MSc System and Network Engineering
Research Project 2

Security By Default

A Comparative Security Evaluation of Default Configurations

Bernardus A. Jansen, BSc
Bernardus.Jansen@os3.nl

July 12, 2018

Supervisors:
Dr.-Ing. T. Fiebig (TU Delft)
R. Koning, MSc

Assessor:
Prof. dr. ir. C.T.A.M. de Laat

Technische Universiteit Delft

Universiteit van Amsterdam

Abstract

Misconfiguration of network services is a serious problem on the Internet. Defaults of software packages are often meant to get the user up and running as soon as possible, or to demonstrate a working setup *out-of-the-box*. Security issues that may be deemed misconfigurations, could stem from insecure default settings. Users unfamiliar with the software they wish to run may find the default supplied configuration and available settings intimidating, and only make minor changes to modify the configuration to their desired setup. As such, the majority of the default settings of configurations supplied with software packages are expected to remain in large numbers of active systems. In this research we propose an automated system to automatically test a number of commonly used internet services on a range of platforms. We then use this framework to automatically test the default configurations of a diverse set of network services when installed on a range of platforms. We find that the security posture of the default configurations differs both between different platforms, as well as between different applications providing the same functionality.

Contents

1 Introduction

Year after year, the number of people with access to the Internet increases. With the availability of low-cost single board computers, virtual private servers and home network attached storage solutions, running Internet facing services is accessible to many of those Internet users. *DIY* minded individuals may for example choose to run their own mail server or setup their own Internet of Things network. Even for seasoned administrators, keeping up to date with updated applications or entirely new services may prove to be a challenge [22].

When running Internet services, usually a set of settings is required for the service to be able to function. A mail server for example requires knowledge of the domains it is serving mail for and the ports and interfaces it should listen on for connections. Such a set of settings is called a *configuration*.

In order to not burden the user with having to specify every single setting required for operation, most software bundles a default configuration with all or a number of settings preconfigured in a default configuration. When installing software a user is unfamiliar with, they may find it daunting to dive into the default configuration of the application, evaluate all settings and change them to their preference. As such, default settings are likely to remain in a running system for an extended period of time, or perhaps indefinitely.

To get a better understanding of the influence of software distributors on the security posture of default configurations, we propose a framework to quickly asses and compare default configurations across distributions and apply it to a sample selection of distributions and software packages.

Research Questions: Specifically, we investigate the following research questions:

- *What is a suitable metric to describe the security posture of default configurations for Internet services?*
- *What role do distributors play in the quality of default configurations for Internet services?*

The second research question is additionally subdivided as follows:

- *Are there significant differences in security performance between distributors?*
- *Which factors influence the security performance of software distributors?*

Structure: The remainder of this document is structured as follows: In Section 2, we introduce the necessary background on software configurations and automated testing. We describe our testing methodology along with the tested applications in Section 3. We then present our results in Section 4, and discuss them in Section 5, where we also describe the limitations of our work. We compare our approach to related work in Section 6, and finally conclude in Section 7, where we also discuss opportunities for further work.

2 Background

Incorrectly configured Internet services can lead to significant security risks. The Open Web Application Security Project has listed misconfiguration of software in its Top 10 of most critical web application security risks for four out of five times it has been published [27, 26, 24, 25].

2.1 Misconfiguration

Misconfigured software can pose a risk to the operator and users of the service for example by opening the server up to users or hosts that should not be able to connect to the service. In some cases, misconfigured software can even affect other Internet users unrelated to the misconfigured system. This can occur for example when an unintentionally open DNS resolver is used to amplify traffic in a denial of service attack or when a mail server configured as an open relay is used to spread spam messages.

In his 2017 dissertation [15], Fiebig defines a misconfiguration as follows: *"A security misconfiguration has occurred, if the way an Internet service is deployed, i.e., configured, enables an attacker to taint either its confidentiality, integrity or availability, and the property could not have been tainted if the service would have been deployed and configured correctly, without restricting availability for legitimate clients."* Following this definition, unchanged insecure default settings in software configurations can also lead to a service that can be considered to be misconfigured.

Default settings for a piece of software can usually be set in two ways. Defaults may be built into the application itself, or set in a bundled configuration file. With built-in defaults, the application will function in accordance to defaults set in its application code unless overridden by an external setting. In this paper we will refer to built-in defaults as *no-setting* defaults.

The second method of applying default settings for an application is through a default configuration file, that is installed simultaneously with the software. This configuration file, if applicable combined with built-in defaults, then dictates the default setup of the application.

Like multiple methods of applying default settings exist, software can be installed in a number of ways. For Unix-like systems, the most common method of installing software is by downloading and installing a package file from a repository. A repository is a catalogue of software managed by the distributors of the respective distribution or operating system. The software in the repository is managed by maintainers. These maintainers can be the developers of the actual software, but can also be a third party or individual unrelated to the maintained software. The maintainer of a piece of software in a repository is responsible for packaging the original application for the respective operating system including for example startup and install scripts, as well as a default configuration for the software. A schematic overview of these relations is available in Figure 1.

In some situations, no package for a piece of software may be available in the official repositories for an operating system. This may for example be due to software licensing conflicts or due to the relative obscurity of a package. In such a situation the developer of the software may provide their own repository, containing only their own software, or provide the packaged application through other means, for example making it downloadable from their website. Another method of installing software is building and installing the application from its source code. In this situation the source code and other files related to the application, such as configuration files, are supplied by the distributor of the source package, which is usually the developer of the software itself.

2.2 Automated Testing

Automated testing of code has received significant attention in recent years as part of the DevOps and test-driven development paradigms. As such, continuous integration tools such as Jenkins [32] and services as Travis CI [33] have seen significant uptake.

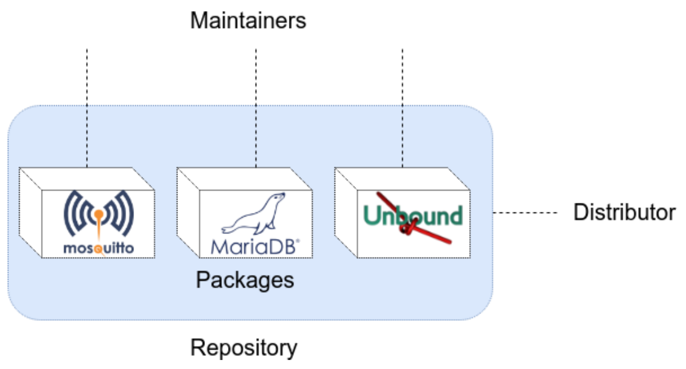

Figure 1: Schematic overview of packages in a repository. Maintainers are responsible for their packages in the repository, distributors are responsible for the overall repository itself.

These tools allow to automatically build and test services according to predefined tests to determine whether recent modifications to the code introduced faults in the operation of the application.

Additionally, tools exist to scan servers or networks whether they run versions of software that may be open to exploits such as unpatched vulnerabilities [23, 28, 3]. With container environments being very popular and containers largely being based on default images, tools specifically designed to scan for vulnerabilities in containers have been introduced, such as Docker security scanning [17], CoreOS Clair [30] and Twistlock [34].

The frameworks mentioned above use different approaches to detect issues. The tests used in continuous integration systems are usually designed based on specific parts of the application code that is to be tested, this is called a *white-box* approach. Vulnerability scanners in their simplest form however match the version of software that is detected to run on the tested system against a database of software versions and vulnerabilities existing in these versions to decide whether a system is vulnerable, without specific knowledge about the tested system or software. This is called a *black-box* approach.

Further, there is a distinction between *dynamic* and *static* analysis. In *dynamic* analysis, a running system is analysed by interacting with the respective application. In static analysis, the application binaries or code are analysed. General purpose vulnerability scanners generally perform dynamic analysis, whereas the described container scanning frameworks perform static analysis, scanning container images for vulnerabilities without actually running the container.

3 Methodology

In this research, we will perform dynamic analysis on our tested set of applications in a *grey-box* approach. Our tests are designed based on the possible configuration of the tested service and judge the security posture of the tested applications by interacting with the service. This reflects the way in which an attacker would access a vulnerable Internet-facing system, and allows us to determine the default configured state of an application, regardless of whether these defaults stem from a configuration file or built-in defaults.

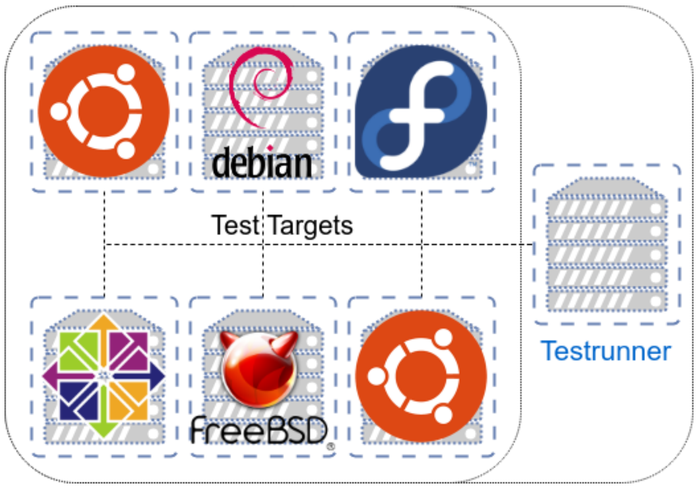

Figure 2: Schematic Overview of Test Setup

3.1 Configuration Testing Framework

In order to conduct our tests in a way that is easily reproducible, we set up an automated framework to conduct tests and gather results for this project. The framework spins up virtual machines on which the tests will be conducted. The tests themselves will also be conducted from a virtual machine in order to increase the portability of our framework. A schematic overview of this setup is available in Figure 2.

We use Vagrant [8] to launch and manage the virtual machines. Vagrant allows to easily launch virtual machines, or *boxes*, without user interaction and supports multiple virtualisation backends. In this work, Virtualbox was used as the virtualisation backend as this is supported by the largest number of publicly available Vagrant boxes. When supported by the boxes, other backends can be used as well.

As we use Vagrant to manage the virtual machines, we use default Vagrant boxes as the basis for the platforms tested in this project. The framework will automatically download and import the Vagrant boxes that are configured to be part of the test.

The Vagrant boxes are provisioned with the tested software using Ansible playbooks [29]. As Ansible is designed to ensure idempotency each time the same playbook is executed on the same system, this ensures our tested systems are in an appropriate and reproducible state upon every execution of the framework. As with the platforms that are to be tested, Ansible also provisions the virtual machine executing the tests with the software that is needed to run the tests.

All tests will be executed from the Testrunner virtual machine, remote to the tested targets. On-host tests are not included in this research.

The basic form of the tests executed by the Testrunner is a wrapper script around a client application for the tested service. When a service is found to be remotely accessible, this client attempts to interact with it. The outcome thereof is interpreted by the script and a result is returned. The actual tests are specific to the tested services and are further described from Section 3.2.1 onward.

3.2 Software and Tests

We have selected a diverse set of Internet services to be tested in this project. As these services provide very different functionality, insecure defaults for these services can have very different security implications. We will be testing a database server, two DNS resolvers, two mail transfer agents, NTPD and the Mosquitto MQTT server.

Though many more applications can be imagined that would be very interesting to test, the scope for this research project has been limited to the described services. As our framework uses default Vagrant images and Ansible playbooks, the platform can be easily extended to test more applications and or platforms. The selected services will be installed on various versions of Ubuntu, Debian, Fedora, CentOS and FreeBSD, as seen in Table 1. All versions tested are currently supported by their respective distributor as of July 2018.

The focus of this research is to uncover any differences in security performance that may exist between different distributions or software packages. As such, while the versions of the tested software differ between different platforms and different versions thereof, we do not expect the version of the tested software itself to be the source of any perceived differences in security performance. Should trends in our results correlate with software versions however, we will further analyse these situations to determine its cause.

Platform	**Version**	**MySQL**	**Unbound**	**Bind**	**NTP**	**MQTT**	**Postfix**	**Sendmail**
Ubuntu	18.04	10.1.29-6	1.6.7	9.11.3	4.2.8p10	1.4.15-2	3.3.0	8.15.2-10
	16.04	10.0.34	1.5.8	9.10.3	4.2.8p4	1.4.8	3.1.0	8.15.2-3
	14.04	5.5.59	1.4.22	9.9.5	4.2.6p5	0.15	2.11.0	8.14.4-4
Debian	Stretch	10.1.26	1.6.0	9.10.3	4.2.8p10	1.4.10-3	3.1.8	8.15.2-8
	Jessie	10.0.35	1.4.22	9.9.5	4.2.6p5	1.3.4-2	2.11.3	8.14.4-8
CentOS	7	5.5.56	1.6.6	9.9.4	4.2.6p5	1.4.15	2.10.1	8.14.7
	6	5.1.73	1.4.20	9.8.2	4.2.6p5	N/A	2.6.6	8.14.4
Fedora	28	10.2.15	1.7.2	9.11.3	4.2.8p11	1.5	3.2.5	8.15.2
	27	10.2.15	1.7.2	9.11.3	4.2.8p11	1.5	3.2.5	8.15.2
FreeBSD	11.1	5.5.60	1.7.3	9.12.1	4.2.8p11	1.4.14_2	3.3.1,1	8.15.2
	10.4	5.5.60	1.7.3	9.12.1	4.2.8p11	1.4.14_2	3.3.1,1	8.15.2

Table 1: Tested Platforms and Software

3.2.1 MySQL

Though other databases and NoSQL solutions having seen significant uptake in recent years, MySQL is still the most popular database server [7], commonly supporting web applications as part of the traditional Linux Apache MySQL PHP 'LAMP' stack. In most current open-source operating systems the original MySQL server has been replaced with its community-driven fork MariaDB. We will therefore be testing MariaDB on all platforms with the exception of CentOS 6, where we will install MySQL as no package for MariaDB is available in the repositories for this platform.

While databases and key-value stores are commonly configured to accept remote connections, remote connections should only be allowed within a trusted network, with most databases and key-value stores not supporting transport-layer encryption, or even user authentication [16]. Remote access to MySQL server resources can be configured in two ways. Firstly, the MySQL server can be configured to only accept connections through a domain socket, or to only accept TCP/IP connections from localhost. The other or additional method is to restrict the hosts a database user is allowed to connect from. While both methods can be used to prevent unauthorised remote access to databases, we consider the former to be more secure as a default configuration than the latter method, as adding a user with a weak password and insufficient host restrictions will not be enough to open the system up to a remote attacker in that case.

In our MySQL test we will attempt to connect to the database remotely as the root user, which exists by default in all MySQL installations.

3.2.2 DNS Resolvers

DNS resolvers are commonly exploited for use in DNS-amplification DDoS attacks, amplifying traffic up to 179 times [36]. Bind and Unbound are two of the most popular DNS resolvers. While Unbound is strictly a DNS resolver, BIND can be utilised as both a resolver and an authoritative nameserver. We will only consider the resolver functionality, as this opens a server up to large-scale abuse, whereas this is less likely for a nameserver. We will test whether Bind and Unbound are configured as recursive resolvers by querying an existing domain name. If the requested record is returned, this indicates the server is configured as an open resolver.

3.2.3 NTP

Starting at the end of 2013, NTP amplification attacks rapidly rose to be the most significant vector for large DDoS attacks [12]. Like with DNS, NTP-based DDoS attacks exploit the UDP-based nature of the application protocol by spoofing the source address in requests to an NTP server and using the NTP server as traffic amplifier. In order to effectively amplify traffic, the NTP server is queried with the `monlist` command, which returns a very large reply. Though `monlist` has the highest amplification factor, other query commands exist in NTP for which the reply is larger than the request [12]. In our test we will query the NTP server with NTP the `monlist` and `peers` commands to determine whether the server is open to queries.

3.2.4 MQTT

The Message Queuing Telemetry Transport (MQTT) protocol, is a very popular protocol supporting Internet of Things devices. MQTT servers, also named *brokers*, can be deployed locally in the same network as the IoT devices it supports, as well as on remote servers that are accessed over the Internet by their client devices. Authentication of clients is possible through username and password as well as through client certificates as part of the MQTT protocol [10]. Authorisation of authenticated clients is handled using access control lists. When no access control is in place, any client is allowed to subscribe and publish to any topic on the MQTT server. Subscribing to all topics on the server additionally is possible using a multi-level wildcard (`#`). In this project we will be testing the Mosquitto [20] MQTT server. This is among the most widely used MQTT servers and is the only MQTT server available through the repositories on most of our test platforms. As no Mosquitto package is available in the repositories for CentOS 6, no MQTT tests are performed for this platform. In our MQTT test we will attempt to connect to the MQTT broker and subscribe to the `$SYS` topic, as well as all other topics by subscribing to a wildcard topic. We will then publish a message to a topic and check whether it is received by the subscribed client.

Platform	Version	MySQL	Unbound	Bind	NTP	MQTT	Postfix	Sendmail
Ubuntu	18.04	+	-	+	+	-	+^{1}	+^{2}
	16.04	+	-	+	+	-	+^{1}	+^{2}
	14.04	+	-	+	+	-	+^{1}	+^{2}
Debian	Stretch	+	-	+	+	-	+^{1}	+^{2}
	Jessie	+	-	+	+	-	+^{1}	+^{2}
CentOS	7	o	+	+	+	-	+^{2}	+^{2}
	6	o	+	+	+		+^{2}	+^{2}
Fedora	28	o	+	+	+	-	+^{2}	+^{2}
	27	o	+	+	+	-	+^{2}	+^{2}
FreeBSD	11.1	o	+	+	+	-	+^{1}	+^{1}
	10.4	o	+	+	+	-	+^{1}	+^{1}

+: Secure default; o: Problematic choice; -: Insecure default;
^{1}Relay access denied ^{2}Bound to localhost

Table 2: Overview of test results.

3.2.5 Mail Transfer Agents

Postfix is one of the currently most used mail transfer agents, while Sendmail has historically been very widely used [2]. Mail Transfer Agents have historically been designed to support forwarding or relaying e-mail destined for other mailservers, should the destination server be (temporarily) unreachable or unavailable to the sender. Nowadays however, open relays are commonly exploited to send unsolicited mail. Our test script will attempt to connect to the mail server installed on the tested system, and to relay e-mail through the server to a remote address without authenticating, both with an external e-mail address as sender, as well as a sender that is expected to exist on the tested host.

3.3 Ethical Considerations

As described in Section 3.1, the tests in this research are conducted on our own infrastructure on clean virtual machines that do not contain any user data. While the tested virtual machines can reach remote hosts, the machines are themselves unreachable outside of the testing network, preventing any insecurely configured services from being exploited and impacting a third party.

Should our tests uncover problematic defaults that are likely to impact a large number of users, we will responsibly disclose our findings to the parties involved.

4 Results

We used our framework to execute the tests on our set of applications and platforms. The results of the tests are displayed in Table 2. The results per service are described in their respective sections from Section 4.1 onward.

4.1 MySQL

On none of the tested platforms was it possible to remotely log into the MySQL server as the root user. On Ubuntu and Debian, this was due to MySQL only listening for connections on the localhost interface, whereas Fedora, CentOS and FreeBSD by default listened for connections on all interfaces, but had the default users configured to only accept login attempts from those users from the local machine. This means that if an administrator were to configure a new user with a weak password and does not properly limit the hosts the user is allowed to login from, the system would immediately be open to be compromised with the default Fedora, CentOS and FreeBSD configuration, whereas this would not be the case with the Ubuntu and Debian defaults.

4.2 DNS

In our DNS tests we find notable differences between our two tested resolvers. While Unbound is by default configured to only listen for queries on the localhost interface, for Bind this is only true for Fedora, CentOS and FreeBSD. Interestingly, the CentOS and Fedora configuration files for Unbound contained a note that binding to all interfaces was disabled, as per the Fedora policy of not listening on all interfaces by default. This is especially notable considering MySQL was seen to listen on all interfaces by default on these platforms, as described in Section 4.1.

A possible explanation for the difference between Bind and Unbound on Ubuntu and Debian may be found in the no-setting defaults for these applications. When the interfaces to listen on are not explicitly set in the configuration, Bind defaults to listening on all interfaces, whereas Unbound defaults to listening only on the localhost interface.

Of note is that while both have Bind configured as an open resolver by default, the very minimal default Unbound configuration for Debian Stretch and Ubuntu 18.04 enables RFC7816 [11] support, minimising queries to upstream nameservers, meant to increase user privacy.

4.3 NTP

While having been used as a vector for some of the largest DDoS attacks ever recorded [12], NTP is among the most well-behaved applications in our testset. Though listening on all interfaces by default, it allows no queries by default on any platform.

As a response to NTP's stake in large DDoS attacks, the `monlist` query has been completely removed from ntpd versions 4.2.7 and up [31]. While CentOS 6 and 7, and Debian Stretch and Ubuntu 14.04 run ntpd 4.2.6, the default configuration for all these platforms has been configured to not accept NTP queries by default.

As large numbers of NTP servers were seen to participate in DDoS attacks, we installed ntpd on Ubuntu 6.06 to find out how NTP was configured on this old system. It turned out however that even in this release NTP is configured to not accept any queries by default. Though no comprehensive history of Ubuntu packages could be found, the overview of Debian NTP package versions [4] revealed NTP was configured not to accept queries by default as far back as 2006 as well.

4.4 MQTT

As suggested by its manpage [19], Mosquitto is seen to be completely open to outside read and write access by default on all tested platforms. While making it very easy to set up an MQTT service, these defaults are troubling considering the fact MQTT is commonly used to forward commands to clients connected to the server. While it is unclear whether these situations are rooted in the insecure defaults we found here, improperly configured MQTT servers have already been seen to have significant security and privacy risks [21, 35].

4.5 Mail Transfer Agents

Both postfix and the older Sendmail can be seen to perform securely by default. With spam and malicious e-mail having been commonplace for years, it is not unlikely the default configurations currently used have been specifically developed in order to not setup mail servers as open mail relays by default. Notable is that CentOS and Fedora have postfix bound to localhost by default, whereas one would usually want a mail server to listen on a publicly accessible interface in order to be able to receive messages from other servers. If binding to localhost is the only security strategy applied, this could potentially make the system less secure than when relay access is denied for unknown users or domains, as users that wish to configure the mail server to receive e-mail from other servers will negate the bind-to-localhost defence.

Sendmail is by default bound to localhost on all platforms except for FreeBSD. This is most likely due to the fact that Sendmail is and has been often used just as an SMTP client, only sending e-mail. In this use case, it is not required to listen on an interface.

5 Discussion

In our results we have found some marked differences in the security posture of services both between the same service running on different platforms, as well as comparable services on the same platform. While looking at the configuration file for Unbound on Fedora, we found explicit mention of a Fedora policy not to listen on all addresses by default. While Fedora has extensive guidelines regarding the approval process for new packages, no mention is made of what should be in a default configuration [6]. Fedora does however require all services that start by default on package installation to not listen for remote connections [5].

Though not explicitly mentioning default configurations, Debian has guidelines in place that aims to audit all packages conforming to certain criteria [1], one of which is providing a remotely reachable service.

While having a spotty reputation in the past and present due to their role in abuse on the Internet, ntpd and both MTAs tested in this research project are securely configured by default on all our tested platforms. As the three respective applications and the service they provide have been around for quite some years, this long timeframe may have contributed to the nowadays relatively sane state of their default supplied configurations.

Though most packages were bound to localhost by default on Fedora and CentOS, notably MySQL was not. This is notable as in the majority of situations wherein MySQL is used, it supports applications that run on the same host as the database itself. In such situations it is not necessary to be able to accept remote connections. Only in situations where a database server should support applications running on other hosts is remote access desired. As described in Section 4.1, this should only be done in a trusted network.

While MySQL by default is bound to localhost on the tested Debian and Ubuntu versions, Bind on the other hand is configured as an open resolver on these systems whereas this is securely configured on Fedora, CentOS and FreeBSD. This again is problematic. While recursive is one of the most common use cases for Bind, and even the only use case for Unbound, in most cases the resolver should only be available to the system itself or a number of trusted systems, rather than the entire Internet. As such, for most services it would be sensible to place the burden of opening the system up as much as is desired to the user, rather than the other way around.

5.1 Limitations

As our tests are executed from a host remote to the tested system, it was not possible to check any further layers of defence, should a service be bound to localhost by default. As such, we did not test whether any further layers of defence are deployed for these services, and what the security posture of these services would be when a user configures the service to be remotely accessible.

In this research project we have executed our tests over IPv4. While we do not believe to find different results if we were to replicate our tests over IPv6 for the software tested in this project, discrepancies in IPv4 and IPv6 availability of services may be found for other applications [13].

As our framework utilises Vagrant for managing the virtual machines, we relied on the distributors of our tested platforms to provide Vagrant compatible images reflecting a default installation of the platform. As building our own images was outside the scope of this research, we could not test platforms that had no official Vagrant images available.

6 Related Work

With misconfiguration of Internet services making for potentially dangerous attack vectors, work exists into analysing software or device configurations for dangerous or inadvisable settings, such as *SCAAMP* [14]. This framework generates a security score for an Apache MySQL PHP (AMP) setup by parsing the respective configuration files and comparing the settings to their recommended values. An approach for detecting insecure configurations in an end-to-end network is presented in the *Network Configuration in A Box* paper [9]. Herein *ConfigChecker* is presented, which allows to express a network of devices with their configurations in a model and then test the security posture of the modelled network. Besides rule-based configuration testing, data-mining has also been applied to detecting misconfigurations, such as *Minerals* [18], which detects misconfiguration in networks with multiple routers by applying association rules mining to router configuration files. Also a data mining approach, *EnCore* includes information about the system environment in misconfiguration detection, taking built-in defaults into account [37].

While a large body of work exists on misconfigurations in software, no work was currently found to exist on the role of insecure default settings in systems that are considered to be misconfigured. In this paper we provide a first look into the security posture of default settings, and the origin of these defaults. We also introduce a framework that allows to easily test default settings for internet services on different platforms.

7 Conclusion

In this paper we have developed an automated framework allowing us to easily install and test software on different platforms without user interaction in a way that is easily reproducible.

In our limited set of tested software and platforms we have already found differences regarding the security posture of default software configurations, notably for different pieces of software serving the same means on the same platform.

Based on the different strategies for securing Internet services observed in this research, a metric quantifying the security posture of an Internet service could be expressed in the number of security layers defending against common exploitation of the respective service. Such a metric however can not necessarily be generalised between different services, with binding to localhost being arguably more effective for database servers than for mail transfer agents, with the former only requiring remote access in some specific use cases and the latter requiring it in practically all.

While distributors of an operating system are usually not responsible for the actual contents of default configurations of packages in their repository, they are responsible for the overall repository and can demand a certain level of security for all packages therein, mandating certain defaults, as referenced in Section 4.2. While we find differences in security posture for Internet services between platforms, we did not identify factors that would prevent or prohibit a software distributor from increasing the default security of their software package. Rather we do see that where security policies are in place prohibiting listening on all interfaces by default, services can be seen to have a better default security posture on those platforms than where those policies are not enforced.

Future Work: While in this research we have seen most services to be shipped with sane default settings, it may be very interesting to research the defaults that may have been shipped for an application throughout the years. This may however be challenging as this would largely rely on the availability of old repository snapshots as these will have been updated throughout the lifetime of the respective distribution.

In addition to default configurations, systems may also be vulnerably configured through configurations from other sources. Many tutorials for example exist how to configure certain Internet services. These tutorials may have been followed by a large number of users and it may be interesting to determine how secure these tutorial configurations really are. Similarly, large numbers of users may be running services on the same Docker images. While these images can automatically be tested for vulnerabilities as described in Section 2, their configuration is not tested, while a single misconfiguration in an image may affect a large number of users spread over many different networks.

In this research project we have tested only a small set of applications and platforms. Scaling up this research to include more platforms and packages could be interesting further research. While we have only focused on applications on general purpose operating systems, analysing the default configurations of embedded devices could additionally be very interesting. Configurations of devices such as consumer routers are often not modified or updated by their users at all, meaning defaults could persist in a network for an extended period of time.

In this paper we have focused on the security implications default configurations. While briefly touched upon in Section 4.2, we have not specifically looked into privacy implications of default configurations. Future work could thus be extending the scripts used in this research to also perform such tests. A new project investigating the default privacy of client applications could also be imagined.

References

[1] Debian – Auditing Package Prioritization Guidelines. `https://www.debian.org/security/audit/packages.en.html`. (Accessed on 2018-07-10).

[2] Mail (MX) Server Survey. `http://www.securityspace.com/s_survey/data/man.201806/mxsurvey.html`. (Accessed on 2018-07-10).

[3] Nessus professional vulnerability scanner. `https://www.tenable.com/products/nessus/nessus-professional`. (Accessed on 2018-07-09).

[4] ntp - snapshot.debian.org. `http://snapshot.debian.org/package/ntp/`.

[5] Packaging:defaultservices - fedora project wiki. `https://fedoraproject.org/wiki/Packaging:DefaultServices?rd=DefaultServices`. (Accessed on 2018-07-10).

[6] Packaging:ReviewGuidelines - Fedora Project Wiki. `https://fedoraproject.org/wiki/Packaging:ReviewGuidelines#Package_Review_Process`. (Accessed on 2018-07-10).

[7] Stack Overflow Developer Survey 2018. `https://insights.stackoverflow.com/survey/2018/#technology-databases`. (Accessed on 2018-07-11).

[8] Vagrant by HashiCorp. `https://www.vagrantup.com/`. (Accessed on 2018-07-11).

[9] Ehab Al-Shaer, Will Marrero, Adel El-Atawy, and Khalid Elbadawi. Network configuration in a box: Towards end-to-end verification of network reachability and security. In *Proc. IEEE International Conference on Network Protocols*, pages 123–132, 2009.

[10] Andrew Banks and Rahul Gupta, editors. *MQTT Version 3.1.1*, OASIS Standard, October 2014.

[11] S. Bortzmeyer. DNS Query Name Minimisation to Improve Privacy. RFC 7816, RFC Editor, March 2016.

[12] Jakub Czyz, Michael Kallitsis, Manaf Gharaibeh, Christos Papadopoulos, Michael Bailey, and Manish Karir. Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks. In *Proc. ACM Internet Measurement Conference*, pages 435–448, 2014.

[13] Jakub Czyz, Matthew Luckie, Mark Allman, and Michael Bailey. Don't Forget to Lock the Back Door! A Characterization of IPv6 Network Security Policy. In *Proc. Symposium on Network and Distributed System Security (NDSS)*, volume 389, 2016.

[14] Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam. Early detection of security misconfiguration vulnerabilities in web applications. In *Proc. IEEE Conference on Availability, Reliability and Security (ARES)*, pages 169–174. IEEE, 2011.

[15] Tobias Fiebig. *An empirical evaluation of misconfiguration in Internet services.* PhD thesis, TU Berlin, 2017.

[16] Tobias Fiebig, Anja Feldmann, and Matthias Petschick. A one-year perspective on exposed in-memory key-value stores. In *Proc. ACM Workshop on Automated Decision Making for Active Cyber Defense*, pages 17–22, 2016.

[17] Lily Guo, Toli Kuznets, and Nandhini Santhanam. Docker security scanning safeguards the container content lifecycle - docker blog. `https://blog.docker.com/2016/05/docker-security-scanning/`, may 2016.

[18] Franck Le, Sihyung Lee, Tina Wong, Hyong S Kim, and Darrell Newcomb. Minerals: using data mining to detect router misconfigurations. In *Proc. ACM SIGCOMM*, pages 293–298, 2006.

[19] Roger Light. mosquitto.conf man page — Eclipse Mosquitto. `https://mosquitto.org/man/mosquitto-conf-5.html`.

[20] Roger A Light. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13), 2017.

[21] Lucas Lundgren. Lightweight protocol! serious equipment! critical implications! `https://www.rsaconference.com/writable/presentations/file_upload/hta-r03-light-weight-protocol-serious-equipment-critical-implications.pdf`, 2016.

[22] Kiran Nagaraja, Fábio Oliveira, Ricardo Bianchini, Richard P Martin, and Thu D Nguyen. Understanding and dealing with operator mistakes in internet services. In *Proc. OSDI*, volume 4, pages 61–76, 2004.

[23] OpenVAS. OpenVAS - OpenVAS - Open Vulnerability Assessment System. `http://www.openvas.org/`.

[24] OWASP Foundation. Top 10 2010-A6-Security Misconfiguration - OWASP. `https://www.owasp.org/index.php/Top_10_2010-A6-Security_Misconfiguration`, 2010.

[25] OWASP Foundation. A10 2004 Insecure Configuration Management - OWASP. `https://www.owasp.org/index.php/A10_2004_Insecure_Configuration_Management`, 2004.

[26] OWASP Foundation. Top 10 2013-A5-Security Misconfiguration - OWASP. `https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration`, 2013.

[27] OWASP Foundation. Top 10-2017 A6-Security Misconfiguration - OWASP. `https://www.owasp.org/index.php/Top_10-2017_A6-Security_Misconfiguration`, 2017.

[28] Rapid7. Top Rated Vulnerability Management Software — Rapid7. `https://www.rapid7.com/products/nexpose/`.

[29] Red Hat Inc. Ansible is Simple IT Automation. `https://www.ansible.com/`. (Accessed on 2018-07-11).

[30] Red Hat Inc. CoreOS Clair Documentation. `https://coreos.com/clair/docs/latest/`.

[31] United States Computer Emergency Readiness Team. NTP Amplification Attacks Using CVE-2013-5211. `https://www.us-cert.gov/ncas/alerts/TA14-013A`, January 2014. (Accessed on 2018-07-04).

[32] The Jenkins Project. Jenkins. `https://jenkins.io/`.

[33] Travis CI GMBH. Travis CI - Test and Deploy with Confidence. `https://travis-ci.com/`.

[34] Twistlock Ltd. Docker security & docker swarm security — twistlock. `https://www.twistlock.com/solutions/docker-security/`.

[35] Henk van Doorn and Bernardus Jansen. Monitoring MQTT monitors, 2018. unpublished.

[36] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC and its potential for DDoS attacks: a comprehensive measurement study. In *Proc. ACM Internet Measurement Conference*, pages 449–460, 2014.

[37] Jiaqi Zhang, Lakshminarayanan Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou. EnCore: Exploiting system environment and correlation information for misconfiguration detection. *ACM SIGARCH Computer Architecture News*, 42(1):687–700, 2014.