



## Cloud Native Security Report

---

# Watching the Honeypots

H2 2018

# Contents

- 3 Introduction**
- 3 About Twistlock Labs**
- 4 Methodology**
- 7 Summary**
- 8 Honeypot Samples**
  - 8 CouchDB
  - 9 Redis
  - 10 MySQL
  - 12 Elasticsearch
  - 12 Apache Tomcat
  - 13 Jenkins
- 14 Conclusion**

## Introduction

At DockerCon 2017 in Copenhagen, the Twistlock Labs team unveiled findings from a months-long investigation into exposed cloud servers. This research made it clear that a large number of containerized applications were being deployed with known vulnerabilities and weak defaults.

Since that time, Twistlock Labs has focused on how developers around the world build and deploy cloud native applications, and how these applications are maintained and operated by companies around the world. This report, the first in a bi-annual series, focuses on common vulnerabilities and attack patterns against the most popular containerized applications in the wild.

To read more research from Twistlock Labs, and to subscribe to updates, visit [Twistlock.com/labs](https://twistlock.com/labs).

## About Twistlock Labs and this report

Twistlock Labs, Twistlock's Security Research Team, is constantly trying to get into the minds of attackers to better understand how they would attempt to gain access to or potentially compromise your containerized and cloud native environments. We then take this data and share details on these threats or work directly with Twistlock R&D to include [protection for real-world attack scenarios](#) in the [Twistlock Platform](#).

This report is the first in a bi-annual series that examines risks and attacks in the cloud native computing ecosystem. The next report will be released in the first half of 2019.

## Methodology

For this report, we decided to focus on two main sampling methods. The first method involves scanning the internet internally and using public scanning services for openly accessible servers. We came up with a narrow list of applications that are commonly used based on statistics from Docker Hub. We then proceeded to scan for their banners for versions and potential vulnerabilities. The main goal of this search was to sample the version distribution for each application, and more specifically the percentage of applications that are potentially (or clearly) vulnerable to known security issues.

For the second method, we set up honeypots of relevant popular cloud native applications. These honeypots are servers that externally mimic the behavior of the real application, i.e. running on the same default port of the real application and responding to requests with the appropriate banner, while running a completely different logic to log all requests and save all incoming data<sup>1</sup>.

The goal of running these honeypots was to detect patterns of attacks on open servers and identify vulnerabilities that are being actively exploited in the wild (or finding zero days, if we were lucky enough). After absorbing a huge volume of traffic, we proceeded to analyze it to extract additional information. For example, figuring out if the attacks were made by bots or by active attackers.

As suspected, the team found a disturbing number of out-of-date applications, with many open to known vulnerabilities (with CVEs). Some of these were vulnerabilities that have been disclosed years ago. Additionally, the team found a great amount of active bots/attackers that search for these applications in attempt to exploit them. By combining these findings it is reasonable to assume that many of the open servers we examined are already compromised by these attackers.

---

<sup>1</sup> “a computer security mechanism set to detect, deflect, or, in some manner, counteract attempts at unauthorized use of information systems.” Source: [https://en.wikipedia.org/wiki/Honeypot\\_\(computing\)](https://en.wikipedia.org/wiki/Honeypot_(computing))

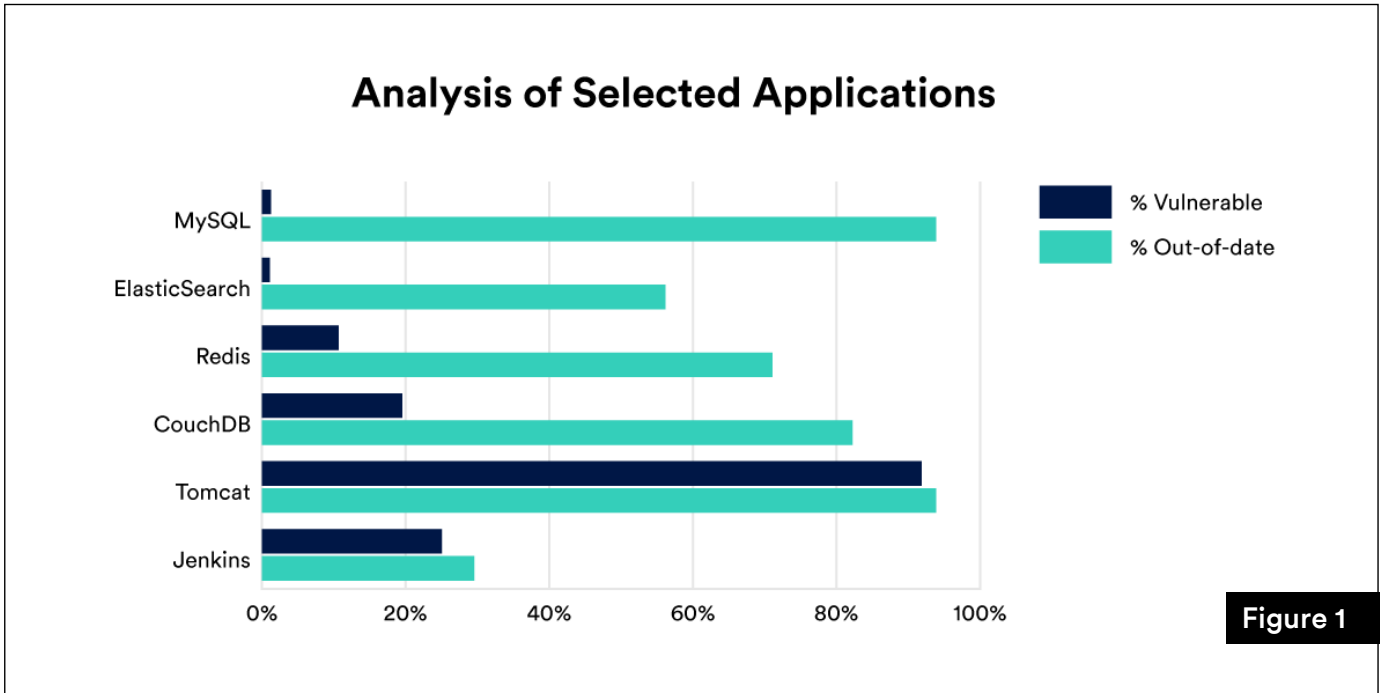


Figure 1 shows the ratio between safe and vulnerable instances of the selected applications we sampled. Most deployments don't actually run the latest releases, but some software vendors release backport patches, making these deployments safe to use. The chart clearly demonstrates which of the applications that we have sampled are more likely to have patches to older versions and which are likely to be ran in the wild with open vulnerabilities.

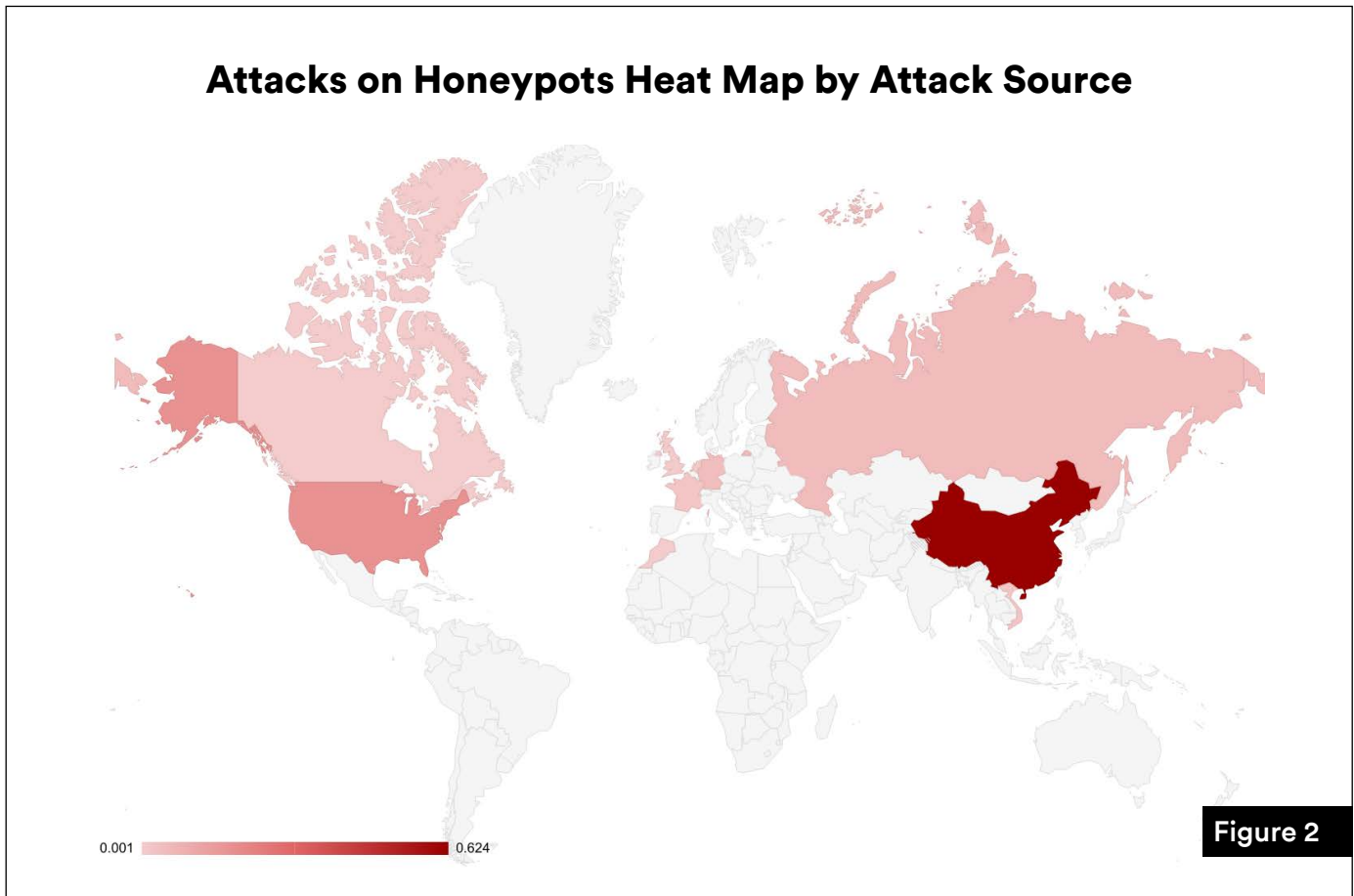


Figure 2 shows the countries from which our attacks originated. The country that immediately stands out is China. The majority of all the attacks we witnessed across all of the services (60%), were sourced from Chinese IP addresses. This should not be surprising to anyone who has had the chance to administrate a server publicly exposed to the internet in recent years. Chinese bot and malware servers are notorious for aggressively scanning the internet and attempting to exploit known vulnerabilities. In fact, in our honeypots we saw attempts to upload different malware binaries from China, and some of these binaries were identical in different honeypot instances. Our honeypot instances were deployed on completely different VMs and there was no visible correlation between them, meaning that the same attackers attempted to use different exploits with the same payload binaries on various publicly exposed cloud applications.

## Summary

To avoid attacks through known issues and vulnerabilities, organizations must make sure all their systems are fully up to date with the latest security releases from their software vendors. Patches and updates should be reviewed on a regular basis and applied to all layers, from operating systems to end applications.

Unfortunately, to our knowledge there is a profound amount of businesses that fail to do so, from the smallest to the biggest and well budgeted ones.

### Key findings include:

- Surveying the top cloud native applications, 25 percent were running with CVEs where a known exploit exists
- MySQL was the most likely to be out of date, with over 80 percent of deployments being at least one version behind. Overall, 60 percent of all cloud native apps are not patched to the latest version
- Over 90 percent of detected attacks were automatically executed — zero touch hacking that focuses on brute force or known exploits
- China plays a significant role in the modern threat landscape with over 60 percent of detected attacks against cloud native applications originated from Chinese IP ranges

It appears that some businesses are still unaware of the issues with running out-of-date software. Some businesses decide to ignore or belittle the danger of it, while some fail to adequately and consistently make sure all of their deployments are updated.

### We hope our research allows you to:

- Better understand recent attack vectors and threat scenarios facing modern applications
- Improve risk management at your organization as it relates to proper software updates, workflows, and patching cadence
- Recognize the benefits from deploying scalable security solutions to stop automated threats

## Honeypot Samples

### An abundance of open and vulnerable applications

We wanted to assess just how common it is for popular cloud native applications that are found in the wild to be out-of-date. By systematically scanning the internet for selected cloud native applications we discovered that around ~24.9% of the applications we sampled were running code that is vulnerable to a known vulnerability/vulnerabilities. We were surprised to find such a high number of hits given we were not specifically targeting vulnerable apps, but rather ran a untargeted search for instances open to the internet.

### CouchDB

[Apache CouchDB](#) is an open source database server based on NoSQL architecture, written in Erlang. The official CouchDB Docker Hub image has more than 10 million pulls, with other automatic builds also exceeding one million pulls<sup>2</sup>. It is one of the oldest NoSQL database around, and is commonly used in cloud environments.

It is not surprising that many attackers use automated bots to scan for CouchDB servers and to attempt access in unprotected instances. In the honeypot traffic, we found that most attacks were either trying to exploit known vulnerabilities or trying to access potentially open keynames.

The majority of these requests were to CouchDB internal keynames. Some of these keynames were historically open to non-admin users, being a major security hole in all of the versions that allowed their usage.

For instance, we found attempts to access `/_all_dbs`. Versions earlier than 1.3.x are vulnerable to this key being openly accessible<sup>3</sup>. Other keys that attackers tried to get were `/_uuids`, `/_users` and `/_config`.

Through the `/_users` keyname, different attackers tried different attacks, such as creating a new user with: `PUT /_users/org.couchdb.user:operator`. With the `/_config` keyname we saw attempts to exploit [CVE-2017-12636](#), specifically by trying to access `"/_config/query_servers/cmd"`. This is an attack has been widely documented online by different researchers, and full, working exploitation chains are easily accessible online.

With only a handful of exceptions, it appears that all of the traffic we received was coming from bots. This observation is based on user agent strings and the timing between each request sent from the same source.

<sup>2</sup> <https://hub.docker.com/search/?isAutomated=0&isOfficial=0&page=1&pullCount=0&q=couch&starCount=0>

<sup>3</sup> See <https://github.com/hoodiehq/hoodie-server/issues/14> and this commit <https://github.com/hoodiehq/hoodie-server/commit/f450125b4a26dc31ad5bd63d2dbce7c457e26f19>



The underlying conclusion with this honeypot is that CouchDB instances are being actively exploited with mostly known attacks. Any instances open to the internet, that are either out-of-date or misconfigured, could have surely been compromised by one of the attacks we had witnessed.

## Redis

[Redis](#) is another open source NoSQL database server. Redis is considered the most popular key-value database server, and it's one of the most popular databases used in containers<sup>4</sup>. It's official Docker Hub image has more than 10 million pulls and it's listed as one of the top images pulled from DockerHub<sup>5</sup>. To put it simply, Redis is one of the most popular databases available for the cloud native nowadays.

Our Redis honeypot attracted a lot of traffic. The traffic came both from bots and from active attackers that tried to execute various commands. Our Redis honeypot server answered to all commands even from unauthenticated users, so we were able to see all kind of the requests attackers would try to run on any similar open server.

Many attackers attempted to run the **INFO** command first. This makes sense as this command could provide the attacker with many details about the Redis instance and the machine. Some of the attackers were likely hands-on, implied by the frequency and content of the commands they tried to execute.

Other commands attackers commonly executed were **CLIENT** and **LIST**. Some attackers were actually destructive enough to execute the **FLUSHALL** command straight away. All other attackers ran automated commands.

### Key attack patterns we recognized were:

- Attempts to modify or change the config with the **CONFIG** command (e.g. `config set dbfilename ...`)
- Attempts to download a file and execute it by setting a cron string in one key
- Attempts to call the **EVAL** command with a Lua payload

4 <https://en.wikipedia.org/wiki/Redis>

5 [https://hub.docker.com/\\_/redis/](https://hub.docker.com/_/redis/)

The attacks using the **EVAL** command were particularly interesting. The payload seemed to be obfuscated and it included various Lua commands such as `loadstring`, `rawset` and many type castings. After some more serious online researching it appears that these payloads were an attempt to exploit a known security issue with Lua of two years ago, found and [reported](#) by researcher `corsix`.

These attacks are meant to escape the Lua VM by abusing Lua bytecode. It has been mentioned in multiple places online and an exploit adapted to be used on Redis instances has been published online<sup>6</sup>.

In any regard, the keys and the remote URLs and shell files that we found were mostly new. We searched for their hashes online and for some found only minimal references, and none for other samples.

## MySQL

MySQL is one of the oldest database servers that are still commonly used and deployed today. Like the other database servers mentioned in this paper, the Docker Hub MySQL image is listed among the most popular images with more than 10 million pulls<sup>7</sup>. Compared to other honeypots, the MySQL honeypot attracted a lot of attack trails. Most of the attacks started with bruteforce on the MySQL login and then continued with execution of queries.

**We noticed two main attack vectors, both aimed at gaining code execution:**

- a.
  - i. Insert a binary ELF (whether shared object or executable file) into the database as a big blob (using **INSERT** query)
  - ii. Write the executable ELF file into the disk using **EXPORT** query
  - iii. If the binary file is a shared object, the attackers used **DLL Hijacking** method to load it into another process. Let's elaborate on this method: applications load external code using shared libraries. The Linux loader is in charge of finding the right library and load it when the applications starts. If the application is known to load a library, which is missing, one can put a malicious library in the right path and it will be loaded into the application memory address by the loader.
  - iv. If it's the binary file is an executable file, the attacker used other queries to execute it.

<sup>6</sup> Not the particular published exploit, but the a blog post by researcher Ben Murphy discussing the issue with Redis: <http://benmmurphy.github.io/blog/2015/06/09/redis-hot-patch/>

<sup>7</sup> <https://hub.docker.com/explore/>



## Elasticsearch

[Elasticsearch](#) is a NoSQL database which is delivered as part of the [ELK stack](#). Elasticsearch provides an easy way to store data and is mostly used for log analysis. Elasticsearch provides convenient solution when doing full text search, which is something that is quite hard with structured databases.

In 2015, a vulnerability was found in the Java Groovy sandbox in Elasticsearch which allows code execution. This vulnerability has been assigned [CVE-2015-1427](#). This vulnerability is likely the most known Elasticsearch vulnerability.

Therefore, we configured our honeypot to report a vulnerable version of Elasticsearch so we could find out if attackers still try to exploit this vulnerability.

As suspected, from the traffic we received it appears the known exploit for this specific vulnerability was being widely spread by attackers.

We did not receive many scan trails nor attack trails. We also found that some of the attacks on our honeypot were connected with the big operation that we saw on the MySQL instances.

## Apache Tomcat

[Tomcat](#) is an application server from the Apache Software Foundation that executes Java servlets and renders web pages that include Java Server Page coding. Described as a “reference implementation” of the Java Servlet and the Java Server Page specifications, Tomcat is the result of an open collaboration of developers and is available from the Apache web site in both binary and source versions.

Our Tomcat honeypot attracted a relatively small number of automated attacks, each of the attacks had the same characteristics in terms of the method of compromise and the end goal of the attacker:

### The attackers attempted to gain access with one of the 2 methods:

- Exploiting a single specific vulnerability: [CVE-2017-12617](#)
- Bruteforcing the admin password

Once the attackers gained the initial access with one of the 2 methods above, they proceeded with uploading a “webshell” - a specially crafted webpage, which when uploaded and served by the Tomcat server can act as a gateway inside the tomcat server, and provide a shell access for the attacker.

Once inside the server (shell access) each and every one of the attackers executed crypto-mining scripts and left the scene.

All of the attacks are presumed automated as the speed of executing the steps is noticeably high. In addition there were some traces of attempts to elevate privileges (once the webshell is uploaded the shell access that is gained from it has the same privileges as the tomcat application, e.g non-root).

## Jenkins

[Jenkins](#) is a popular CI (continuous integration) platform written in Java. It's a versatile tool that has become very common for use within software development deployments. It allows developers to automate build processes and find bugs before releasing their software. The official Jenkins image in Docker Hub has more than 10 million pulls.

Our Jenkins honeypot was attacked with a only a single vulnerability: [CVE-2017-1000353](#)

This vulnerability allows an attacker to execute commands on the host with the context of the Jenkins user.

Judging by timing between commands we can assume that all the attacks were automated. By examining the attacks' exploit chains we found two major categories of payloads:

1. Deployed mining operations after validating that command execution was successful
2. Establishing persistent control of the machine

## Conclusion

By deploying honeypot servers and sampling deployments available on the internet we were able to gather statistical information regarding the state of the cloud native world. We learned that the majority attacks on cloud native applications are automated and sourced from bots. Moreover, the larger part of these attacks rely on known vulnerabilities, which in many cases have assigned CVEs or can otherwise be easily be found online. We were not surprised to find attacks relying on exploits that are published online (“skiddie” attacks).

As always, we recommend that all our readers put a greater effort on foundational security practices, such as not exposing internal applications to the internet unless strictly required. Our main recommendation is to systematically verify that all software deployments are up-to date with the latest security patches.

---

## About Twistlock

Twistlock is the most complete, automated and scalable container cybersecurity platform. Automated and built for scale, Twistlock provides container security for teams using Docker, Kubernetes, and other technologies.

### Learn More

[www.twistlock.com](http://www.twistlock.com)



### Contact Us

[contact@twistlock.com](mailto:contact@twistlock.com)

### Portland

411 NW Park Ave  
Portland, OR  
97209