



DISTRIBUTED NETWORK DISCOVERY

BUILDING A NEW PORT SCANNER
FROM SCRATCH

By edermi

AlligatorCon Europe 2019

META!

- x Security Consultant from Munich (pentesting, social engineering)
- x  @edermi
- x  @michael_eder_ (no this is not my real name *cough*)

- x Slides: SlidesCarnival – Logos: Fontawesome – Drawings: Draw.io
- x You may take photos of the talk



INTRO: NETWORK WHAT?

NETWORK DISCOVERY AKA PORT SCANNING

- ✗ Find out if a TCP/UDP port is open
- ✗ Concept is around for decades
- ✗ Several different approaches (TCP connect, TCP SYN, just send as much as possible and see who replies)
- ✗ Usually one of the first tasks in order to find vulnerable systems
- ✗ Base data for more in-depth analysis

2.

PROBLEM STATEMENT:
PORT SCANS ARE REGULAR
PART OF YOUR JOB

“THE PROBLEM IS ALREADY SOLVED!!!”

- ✗ Nmap – already supports every scan type you can imagine
- ✗ Masscan – super fast
- ✗ ZMap – academics love that one
- ✗ Unicornscan – somehow distributed, unmaintained, but still used by some
- ✗ Wolpertinger – distributed, unmaintained (?)
- ✗ Metasploit, thousands of random Powershell/Python scripts
- ✗ Nessus, Qualys, Nexpose (\$\$\$, shiny web interfaces)

Why write your own?

DEPLOYMENT

- ✗ They often require root / administrative privileges (raw sockets)
- ✗ Some of them are picky regarding OS support
- ✗ They often have runtime dependencies (pfring, libpcap, glibc, ...)
- ✗ Creating a static binary that runs on anything apart from amd64-linux is a time-consuming adventure



DEPLOYMENT

- ✗ Often no direct network access (and if, they somehow managed to give you the place attached to the 10MBit switch)
- ✗ How to gain insight into DMZ, IoT, production or administrative networks?
- ✗ Cloud environments aka 172.16.0.0/12, but somewhere on port 31xxx you are probably going to find the holy grail
- ✗ Usually, pivot via jump server, web shell, phished employee – no choice of environment

TARGET SELECTION, RESULT REPORTING

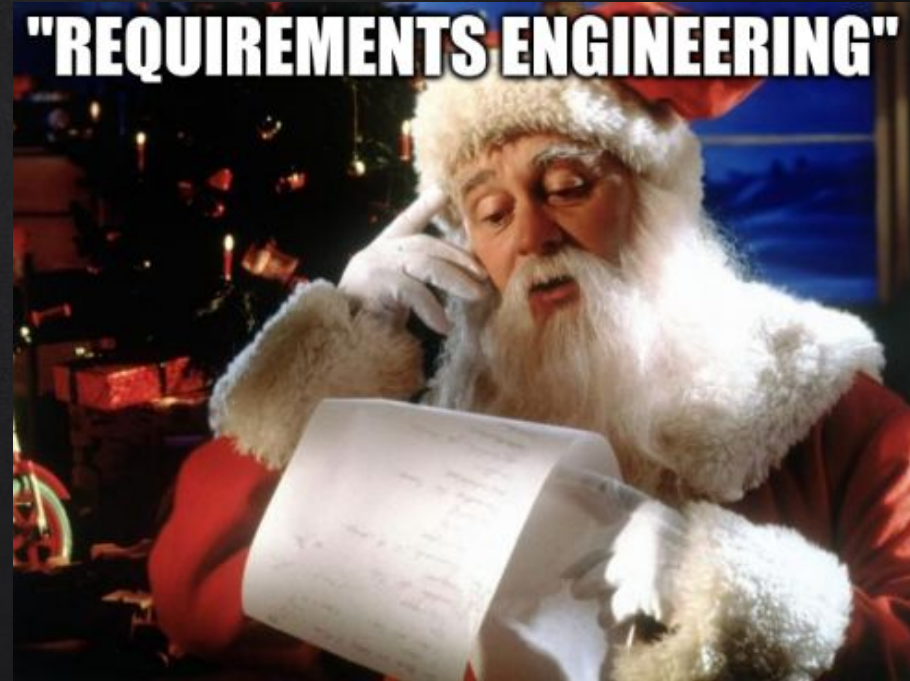
- ✗ “Static” target selection: List of CIDR networks
- ✗ Results are either printed to stdout or written to something that needs 2 hours of scripting to be parsed (.nmap, .XML)
- ✗ “Give me a file with targets/networks and I’ll give you some XML”



REINVENTING THE WHEEL

FEATURE WISHLIST

- ✗ Scan ports (obviously)
- ✗ No runtime dependencies
- ✗ Run as unprivileged user
- ✗ Easy to build for \$OS and \$architecture
- ✗ Pluggable target selection modules
- ✗ Event-based, flexible reporting



FEATURE WISHLIST

- ✗ Scan ports (obviously)
- ✗ No runtime dependencies
- ✗ Run as unprivileged user
- ✗ Easy to build for \$OS and \$architecture
- ✗ Pluggable target selection modules
- ✗ Event-based, flexible reporting
- ✗ Bonus: Serve as project for my master's thesis



Department of Informatics
Technical University of Munich



TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

Network Discovery Orchestration

Michael Eder

ALSO NICE TO HAVE

- ✗ Config file (commentable, versionable, no more do-nmap.sh)
- ✗ Trigger application layer scans, e.g. if tcp/22 is open, do a more in-depth SSH-scan
- ✗ Create data to easily build your own, private Shodan / Censys / Binaryedge / \$wescanforyou

✗ Distributed scanning



NRAY

NRAY

- ✗ Free software (GPLv3)
- ✗ Written in *pure* Go
 - architecture and OS independent
 - easy to build and cross-compile
 - one static binary, no runtime dependencies
 - goroutines: high level of concurrency internally

NRAY

- ✗ TCP connect scan:
 - requires no special permissions, restricted guest user in locked down container is sufficient
 - with goroutines, it is easy to have hundreds of connections in parallel
 - default config still magnitudes faster than Nmap's defaults
- ✗ UDP scan: implemented
- ✗ ZGrab2 (the engine behind censys.io) application layer scanning
 - HTTP and SSH implemented
 - More to come, but manual gluing required (SMB, FTP, \$mail, \$SQL)
- ✗ Distributed scanning: server-client architecture with control server and scanning nodes (optionally with TLS 1.2 and even mutual authentication if required)

DISTRIBUTED NETWORK DISCOVERY!

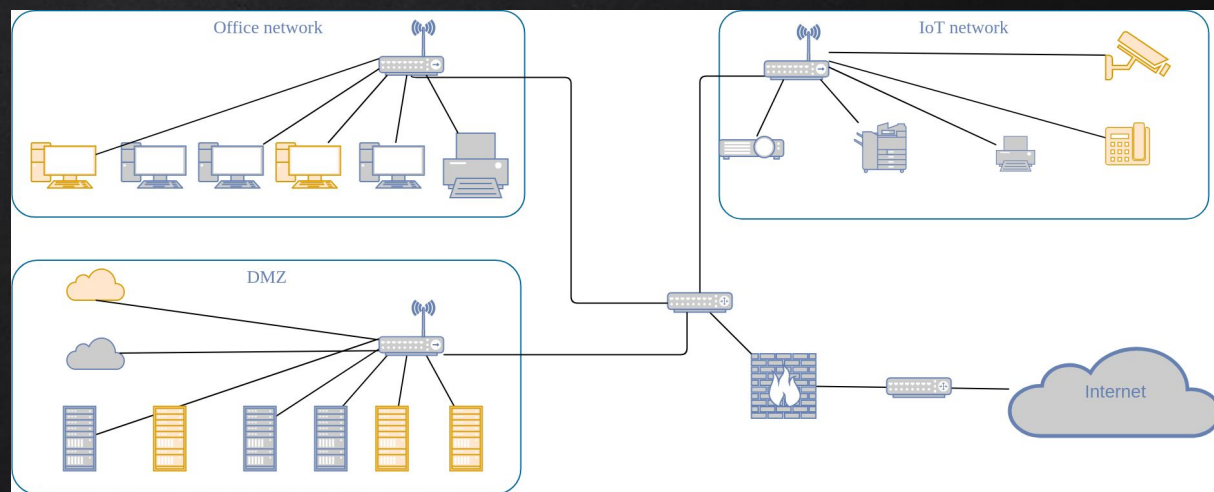
- ✗ Deploy scanning nodes anywhere!
- ✗ Nodes connect to server (same binary!), query for work, perform a scan, report back and query for more work
- ✗ Pull-based approach (nodes connect to server):
 - Easier traverse Firewalls, NAT etc. (getting out is usually easier, getting back easy because of stateful firewalls)
 - More flexible; Nodes may come and go, server does not need to know them before starting the scan
 - AFAIK unique to nray, everybody else did/does push-based
- ✗ Bonus: Resilience against I{D,P}S / angry network guys kicking you off the network

POOLING

- ✗ Create views by scanning from different vantage points
- ✗ Scale scan speed linearly with number of nodes
- ✗ Each pool contains all targets
- ✗ Each pool may have arbitrary scanner nodes working on it
- ✗ Allows to create views while still being able to speed up the scan

Here:

- ✗ IOT pool, 2 nodes → double scan speed
- ✗ Office pool, 2 nodes → double scan speed
- ✗ DMZ pool, 4 nodes → 4x scan speed



NRAY

- ✗ Many data sources
 - Hosts, networks (like any other scanner)
 - Certificate transparency logs (e.g. scan all mail.*.org on port 25)
 - LDAP (e.g. query DCs for computer accounts)
 - Easy to extend – attach your CMDB, asset-Excel, metasploit etc.
- ✗ Event-based results
 - Print to stdout, built-in filtering
 - JSON (easy to query with jq, also during a running scan)
 - Elasticsearch
 - Splunk (native adapter in work, uploading json works fine)
 - Also, easy to extend

```
certificatetransparency:
  enabled: false
  # For regex debugging: https://play.golang.org/p/jgDiTmPlqdW
  domainRegex: '^((www[.])?.*([.]com)$)'
  tcpports: [top25]
  udpports: [top25]
  blacklist: []
  maxHostsPerBatch: 150
  maxTcpPortsPerBatch: 25
  maxUdpPortsPerBatch: 25
ldap:
  enabled: false
  ldapSearchString: "(objectCategory=computer)"
  baseDN: "dc=contoso,dc=com"
  ldapAttribute: "dNSHostName"
  ldapServer: ""
  ldapPort: 636
  insecure: false
  ldapUser: ""
  ldapPass: ""
  tcpports: [top25]
  udpports: [top25]
  blacklist: []
  maxHostsPerBatch: 5
  maxTcpPortsPerBatch: 25
  maxUdpPortsPerBatch: 25
```

IT'S DOCUMENTED!

HTTPS://NRAY-SCANNER.ORG

1. Introduction

2. Installation

3. Usage

3.1 Configuration file (server)

3.1.1 Basic settings

3.1.2 TLS

3.1.3 Internal

3.1.4.0 targetgenerator

3.1.4.1 standard

3.1.4.2 certificatetransparency

3.1.4.3 ldap

3.1.4.4 DNS zone transfer

3.1.5.0 scannerconfig

3.1.5.1 tcp

3.1.5.2 udp

3.1.5.3 zgrab2

3.1.5.3.1 ssh

3.1.5.3.2 http

3.1.6.0 events

3.1.6.1 terminal

3.1.6.2 json-file

3.1.6.3 elasticsearch

3.2 Command line flags (node)

3.3 Pooling


3.4 TLS setup

[nray](#) > [Usage](#) > [Configuration file \(server\)](#) > [targetgenerator](#) > [standard](#)

[Edit this page](#)

STANDARD

This is the standard target generator that is probably going to be used most of the time. It takes a list of IPs, CIDR networks or domain names as targets and black list as well as a list of TCP and UDP ports to be scanned.

enabled: `true` 


Enables or disables this target generator.

targets: `[]` 


A list of targets to scan. May be (also a mix of) CIDR networks, IPs or domain names.



Example:

```
standard:
  enabled: true
  targets: ["192.168.1.1/24", "192.168.16.0/22", "somesystem.domain.local", "192.168.12.14"]
  tcpports: ["top25"]
  udpports: ["top25"]
  maxHostsPerBatch: 150
  maxTcpPortsPerBatch: 25
  maxUdpPortsPerBatch: 25
```

targetFile: `""` 

A file containing a list of targets to scan. May be CIDR networks, IPs or domain names, newline separated.

tcpports: `["top25"]` 

List of TCP ports to scan. Supports enumerations (`[20,21,22,23]` ) , ranges (`[8000-8100]` ) or nmap-style top-lists (e.g. `["top50"]` ) . You may also mix, e.g. `[20,21,22,23,8000-8100,"top50"]`  . Duplicates are filtered out ☺

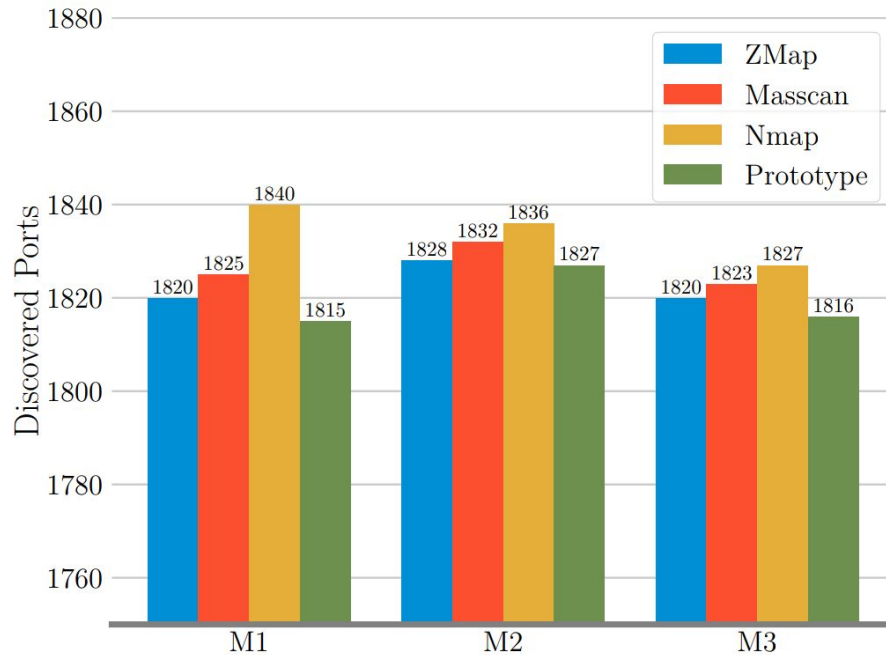


STORIES FROM THE FIELD

REAL WORLD SCANNING RESULTS AND COMPARISONS

- ✗ Scanning the network of the faculties Mathematics and Informatics from the Internet and from internal (/16 with exclusions, ~50.000 IPs in scope)
- ✗ TCP top 50
- ✗ Compare results with Nmap, ZMap and Masscan
- ✗ Closer look on real world certificate usage: internal vs. Internet

MEASUREMENTS: TOP 50 TCP AGAINST REAL /16



Times:

✗ ZMap*:	~7m22.848s
✗ Masscan:	~0m46.63s
✗ Nmap**:	~1169m23.497s
✗ nray ("Prototype"):	~96m031s

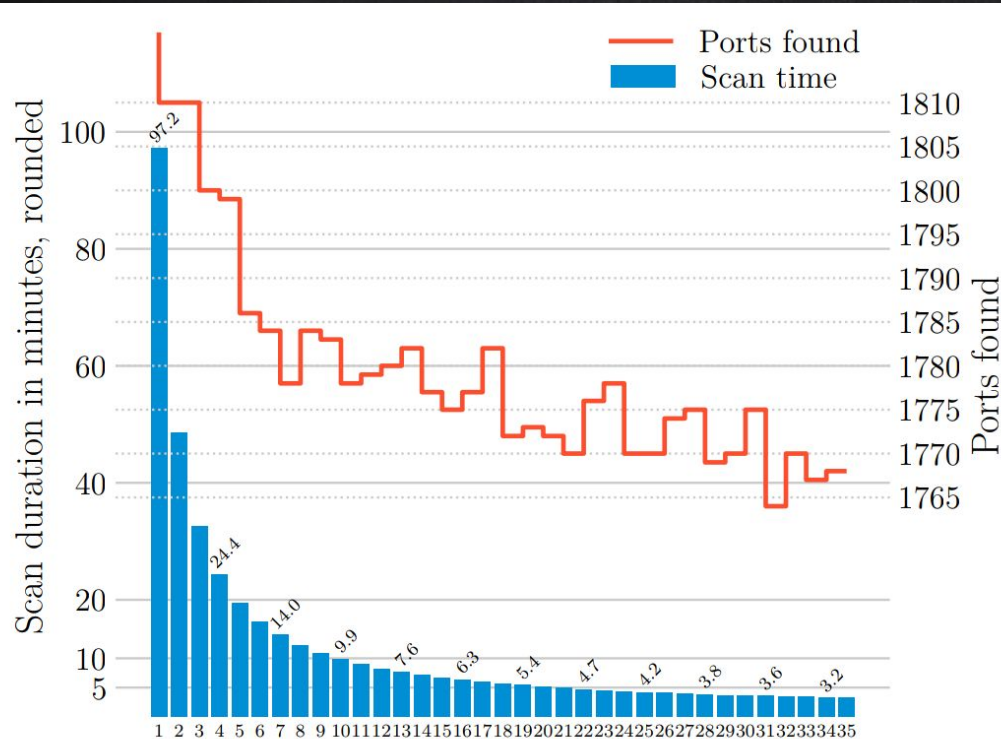
"Ports per minute":

✗ ZMap*:	246.95
✗ Masscan:	2350.66
✗ Nmap**:	1.57
✗ nray:	18.797

* ZMap: sequential execution for each of 50 tested ports

** Nmap: `nmap -v -T4 -Pn -n --max-retries 0 --randomize-hosts --excludefile ./excludes.txt --top-ports 50 -oA scan.nmap x.x.x.x/16`

MULTIPLE SCANNER NODES ON SAME SYSTEM



- ✗ On Linux: max 1024 FD per process (default)
- ✗ We need some spare (connection to server, randomness, etc.)
- ✗ Scanning is possible with ~1000 workers per process
- ✗ Idea: Spawn multiple processes on same machine

EXAMPLE EVALUATION: TAKE CLOSER LOOK AT TLS CERTIFICATES

	M1		M2		M3	
	int	ext	int	ext	int	ext
Unusual CAs	117	64	120	65	122	62
DFN / TUM / LRZ	279	236	274	236	280	242
Let's Encrypt	65	68	70	71	71	73
Total	461	368	464	372	473	377

	M1		M2		M3	
	int	ext	int	ext	int	ext
2006	1	0	1	0	1	0
2007	2	1	2	1	2	1
2008	0	0	0	0	0	0
2009	2	2	2	2	2	2
2010	1	1	1	1	1	1
2011	1	1	1	1	1	1
2012	2	1	1	0	2	1
2013	2	1	2	1	2	1
2014	3	2	3	2	3	2
2015	1	1	1	1	1	1
2016	19	4	18	4	18	3
2017	13	6	12	7	13	7
2018	14	11	12	10	12	10
2019	3	3	6	6	4	4
Total	64	34	62	36	62	34

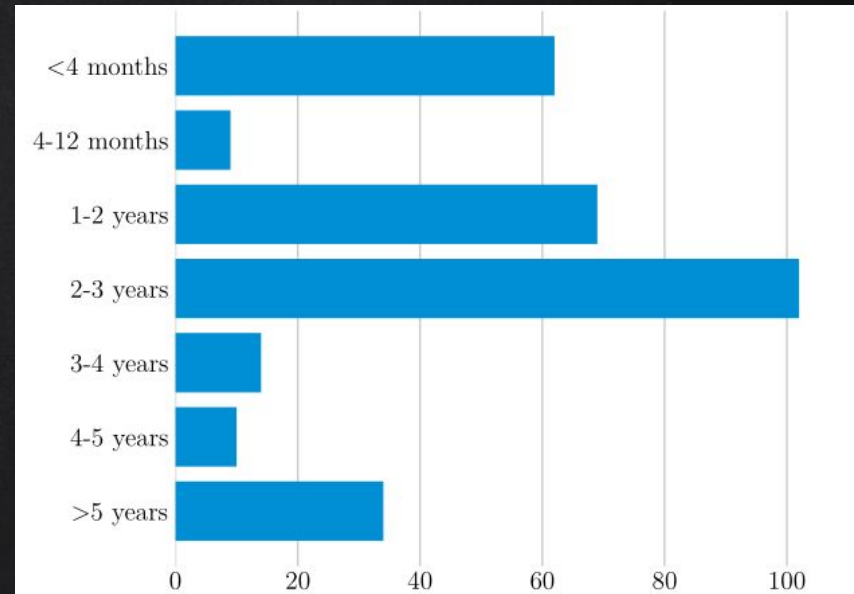
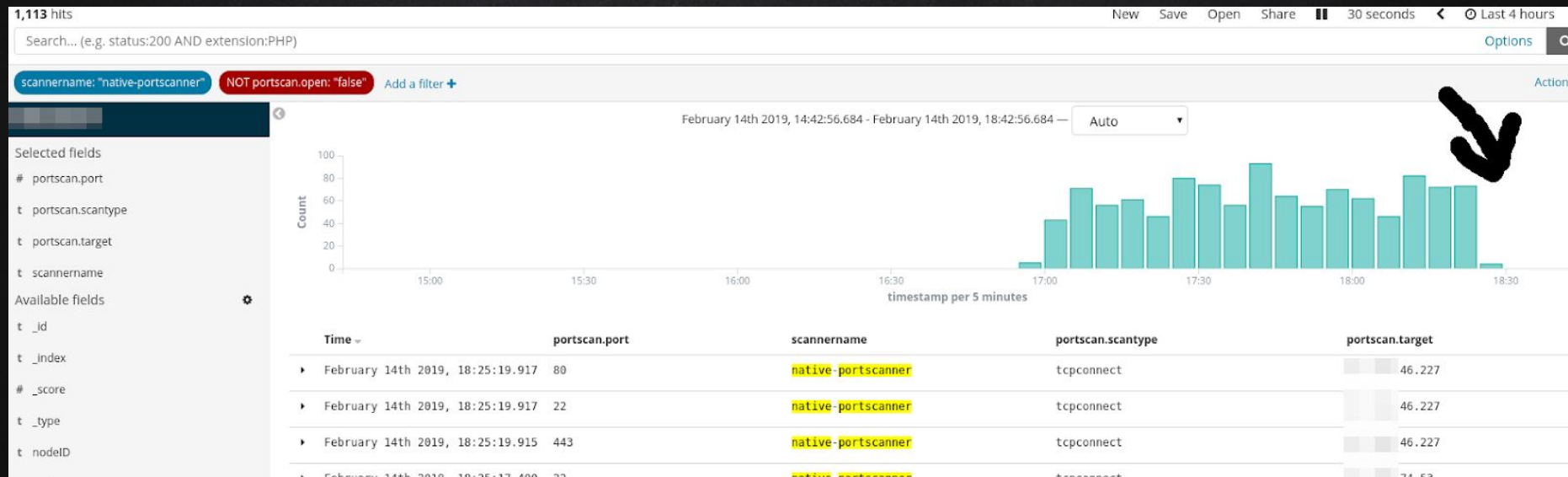


FIGURE 5.4: Lifetime of certificates observed in M1 from external.

GOT BLOCKED BY IDS?

NOTICED BY VISUAL INSPECTION OF EVENT STREAM



PENTEST: ATTEMPT TO FIND OTHER SERVICES IN OPENSIFT (RED HAT'S KUBERNETES) CLUSTER

OPENSIFT CONTAINER PLATFORM Application Console ▾

Test Test Test ▾


- Overview
- Applications** ▸
- Builds ▸
- Resources ▸
- Storage
- Monitoring
- Catalog

Deployments ▸ **debian-pentest-2** ▸ #3

debian-pentest-2-3 created 40 minutes ago

`nray-node` `nray-node` `openshift.io/deployment-config.name` `debian-pentest-2`


[Details](#) [Environment](#) [Metrics](#) [Logs](#) [Events](#)

Status:  Active

Deployment Config: [debian-pentest-2](#)


Status Reason: config change

Selectors: deployment=debian-pentest-2-3
deploymentconfig=debian-pentest-2
nray-node=nray-node

Replicas: 400 current / 400 desired 


400 pods


Template

 Container debian-pentest-2 does not have health checks to ensure your application is running correctly. [Add Health Checks](#)

Containers

debian-pentest-2

 Image: `902753a` 286.1 MiB

 Command: `/home/pentest/nray-injected-x64-linux`



AND NOW?

GRAB IT WHILE IT'S HOT!

- ✗ <https://nray-scanner.org>
- ✗ <https://github.com/nray-scanner/nray>
- ✗ Fork it, improve it, break it

USAGE IDEAS

- ✗ Bug bounties / shells: Monitor for certificates issued
 - these systems are often fresh and not hardened yet
 - weak/default passwords
 - open mail relays
 - installed from 4 month old iso and not patched yet
- ✗ Pentest: easily spot high value systems by
 - Creating network views from different positions
 - Deploying nodes on multiple systems for better speed and resilience
- ✗ Blue Team / network team:
 - Identify and monitor assets, spot rogue devices or newly opened ports 4444
 - Build your own Shodan (and answer patchday questions like “OMFG which systems have RDP exposed and who can reach them” within 2 minutes)
 - Big data, build dashboards, give others access
 - Track changes over time (and make for example progress visible)



THANKS!

Any questions?