No Keys to the Kingdom Required: A Comprehensive Investigation of Missing Authentication Vulnerabilities in the Wild

Manuel Karl*
TU Braunschweig
m.karl@tu-braunschweig.de

Marius Musch*
TU Braunschweig
m.musch@tu-braunschweig.de

Guoli Ma
Google
magl@google.com

Martin Johns
TU Braunschweig
m.johns@tu-braunschweig.de

Sebastian Lekies
Google
slekies@google.com

ABSTRACT

Nowadays, applications expose administrative endpoints to the Web that can be used for a plethora of security sensitive actions. Typical use cases range from running small snippets of user-provided code for rapid prototyping, administering databases, and running CI/CD pipelines, to managing job scheduling on whole clusters of computing devices. While accessing these applications over the Web make the lives of their users easier, they can be leveraged by attackers to compromise the underlying infrastructure if not properly configured.

In this paper, we comprehensively investigate inadequate authentication mechanisms in such web endpoints. For this, we looked at 25 popular applications and exposed 18 of them to the Internet because they were either vulnerable in their default configuration or were easy to misconfigure. We identified ongoing attacks against 7 of them, some were even compromised within a few hours from the deployment. In an Internet-wide scan of the IPv4 address space, we examine the prevalence of such vulnerable applications at scale. Thereby, we found 4,221 vulnerable instances, enough to create a small botnet with little technical knowledge. We observed these vulnerable instances and found that even after four weeks, more than half of them were still online and vulnerable.

Currently, most of the identified vulnerabilities are seen as features of the software and are often not yet considered by common security scanners or vulnerability databases. However, via our experiments, we found missing authentication vulnerabilities to be common and already actively exploited at scale. They thus represent a prevalent but often disregarded danger.

CCS CONCEPTS

• **Security and privacy** → **Web application security**; **Vulnerability scanners**;

*Both authors contributed equally to this research.

IMC '22, October 25–27, 2022, Nice, France

ACM ISBN 978-1-4503-9259-4/22/10.
https://doi.org/10.1145/3517745.3561446

ACM Reference Format:
Manuel Karl, Marius Musch, Guoli Ma, Martin Johns, and Sebastian Lekies. 2022. No Keys to the Kingdom Required: A Comprehensive Investigation of Missing Authentication Vulnerabilities in the Wild. In *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22), October 25–27, 2022, Nice, France.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3517745.3561446

1 INTRODUCTION

Remotely administering software, servers, or whole clusters of computing devices can be a difficult task. Thus, a plethora of applications offer easy and convenient remote access to administrative functions such as querying a database, uploading files to a server, or executing OS commands via a HTTP(S) endpoint. Examples for such applications are CI/CD interfaces, content management systems, cluster management systems, notebooks for code, or database control panels. Many of these applications have been designed for usage within an intranet or on localhost and thus most of them are lacking strong authentication mechanisms. However, with the advent of cloud computing, more and more of these applications are getting deployed to the cloud and need to be accessed remotely.

Due to the lack of authentication mechanisms, it is dangerous to expose such applications to the Internet. For example, an exposed CI/CD admin panel can be leveraged by an attacker to push a malicious binary to a build server. Most administrative operations, such as querying a database, uploading a file, or running jobs on a computing cluster, can be turned into arbitrary code execution. Therefore, exposing an administrative web endpoint to the Internet without an authentication mechanism in place can easily lead to exposing a remote code execution vulnerability.

In this paper, we study the exploitation and prevalence of such missing authentication vulnerabilities in the wild. To the best of our knowledge, we are the first to systematically investigate this problem at scale and to quantify the potential of maliciously exploiting it. For this, our paper makes the following contributions:

- First, we manually investigated 25 popular applications across five categories for publicly exposed sensitive functionality. Thereby, we found that 18 out of 25 software products contain sensitive endpoints without authentication. Hence, when exposed to the Internet, such applications can be compromised by attackers (Section 2).

- Subsequently, we scanned the entire IPv4 range for missing authentication vulnerabilities. Our scan took less than one day to complete and discovered 4,221 instances that exposed sensitive application functionality without authentication. Effectively, each of these instances is vulnerable to remote code execution and we could have created a small botnet by abusing these vulnerabilities (Section 3).
- To study whether attackers actively exploit administrative web endpoints in practice, we set up honeypots for the 18 vulnerable applications. Within four weeks we recorded 2,195 attacks leading to compromises of 7 of the 18 vulnerable applications, often even within a few hours of exposure (Section 4).
- Finally, we use two commercial industry-leading security scanners to show that defenders are mostly not aware of the risks, as they only detect 5 and 3 out of the 18 vulnerabilities. Our data analysis shows that applications relying on secure-by-default mechanisms are considerably less prone to compromises (Section 5 and Section 6).

2 MISSING AUTHENTICATION VULNERABILITIES

In this paper, we focus on applications that expose *administrative web endpoints (AWEs)*. Conducting administrative actions over them can either be their main purpose, e.g., managing a single server or a whole cluster, or these endpoints can be part of a larger application in the form of an additional admin panel, e.g., a management interface for a blog. Note *web endpoints* does not only include traditional web applications like *WordPress*, but also applications that additionally expose an HTTP API like *Kubernetes* or *Docker*.

Due to their powerful capabilities, accessing these endpoints should only be allowed for the legitimate owner of the application. An attacker would not need to abuse "real" vulnerabilities such as buffer overflows to gain control of the program flow, as these applications provide arbitrary code execution *by design*. We call the absence of an authentication mechanism in AWEs a *missing authentication vulnerability (MAV)*. As this attack vector is based on a feature and not a bug, even keeping the system up-to-date does not prevent exploitation of MAVs. Simply exposing these AWEs to the Internet is already a security risk, as an attacker can use the provided functionality to run malicious code.

Flavors of MAVs In the simplest case, AWEs allow their users to directly execute system commands on the server via a web interface, such as a Web page or an HTTP endpoint. For example, by offering a terminal to execute bash commands or the possibility to write code that is executed on the server. Others offer their administrative capabilities through an HTTP API which basically serves as a wrapper for the system commands. This is a bit more hidden than a graphical interface and sometimes not clear to the user that an HTTP API even exists by default after installation.

A special case is the lack of authentication during installation only. This happens, if the installation starts with extracting the application files on the web server and then configuring the rest, including the admin password, in the browser. In this case, the first person connecting to the server is assumed to be the legitimate owner, so these applications basically rely on trust on first use. It should be noted that once the installation is complete, these applications require authentication and are secure by default.

Out of scope For this paper, we consider insecure default credentials that can be easily guessed or found in the documentation, like *admin:admin*, to be out of scope. While the employed authentication is weak, these systems are not *missing* authentication. Another more practical reason is that we would not be able to test for default credentials in an ethical and non-invasive manner, i.e., without conducting POST requests. Moreover, we also do not investigate authentication bypasses, where a strong but buggy authentication mechanism exists. However, we *do* consider applications that can be misconfigured in a way that skips the authentication, e.g., by providing an empty password.

2.1 Manual Investigation of MAVs

We divided the AWEs into five different categories and selected the five most popular open source representatives based on their number of Github stars. We outline if and how they are affected by MAVs and investigate if they are insecure in their default configuration. Moreover, we discuss if this changed over time, e.g., because they changed their defaults or introduce other countermeasures to prevent attacks. If the application is insecure by default, we investigate if there are any warnings, e.g., in their installation documentation, during startup, or when first visiting the application. We summarize our findings in Table 1.

Continuous Integration (CI) These systems help to automate the deployment of software through automatic builds of the project each time the code changes or on a certain schedule. For this category, we analyzed *Gitlab, Drone CI, Jenkins, Travis CI*, and *GoCD*. By default, only GoCD allows unauthenticated users to run arbitrary commands as part of the build process. Their documentation [30] states that "A newly installed GoCD server does not require users to authenticate. This is great for a trial. However, it is one of the first things you should change [...]". Older versions of Jenkins before v2.0 were also vulnerable, however they introduced a countermeasure and now create an admin account with a random password during the installation [56].

Content Management Systems (CMS) These systems allow users to create and upload new content such as text and media, organize this content and provide extensive capabilities to customize the appearance of this content. By far, the most popular CMS is *WordPress*, which is estimated to manage content on over 40% of all websites worldwide [78]. In the CMS category we selected *Ghost, WordPress, Grav, Joomla*, and *Drupal* for manual analysis. All five include an password-protected admin panel. However, this password is initially set during the installation process on a publicly reachable web page, meaning an attacker can just set whatever they want. Therefore, the CMSes are generally in scope, as long as they have not yet been installed. While Ghost offers an admin panel to edit the content, it does not allow the user to change the actual code of the application and is therefore out of scope. On the other hand, all other four CMSes offer the ability to add or edit PHP code either directly or through the upload of custom extensions and thus

allow an attacker to execute arbitrary code on the server. Joomla is the only one to include a countermeasure to such installation hijacks since version v3.7.4. If you do not know the password of the local MySQL server and want to connect to a remote one, then the installation will only continue if you delete a file with a random name on the server, proving your ownership [44].

Cluster Management (CM) Software in this category allows managing clusters by creating, updating or deleting nodes of the cluster or running jobs on the nodes. For the CM category, we look into *Kubernetes, Docker, HashiCorp Consul, Apache Hadoop*, and *HashiCorp Nomad*. Each of them offers an API that is enabled by default and allows code execution. By default, only Kubernetes does not expose this API to the Internet. However, this can be changed via the configuration [50]. Consul exposes its API by default, but is only vulnerable if either the `enableScriptChecks` or the `enableRemoteScriptChecks` option is manually enabled [10]. Both these options allow an remote attacker to execute arbitrary code: The first enables health checks that regularly execute previously the provided code, while the latter is just outright designed to execute code from remote sources. Of the remaining three products, we only found a prominent warning for Nomad, which said "Nomad is not secure-by-default" [55].

Notebooks (NB) Notebooks typically allow to write documents that contain a mix of text, code, and figures. To ease debugging and other administrative tasks, they often even provide a web terminal to execute bash commands on the server. We looked at *Jupyter Lab, Jupyter Notebook, Apache Zeppelin, Polynote*, and *Spark Notebook*. Both Jupyter products can be misconfigured by passing an empty string via the `–NotebookApp.password` option to disable all authentication. For Jupyter Notebook, the generation of a random password during the installation was only later introduced in the update v4.3 [63]. On the other hand, Zeppelin and Polynote do not enable any authentication by default. However, Polynote has a warning on the download page that says "Like other notebook tools, a large part of its usefulness relies on arbitrary remote code execution [...] and relies entirely on the user deploying and configuring it in a secure way" [62]. Spark Notebook seems to be discontinued with no updates since February 2019 and was thus excluded from our study.

Control Panels (CP) In this category, we include all other control panels with administrative functionality that are not part of one of the previous four categories. This include generic admin panels which provide system statistics and control panels to manage database servers. We looked at *Ajenti, phpMyAdmin (PMA), Adminer, VestaCP,* and *OmniDB*. Ajenti requires the credentials of a local OS account by default. It comes with an `–autologin` option, but the documentation explicitly warns that "this is a security issue if your system is public" [3]. PMA and Adminer both require valid credentials for a SQL user. PMA does not allow supplying an empty password unless the `allowNoPassword` option is explicitly enabled. By default, the option is set to false to prevent logging in without a password and prevent unwanted access to the MySQL server. Adminer does not allow an empty password at all anymore since mid 2018 [76]. VestaCP and OmniDB automatically set or generate a password during the installation, with no option to misconfigure, and are therefore out of scope.

Out of scope We have chosen the five categories carefully to address a large number of different sensitive functionalities that should be protected by authentication. We consider applications in similar categories or with a similar functionality to be out of scope for this paper. For example, in the case of CMSes we are looking at indirect code execution via the modification of template files or via installing malicious plugins. It makes little difference whether the vulnerable application allows the management of a blog, a forum, or any other type of web presence, as the attack vector is the same.

Summary of our manual analysis Overall, we looked at 25 popular products in five different categories and found that 18 are in scope for our study on missing authentication vulnerabilities. Of these, 9 are insecure by default, 4 were insecure by default in an older version, and another 5 are easy to misconfigure by exposing a port or changing a setting.

Table 1: Summary of all investigated applications with their corresponding attack vector and Github ranking.

Type	App	Stars	Vuln	Default MAV	Warn
CI	Gitlab [29]	23k	–	–	–
CI	Drone [15]	23k	–	–	–
CI	Jenkins [42]	18k	Syscmd	< 2.0 (2016)	–
CI	Travis [74]	8k	–	–	–
CI	GoCD [72]	6k	Syscmd	✓	✓
CMS	Ghost [28]	38k	–	–	–
CMS	WordPress [80]	15k	Install	✓	✗
CMS	Grav [27]	13k	Install	✓	✗
CMS	Joomla [43]	4k	Install	< 3.7.4 (2017)	–
CMS	Drupal [16]	4k	Install	✓	✗
CM	Kubernetes [49]	78k	API	✗	–
CM	Docker [12]	23k	API	✓	✗
CM	Consul [36]	22k	API	✗	–
CM	Hadoop [6]	12k	API	✓	✗
CM	Nomad [37]	9k	API	✓	✓
NB	J-Lab [46]	11k	Syscmd	✗	–
NB	J-Notebook [45]	8k	Syscmd	< 4.3 (2016)	–
NB	Zeppelin [7]	5k	Syscmd	✓	✗
NB	Polynote [61]	4k	Syscmd	✓	✓
NB	Spark NB [5]	3k	–	–	–
CP	Ajenti [2]	6k	Syscmd	✗	✓
CP	phpMyAdmin [60]	6k	SQL	✗	✗
CP	Adminer [77]	5k	SQL	< 4.6.3 (2018)	–
CP	VestaCP [75]	3k	–	–	–
CP	OmniDB [58]	3k	–	–	–

On the other hand, 7 of these 18 AWEs allows the attacker to directly execute system commands, 5 expose a critical API, 2 allow to execute SQL commands and 4 are unsafe in their pre-installation state. While on the surface an unfinished WordPress installation might appear to be quite different from a Jupyter Notebook server, when exposed to the Internet their impact is actually the same as they both give the attacker the ability to execute arbitrary code.

2.2 Research Questions

So far, our manual analysis confirmed that with 18 out of 25 AWEs being vulnerable by default or easy to misconfigure, MAVs are, at least in theory, a common occurence. In the remainder of this paper, we will now investigate how this relates to the practice conducted by *users, attackers, and defenders* in the real world.

To comprehensively explore this phenomenon, we ask and answer seven *research questions (RQs)* on the following three major

aspects. First of all, in Section 3, we determine the *usage* of AWEs in the wild to see how often and for how long these endpoints are exposed in a vulnerable state.

- **RQ1:** Is this a prevalent issue, i.e., how often are AWEs and their MAVs exposed on the Internet?
- **RQ2:** How many of them are up-to-date but vulnerable due to insecure defaults or misconfiguration?
- **RQ3:** How long do endpoints with MAVs stay online and exposed in a vulnerable state?

Then, in Section 4, we also want to find out if *attackers* are aware of this problem and if so, how the attacks are distributed across both time and malicious actors.

- **RQ4:** Are administrative endpoints compromised in the wild and to what purpose?
- **RQ5:** How long does a vulnerable application survive without getting compromised?
- **RQ6:** How is the attack landscape shaped, e.g., how many attackers compete for the same vulnerabilities?

Finally, we also briefly investigate if *defenders* are already aware of this problem in Section 5.

- **RQ7:** Are defenders aware of MAVs, i.e., do security scanners detect this vulnerability?

3 PREVALENCE OF MAVS

To comprehensively answer RQ1-3, we used an Internet-wide scan of *all* IPv4 hosts, as we do not expect AWEs to attract enough visitors to be represented in lists such as the Tranco Top 1 Million [51]. In the following, we describe our three-stage methodology for this data collection, as well as briefly outline the setup of our experiment. Finally, we report on our results in detail.

3.1 Scanning Methodology

On a high level, our methodology consists of three stages that each act as a filter, i.e., the earlier stages remove targets that are out of scope for the later, slower stages. Finally, we use a fingerprinter to extract version information from the scanned application for an analysis on their default settings.

Stage I - Masscan As the first step, we checked which hosts are actually online and exposing an endpoint to the Internet. For this, we made use of the existing tool *Masscan* [34], which is an extremely fast port scanner written in C. We excluded all IANA reserved allocations [39], such as those reserved for Multicast, private use, or the US Department of Defense, leaving us with roughly 3.5B IPv4 addresses. We limited our scan to the 12 most important ports for our study: 80, 443, and all default ports of the 18 selected applications (which have some overlap). To prevent running the next two stages on hosts that went offline in the meantime, we always selected and scanned a fraction of all hosts with our full pipeline before we continued the port scan with the next batch.

Stage II - Prefilter For all open ports identified by the first stage, we then checked if they speak HTTP or HTTPS, except port 80 where we only tested HTTP, and port 443 where we only tested for HTTPS. If one or both connections succeeded, we followed redirects until we received a response body and searched it with our *prefilter signatures*. These signatures are short regular expressions, e.g., `wp-json` for WordPress and `/static/yarn.css` for Hadoop, were crafted manually to indicate that the response was generated by one of the 18 applications of our study. In total, we created 90 such signatures, an average of 5 per application. This stage thus acts as a *prefilter* that discards all hosts that are not relevant for our study.

Stage III - Tsunami At this point, we confirmed that the application is in scope for our study and now need to verify whether it suffers from a MAV. For this, we created and open sourced a generic network security scanner called *Tsunami* [33]. It has an extensible plugin system and each MAV verification logic is implemented as a dedicated plugin. Based on the port and application information from *Stage I* and *Stage II*, Tsunami selects the appropriate MAV detection plugins for each matching application. For example, in order to check whether a WordPress installation can be hijacked, we have a plugin that queries the `/wp-admin/install.php` page and checks whether the WordPress installation process is served on that link. The full details of all detection steps of all the plugins can be found in Appendix A. Moreover, we are in the process of open-sourcing all detection plugins and many of them are already available [32].

Version fingerprinting For more detailed analyses of the hosts, we also tried to determine the version of all applications that are in scope for our study to infer their default settings (which could have changed over time), as well as their up-to-dateness in general. Therefore, we first try to extract the exact version number from the 13 applications where this information is usually voluntarily revealed, e.g., Kubernetes has the `/version` API endpoint while Consul includes a HTML comment. For the five remaining applications, as well as cases where this version number was removed, we employed a more elaborate fingerprinting mechanism. It is based on two major components, a *knowledge base* and a *crawler*, and is implemented as an additional Tsunami plugin. The knowledge base is built using the repositories of the open-source applications and includes hashes of their static files such as images, scripts and stylesheets. To identify an unknown application we first crawl the application and collect all static files from the responses. Then we match their hashes against the knowledge base to identify the application and version. This fingerprinter is also publicly available, including our knowledge base for all the applications [31].

Threats to validity To ensure that our prefilter signatures and Tsunami's MAV detection plugins work on older versions of the targeted software, we tested them on both the newest and oldest stable releases we could find, as well as random samples collected during preliminary, smaller scans. We refined them and added additional checks and signatures as needed, looking for strings and endpoints that appeared stable across all the different versions. However, there is a small chance that some version in between introduced a breaking change that was later rolled back, meaning we would miss some results.

3.2 Experiment Setup

Using the three-stage scanning methodology described above, we conducted a scan of all IPv4 addresses on June 03, 2021. To conduct this large scan within one day we spun up 64 machines through a large cloud provider, each with 48 cores and 384 GB of memory.

With this setup, the experiment lasted about 22 hours. Additionally, we used a smaller machine to act as an *observer* to periodically repeat the scan of the vulnerable machines, to see whether they were still running and vulnerable.

Ethical considerations As the first two stages only conduct a port scan and a maximum of three HTTP(S) requests, the burden on one individual system is small. Moreover, to prevent flooding a whole network with our requests, we scanned the IPv4 range in a random order of 24 blocks instead of scanning them sequentially. The third stage of our scanning pipeline only verifies the existence of sensitive UIs or APIs in a non-intrusive way. To prevent both ethical and legal issues, we did not try to circumvent existing authentication mechanisms, nor did we abuse the MAVs to gather more information about the system, e.g., by extracting hardware information of the host. Instead, our scanner is limited to non-state-changing GET requests and thus acts very similar to a general Web crawler. This is even true for PhpMyAdmin and Adminer, which do not need POST requests to test if an empty password would be accepted. Therefore, we can only infer the presence of a MAV from the presence of the web pages that contain the vulnerable functionality without actually executing that functionality. The potential consequences of this on our scanning results are further discussed in the limitations section.

Responsible disclosure Reporting vulnerabilities discovered during an IP scan is a non-trivial problem, as no direct connection to a domain name and thus email address exists. To nevertheless notify as many owners of vulnerable servers as possible, we first checked if the IP address belongs to one of the large cloud providers and contacted them with a list of all their affected assets. For all other IPs, we try to connect to each via HTTPS and inspected the returned certificate (if any) to see if it contains a domain we can contact. In these cases, we directly contacted the owners via `security@domain` about the vulnerable server. Moreover, we are in the process of reporting the results of our studies to the vendors of the affected software, in particular those that are currently insecure by default, to make them consider switching to a securer alternative.

3.3 Scanning Results

Prevalence of endpoints and vulnerabilities (RQ1) First, we investigated the number of open ports and subsequent HTTP(S) responses on those ports as shown in Table 2. Unsurprisingly, the generic ports 80 and 443 were most commonly available, together making up for about two thirds of all open ports and 85% of all responses. On the other hand, Docker's 2375 and Nomad's 4646 were seen the fewest times, probably because they are rather exotic ports compared to the 8XXX range, which is more commonly associated with HTTP(S) traffic. It should be noted that some applications answered both HTTP and HTTPS on the same port, however usually only to tell the user that the application is only served via HTTPS. Moreover, we found 3.0M hosts that appeared to always have *all* ports open. However, when actually trying to connect to some of these hosts, we did not find any of the applications that are usually associated with the respective ports and conclude this was either an accidental or intentional network misconfiguration. Therefore, we excluded them from Table 2, as they distorted the results.

Table 2: Open ports and corresponding amount of HTTP(S) responses we received.

Port	# Open	# HTTP	# HTTPS
80	56.8M	51.3M	—
443	50.1M	—	35.9M
2375	120k	11k	2k
4646	180k	24k	4k
6443	553k	304k	322k
8000	5.5M	1.6M	293k
8080	9.0M	7.6M	667k
8088	2.6M	857k	943k
8153	291k	171k	3k
8192	331k	175k	7k
8500	384k	62k	107k
8888	2.4M	1.8M	192k
Total	164.8M	64.0M	38.5M

In the following, we only focus on those hosts that run at least one of the in-scope AWEs on one of the ports we scanned. If the same application was running on multiple ports of the same host, we counted it only once. As shown in Table 3, we found around 2.5M hosts running an AWE, of which WordPress and Kubernetes were by far the most commonly found applications. WordPress alone was responsible for over half of all results in the second stage of our scanning pipeline. Combined, the five most common AWEs were responsible for over 98% of all findings. Overall, we can see that the frequency of these applications varies widely. However, just because an AWE was infrequently represented in our results, that does not mean that it is seldom used, but rather means that it is seldom *exposed* to the Internet.

Table 3: Prevalence of AWEs on the Internet and how many of them suffered from a vulnerability. The default row indicates whether the endpoint is secure by default (✓), had changed over time (†), or a MAV exists by default (✗)

Type	App	# Hosts	# MAVs	Default
CI	Jenkins	2,440 (0.10%)	80 (3.3%)	†
CI	Gocd	587 (0.02%)	36 (6.1%)	✗
CMS	WordPress	1,462,625 (58.33%)	345 (0.0%)	✗
CMS	Grav	2,617 (0.10%)	4 (0.2%)	✗
CMS	Joomla	50,274 (2.00%)	16 (0.0%)	†
CMS	Drupal	65,414 (2.61%)	258 (0.4%)	✗
CM	Kubernetes	706,235 (28.16%)	495 (0.1%)	✓
CM	Docker	893 (0.04%)	657 (73.6%)	✗
CM	Consul	9,447 (0.38%)	190 (2.0%)	✓
CM	Hadoop	923 (0.04%)	556 (60.2%)	✗
CM	Nomad	1,231 (0.05%)	729 (59.2%)	✗
NB	J-Lab	1,369 (0.05%)	53 (3.9%)	✓
NB	J-Notebook	9,549 (0.38%)	313 (3.3%)	†
NB	Zeppelin	1,033 (0.04%)	82 (7.9%)	✗
NB	Polynote	8 (0.00%)	8 (100.0%)	✗
CP	Ajenti	1,292 (0.05%)	0 (0.0%)	✓
CP	Phpmyadmin	184,968 (7.38%)	396 (0.2%)	✓
CP	Adminer	6,621 (0.26%)	3 (0.0%)	†
	Total	2,507,526	4,221	

As shown in Table 3, we found 4,221 hosts with a MAV. The relative prevalence of the vulnerabilities is very different between applications and also between their larger categories. For example, MAVs in CMSes could be found only very infrequently, as they are only vulnerable in their pre-installation state, which they usually do not reside in for a long time. On the other hand, the CM

products Docker, Hadoop, and Nomad, if exposed to the Internet, were vulnerable in the majority of cases. Compared to their overall prevalence, Kubernetes and Consul were a lot less often vulnerable. However, they still contributed to the total of all MAVs, with the cluster management software being responsible for roughly half of all discovered MAVs. Furthermore, comparing the last column in Table 3 with the relative number of MAVs shows that applications with a default MAV were most frequently found to be vulnerable, except for the short-lived installation pages for CMSes.

Overall, only a fraction of all running software was found to be vulnerable, but, on the other hand, those vulnerable hosts would be trivial to abuse, e.g., as part of a botnet, as they do not have *any* form of protection at all. Moreover, we used an IP meta data service [40] to determine the location of these vulnerable hosts and found that approximately 64% of them are running in the network of a dedicated hosting provider. On one hand, these are valuable for attackers as they are more likely to provide a static IP address and continuous uptime. On the other hand, compromised servers in what appears to be residential and smaller company networks can be useful in circumventing bot detection mechanisms based on IP reputation. A breakdown of the most commonly affected countries and autonomous systems can be found in Table 4.

Table 4: The five most common countries hosting vulnerable applications on the left side and the five most common autonomous systems on the right side.

Country	Hosts
United States	2104
China	1000
Germany	172
Singapore	97
France	96

AS	Provider	Hosts
AS16509	Amazon EC2	913
AS37963	Alibaba	542
AS14618	Amazon AES	329
AS14061	DigitalOcean	244
AS396982	Google Cloud	221

Deployed software versions (RQ2) Next, we use the results from our version fingerprinter to investigate how up-to-date the discovered software is. To make the versions of all the different software comparable, we only report on their release date and not the version numbers themselves. Overall, we found that about 65% of the discovered versions had been updated or newly installed within the last 6 months. Moreover, 25% were released in 2020, while only about 10% of all discovered versions were released before 2020. However, these numbers are biased as the software has widely different prevalence and WordPress, with its automatic updates, is responsible for most of these results. Looking into the different categories in more detail, we found CMSes to be the most up-to-date of all the five categories, with their median release date in May 2021. Software of the continuous integration and cluster management categories were also rather new, both with a median release date pointing to January 2021. On the other hand, notebooks were running much older versions with a median of January 2020 and control panels were most often out-of-date with half of them older than September 2019.

Next, we investigated how this differs between secure[1] installations and those suffering from a MAV. As Figure 1 shows, hosts with a MAV are, in general, older than those without one. Looking more specifically into Jupyter Notebook, which changed their insecure defaults in December 2016, we can see the impact this change had on MAVs in the wild, as most vulnerable instances are running very old versions. However, interestingly most of the insecure notebooks are now actually running releases *slightly after* this change that require a password by default. When investigating this issue, we found many reports of users who suddenly could no longer access the Notebook they had used for years. The accepted answer to this problem in a StackOverflow question with over 80,000 views says "The following is very unsafe, but you can remove the password completely with [...]" [69] and reveals the problems of changing the default settings of widely used software. On the other hand, when we compare these results with Hadoop, which did not change their insecure defaults over the years, we can see that MAVs are rather evenly distributed across the different releases, with new and insecure instances still getting installed and exposed to the Internet. For Jupyter Notebook it's mostly the older versions that are problematic.

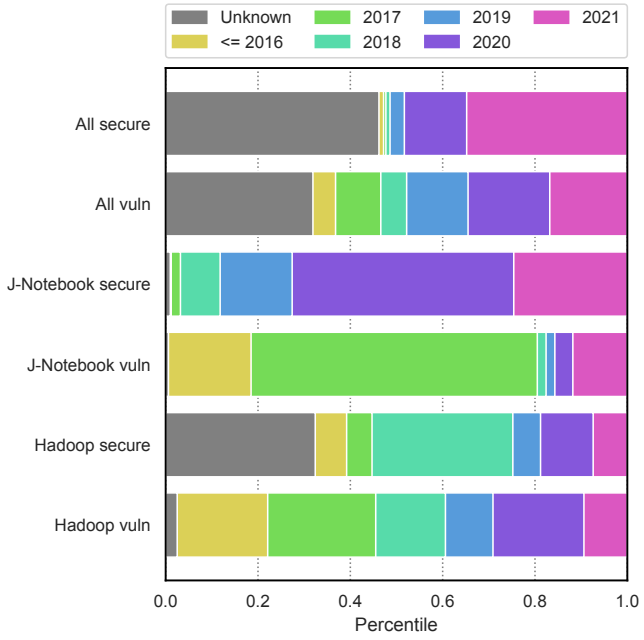

Figure 1: Software release dates separated into 7 bins and compared between secure and vulnerable instances. For brevity, only two relevant software products are shown in detail.

Longevity of vulnerabilities (RQ3) Another goal of our study was to analyze how long the discovered instances with a MAV stayed online and vulnerable. For this, we repeated our scan on the 4,221 vulnerable hosts every three hours over a time span of four weeks. We found that initially, there is a quick decrease and about 10% of them were no longer vulnerable within the first six hours. However, after that the number of machines only decreases by 5-10% per week and, as the first row in Figure 2 shows, over two thirds of them are still online and vulnerable after two weeks and over half of them

[1] In this case, secure only means not suffering from a MAV. Especially if they are out-of-date, other vulnerabilities could exist.

after four weeks. While initially, the insecure-by-default instances decrease quicker on the first day, they afterward decrease at roughly the same rate as their explicitly modified instances. Moreover, there is also a considerable difference between the CI and the notebook category, with the notebooks staying vulnerable for much longer overall.

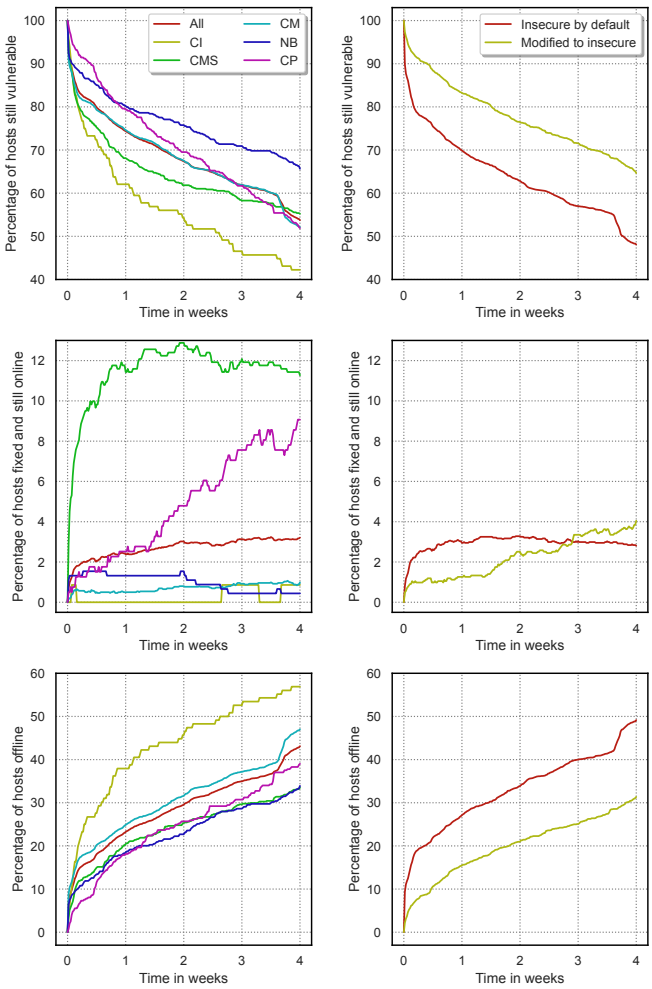

Figure 2: Longevity of the detected MAVs. The left column is grouped by application, the right column grouped by defaults. The legend is shared across graphs in the same columns. The rows are separated into percentage of vulnerable, fixed, and offline hosts.

For the hosts that are no longer vulnerable, we have to differentiate between those that fixed the MAV and continued to stay online and those that just disappeared, e.g., because they were shut down or firewalled. As the second row in Figure 2 shows, the number of fixed hosts is tiny with just 139 hosts (3.2%). Most of those within the first few days are caused by the CMS category, where completing the installation "fixes" the vulnerability as initializing the admin credentials can only be done once. Unfortunately, we can not tell how many of these installations were completed by the original owners. Therefore, the number of fixed vulnerabilities is likely even lower as some of them were actually attackers that completed the installation, thus locking out everyone else. This means, that in almost all cases where a vulnerable instance stopped being vulnerable, it was no longer reachable instead, because the application was either firewalled or taken offline on 1,823 hosts (43.2%) by the end of the four weeks. We can also see that on the first day of exposure more insecure-by-default instances are quickly taken offline again. Afterward, those instances that were explicitly modified to be insecure are a bit more likely to be fixed than taken offline compared to their insecure-by-default counterparts.

Overall, Jenkins and WordPress were on average vulnerable for the shortest time while Joomla and Drupal remained vulnerable for the longest time. However, this might only mean that attackers are, so far, not focusing on those products that remain online for the longest. Moreover, we also continued to apply our fingerprinter to all vulnerable hosts, to see if some of them were updated, but found only a small number of 101 hosts (2.4%) where their version was updated during the four weeks of observation.

4 ATTACKER AWARENESS

In this section, we investigate whether attackers are aware of these vulnerable applications, thereby answering RQ4-6. For this, we set up the 18 applications as high-interaction honeypots to observe and monitor potential attackers in a four weeks-long study. In the following, we first describe our methodology and briefly outline our setup. Finally, we report on the results in detail.

4.1 Methodology

On one hand, studying the attacker awareness requires setting up the 18 applications in a vulnerable state. On the other hand, we need a monitoring that intervenes as soon as we detected actual abuse of the system.

Vulnerable applications We installed each application on a separate server hosted by a large cloud provider. To observe potential attacks, we either left the applications in an insecure-by-default state, or enabled insecure settings for secure-by-default applications. Due to the nature of this study, where we included many different applications from different categories, interactions with them are also very different. For example, in the case of the notebooks a command can be executed directly via the terminal, whereas in WordPress the installation must be completed before the PHP templates can be edited to run arbitrary commands. Therefore, different interactions count as an attack, depending on the targeted application. In general, we are only interested in invasive interactions that are not triggered by a normal web scanner and show signs of malicious or illegal behavior. Moreover, we intentionally did not limit the interaction possibilities with the applications to the execution of a single command, because often a payload script is loaded during first interaction, for example via `wget` or `curl`.

Monitoring To monitor our honeypots, we relied on three open-source products: *Packetbeat* [22], *Auditbeat*[20], and *ElasticSearch* [21]. Packetbeat allowed us to record the network traffic, such as HTTP, by directly reading from the network interface. This way, we obtain much more data than would be contained in simple web server logs, e.g. we also collect POST request bodies and WebSocket traffic. Auditbeat interacts directly with the Linux audit framework

and captures data, such as system calls or file accesses, along with information about the user and function parameters. To prevent an attacker from changing the log afterwards, all honeypots send their logs to a central, append-only log under our control. This central log runs on a standalone server and uses Elasticsearch under the hood, which subsequently allowed us to search and analyze the indexed data.

To make sure that no interactions with our honeypots were possible during setup, we temporarily set up a firewall to block all incomoing requests. At the end of the installation, we took a snapshot of the server to create a copy of the finalized honeypot. Throughout our study, we monitored the continued availability of the applications to detect possible attacks that prevent further exploitation of the application. This is important because some vulnerabilities can only be exploited once (e.g., trust on first use) and thus reverting these applications to their initial state is critical to observing multiple attacks. If we detected a successful attack abusing the resources of the server, we shut down the infected machine and restored the snapshot.

4.2 Experiment Setup

We set up 18 honeypots and ran them for four weeks from June 09, 2021 to July 07, 2021. Each was set up on a dedicated machine using 2 CPU cores and 8 GB of memory. Furthermore, each instance was assigned a static IPv4 address and Packetbeat version 7.13.0 and Auditbeat version 7.13.0 were deployed. Besides the honeypots, we deployed a dedicated log server with 8 CPU cores and 64GB of memory. To analyze the logs we installed Elasticsearch in version 7.13.1. Additionally, this machine was also responsible for monitoring the resource utilization and restoring the snapshots.

Ethical considerations To prevent abuse, we implemented a resource monitor to observe CPU and network bandwidth usage. Specifically, we wanted to ensure that the installed malware was not able to attack additional machines or cause any other harm. To do so, we defined thresholds based on usage patterns we have observed before exposing the vulnerabilities to the Web. Once a threshold was exceeded, we shut down the honeypot and restored the initial state of the server using the previously created snapshot. In addition to shutting down compromised machines, we preemptively blocked outgoing traffic to port 22 to prevent brute-force attacks on SSH. Importantly, both the firewall and our resource monitoring were provided out-of-band in the administrative interface of the cloud provider that hosted our servers. Therefore, an attacker could not drop firewall rules or disable our resource monitoring, even if they are root on the machine.

4.3 Results on Attacker Awareness

Compromised software (RQ4) First, we investigated which applications were attacked and how many attacks we observed by detecting the execution of a system command through the exposed sensitive functionality. If multiple commands were executed from the same source IP within 15 minutes, we counted all of the commands as a single attack. Note that we only count the successful execution of system commands. Therefore, other requests, such as by scanners not explicitly targeting our applications, are not counted as an attack. In addition to the total number of attacks, we also tried to determine the number of unique attacks based on grouping attacks by payloads and source IP addresses. To ensure the accuracy of our analysis we performed this step in a semi-automatic fashion.

Our results in Table 5 show that applications from four of our five categories were attacked, leaving only for the control panels completely unexploited. Looking at the individual categories and taking into account the number of representatives of the category, it can be seen that cluster management applications were attacked the most, followed by notebooks, CMSes and CI systems in that order. This shows that CMSes, which are often used as potential targets in honeypots with known vulnerabities [e.g., 9, 52], are far from being the most attractive targets for attackers when studying MAVs. Looking into the individual applications, our results show that 7 of the 18 applications were attacked at least once during the four weeks of our study. Of these, Hadoop was by far the most popular target, followed by Docker, Jupyter Notebook, and Jupyter Lab. In total, we observed 2,195 attacks from 160 different IP addresses.

Table 5: Attacks distributed across the different applications. The total is not merely the sum of the rows, as some actors attacked multiple applications.

Type	App	# Attacks	# Uniq. Attacks	# Uniq. IPs
CI	Jenkins	4	3	3
CMS	WordPress	9	4	5
CMS	GravCMS	1	1	1
CM	Docker	132	12	22
CM	Hadoop	1,921	49	81
NB	J-Lab	29	13	13
NB	J-Notebook	99	50	50
	Total	2,195	122	160

In addition to identifying the actively compromised applications, we also wanted to discover the purpose behind these attacks. To this end, we manually analyzed a random sample of the compromised machines and found them mostly to be abused for cryptojacking. In the following, we briefly describe three cases that stood out in particular by exposing interesting characteristics. The first case is a Monero cryptominer that terminates processes of other malicious applications that take up valuable computing resources that their own miner needs. Furthermore, the script creates a `cronjob`, thus, even if the server is restarted, the attack persists and starts again. We were able to observe this specific attack four times on Hadoop executed by two different IP addresses. Moreover, we also observed a lot of other cryptomining cases that can be attributed to the well-known *Kinsing* malware campaign [65] which has been active since the beginning of 2020. Interestingly, through our study we could observe that the campaign, which initially focused on insecure Docker instances, is now also spreading to Hadoop. Finally, we also noticed a vigilante attacker during our analysis, who visited our Jupyter Lab several times and attempted to stop our server by executing the `shutdown` command in the terminal. While this can be annoying if someone is currently using the system productively, it also reveals that the system is insecure without any malicious action and prevents others from abusing the server.

Time until compromise (RQ5) Another important goal of our study was to measure how long applications with a MAV survive in the wild until they get compromised. As Table 6 shows, with Hadoop

the first application was successfully exploited *less than one hour* after our study began. WordPress followed within approximately three hours and Docker after about six hours. On the other hand, for the remaining 4 applications that were attacked at least once, it took at least two days until the first attack. For GravCMS it was over two weeks until we observed an attack. As the full timeline of all attacks in Figure 3 shows, the fast attack on WordPress is likely a coincidence, as after the initial fast attack, no more attacks happened for over one week.

Table 6: Time until compromise in hours. First contains the whole time since the start of our study, the other columns contain the time between attacks.

Application	All attacks		Unique attacks		
	First	Average	Shortest	Longest	Average
Jenkins	172.4	159.9	90.1	377.0	213.1
WordPress	2.8	70.7	2.8	451.0	159.2
GravCMS	355.1	355.1	355.1	355.1	355.1
Docker	6.7	5.0	6.5	193.2	59.4
Hadoop	0.8	0.3	0.7	94.3	18.0
J-Lab	133.7	22.6	2.5	173.0	50.4
J-Notebook	48.0	6.7	0.1	58.8	13.4

Therefore, we also have to investigate the time between all attacks over the whole span of our study. As Table 6 also shows, the average time between attacks is as low as approximately 20 minutes for Hadoop, if we count all attacks. The table also reports on the times between *unique attacks*, which excludes attacks from known IPs as well as attacks with known payloads that we observed before. For half of the applications, we observed at least one very fast attack happening within less than 3 hours after another attack. If we look at the longest time without a new attack, it usually took at least 3-4 days, often even over one week until a new attack occurred.

Overall, we can see both in the table and the corresponding timeline graph, that Hadoop is constantly attacked, while Docker and J-Notebook are attacked at least every other day and there are no longer breaks without attacks for these applications. On the other hand, especially for the CMSes and Jenkins, scanning for vulnerable targets seems to happen at a much slower rate on the attackers side. Jupyter Lab has the most conspicuous timeline of the 7 applications, where in the beginning attacks are very rare, but then happen much more frequently towards the end of our study.

Attack landscape (RQ6) Next, we analyzed the attackers behind the observed compromises. Only five attackers were responsible for 1,492 (67%) of all compromises listed in Table 5, while the ten most active attackers were responsible for 1,845 (84%) of all compromises. Our results demonstrate that there is a small group of attackers performing most attacks. On the other hand, the number of unique attacks, 122, implies that there were also several attackers who only interacted with our honeypots once or twice. If we look at the attack behavior for the individual applications, Hadoop stands out with one attacker who performed 719 attacks. This is also the most frequently observed attacker. For Docker, the most active attacker performed a total of 63 attacks. We assume that these attackers regularly scan the IPv4 range for vulnerable applications and thus frequently attack all exposed instances on the internet.

In addition to the pure number of attacks by individual attackers, we analyzed whether the unique attackers used multiple IPs for their attacks and if they attacked different applications. To group the unique attackers, we used the same methodology as for RQ4. Figure 4 shows unique attackers who attack at least two different applications. It is noticeable that only attacker I attacked Docker and J-Notebook, all others attack either Hadoop and Docker or J-Lab and J-Notebook. On the other hand, Jenkins, WordPress, and GravCMS are not shown in the figure, since we did not observe any attackers attacking more than one of these applications. In total, the attackers shown in Figure 4 are responsible for 419 of the 2,195 attacks. Attacker II, in particular, stands out, with a total of 326 attacks on Hadoop and Docker, followed by attacker III with 35 attacks on Docker and Hadoop. In the ranking of the ten most active attackers from above, attacker II is in second place and attacker III is in ninth place. Attacker I represents the largest attacker regarding the number of IP addresses, with a total of 14 different IP addresses used. However, in the ranking of the most active attackers, attacker I is only in 11th place, which shows that the number of IPs used is not in direct proportion to the number of attacks.

Table 7: The ten countries with the most attacks and the respective number of involved ASes from that country.

Country	# Attacks	# AS	Country	# Attacks	# AS
Netherlands	496	10	Moldova	136	2
Brazil	398	4	United Kingdom	71	2
United States	359	36	Poland	69	4
Russia	192	11	India	52	22
Singapore	168	3	Switzerland	51	1

Additionally, we used an IP meta data service [40] to obtain more information about the observed attacks. Thereby, we found that the 2195 attacks from Table 5 originated from 66 different autonomous systems across 28 countries. In Table 7, we list the ten most common countries from which the we observed attacks, which together make up 1992 of the attacks. Moreover, we also took a look at the distribution of the attacks among the different ASes. The result is shown in Table 8, where we list the five ASes with the most attacks and the number of involved countries. However, we do not want to imply that these countries and ASes are actually "responsible" for the attacks, as attackers often use proxies and previously hacked servers to commit further attacks. Instead, this merely represents the attack origins that *we could observe* and should not be used to reason about attacker locations.

Table 8: The top five ISPs from which most attacks originate and the number of involved countries.

AS	Provider	# Attacks	# Countries
AS211252	Serverion BV	469	2
AS268624	Gamers Club	396	2
AS14061	DigitalOcean	351	14
AS200019	Alexhost	135	1
AS16509	Amazon EC2	78	4
		1429	23

5 DEFENDER AWARENESS

Finally, to answer RQ7 on whether defenders are aware of missing authentication vulnerabilities in the investigated software, we analyzed whether commercial security scanners could detect the

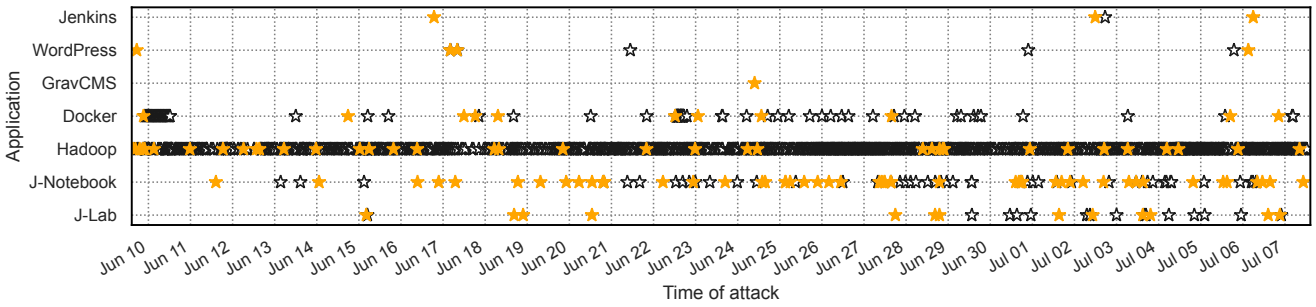

Figure 3: Distribution of attacks during our four weeks of study. Each star represents one attack on the respective application: Yellow stars represent a new attack, black stars represent repeated attacks with known payloads.

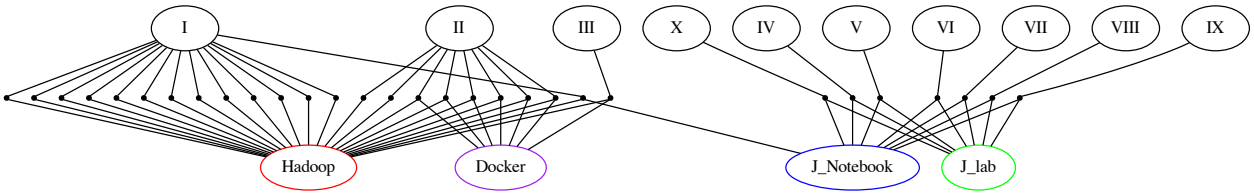

Figure 4: Unique attackers numbered from I to X with their used IP addresses as dots connected to the target applications. For brevity, only attackers who attack at least two different applications are listed.

identified vulnerabilities. To do so, we scanned all our honeypots with two commercial, industry-leading security scanners. As our main goal here is to determine defender awareness and not to blame vendors for missing scanning capabilities, we decided to not disclose their names. Moreover, we do not want to imply that the following findings are in any way correlated to the capabilities of attackers, as they are unlikely to employ enterprise scanning tools intended for defenders.

When configuring the scanners, we aimed at enabling as many scanning capabilities as possible, such as special web scanning features or scanning of extended port ranges. Subsequently, we performed a scan with the most detailed reporting level and manually analyzed the results:

Scanner 1 The first scanner identified 5 out of 18 vulnerabilities: Consul, Docker, Jupyter Notebook, WordPress, and Hadoop. Interestingly, we have observed active attacks on all but one of these applications. However, the scanner did not identify issues in actively exploited applications, such as Jenkins, GravCMS, and Jupyter Lab, as well as most of the non-exploited vulnerabilities.

Scanner 2 The second scanner detected and flagged 3 out of 18 vulnerabilities: Consul, Docker, and Jenkins. Additionally, the scanner flagged installations of Joomla, PhpMyAdmin, Kubernetes, and Hadoop as an informational finding. However, the scanner did not raise a vulnerability for them. Our results suggest that the corresponding vendor is not aware of the missing authentication issues in the listed software. Moreover, entire scan took several hours to complete. During the time of the scan, multiple instances got compromised by malicious actors. Hence, a scan with this scanner would be too slow to detect and remediate internet-exposed vulnerabilities.

Given the low detection rates of 5 and 3 vulnerabilities out of 18, we conclude that defenders who rely on these security scanners to prevent compromise will not be made aware of missing authentication vulnerabilities in many cases.

6 DISCUSSION

In the following, we condense the results of the studies into combined insights and discuss limitations of our approach.

6.1 Insights and Lessons Learned

Table 9 summarizes our most important findings of the previous four chapters and reveals insight only possible due to our comprehensive investigation of MAVs combining prevalence, attack, and defender awareness.

Defaults are important The relative number of vulnerable AWEs is much higher for those insecure by default. Expect for the cases with short-lived insecure installation processes, *all* products where about 5% or more of the exposed AWEs were vulnerable, they were so because of insecure defaults. This demonstrates that defaults are rarely changed by their users even for critical security settings and suggest that requiring authentication by default is a good solution and thus should become commonplace for all administrative software.

Table 9: Summary of our results. The default row indicates whether the endpoint is secure by default (✓), had changed over time (†), or a MAV exists by default (✗). S1 and S2 are the two scanners from Section 5.

Type	App	Default	Vulnerable	Attack	Defend
CI	Jenkins	†	80 (3.3%)	4	S2
CI	Gocd	✗	36 (6.1%)	0	✗
CMS	WordPress	✗	345 (0.0%)	9	S1
CMS	Grav	✗	4 (0.2%)	1	✗
CMS	Joomla	†	16 (0.0%)	0	✗
CMS	Drupal	✗	258 (0.4%)	0	✗
CM	Kubernetes	✓	495 (0.1%)	0	✗
CM	Docker	✗	657 (73.6%)	132	S1&2
CM	Consul	✓	190 (2.0%)	0	S1&2
CM	Hadoop	✗	556 (60.2%)	1,921	S1
CM	Nomad	✗	729 (59.2%)	0	✗
NB	J-Lab	✓	53 (3.9%)	29	✗
NB	J-Notebook	†	313 (3.3%)	99	S1
NB	Zeppelin	✗	82 (7.9%)	0	✗
NB	Polynote	✗	8 (100%)	0	✗
CP	Ajenti	✓	0 (0.0%)	0	✗
CP	Phpmyadmin	✓	396 (0.2%)	0	✗
CP	Adminer	†	3 (0.0%)	0	✗

Changing defaults is effective, but slow When taking a closer look at the software that instead changed their defaults over its lifetime, we found that for Jupyter Notebook this was indeed effective at reducing MAVs in the wild. As our fingerprinting of the affected servers revealed in Figure 1, the less than 5% of all instances running very old versions were responsible for 80% of *all* vulnerable Jupyter Notebook servers in the wild. On the other hand, we *still* observed hundreds of vulnerable-by-default notebooks, despite them having fixed this over 5 years ago. This means that even if other vendors would react now and change their defaults, it will take many years to completely get rid of the problem.

Defenders are behind We observed attacks for 7 of the MAVs, but the security scanners only detected 3 to 5 of these applications as insecure, thus missing vulnerable applications that are already under attack. Even more importantly, with the exception of Consul, none of them support vulnerable applications that have not yet been exploited in the wild. This shows how defenders are behind, as they only react *after* exploitation is already happening in the wild and miss cases such as Jupyter Lab. To be ahead in this cat-and-mouse game, defenders must stop reacting and roll out more proactive defenses.

There is no consensus on MAVs The overlap of the detected vulnerabilities by the two scanners is also low, with only Docker and Consul detected by both and 4 other applications only detected by one scanner. This shows a lack of consensus on the question whether missing authentication in these applications should be considered a vulnerability. Moreover, some websites of the vulnerable applications still sell the missing authentication as a *feature* that allows a quick and convenient setup. However, our scanning study has shown that many users are unaware of the risks as we found 4,221 vulnerable instances on the Internet.

Data on warnings is inconclusive Some of the applications opted to show explicit warnings, either directly when downloading the software or at least in their documentation. Regarding the effectiveness of these warnings, when looking at Polynote it appears like the warnings on their website had no effect because 100% of the Polynote installations we found were vulnerable. However, as we only discovered 8 of these installations in total, it could also be the case the warnings were effective and most of the installations were *not exposed to the Internet* in the first place, thus missing from our statistics. Without knowing the overall installation numbers of these applications, the effectiveness of warnings remains unclear and is left for future work.

6.2 Limitations of our Studies

Completeness First of all, our results might be biased because we were limited to open source software, as setting up the honeypots requires access to the product free of charge. Furthermore, the correlation between Github stars and actual usage appears to be low. Nevertheless, we think our selection comprises a representative sample for the respective categories, but the results might not necessarily generalize for other types of software.

False positives The MAV detection plugins in our pipeline make very specific requests to the application, which makes it highly unlikely that a false positive ocurs. Though without actually exploiting the vulnerability by running code ourselves, which would be neither ethical nor legal, we cannot guarantee a zero false positive rate. We are in the process of open-sourcing the MAV detection plugins [32] so that interested readers can verify the code. Moreover, due the static IPs we used for our honeypots, we have observed many repeated attacks, which might have inflated our results. Nevertheless, we think this is the most realistic scenario, as real servers would also have used a static IP address. Additionally, dynamic IP addresses in the vulnerable hosts that we found could have led to counting the same application twice, though we tried to avoid this by conducting the whole scan as fast as possible.

False negatives We tested our scanning pipeline on both the newest and oldest available versions, however there is no guarantee that all versions in between are also correctly detected. Moreover, we performed only a single Internet-wide scan and this only on the common ports like 80 and 443, and the default ports of the applications. Therefore, we missed hosts that were unresponsive, temporarily unavailable, or not running on the default port.

Under counting Furthermore, we conducted our scan only on IP addresses and not domain names, thus, e.g., missing applications running on shared hosting services that are distinguished by the *Host* header. In particular, attackers could increase the likelihood to discover unsecured applications and unfinished installations by using Certificate Transparency (CT) logs to discover newly registered domains [8] and scan those preferably instead of a full sweep of the IPv4 space. Overall, our scanning results should thus be seen as a *lower bound* and the actual number of MAVs in the wild is likely even higher.

7 RELATED WORK

Honeypots Honeypots are used to analyze attacks on a wide variety of targets, like IoT or embedded devices [e.g., 25, 35, 70], Industrial Control Systems [53] or Instant Messaging services [4].

In this work, we are only concerned with high-interaction web application honeypots. Back in 2008, Müter et al. [54] presented a toolkit to convert arbitrary PHP applications into high-interaction honeypots. Closer related to our experiments, Canali and Balzarotti [9] ran 500 honeypot websites with known vulnerabilities, such as SQL injection, to analyze the behavior of the attackers after a successful exploit. Our honeypot study is orthogonal to their work, as we study compromises due to missing authentication, which is not widely considered a vulnerability yet. Recently, Li et al. [52] characterized requests to CMSes to detect malicious bots. While there are some similarities to our approach to identify attackers, we are also interested in manual attacks and investigate a much wider range of application categories and compare them with each other.

Internet-wide Security Scanning The two most popular tools to conduct fast port scans are *masscan* [34] and *ZMap* [17]. Similarly notable are the search engines *Shodan* [67] and *Censys* [19], which conduct regular scans and support researchers without the means to conduct scans themselves. Subsequent works tried to reduce the time needed to conduct a scan [e.g., 1, 41, 48, 66], used large network telescopes to observe the scanning behaviors of others [18, 64], and investigated biases introduced by the origin of the scan [79]. The second stage of our scanning pipeline, on the other hand, is more related to fingerprinting tools like *WhatWeb* [38] and *BlindElephant* [71]. However, WhatWeb can only extract voluntarily disclosed version information. BlindElephant relies on file hashes of resources embedded on the website, but is severely outdated with the last update almost 10 years ago. Therefore, we created our own fingerprinting pipeline, which combines the advantages of both these approaches into one. Finally, we try to identify whether the deployed application suffers from a missing authentication vulnerability, which makes it related to works on *black-box web security scanners* [e.g., 13, 14, 23, 47, 59]. However, these tools mostly focus on more widely known vulnerabilities like SQL injection and cross-site scripting. Our tool, on the other hand, tries to detect MAVs in web applications, which were, to the best of our knowledge, not yet studied by the academic community and are also often not yet considered by commercial scanning tools, as our experiment in Section 5 has shown.

Vulnerability Research and Software Misconfigurations So called *Google Dorks* use specific search engines queries to automatically locate vulnerable targets, without the need to conduct a scan themselves [73]. Others tried to automatically match known vulnerabilities from CVE feeds with the results indexed by Shodan and Censys [26, 57]. In contrast to these works, we focus on missing authentication vulnerabilities, which are usually not assigned a CVE as they are just seen as a missing feature rather than a real vulnerability. Thus, studies of sensitive information disclosure without authentication, such as a study of the FTP ecosystem by Springall et al. [68], or a study of Amazon S3 buckets by Continella et al. [11], are more related to us. Most recently in 2020, Ferrari et al. [24] investigated the prevalence of misconfigured NoSQL databases. In contrast to these previous works that studied specific protocols with a limited number of possible interactions like FTP and NoSQL, we studied the prevalence and exploitation of misconfigurations in 18, in some cases totally different, Web applications of five different categories. Moreover, we do not focus on leaks of sensitive data, but attackers that achieve remote code execution through the affected administrative web endpoints.

8 SUMMARY & CONCLUSION

In this paper, we comprehensively investigated missing authentication vulnerabilities. For this, we looked at 25 popular applications in 5 different categories and found that 18 of them are prone to such MAVs. Our scanning study revealed 4,221 vulnerable instances in the wild, confirming that this is not a theoretical risk. We found that default settings play an important role, with vulnerabilities in insecure by default applications being significantly more prevalent during our scans. Moreover, we show these instances would be a valuable target for attackers, as more than half of them were still exposed in a vulnerable state after four weeks of observation. Our study on the attacker's awareness revealed that 7 of these applications are already actively exploited. Exploitation can happen quickly, sometimes within one hour of exposure, meaning attackers are using considerable resources to continually scan the whole Internet for these vulnerable applications. Furthermore, we found that relatively few attackers are responsible for most observed attacks and that they even target multiple vulnerable applications at the same time. Finally, we investigated if defenders are aware of this problem and discovered that two state-of-the-art commercial security scanners only found 3 and 5 of these 18 vulnerabilities.

Overall, we found MAVs to be a prevalent, overlooked and underrated problem in application security. It seems that, so far, our community focuses so much on *insecure programming* practices that insecure configuration and sane defaults receive too little attention. Therefore, we think it is important to raise awareness and to consider missing authentication as a vulnerability rather than a feature, so that more software products require authentication by default and more security scanners look for these flaws.

ACKNOWLEDGEMENTS

We would like to thank our shepherd Robert Beverly and all anonymous reviewers for their valuable comments and suggestions. The authors gratefully acknowledge funding from the German Federal Ministry of Education and Research (BMBF) under the project IVAN (16KIS1168) and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

A MAV DETECTION DETAILS

Table 10 provides a short description of the steps taken to identify MAVs in the 18 applications. Many of them have already been open-sourced and can be found in the Tsunami plugins repository [32].

REFERENCES

[1] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman. Zippier zmap: internet-wide scanning at 10 gbps. In *Proc. of USENIX Workshop on Offensive Technologies (WOOT)*, 2014.
[2] Ajenti. Ajenti. Online https://github.com/ajenti/ajenti, 2021.
[3] Ajenti Docs. Running ajenti. Online https://docs.ajenti.org/en/latest/man/run.html, 2021.
[4] S. Antonatos, I. Polakis, T. Petsas, and E. P. Markatos. A systematic characterization of im threats using honeypots. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2010.
[5] Apache. Spark notebook. Online https://github.com/spark-notebook/spark-notebook, 2019.

Table 10: The MAV detection steps from our Tsunami plugins in pseudo-code. Unless otherwise noted, we only detect a MAV if *all* steps are successful.

Application	MAV detection steps
Jenkins	1. Visit '/view/all/newJob' 2. Check that body contains 'Jenkins' and is valid HTML 3. Parse HTML response and verify that element 'form#createItem' exists
GoCD	1. Visit '/go/home' 2. Check that body contains 'Create a pipeline - Go' and 'pipelines-page', or 'Add Pipeline' and 'admin_pipelines', or 'Dashboard - Go' and '/go/admin/pipelines/', or 'Pipelines - Go' and '/go/admin/pipelines'
WordPress	1. Visit '/wp-admin/install.php?step=1' 2. Check that body contains 'WordPress' and is valid HTML 3. Parse HTML response and verify that elements 'form#setup' and 'form#setup input#pass1' exist
Grav	1. Visit '/' and check that body contains 'The Admin plugin has been installed' and 'Create User' 2. If step 1 is not successful, visit '/admin' and check that body contains 'No user accounts found' and 'create one'
Joomla	1. Visit '/installation/index.php' 2. Check that the body contains 'Joomla! Web Installer' or 'Enter the name of your Joomla! site'
Drupal	1. Visit '/core/install.php?langcode=en&profile=standard&continue=1' 2. Remove all whitespace from response, as their placement differs across versions 3. Check that body contains '<liclass="is-active">Setupdatabase'
Kubernetes	1. Visit '/' and check that body contains 'certificates.k8s.io' and 'healthz/ping' 2. Visit '/api/v1/pods', remove all whitespace from the response and check that it contains '"phase":"Running"' 3. Parse the response as JSON and check that the 'items' array exists and is not empty
Kubernetes	1. Visit '/' and check that body contains 'certificates.k8s.io' and 'healthz/ping' 2. Visit '/api/v1/pods', remove all whitespace from the response and check that it contains '"phase":"Running"' 3. Parse the response as JSON and check that the 'items' array exists and is not empty
Docker	1. Visit '/' and check that body contains '{"message":"page not found"}' 2. Visit '/version', convert response to lower case and check that it contains 'minapiversion' and 'kernelversion'
Consul	1. Visit '/v1/agent/self' and check that response is valid JSON 2. Parse JSON response and check that the 'debugConfig' property does exist 3. Check that at least one of 'debugConfig.enableScriptChecks' and 'debugConfig.enableRemoteChecks' is enabled
Hadoop	1. Visit '/cluster/cluster' and convert response to lower case 2. Check that response contains 'hadoop', 'resourcemanager' and 'logged in as: dr.who' 3. Visit '/ws/v1/cluster/apps/new-application' and check that it is valid JSON 4. Parse the JSON response and check that it contains the 'application-id' object
Nomad	1. Visit '/v1/jobs' 2. Check that response contains '<title>Nomad</title>'
J-Lab	1. Visit '/api/terminals' 2. Check that response contains 'JupyterLab'
J-Notebook	1. Visit '/api/terminals' 2. Check that response contains 'Jupyter Notebook'
Zeppelin	1. Visit '/api/notebook' 2. Check that response contains '{"status":"OK",'
Polynote	1. Visit '/' 2. Check that response contains '<title>Polynote</title>'
Ajenti	1. Visit '/view/' 2. Check that response contains 'customization.plugins.core.title || 'Ajenti'' and 'ajentiPlatformUnmapped'
phpMyAdmin	1. Visit '/' and check that it contains 'Server connection collation' and 'phpMyAdmin documentation' 2. If step 1 is not successful, visit '/phpmyadmin' and check that it contains the same two strings
Ajenti	1. Visit '/adminer.php?username=root' and check that it contains 'through PHP extension' and 'Logged as' 2. If step 1 is not successful, visit 'adminer/adminer.php?username=root' and check that it contains the same two strings

[6] Apache. Hadoop. Online https://github.com/apache/hadoop, 2021.
[7] Apache. Zeppelin. Online https://github.com/apache/zeppelin, 2021.
[8] H. Böck. Hacking web applications before they are installed. Online https://www.golem.de/news/certificate-transparency-hacking-web-applications-before-they-are-installed-1707-129172.html, 2017.
[9] D. Canali and D. Balzarotti. Behind the scenes of online attacks: an analysis of exploitation behaviors on the web. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2013.
[10] Consul Docs. Configuration. Online https://www.consul.io/docs/agent/options, 2021.
[11] A. Continella, M. Polino, M. Pogliani, and S. Zanero. There's a hole in that bucket! a large-scale analysis of misconfigured s3 buckets. In *Proc. of Annual Computer Security Applications Conference (ACSAC)*, 2018.
[12] Docker. Docker. Online https://github.com/docker/compose, 2021.
[13] A. Doupé, M. Cova, and G. Vigna. Why johnny can't pentest: An analysis of black-box web vulnerability scanners. In *Proc. of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2010.
[14] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna. Enemy of the state: A state-aware black-box web vulnerability scanner. In *Proc. of USENIX Security Symposium*, 2012.
[15] Drone. Drone. Online https://github.com/drone/drone, 2021.
[16] Drupal. Drupal. Online https://github.com/drupal/drupal, 2021.
[17] Z. Durumeric, E. Wustrow, and J. A. Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Proc. of USENIX Security Symposium*, pages 605–620, 2013.

[18] Z. Durumeric, M. Bailey, and J. A. Halderman. An internet-wide view of internet-wide scanning. In *Proc. of USENIX Security Symposium*, 2014.

[19] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A search engine backed by internet-wide scanning. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2015.

[20] Elasticsearch. Lightweight shipper for audit data. Online https://www.elastic.co/beats/auditbeat, 2021.

[21] Elasticsearch. The heart of the free and open elastic stack. Online https://www.elastic.co/elasticsearch/, 2021.

[22] Elasticsearch. Lightweight shipper for network data. Online https://www.elastic.co/beats/packetbeat, 2021.

[23] B. Eriksson, G. Pellegrino, and A. Sabelfeld. Black widow: Blackbox data-driven web scanning. *Proc. of IEEE Symposium on Security and Privacy*, 2021.

[24] D. Ferrari, M. Carminati, M. Polino, and S. Zanero. Nosql breakdown: A large-scale analysis of misconfigured nosql services. In *Proc. of Annual Computer Security Applications Conference (ACSAC)*, 2020.

[25] D. Fraunholz, D. Krohmer, H. D. Schotten, and C. Nogueira. Introducing falcom: A multifunctional high-interaction honeypot framework for industrial and embedded applications. In *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2018.

[26] B. Genge and C. Enăchescu. Shovat: Shodan-based vulnerability assessment tool for internet-facing services. *Security and communication networks*, 2016.

[27] Getgrav. Grav cms. Online https://github.com/getgrav/grav, 2021.

[28] Ghost. Ghost. Online https://github.com/TryGhost/Ghost, 2021.

[29] Gitlab. Gitlab. Online https://github.com/gitlabhq/gitlabhq, 2021.

[30] GoCD Docs. Authentication. Online https://docs.gocd.org/current/configuration/dev_authentication.html, 2021.

[31] Google. Tsunami web service fingerprinter. Onlince https://github.com/google/tsunami-security-scanner-plugins/tree/master/google/fingerprinters/web, 2022.

[32] Google. Tsunami security scanner plugins. Onlince https://github.com/google/tsunami-security-scanner-plugins, 2022.

[33] Google. Tsunami security scanner. Onlince https://github.com/google/tsunami-security-scanner, 2022.

[34] R. D. Graham. Masscan: Mass ip port scanner. Online https://github.com/robertdavidgraham/masscan, 2013.

[35] J. D. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici. Siphon: Towards scalable high-interaction physical honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, 2017.

[36] Hashicorp. Consul. Online https://github.com/hashicorp/consul, 2021.

[37] Hashicorp. Nomad. Online https://github.com/hashicorp/nomad, 2021.

[38] A. Horton and B. Coles. Whatweb. Online https://github.com/urbanadventurer/WhatWeb, 2021.

[39] IANA. Ipv4 address space registry. Online https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml, 2021.

[40] IPHub. Proxy & vpn detection api. Onlince https://iphub.info/, 2022.

[41] L. Izhikevich, R. Teixeira, and Z. Durumeric. LZR: Identifying unexpected internet services. In *Proc. of USENIX Security Symposium*, 2021.

[42] Jenkins. Jenkins. Online https://github.com/jenkinsci/jenkins, 2021.

[43] Joomla. Joomla cms. Online https://github.com/joomla/joomla-cms, 2021.

[44] Joomla! Documentation. Secured procedure for installing joomla with a remote database. Online https://docs.joomla.org/J3.x:Secured_procedure_for_installing_Joomla_with_a_remote_database, 2017.

[45] Jupyter. Notebook. Online https://github.com/jupyter/jupyter, 2020.

[46] Jupyter. Lab. Online https://github.com/jupyterlab/jupyterlab, 2021.

[47] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic. Secubat: a web vulnerability scanner. In *Proc. of the International World Wide Web Conference (WWW)*, 2006.

[48] J. Klick, S. Lau, M. Wählisch, and V. Roth. Towards better internet citizenship: Reducing the footprint of internet-wide scans by topology aware prefix selection. In *Proc. of Internet Measurement Conference (IMC)*, 2016.

[49] Kubernetes. Kubernetes. Online https://github.com/kubernetes/kubernetes, 2021.

[50] Kubernetes. Controlling access to the kubernetes api. Online https://kubernetes.io/docs/concepts/security/controlling-access/, 2021.

[51] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2019.

[52] X. Li, B. A. Azad, A. Rahmati, and N. Nikiforakis. Good bot, bad bot: Characterizing automated browsing activity. In *Proc. of IEEE Symposium on Security and Privacy*, 2021.

[53] E. López-Morales, C. Rubio-Medrano, A. Doupé, Y. Shoshitaishvili, R. Wang, T. Bao, and G.-J. Ahn. Honeyplc: A next-generation honeypot for industrial control systems. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2020.

[54] M. Müter, F. Freiling, T. Holz, and J. Matthews. A generic toolkit for converting web applications into high-interaction honeypots. *University of Mannheim*, 2008.

[55] Nomad Docs. Overview. Online https://www.nomadproject.io/docs/internals/security, 2021.

[56] J. Nord. Jenkins should be secure out of the box by default. Online https://issues.jenkins.io/browse/JENKINS-30749, 2015.

[57] J. O'Hare, R. Macfarlane, and O. Lo. Identifying vulnerabilities using internet-wide scanning data. In *2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3)*, 2019.

[58] OmniDB. Omnidb. Online https://github.com/OmniDB/OmniDB, 2020.

[59] G. Pellegrino, C. Tschürtz, E. Bodden, and C. Rossow. jäk: Using dynamic analysis to crawl and test modern web applications. In *Proc. of International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2015.

[60] phpMyAdmin. phpmyadmin. Online https://github.com/phpmyadmin/phpmyadmin, 2021.

[61] Polynote. Polynote. Online https://github.com/polynote/polynote, 2021.

[62] Polynote Docs. Installing polynote. Online https://polynote.org/docs/01-installation.html, 2021.

[63] Project Jupyter. Security release: Jupyter notebook 4.3.1. Online https://blog.jupyter.org/security-release-jupyter-notebook-4-3-1-808e1f3bb5e2, 2016.

[64] P. Richter and A. Berger. Scanning the scanners: Sensing the internet from a massively distributed network telescope. In *Proc. of Internet Measurement Conference (IMC)*, 2019.

[65] S. Schick. Kinsing malware hits container api ports with thousands of attacks per day. Online https://securityintelligence.com/news/kinsing-malware-hits-container-api-ports-with-thousands-of-attacks-per-day/, 2020.

[66] Z. Shamsi, D. B. Cline, and D. Loguinov. Faulds: A non-parametric iterative classifier for internet-wide os fingerprinting. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2021.

[67] Shodan. Search engine. Online https://www.shodan.io/, 2021.

[68] D. Springall, Z. Durumeric, and J. A. Halderman. Ftp: The forgotten cloud. In *Proc. of Conference on Dependable Systems and Networks (DSN)*, 2016.

[69] StackOverflow. How to disable password request for a jupyter notebook session? Online https://stackoverflow.com/a/47509274, 2016.

[70] A. Tambe, Y. L. Aung, R. Sridharan, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici. Detection of threats to iot devices using scalable vpn-forwarded honeypots. In *Proc. of ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2019.

[71] P. Thomas. Blindelephant. Online https://sourceforge.net/projects/blindelephant/, 2012.

[72] ThoughtWorks. Gocd. Online https://github.com/GoCD/GoCD, 2021.

[73] F. Toffalini, M. Abbà, D. Carra, and D. Balzarotti. Google dorks: Analysis, creation, and new defenses. In *Proc. of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.

[74] Travis. Travis. Online https://github.com/travis-ci/travis-ci, 2020.

[75] Vesta CP. Vesta cp. Online https://github.com/serghey-rodin/vesta, 2020.

[76] J. Vrána. Accessing a database without a password. Online https://www.adminer.org/en/password/, 2018.

[77] J. Vrána. Adminer. Online https://github.com/vrana/adminer, 2021.

[78] W3Techs. Usage statistics of content management systems. Online https://w3techs.com/technologies/overview/content_management, 2021.

[79] G. Wan, L. Izhikevich, D. Adrian, K. Yoshioka, R. Holz, C. Rossow, and Z. Durumeric. On the origin of scanning: The impact of location on internet-wide scans. In *Proc. of Internet Measurement Conference (IMC)*, 2020.

[80] WordPress. Wordpress. Online https://github.com/WordPress/WordPress, 2021.