that most tools support only external scanning. This is well-expected as: (a) the very nature of these tools focuses on web application security only and has external scanning as its design cornerstone; (b) in the past, web applications were offered in private, more difficult to penetrate infrastructures. As such, there was no need to supply an internal scanning mode to find inner environment vulnerabilities. However, nowadays, due to the change of focus with respect to the hosting environment and its public nature, this environment is more vulnerable to attacks from adversaries, also due the fact that hardware resources are shared between user applications. This focus shift then requires supporting also an internal scanning mode. This real necessity, quite demanding for cloud environments, seems to have been picked up by some tool providers. In particular, one third of the tools seem to support both scanning modes. This concerns both the most advanced scanning tools, i.e., OpenVAS and OpenSCAP, and those academic approaches adopting OpenVAS (or its parent tool, i.e., Nessus).

Vulnerability Database. The existence of a VDB enhances a tool's reporting capability while assists in better detecting and identifying vulnerabilities. Coupled with the support to standards, the amount and credibility of reported information can reach high levels and thus make the tool more attractive to users. The above arguments must have touched the tool providers as almost half of the tools encompass a VDB in one or another form with varying capabilities. Most approaches encompassing a VDB also support one or more standards, which enables to exhibit the advantages that were previously mentioned. On the other hand, tools with no VDB seem to be old and clearly not supporting standards. As such, these tools could be considered as outdated and not attractive any more to users. This has thus impacted their communities, also explaining their infrequent updating.

The need to encompass a VDB has been well recognised by the academic community as there is a 100% support for this feature by all academic efforts.

Modularity. A modular tool can be individually extended as needed. It also allows making a focused scanning by using only those modules necessary for detecting the right vulnerabilities for an application. This need for modular tools seems to be picked up by the tool providers as most tools are modular.

Two different forms of modularity exist across the examined tools. Full modularity enables specifying scanning profiles including a configurable scanning rule set. Such scanning profiles are special-purpose as they focus only on a certain set of vulnerability kinds which might be exhibited by particular kinds of applications or their components. Partial modularity, on the other hand, is a limited modularity form that implies the existence of one profile from which the user can select some of the scanning rules contained. In some cases, the addition of new rules might be also possible in such a profile.

Based on the above definitions, we can observe that most tools are fully modular. This is a very good result for the prospective of tool users as they have the best possible flexibility in configuring such tools with the right set of scanning rules. This also increases the usability level of these tools.

On the other side, non modular tools are mostly those with low vulnerability coverage. Such tools might be special-purpose, focusing on few vulnerability kinds, such that making them modular would not make sense.

In most cases, tools which expose a VDB also exhibit modularity. This is a promising result, also enabling possible vulnerability scanning adopters to assess a tool's coverage level by inspecting the vulnerability kinds it can cover from the available modules and their mapping to the vulnerabilities stored in the VDB. The VDB alone, though, cannot tell the whole truth as it could be made as complete as possible (as in case of NVD VDB) without having a one-to-one mapping of vulnerabilities to the tool's detection capabilities.

OS Support. The variety of OSs supported by a tool impacts the extensiveness in the tool's usage and applicability. This support can come in different forms: (a) with respect to the tool's execution environment; (b) with respect to the scanned component's execution environment. Obviously, the latter form affects the former as it is expected that when one tool is written to operate in one OS, it is rather strange to apply it to applications written in other OSs. However, this expectation holds mainly for internal-mode tools. As such, this criterion should not be inspected alone when attempting to discover OS restrictions on the application side. On the contrary, by knowing that a tool works in external mode, this tool can be applied for applications operating in any OS. Further, a tool that works in both modes would then be able to cover externally any application and internally only an application operating on those OSs where this tool also operates.

Some tools like Clair are able to operate on the level of images. This then enables them to work on an "external" mode as they just inspect the images for vulnerabilities before they are deployed in the cloud. As such, this kind of tools can support the scanning of images of any OS.

The evaluation results with respect to the first OSsupport type signify that most tools support Linux-based OSs. This is well expected by considering that such OSs have been deemed more secure (at initial deployment time without any extra hardening support [48]). Windows come next with around two third of tools able to operate in Windows environments. OS-X comes third but still having more than half of the tools supporting it.

Around one third of the tools support any OS, which is an encouraging result, signifying that the respective need has been well recognised by tool providers. However, contrary to other criteria, this need is not well reflected in academic tools. This is well anticipated as the internal or community resources dedicated to developing these tools are limited. Rather, a more focused development on improving these tools' core capabilities is expected.

By combining the results from this and the *Usage Level* criterion, we can deduce that: (a) tools only operating on Linux have an external or both modes; (b) tools operating on Windows and Linux exhibit both modes; (c) tools operating on any OStend to support only external usage. This result unveils possibly the fact that tools with external usage tend to increase their applicability to improve their market share position by supporting more OSs to alleviate for their non-high vulnerability coverage. It further indicates that tools focusing on a limited OS set tend to be more extensive in terms of their usage level. This is natural as in this case such tools have a better coverage and applicability over these OSs as they invest on these OSs' popularity.

Hardware Requirements. This should be inspected in conjunction with the *usage level* criterion. This is due to the fact that as long as a tool is not intrusive, it can be regarded as not demanding for resource requirements. However, this should not be treated as a panacea as there exist non-intrusive tools like ZAP that have high hardware requirements. On the other hand, an intrusive tool should be either invoked irregularly or have low resource requirements to not interfere with the scanned application's normal operation.

The evaluation results indicate that more than half of the tools have low resource requirements. These tools' providers have decided to make them lightweight to be more appealing to prospective adopters. However, this design choice seems to affect the tool capabilities, as most tools with low requirements do not have a good vulnerability coverage. On the other hand, sophisticated tools, like OpenVAS, or tools that re-use them are more heavy-weight and require either high-disk VMs (Nessus) or medium CPU & memory VMs (OpenVAS). However, these tools' usage level is extensive such that they can be executed also in external mode. As such, they are not necessarily intrusive to the application such that they can jeopardise its normal operation. Further, we are discussing mostly requirements at the server side. For the client side, the resource requirements can be significantly less. So, the burden in using these tools comes mainly with the operation cost of the server which can be outside the VMs hosting the scanned application.

Access Control Mode. This mode impacts a tool's vulnerability coverage plus its administration. It might also raise security concerns about how safe are clients operating in admin mode in a client-server architecture. Thus, tools operating in a normal, user mode are less intrusive and do not access critical assets via internal scanning. However, as such assets are uncovered, vulnerability coverage is low. On the other hand, an admin mode enables accessing (possibly infected) critical system assets and thus caters for a better vulnerability coverage. However, acting in this mode comes with issues that might involve the incautious exchange of credentials and the possible scanner infection thus exposing the respective system to great vulnerability.

The evaluation signifies that most tools support the normal access mode. This is a natural result as: (a) a tool focusing solely on external web application scanning does not require using any credentials for authentication and authorisation purposes; (b) less vulnerability to attacks can be caused by the tool's compromise; (c) possible undesirability of users to provide credentials for this scanning type. On the other hand, almost one quarter of tools support both access modes. We believe that such tools are more flexible in vulnerability scanning as they enable the user to choose the most desirable access control mode according to the current context and application kind. It is not surprising to see that the most representative tools of this kind are OpenVAs and OpenSCAR, while the rest of this kind's tools just reuse them. This indicates that these two tools are designed based on suitable requirement sets originating from actual users and their needs. This answers well these tools' excellent performance in many of the considered criteria.

Best Tool Nomination for the Configuration Category. By considering the overall evaluation results of this criteria category, OpenVAS and OpenSCAP can be considered as the best. OpenVAS seems better in terms of OS support while OpenSCAP is better with respect to hardware requirements. As the OS support can be less important than hardware requirements, we might then nominate OpenSCAP as the best. In both cases, both tools seem to support only one kind of architecture (deployment). Thus, we can deduce that there is still room for improvement of these tools.

On the other hand, academic tools require significant improvement over this criteria category, especially for the criteria of *usage level*, *OS* support and *hardware requirements*. However, in contrast to non-academic tools, we do not expect such improvement to take place soon due to the limited resources available for developing the academic tools and their development priorities.

5.1.2.4. Overall evaluation results analysis. By considering the results across all categories plus the derived results from the analysis of each category, we can discern two main tools: OpenVAS and OpenSCAP. OpenVAS is the best in terms of the functional category while OpenSCAP in the support category. While they score equally in the configuration category. However, in our opinion, while OpenVAS might seem better in categorisation and tool coverage as well as OSsupport, we would promote OpenSCAP as the best based on the following justification: (a) OpenSCAP is the sole tool able to mitigate vulnerabilities; (b) it has lower resource requirements; (c) it supports more standards while it has a similar community support with OpenVAS. Further, it seems to have a slight better vulnerability coverage based on the quantitative results reported in Appendix B. To this end, OpenSCAP is the best possible tool that can be selected by respective practitioners while OpenVAS could be selected as the second best.

If the analysis is restricted in the academic tool partition, different tools can be distinguished as best in each criteria partition. HPI-Vuln is better in terms of configuration, AWS-Vuln in terms of functionality and Vulcan in terms of support. In overall, we would actually discern two tools for different reasons. AWS-Vuln is discerned not only due to its perfect functional coverage but also as it seems to have good community support and supports well-known standards. By considering its functional coverage, we should highlight its capability to complement vulnerability detection with malware/antivirus protection, a highly-required and innovative feature. However, it still needs to be improved in some aspects like resource consumption, which comes with its design choice to select Nessus as its core vulnerability scanning sub-tool.

On the other hand, we discern Vulcan for three main reasons: (a) its excellent support level to standards; (b) its inference capability; (c) its low resource requirements. However, Vulcan still needs to be improved with respect to its low vulnerability and object coverage plus its limited OSsupport. Further, it needs to be expanded to use a very good vulnerability scanner, like OpenVAS or OpenSCAP. In any case, we believe that all academic tools should be made publicly available as normal vulnerability scanning tools to increase their sustainability and potential for improvement and evolution.

5.2. Benchmark evaluation

As indicated in Section 3, research question Q_3 required a special handling to enable assessing the properties of scanning time, accuracy and coverage. This handling is also due to the fact that there is no reporting currently for most tools about their performance for these properties. Nevertheless, even if this reporting was available, it could be old or rely on simple benchmarks.

This difficult situation actually created the need to search for a suitable benchmark to conduct the required tool assessment. Such a benchmark should exhibit certain features, including the capability to objectively assess all these properties while, in addition, catering for creating an as realistic as possible assessment environment that covers multiple vulnerability areas.

As such, we have conducted a small web-based survey to identify candidate vulnerability tool evaluation benchmarks. This led to discovering 6 benchmarks which are shortly presented in Table 6. From these benchmarks, we have finally selected the OWASP benchmark for the following reasons: (a) it is as realistic as possible as it relies on a web-based application built according to well-known design patterns; (b) it has the widest number of vulnerability areas covered; (c) it enables to assess all three properties. This includes a suitable scanning accuracy metric that considers a tool's returned evaluation results across all the vulnerability areas covered; (d) it supplies code which employs an easy integration point with respect to the adoption of a scanning tool. The code is also more frequently updated than the other benchmarks; (e) it already includes some scanning tools in its distribution.

After selecting the right benchmark, we came to a certain dilemma: should we evaluate the plethora of selected scanning tools? As Section 3 indicated, this required a huge effort; so, we followed a different

Table 6
Vulnerability tool assessment benchmarks.

Name	Description
OWASP Benchmark	An open test suite focusing on evaluating the speed, accuracy and coverage of vulnerability detection tools. This suite adopts the Youden Index, which is a standard way of measuring accuracy for test sets. Its tests are derived from coding patterns in real-world applications.
WavSep	Relies on supplying a vulnerable web application comprising a collection of vulnerable web pages to be used for testing the features, quality and accuracy of scanning tools. The assessment is performed via a set of test cases which cover different vulnerability areas.
DVWA	This is a PhP/MySQL vulnerable web application that can assist in assessing scanning tool accuracy. It exhibits some common vulnerabilities which come with different levels of difficulty.
Android App Vulnerability Benchmarks [79]	This benchmark repository captures 25 known Android application vulnerabilities. Each benchmark captures a unique vulnerability while maps to a pair of benign and malicious applications. The covered areas include inter-component communication, storage, system and web.
WebGoat	It is a deliberately insecure J2EE application developed to teach web application security lessons. Once the application is installed and executed, the user can go through a set of lessons, each focusing on identifying a different issue mapping to a different vulnerability.
SQL Injection Framework [80]	It can be used to evaluate web scanning and penetration tools. Covers only the SQL injection vulnerability area. It can dynamically create a testbed and map it to ideal assessment results. Tools can be run against this testbed to compare their results with the ideal. 3 aspects can be evaluated: deployment requirement, SQL injection coverage, and certain evaluation parameters.

approach. In particular, we decided to use the tools that the benchmark already provides plus open-source tools, like Zap, which provide guidance about how they can be assessed with this benchmark. This would enable us to maintain the assessment effort to the minimum while still enabling to answer appropriately research question Q_3. From the reader's perspective, we continue the assessment of the previous sub-section with the capability to rank the top results against the three major properties of scanning accuracy, time and coverage. To inspect the trade-offs between the three main properties, we also tried to evaluate the scanning time and accuracy by varying a tool's vulnerability coverage. This was performed by activating or deactivating the respective vulnerability scanning rules, in case this possibility was available.

In the following, we first acquaint the reader via the use of Table 7 with some of the assessed tools, which were not covered in the previous subsection. Then, we present the evaluation results and analyse them.

Tables 8 and 9 show the results of the benchmark-based evaluation. The first table focuses on depicting the tool involved, its operation mode, scanning accuracy and time while also supplies some extra information related to the interpretation of each result. The second table reports the accuracy of each tool per each vulnerability area covered by the benchmark.

Based on the above results, the best tool with respect to the overall scanning accuracy and time is FindSecurityBugs. In fact, this tool achieves a high accuracy on many vulnerability areas. This might be regarded as unexpected as this tool only includes a small set of security rules. However, it can be well justified by the fact that this tool operates directly on the source code so it can indicate with a higher accuracy if a certain issue holds.

Similar results were expected for SonarQube. However, this occurred only for scanning time but not accuracy. This could be due to the fact that FindSecurityBugs focuses only on Java source code, on which the OWASP benchmark is based, and is thus more optimal with respect to this programming language. Thus, as Sonarqube has put equal focus on different languages, some required scanning rules for Java programs might be still missing. Sonarqube, though, was able to detect minor security issues, not foreseen in the OWASP benchmark. This led to a reduction in its accuracy which was bypassed by removing the security rules detecting these issues.

ZAPProxy exhibited a much worse performance in both scanning time and accuracy. The bad performance in scanning time can be justified by the fact that ZAPProxy has to perform a kind of an extensive attack on the whole application to detect vulnerabilities. In such an attack, each vulnerability area testing requires executing a test set on each application part. Thus, such an extensive attack takes a considerable amount of time to execute in contrast to source code inspection that is much faster.

This issue was considerably improved, but not still reaching the scale of source code scanning time, by configuring ZAPProxy to cover only certain vulnerability areas and running it at the user VM level. This led, for instance, to the case where in full scanning but local mode, the scanning time is much better than the remote (partial) scanning mode one.

ZapProxy's bad performance in scanning accuracy can be justified as follows: (a) it is harder to find vulnerabilities when source code is not inspected – this justifies the fact that some vulnerability areas were not covered at all; (b) it is more difficult to have a high confidence for each vulnerability kind.

Based on the results of Table 8, a trade-off exists between scanning time and accuracy only for dynamic vulnerability scanning. In fact, by varying the number of scanning rules, we can reach different levels of scanning time and accuracy. However, when the scanning is statically applied on the application source code, scanning time is very similar, almost independently from the number of security rules considered. This is related to the fact that in all examined static analysis tools the number of security rules is small, such that varying the partitions of the security rule set does not significantly impact scanning time. Thus, it is advocated that static analysis tools should always be used on full scanning mode. On the other hand, dynamic scanning should be more focused, mapping to the need to create suitable scanning profiles to cover different kinds of applications or components. Due to the nature of such tools and the workload they incur on the underlying resources, it is advocated that such tools are used either remotely and infrequently due to their impact on application performance, or internally, when sufficient internal resources are available to enable the scanning to be normally performed.

The results of Table 9 indicate that FindSecurityBugs covers all vulnerability areas. However, this coverage is not always deep. On the other hand, Sonarqube touches just three vulnerability areas and has good accuracy performance in only two of them. This signifies that FindSecurityBugs is recommended, due to its coverage, for use when the application is written in Java. While Sonarqube can be exploited for applications written in different languages. However, in this case, Sonarqube requires to be coupled with another scanning tool to cover additional vulnerability areas.

In the context of dynamic analysis, ZAP covers mainly 5 vulnerability

Table 7
Additional (static analysis) tools in OWASP benchmark.

Name	Description
FindSecurityBugs [81]	The FindBugs tool [82] enables conducting static analysis to assess Java source code quality. This extension enhances FindBugs with the capability to find security bugs. However, it does not enable filtering the security bug detection rules. Only explored possibility is to either apply only the security rules or all bug detection rules, i.e., including those originally constituting the rule base of FindBugs.
Sonarqube [83]	It assesses quality of code implemented in many programming languages. It has a more configurable rule base, allowing to filter some security rules, especially those detecting minor security issues.

Table 8
Scanning time and accuracy benchmark results.

Tool	Operation Mode	Accuracy	Time	Comments
FindSecurityBugs	ONLY_SEC	39.10%	3:06	Only security rules applied
FindSecurityBugs	FULL	39.10%	3:41	All bug scanning rules applied
Sonarqube	FULL	10.00%	3:37	Found 23582 vulnerabilities, 1176 bugs and 15448 code smells.
Sonarqube	PART_SEC	19.00%	3:41	Removed 13 security rules creating false positives in two vulnerability areas. This enabled reaching a 100% accuracy in these two areas. Total vulnerabilities were reduced to 242.
ZAPProxy	FULL_REMOTE	15.65%	12:00:00	Remote scanning with all vulnerability scanning plugins. Reached 48% progress and then the benchmark app went down. Bad accuracy is due to many false negatives.
ZAPProxy	PART_REMOTE	15.44%	501:40	Reduced version with only the most appropriate vulnerability scanning plugins. Accuracy almost the same. However, the command injection category was additionally covered. This might be due to the previous mode interruption, which did not enable to apply the corresponding plugin.
ZAPProxy	FULL_LOCAL	7.75%	2:42:00	Full scanning mode applied internally to the benchmark application. Scanning time was quite accelerated. However, accuracy became worse as the performance on path traversal and SQL injection vulnerability areas was bad with respect to remote scanning.

areas but exhibits good accuracy performance only in two of them. Compared to FindSecurityBugs, it is slightly better in two vulnerability areas: *command injection* and *path traversal* while much better in *cross-site scripting* and *SQL injection*. This signifies that the agglomeration of these two tools will enable to have a better vulnerability area coverage and thus reach a much higher overall accuracy level. However, some vulnerability areas are not yet deeply covered, including: *command injection*, *LDAP injection*, *path traversal*, *trust boundary violation* and *XPath injection*. This outlines the need to consider another scanning tool to complementarily cover more deeply these areas.

All the above observations signify that scanning tools should not be individually used but orchestrated to reach a suitable vulnerability coverage level. Further, it has been highlighted that there is a need for a scanning tool strategy or workflow to enable executing different vulnerability tools based on the kind, nature and content of respective applications. Finally, it has been indicated that tools might require being configured with different rule profiles to enable a more focused scanning to cover only those vulnerability areas relevant for an application. Such rule profiling would also guarantee a suitable scanning time without overloading much the scanned application.

6. Future work directions & challenges

Based on the comparative and empirical evaluation results of the vulnerability scanning tools and databases selected, this section elaborates on the consequences of the main findings in terms of on-going challenges and future work directions. The presentation is separated based on the tool kind considered, i.e., a vulnerability DB or tool, in the following two sub-sections.

6.1. Vulnerability database challenges

6.1.1. Interfacing

Most VDBs exhibit just a limited web-search interface. In our opinion, there is a need to go beyond this and offer an API allowing to: (a) pose more advanced query forms; (b) subscribe to certain events (e.g., new vulnerabilities introduction); (c) manage vulnerability information. Such an API could be quite beneficial to both scanning tools and users. The former could retrieve the required information in a sophisticated and precise manner. Further, they could supply different vulnerability reporting levels to users. On the other hand, users would benefit from the extra information that could be retrieved for a certain vulnerability as well as from the existence of an interface for introducing or updating vulnerabilities in case that they concern their own software. Further, such an API could enable to abstract away from the information source technology and enable the VDB to obtain and integrate information from multiple, disparate sources.

6.1.2. Semantics

While in terms of information coverage reasons a relational DB technology could be selected for implementing a VDB, we believe that semantic technology should be the norm instead due to the following reasons: (a) a semantic representation can be richer, closer to human perception, while enabling to better link different information perspectives and entities; (b) SPARQL as a query language can be more easy to use and manipulate for evaluating the relations between different vulnerabilities; (c) semantic technology enables inferencing over the stored information which can be beneficial to derive new knowledge that could take different forms (see details in next subsection); (d) in case that different information sources need to be integrated together, semantic technology can be the most optimal way to achieve this integration.

Due to the main benefits that semantic technology brings about, we expect a proliferation of semantic VDBs in the near future. Currently, only Vulcan offers a semantic VDB. However, this VDB is not rich enough and coupled with the right rules to allow inferring various knowledge kinds. Further, it seems to manually and not automatically integrate

Table 9
Scanning accuracy benchmark results per vulnerability area.

Vulnerability Area	Find SecurityBugs	Sonarqube [Full]	Sonarqube [PART]	ZapProxy [FullRemote]	ZapProxy [PartialRemote]	ZapProxy [FullLocal]
Command Injection	11.20%	0.00%	0.00%	0.00%	13.49%	12.70%
Cross-Site Scripting	37.32%	0.00%	0.00%	55.69%	55.69%	8.13%
Insecure Cookie	100%	55.56%	100.00%	55.56%	55.56%	55.56%
LDAP Injection	15.63%	0.00%	0.00%	0.00%	0.00%	0.00%
Path Traversal	9.57%	0.00%	0.00%	11.28%	11.28%	1.50%
SQL Injection	9.48%	5.53%	5.53%	49.63%	33.82%	7.35%
Trust Boundary Violation	18.60%	0.00%	0.00%	0.00%	0.00%	0.00%
Weak Encryption Algorithm	54.31%	50.77%	100.00%	0.00%	0.00%	0.00%
Weak Hash Algorithm	68.99%	0.00%	0.00%	0.00%	0.00%	0.00%
Weak Random Number	100%	0.00%	0.00%	0.00%	0.00%	0.00%
XPath Injection	5%	0.00%	0.00%	0.00%	0.00%	0.00%

information from multiple sources.

6.1.3. External source integration

In some cases, VDBs attempt to draw information from various information sources but only in the context of typical vulnerabilities. In our opinion, a VDB should be equipped with the capability to automatically collect information from multiple sources that cover both vulnerabilities and software bugs. The latter could be drawn from well-known bug repositories [46] and enable inspecting whether certain application components are safe to be exploited, unveiling bugs related to software vulnerabilities, and highlighting the most secure and reliable component versions to adopt. The glue between vulnerabilities and bugs could be achieved by using semantic technology, realisable by adopting suitable semantic models and mapping techniques to integrate information from multiple, heterogeneous information sources.

6.1.4. Vulnerability relation coverage

CWE seems to propose a rich vulnerability and threat taxonomy. However, this taxonomy should be extended accordingly to become a semantic security meta-model able to both cover the most important security concepts and their relations. This coverage along with semantic rules incorporation would then enable inferring further relations between security concepts which would never be acquired by a syntactic modelling approach. Apart from this, such relations would improve the scanning accuracy as explicated below.

6.2. Vulnerability scanning tools

6.2.1. Tool support

A good support level already exists for many of the tools examined. However, we see some places for further improvement based on two main directions: (a) additional support to standards; (b) community engagement.

The first direction indicates that, as there are many special-purpose standards, most of them must be supported by the scanning tools so as to become more complete. Further, this promotes interoperability, especially in terms of tool re-use or orchestration, as it will be shown in the next sub-section. For instance, support to CVE and CVSS can enable to merge vulnerability reporting results from different scanning tools. However, as some standards seem already prevailing while others promising, possibly different priorities must be given to different standards for their adoption. In our opinion, apart from supporting CVE, CWE and CVSS, there is a great need to also support OVAL and SCAP. The first can enable a uniform way to specify and address vulnerabilities while the second their automatic mitigation. The support to SCAP will be further elaborated in a later sub-section.

The second direction signifies the need to find multiple ways via which communities can be engaged in tool usage and enhancement. Apart from the current adoption of well-known code development and support practices, the tool providers must think new ways community engagement can occur: (a) organisation of specialised events to promote the use of scanning tools and highlight the criticality of addressing application vulnerabilities; (b) organisation of conferences, workshops and competitions to highlight recent vulnerability scanning advancements as well as the top tools possibly categorised under different competition areas; (c) supply of benchmarks or scanner selection tools to assist users in evaluating and selecting the right scanners; (d) supply of interfaces and mechanisms via which valuable user input (e.g., feedback, plugins) can be provided in a more natural and user-intuitive way.

While most tools provide some engagement support, this is not necessarily the case for academic tools. Such tools, while presented in academic forums, are not always made publicly available or offered as open-source. Further, such tools do not supply usual engagement facilities like mailing lists and code development portals which hinders their sustainability and further evolution.

6.2.2. Tool orchestration

As indicated by the evaluation results plus the existence of scanning tools that agglomerate different security tools, there is a high need to orchestrate scanning tools for various reasons. First, as their vulnerability coverage and extent must be enhanced. We have already observed that there exist tools like OpenVAS and OpenSCAP which exhibit a very nice vulnerability coverage. However, due to the their current focus, these tools' detection recall is not high, especially in terms of web application vulnerabilities. We have recently experienced this when using OpenVAS in the OWASP benchmark where OpenVAS was not able to cover all operation space parts of the benchmark application. This indicates that tools like OpenVAS must be complemented with traditional web application vulnerability scanners.

Second, as witnessed by the empirical evaluation, even web application scanners are not so good and must be complemented by source code analysis tools. If fact, as advocated in AWS-Vuln, extra tools (e.g., antivirus) apart from vulnerability scanners are needed. Thus, it is actually advocated that there is a need to orchestrate a great number of different tool kinds to achieve the best possible coverage against the whole application system in a continuous manner. Only in this way, applications can be permanently and fully protected during their whole lifetime as both applications, their components plus penetration methods and techniques continuously evolve over time.

Third, while the orchestration of scanning tools is exhibited in the market, it is not perfect. In particular, such an orchestration does not consider various conflicting factors, including: the complementarity of the tools, their integration level, the user requirements and preferences and the application kinds to be addressed. By focusing on user requirements, we see a trade-off between the following properties: scanning time, accuracy and overhead. Traditionally, there is an usual trade-off between time and accuracy. Further, as accuracy can lead to a quite intensive processing for retrieving all possible vulnerabilities, it can also affect the scanning overhead in terms of resources needed to support this scanning. This is especially true for the internal scanning mode as precious application resources can be stolen by the respective scanner. We should also not ignore the scanning frequency which could be coupled with the scanning overhead and mode. For example, it might not be suitable to frequently execute a heavy scanner in an internal mode. However, if this scanner was used externally, it could be exploited frequently.

To address the suitable orchestration of vulnerability scanning software, we believe that the following directions should be followed:

1. There is a need for complete vulnerability benchmarks that focus on evaluating scanners against different application kinds to better explore their complementarity. These benchmarks must also employ suitable scanning accuracy measures and be constructed by considering the ideal vulnerability coverage per each application kind;
2. There is a need for dynamically agglomerating vulnerability scanners according to the current context. Such a dynamic approach would enable addressing both the evolution of applications, of their requirements and of the components that comprise their system. It would also enable checking the current application context and the current ways interference can be achieved to perform the required vulnerability scanning by using the best possible tool agglomeration. For instance, we could sense that the application is under a heavy load such that we might attempt to either postpone its vulnerability scanning or perform it in a very lightweight manner. As another example, if there is a great potential for application penetration, it might be decided to perform a full scan, no matter what is the current application workload.
3. There is a need for a proper agglomeration strategy encompassing the appropriate use of the right tools of the right kind at the right moment. For instance, vulnerability scanning might be decided not to be performed in conjunction with application protection as a very heavy load could be put on the application, especially as protection testing is performed in an intrusive mode. Further, it could be decided that source code analysis could be performed each time the application is modified and in random moments in case we need to detect unexpected and irregular modifications in the application source or binary code.
4. In any case, we believe that vulnerability scanning should start from the very beginning with the checking of the application source code and the images on which it can be deployed. The latter is actually advocated by AWS-Vuln and Clair which indicate that images, no matter who creates them, usually incorporate vulnerabilities that could be due to using old software or misconfiguring system components (e.g., the OS).

6.2.3. *Inferencing*

Apart from the capability to reason about the vulnerabilities that can be exhibited by component agglomerations, the following forms of extra vulnerability-related knowledge should be captured [11]: (a) discovery of relations between vulnerabilities; (b) discovery of new vulnerabilities.

The first form enables producing a suitable mitigation plan, focusing on fixing core vulnerabilities. As such, vulnerabilities are addressed at their very root before they propagate to other vulnerabilities, thus making the whole system highly vulnerable. To achieve the production of this knowledge kind, there is a need to employ and enhance categorisations like CWE to check possible relations between vulnerabilities which need to be coupled with knowledge about what constitutes the application system and how vulnerabilities can be propagated.

The second form can enhance a tool's capability to discover new vulnerabilities and thus extend its current coverage. This can also enable the IT world to benefit from such a discovery to more rapidly react to new vulnerabilities before becoming exploitable via the development of respective code by adversaries. A tool, which exhibits such functionality, would become immediately the most utilised one as it would be adopted by most systems in the IT world. To support new vulnerability inferencing, there is a need to incorporate (semantic) rules attempting to deduce the existence of a vulnerability based on security facts or incidents. Such rules could be, for instance, derived by employing event pattern mining techniques over security logs. While this is a practice currently followed in a manual manner by well-known software providers, it could be an added-value to automate and integrate it in the cloud application management system. This automatic rule production would reduce detection costs and accelerate application evolution towards resolving the vulnerabilities detected. By also employing an approach towards properly identifying and publishing vulnerabilities, the community will benefit via: (a) the reduction in effort and time in vulnerability publishing; (b) the rapid addressing of new vulnerabilities.

6.2.4. *Mitigation*

The vulnerability world is dynamically evolving constantly such that new vulnerabilities are detected each day. As such, a user is usually faced with a great number of vulnerabilities inferred just for a single application. While a prioritisation of vulnerabilities based on their risk could enable the user to focus more on the most critical ones, we believe that this should be complemented with the capability for automatic vulnerability mitigation. The latter can enable the user to be burdened only by those vulnerabilities still critical and not automatically solvable.

The way to achieve this is to follow a twofold approach. First, an approach based on SCAP must be followed via which vulnerabilities can be automatically mitigated. This requires vulnerabilities to be coupled with mitigation specifications which can be automatically executed by scanning tools once the vulnerabilities are detected with a high confidence. This requires a great community effort, similar to that devoted already for OpenSCAP, involving major software providers and users. The main aim is to produce a global repository for vulnerability mitigation, exploitable by any scanning tool.

While, as indicated in a previous direction, proper engagement of all these providers needs to be achieved, the existence of such repository is not sufficient until two extra correlated issues are addressed. First, a vulnerability might not be always mitigated just with one piece of software or script. In fact, in different circumstances, a vulnerability might be handled by multiple mitigation actions. The selection of such actions could depend on the current situation and especially the way the application is configured. As such, mitigations need to be assorted with descriptions about their pre- and post-conditions to allow vulnerability scanning tools to more optimally select the best possible mitigation alternative. Second, in some cases, a vulnerability might not be certain to exist. This is possible, e.g., in the context of external scanning where the existence of a vulnerability can be inferred based on a respective confidence interval. This creates the need to both increase the confidence interval to a level acceptable for reaction as well as couple the vulnerability mitigation rules with conditions over this interval to avoid performing unnecessary mitigation actions. Such actions might lead to increased costs and undesirable application interruptions. Thus, they need to be kept to the minimum and performed only when needed.

6.2.5. *Architecture & APIs*

The evaluation results indicated that different tools employ different architectures in vulnerability scanning, i.e., the standalone and client-service ones. In our opinion, both architectures are valid and must be supplied by a tool to cover all possible users and the diversification of their requirements. This can certainly improve a tool's applicability. It can also enable it to scale well to cover scanning big applications. Scanning tools also need to be accompanied with an API due to the great benefits it offers, including: (a) ability to integrate the tool in a standardised way with other tools with similar or complementary functionality; (b) ability to abstract away from technical specificities and enable a clearer and more user-intuitive tool usage and administration; (c) ability to interface with a tool to create suitable visual vulnerability scanning and risk assessment UIs, enabling users to better browse and inspect in a clearer and user-intuitive way the scanning results.

All such abilities would lead to the ability to integrate scanning tools into unified compositions that can better cover all possible application vulnerabilities. Thus, the use of APIs is more than recommended to tool providers which should move away from the previous practice of providing not easily integrable tools which are not even understandable by users in some cases.

Apart from abstracting over different vulnerability scanners, the use of APIs could lead to Vulnerability Assessment as a Service scenarios. In such scenarios, users are not burdened any more with tool maintenance, administration and configuration. They can also benefit from this service's flexible pricing model to reduce costs by using the scanner only when needed. Such a service can be also easily configurable by users based on their requirements by abstracting from any tool specificities. Such a configuration could come via profiles playing the role of templates, further evolvable by users based on their requirements. Such a service could also encompass an extra-priced tool agglomeration feature, enabling to optimally satisfy user requirements.

6.2.6. Risk assessment

Most tools usually report a set of vulnerabilities assigned to a certain risk based on either a manual mapping approach or the existence of already modelled mapping knowledge in form of CVSS descriptions. In both cases, however, we see just the reporting of individual vulnerability risks with no capability to aggregate them to deduce the overall application risk. The latter capability is already exhibited by some prototypes [84,85] but not to the full possible extent covering the whole application system. As such, we expect that research should advance the current risk computation algorithms to cover the whole application system while tool providers should borrow the main research results and incorporate them in their tools for advanced reporting reasons. As such, this advanced reporting capability will increase the tool's added-value can be beneficial for users as: (a) knowledge about the whole application risk can enable selecting the most suitable mitigation strategy; (b) a deeper root cause analysis could be followed to unveil further vulnerabilities and better resolve them.

6.2.7. Composite vulnerability detection

While most tools focus on each application component individually, this is not sufficient as illegal behaviour could be detected also in component interactions. As such, there is a need to consider the application topology to have a holistic view about the whole application, its structure and how its components interact with each other along with their dependencies (e.g., hosting, communication). Further, we should focus on both the current topology description and its extension derived via reasoning. For instance, while examining a component that might be hosted on a certain container, we should not focus just on both of them but also on other components not currently covered by the topology, such as improper or non-initiated system processes. Such information could then need to be exploited towards constructing suitable vulnerability detection rules focusing on scanning the interactions between different application components.

The above direction requires to constantly observe the application's current, effective topology and to reason about its extension. Such a requirement could only be fulfilled by the proper cooperation between the scanning tool and the (cloud) application management system. The latter, in particular, should have the right, standardised interface via which the tool could obtain the extended application topology. Further, it should possess the right abilities and sensing mechanisms to sense the extra components involved in a certain system to correlate them to the current application topology.

Going beyond topology models, there can be cases where different user applications communicate to each other. Thus, one serious vulnerability in one application might have the risk to be propagated to another. However, by knowing the topology models of applications that communicate to each other, the scanning process can focus on also checking cross-application vulnerabilities (e.g., side effects of wrong transactions initiated by one, already infected application and handled by another application). This is a novel research direction, not considered before in the literature.

However, this direction might be hard to address as: (a) the knowledge about which application pairs communicate with each other might not be given by the user. In fact, it is possible that each application is handled by a different management system or different instances of such a system. As such, it should be the vulnerability assessment system that must infer this knowledge; (b) actual cross-application vulnerabilities might be hard to be defined and detected while requiring special knowledge to be given by the user (e.g., what is a wrong transaction and how someone can detect it).

The way this can be resolved is twofold. First, by usually inspecting each application from the pair in an individual manner. This relies on the rationale that each application could be considered as an end-user for the other. As such, this end-user could be considered as one source of vulnerabilities, usually checked by data validation testing and other scanning technique kinds. Second, by allowing users to supply their own checks that focus on detecting those inconsistent system states which could make the whole system or an application as vulnerable. This can be considered as a kind of user-specific vulnerability detection, enabling a scanning tool to go beyond the detection of known vulnerabilities towards application/domain-specific ones. This would certainly contribute to a better tool completeness and suitability. Coupled also with the ability to define certain mitigation actions when anticipating such inconsistencies would make the vulnerability scanning tool a full, advanced application security protection software, further enhancing its added-value and applicability.

7. Conclusions

Malware and antivirus software is widely adopted by both organisations and individuals due to the continuous and, in many cases, sophisticated risk mitigation support that it features. However, such software takes a reactive approach in dealing with security issues. On the other hand, while vulnerability scanning tools follow a proactive approach by identifying those places in the application system that need improvement to avoid security issues from happening, they are not widely used in practice. Further, even when decided to be adopted, the great tool diversity and varied coverage makes it hard for a practitioner to choose the right vulnerability scanning tool. To this end, this article offers the following contributions: (a) it guides practitioners towards making an informed decision about which vulnerability scanning tools and databases to select for their applications. Such a guidance is supplied through a comparative evaluation approach that relies on a two-level, hierarchical comparison criteria framework for both scanning tools and databases. The evaluation results supplied unveil which are the best scanning tools and databases per each criterion, category of criteria and in overall; (b) the framework could be adapted to include different weights (currently equal weights are assumed) on the respective criteria and categories of criteria highlighting their relative importance. This could then enable to modify the evaluation results to make them fit to the current usage context; (c) being one of the survey goals, the article proves that vulnerability scanning tool orchestration can enable to reach a higher vulnerability detection coverage; (d) by moving to the academic side, the evaluation results also highlight the exact places for tool improvement, presented in the form of certain challenges which have not yet been confronted in research; (e) finally, this article explains in which application lifecycle places vulnerability scanning can be performed and supplies valuable insights towards better securing cloud applications.

Declaration of Competing Interest

The authors declare no conflict of interest.

Acknowledgements

We thankfully acknowledge the support of the UNICORN (Grant Agreement no. 731846) and CONCORDIA (Grant Agreement no. 830927) H2020 European projects.

Appendix A. OWASP Taxonomy

OWASP has proposed a vulnerability tool categorisation/taxonomy which comprises three levels. At the first level, the two top categories of: (a) *web application vulnerability detection* tools and (b) *web application protection* tools (e.g., malware and antivirus tools) exist. The former tools are then categorised at the second level into: (i) *threat modelling* tools, (ii) *source code analysis* tools (SAST), (iii) *vulnerability scanning* tools, (iv) *interactive application security testing* tools (IAST), and (v) *penetration testing* tools. Penetration testing tools are further classified based on the following partitions: *information gathering* tools, *configuration management testing* tools, *authentication testing* tools, *session management testing* tools, *authorisation testing* tools, *data validation testing* tools, *denial of service testing* tools, *web services testing* tools, *Ajax testing* tools, *HTTP traffic monitoring*, *encoders/decoders*, and *web testing frameworks*.

Appendix B. Approach for Evaluating Vulnerability Coverage of Scanning Tools

In order to evaluate the coverage of each vulnerability scanning tool, we have relied on a three-step approach. This approach had the rationale that coverage means the actual percentage of the different kinds of vulnerabilities that can be detected by a certain tool.

In this respect, the first approach step involved the proper identification of the different vulnerability kinds that can be detected. Towards realising this step, we have made an investigation of different frameworks that might be available in the literature. Such an investigation ended-up in selecting the framework in sectoolmarket. com which comprises a vast amount of the most frequent web application vulnerabilities categories. The focus of this framework is on web application with the rationale that usually the focus is mainly on this kind of vulnerable object and not all possible ones. In fact, we can actually deduce that there is a lack of frameworks that attempt to categorise vulnerabilities beyond this object kind.

As such, based on this framework, 33 vulnerability categories were identified, for which the definition can be inspected in.[1] These categories are shown in the first column of Table 10.

Once the right set of vulnerability categories was identified, the second step of the followed approach involved the assessment of each vulnerability scanning tool considered based on whether it covers each of the categories identified. Towards this goal, we have relied both: (a) on the actual findings[2] from a previously conducted research that is reflected in sectoolmarket. com about the coverage of a certain set of open-source vulnerability scanners as well as (b) on an individual evaluation, conducted by us, of those scanners which were not included in the aforementioned evaluated set.

Table 10
Mapping of scanning tools to the vulnerability kinds they cover.

Vulnerability Kind	Vulnerability Scanning Tool
SQL Injection	WGrabber, Nikto2, Vega, Wapiti, ZAP, OpenVAS, w3af, arachni, IronWASP, OpenSCAP, Clair
Time-Based SQL Injection	Grabber, Vega, Wapiti, ZAP, OpenVAS, w3af, arachni, IronWASP, OpenSCAP
Server-Side Javascript Injection	OpenVAS, arachni, OpenSCAP
Reflected Cross-Site Scripting	Grabber, Vega, Wapiti, Xenotic XSS, ZAP, OpenVAS, w3af, arachni, IronWASP, OpenSCAP, Clair
Persisted Cross-Site Scripting	Wapiti, Xenotic XSS, ZAP, OpenVAS, w3af, IronWASP, OpenSCAP
DOM-based Cross-Site Scripting	Xenotic XSS, OpenVAS, w3af, arachni, IronWASP, OpenSCAP
JSON Hijacking	OpenVAS, OpenSCAP
Local File Inclusion	Grabber, Nikto, Vega, Wapiti, ZAP, OpenVAS, w3af, arachni, IronWASP, OpenSCAP, Clair
Remote File Inclusion	Wapiti, Nikto, ZAP, OpenVAS, w3af, arachni, IronWASP, OpenSCAP
Command Injection	Nikto2, Vega, Wapiti, ZAP, OpenVAS, w3af, arachni, IronWASP, OpenSCAP, Clair
Unrestricted File Upload	Wapiti, OpenVAS, w3af, arachni, OpenSCAP
Open Redirect	Vega, ZAP, OpenVAS, w3af, arachni, IronWASP, OpenSCAP
CRLF Injection	Vega, Wapiti, ZAP, OpenVAS, w3af, arachni, IronWASP, OpenSCAP, Clair
LDAP Injection	Wapiti, ZAP, OpenVAS, w3af, arachni, IronWASP, OpenSCAP
XPath Injection	Wapiti, ZAP, w3af, arachni, IronWASP
Email Injection	OpenVAS, w3af, OpenSCAP
Server-Side Includes	ZAP, OpenVAS, w3af, IronWASP, OpenSCAP
Format String Attack	Vega, OpenVAS, w3af, OpenSCAP
Code Injection	ZAP, OpenVAS, arachni, IronWASP, OpenSCAP
XML Injection	Vega, OpenVAS, arachni, OpenSCAP
EL Injection	IronWASP, OpenSCAP
Buffer Overflow	OpenVAS, w3af, OpenSCAP, Clair
Integer Overflow	Vega, OpenVAS, OpenSCAP, Clair
Source Code Disclosure	Vega, OpenVAS, w3af, arachni, OpenSCAP
Backup Files	Grabber, Wapiti, ZAP, OpenVAS, w3af, arachni, OpenSCAP
Padding Oracle	OpenVAS, OpenSCAP
Authentication Bypass	Nikto2, ZAP, OpenVAS, w3af, OpenSCAP, Clair
Privilege Escalation	OpenVAS, OpenSCAP, Clair
XXE Injection	Wapiti, OpenVAS, arachni, OpenSCAP
Weak Session Identifier	w3af, OpenVAS, IronWASP, OpenSCAP
Session Fixation	ZAP, OpenVAS, arachni, IronWASP, OpenSCAP
Cross-Site Resource Forgery	ZAP, OpenVAS, w3af, arachni, OpenSCAP, Clair
Application Denial of Service	Nikto2, Wapiti, OpenVAS, w3af, OpenSCAP, Clair

This second step resulted in the mapping of each vulnerability category into a set of vulnerability scanning tools that cover it which is reflected in the second column of Table 10. Based on the results of this step, the last approach step involved the calculation of the actual coverage percentage of this tool by computing the division between the vulnerability categories covered and all possible ones. This coverage percentage was then mapped to a

[1] http://www.sectoolmarket.com/audit-features-comparison-unified-list.html#Glossary.
[2] http://www.sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-opensource-list.html.

qualitative scale comprising 6 main values according to the following rule set:

- *percentage* <= 16.66% → coverage = "Very Low"
- 16.66% < *percentage* <= 33.33% → coverage = "Low"
- 33.33% < *percentage* <= 50.00% → coverage = "Medium"
- 50% < *percentage* <= 66.66% → coverage = "Good"
- 66.66% < *percentage* <= 83.33% → coverage = "High"
- 83.33% < *percentage* <= 100.00% → coverage = "Very High"

The final result is depicted in Table 11 which is a subset of Table 4 focusing solely on the *Vuln. Coverage* functional criterion.

Table 11
Quantitative & Qualitative Vulnerability Coverage.

Tool/Approach	Percentage	Coverage
Grabber	15.15%	Very Low
Lismero	93.93%	Very High
Nikto2	18.18%	Low
Vega	33.33%	Low
Wapiti	42.42%	Medium
OWASP Xenotic XSS	9.09%	Very Low
OWASP ZAP	51.51%	Good
OpenVAS	93.93%	Very High
Arachni	60.60%	Good
IronWASP	51.51%	Good
w3af	69.69%	High
OpenSCAP	96.96%	Very High
Clair	33.33%	Low
Vulcan	%	Very Low
HPI-Vuln	93.93%	Very High
UC-OPS	93.93%	Very High
AWS-Vuln	93.93%	Very High

It might be observed that some scanning tools have the same coverage. This is due to the fact that all these tools rely on OpenVAS. As we were not able to assess them easily over their actual coverage, we have considered that their accuracy could not be less than that of the sub-tool that they exploit. In this sense, as the coverage of OpenVAS is the best possible, mapping to the top coverage partition, our consideration is precise according to the scale that has been adopted. Further, we should also note that we were not able to assess Vulcan as its code was not available. However, based on its respective documentation, it relies on a certain limited scanning framework which is expected to have a very low coverage. So, our estimation should be correct here.

References

[1] Chromy JR. Encyclopedia of survey research methods. Thousand Oaks: SAGE Publications, Inc.; 2019.
[2] Common vulnerabilities and exposures. 2019. https://cve.mitre.org.
[3] Tsipenyuk K, Chess B, McGraw G. Seven pernicious kingdoms: a taxonomy of software security errors. IEEE Secur Priv 2005;3:81–4.
[4] OWASP top 10. OWASP_Top_Ten_Project; 2019. https://www.owasp.org/index.php/Category.
[5] National vulnerability database. 2019. https://nvd.nist.gov/.
[6] Common vulnerability scoring system. 2019. https://www.first.org/cvss/.
[7] Common weakness enumeration. 2019. https://cwe.mitre.org/.
[8] Open source vulnerability database. 2019. https://en.wikipedia.org/wiki/Open_Source_Vulnerability_Database.
[9] Hasso-plattner vulnerability database. 2019. https://hpi-vdb.de/vulndb/.
[10] Cheng F, Roschke S, Schuppenies R, Meinel C. Remodeling vulnerability information. In: Bao F, Yung M, Lin D, Jing J, editors. Information security and cryptology. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010. p. 324–36.
[11] Kamongi P, Kotikela S, Kavi K, Gomathisankaran M, Singhal A. VULCAN: vulnerability assessment framework for cloud computing. In: Sere. Washington, DC, USA: IEEE Computer Society; 2013. p. 218–26.
[12] Rapid7 vulnerability and exploit database. 2019. https://www.rapid7.com/db.
[13] Trusted vulnerability & threat intelligence database. 2019. https://vfeed.io/.
[14] Redhat embedded vulnerability detector. 2019. https://access.redhat.com/labsinfo/jevd.
[15] Securityfocus vulnerabilities. 2019. https://www.securityfocus.com/bid.
[16] Vuldb. 2019. https://vuldb.com/.
[17] Securitytracker. 2019. https://securitytracker.com/.
[18] Zeroday published advisories. 2019. https://www.zerodayinitiative.com/advisories/published/.
[19] Offensive security's exploit database archive. 2019. https://www.exploit-db.com/.
[20] NCCIC. 2019. https://ics-cert.us-cert.gov/.
[21] Japan vulnerability notes. 2019. https://jvn.jp/en/.
[22] Auscert security bulletins. 2019. https://www.auscert.org.au/bulletins/.
[23] CERT-EU security advisories. 2019. https://cert.europa.eu/cert/newsletter/en/latest_Security\'\%20Bulletins_.html.
[24] Common platform enumeration. 2019. https://cpe.mitre.org.
[25] Common attack pattern enumeration and classification. 2019. https://capec.mitre.org/.
[26] Open vulnerability and assessment language. 2019. https://oval.mitre.org/.
[27] Web application security Consortium project page. 2019. http://projects.webappsec.org/w/page/13246927/FrontPage.
[28] OWASP taxonomy. Tools_Categories; 2019. https://www.owasp.org/index.php/Category.
[29] Grabber security and vulnerability analysis. 2019. https://github.com/amoldp/Grabber-Security-and-Vulnerability-Analysis-.
[30] Golismero group. 2019. http://www.golismero.com/.
[31] Nikto2. 2019. https://cirt.net/Nikto2.
[32] Vega vulnerability scanner. 2019. https://subgraph.com/vega/.
[33] Wapiti vulnerability scanner. 2019. http://wapiti.sourceforge.net/.
[34] OWASP xenotix XSS exploit framework. 2019. https://www.owasp.org/index.php/OWASP_Xenotix_XSS_Exploit_Framework.
[35] OWASP zed attack proxy project. 2019. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.
[36] Openvas vulnerability scanner. 2019. http://www.openvas.org/.
[37] Arachni web application security scanner. 2019. http://www.arachni-scanner.com/.
[38] IronWASP. 2019. https://ironwasp.org/.
[39] w3af web application attack and audit framework. 2019. http://w3af.org/.
[40] OpenSCAP. 2019. https://www.open-scap.org/.
[41] Scherf T. Openscap: security compliance with openscap. 2015. Admin.
[42] NIST security content automation protocol. 2019. https://csrc.nist.gov/projects/security-content-automation-protocol/.
[43] Common configuration enumeration. 2019. https://cce.mitre.org/about/index.html.
[44] Xccdf - the extensible configuration checklist description format. 2019. https://csrc.nist.gov/projects/security-content-automation-protocol/scap-specifications/xccdf.
[45] Vulnerability static analysis for containers. 2019. https://github.com/coreos/clair.
[46] Torkura KA, Cheng F, Meinel C. Aggregating vulnerability information for proactive cloud vulnerability assessment. J Internet Technol Secur Trans 2015;4:387–95.

[47] Schwarzkopf R, Schmidt M, Strack C, Martin S, Freisleben B. Increasing virtual machine security in cloud environments. J Cloud Comput: Adv Syst Appl 2012;1:12.
[48] Balduzzi M, Zaddach J, Balzarotti D, Kirda E, Loureiro S. A security analysis of amazon's elastic compute cloud service. In: SAC, ACM; 2012. p. 1427–34. Trento, Italy.
[49] Nessus professional. 2019. https://www.tenable.com/products/nessus/nessus-professional.
[50] Javascript lint. 2019. http://www.javascriptlint.com/.
[51] PHP-sat. 2019. http://program-transformation.org/PHP/PhpSat.
[52] Nmap. 2019. https://nmap.org/.
[53] SSLScan. 2019. https://github.com/DinoTools/sslscan.
[54] OWASP DirBuster project. OWASP_DirBuster_Project; 2019. https://www.owasp.org/index.php/Category.
[55] JBroFuzz project. 2019. https://www.owasp.org/index.php/JBroFuzz.
[56] Greenbone security assistant. 2019. https://github.com/greenbone/gsa.
[57] ike-scan. 2019. https://github.com/royhills/ike-scan.
[58] snmpwalk. 2019. http://net-snmp.sourceforge.net/tutorial/tutorial-5/commands/snmpwalk.html.
[59] thc-amap. 2019. https://www.thc.org/thc-amap/.
[60] Ldapsearch - LDAP search tool. 2019. http://www.openldap.org/software//man.cgi?query=ldapsearch&apropos=0&.
[61] Security local auditing daemon. 2019. http://www.openvas.org/compendium/security-local-auditing-daemon.html.
[62] tOVAL Interpreter. 2019. https://sourceforge.net/projects/ovaldi/.
[63] Pnscan - a parallell network scanner. 2019. https://github.com/ptrrkssn/pnscan.
[64] portbunny. 2019. http://www.recurity.de/portbunny/portbunny.shtml.
[65] WiHawk tool. 2019. https://github.com/Anamika21/WiHawk.
[66] XMLCHOR xpath injection exploitation tool. 2019. https://github.com/Harshal35/XMLCHOR.
[67] IronSAP. 2019. https://github.com/prasanna-in/IronSAP.
[68] SSLSecurityChecker - IronWASP module. 2019. https://github.com/GDSSecurity/SSLSecurityChecker.
[69] OWASP Skanda SSRF exploitation framework. 2019. https://www.owasp.org/index.php/OWASP_Skanda_SSRF_Exploitation_Framework.
[70] Hawas. 2019. https://github.com/lavakumar/hawas.
[71] CSRF PoC generator. 2019. https://github.com/jayeshchauhan/csrf_poc_generator.
[72] ClamAV antivirus engine. 2019. https://www.clamav.net/.
[73] Chk-root-kit. 2019. http://www.chkrootkit.org/.
[74] Rkhunter - rootkit hunter project. 2019. http://rkhunter.sourceforge.net/.
[75] Sysinternals utilities index. 2019. https://docs.microsoft.com/el-gr/sysinternals/downloads/.
[76] John the Ripper password cracker. 2019. http://openwall.com/john/.
[77] Extundelete - recover deleted files. 2019. http://extundelete.sourceforge.net/.
[78] WhatWeb web scanner. 2019. https://github.com/urbanadventurer/WhatWeb.
[79] Mitra J, Ranganath V. Ghera: a repository of android app vulnerability benchmarks, CoRR abs/1708. 2017, 02380.
[80] Tajpour A, Ibrahim S. A framework for evaluation of SQL injection detection and prevention tools. Int J Info Commun Technol Res 2013;5:55–62.
[81] Find security bugs scanner. 2019. https://find-sec-bugs.github.io.
[82] Find bugs scanner. 2019. http://findbugs.sourceforge.net/.
[83] Sonarqube scanner. 2019. https://www.sonarqube.org.
[84] Li HC, Liang PH, Yang JM, Chen SJ. Analysis on cloud-based security vulnerability assessment. In: Icebe. IEEE Computer Society; 2010. p. 490–4.
[85] Saripalli P, Walters B. QUIRC: a quantitative impact and risk assessment framework for cloud security. In: Cloud. IEEE Computer Society; 2010. p. 280–8.