A Study on Container Vulnerability Exploit Detection

Olufogorehan Tunde-Onadele, Jingzhu He, Ting Dai, Xiaohui Gu
NC State University

IC2E '19

Motivation

- Great deployment benefits
  - Portability
  - Consistency
  - Isolation

- Recent platform still facing security issues

Real-World Examples

Juniper Networks: Cryptomining Exploit Targeting Docker Containers

by Michael Vizard on November 15, 2018

Attackers Continue to Target Docker Vulnerabilities

Brief, Security

Oct 29, 2018 By Eric Carter, ProgrammableWeb Staff

DIRTY COW

Doomsday Docker security hole uncovered

A security vulnerability has been disclosed for a flaw in runc, Docker and Kubernetes' container runtime, which can be used to attack any host system running containers.

By Steven J. Vaughan-Nichols for Linux and Open Source | February 11, 2019 -- 18:53 GMT (10:53 PST) | Topic: Security

Detection Approaches

- 2 main categories of vulnerability detection
  - Static detection on container images
  - Dynamic detection on container instances

Static Detection Problems

Clair

- Container image scanning
- Depends on
  - Packages and versions
  - Vulnerability records

- Limitation
  - High false positive rate
  - Low detection rate

Dynamic Detection Challenges

1. Containers are often short-lived

2. Containers have dynamic available resources & workloads

3. Containers are lightweight

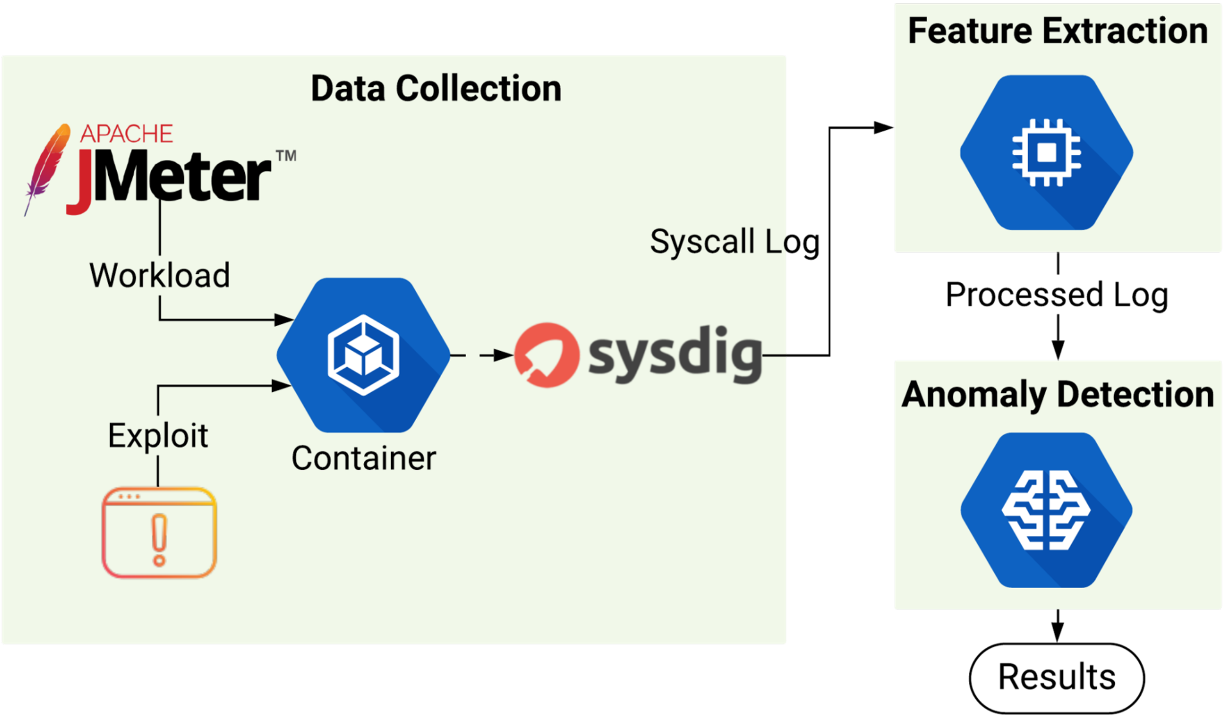
Dynamic Detection

- Anomaly detection on system calls

- Studied Algorithms
  - k-Nearest Neighbors (k-NN)
  - Principal Component Analysis (PCA) + k-Nearest Neighbors (k-NN)
  - k-Means
  - Self Organizing Map (SOM)

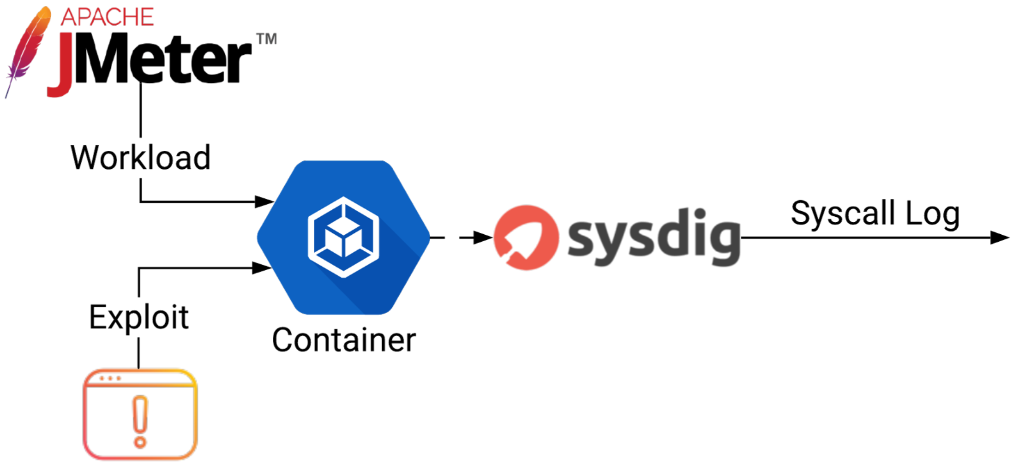
Experiment

- Setup



Data Collection

- Deliver workload for a particular application
- Trigger the exploit at appropriate time
- Collect a detailed log of the system call information



Data Processing

- Frequency vector
  - System call occurences per sample

Timestamp	System Calls			
	write	read	futex	epoll_wait
1516544689186	100	256	430	78
1516544689286	300	759	726	356

- Time vector
  - System call runtime per sample

Case Study: ActiveMQ (CVE-2016-3088)

- File write vulnerability
  - Allows upload and execution of arbitrary files
  - Achieved using an HTTP PUT followed by an HTTP MOVE request

- Exploit requires
  - Knowledge of web directory
  - ActiveMQ run as root

Case Study: ActiveMQ (CVE-2016-3088)

- Exploit (Vulhub, 2017)
  - Set up a waiting shell
  - HTTP PUT payload with crontab commands
  - HTTP MOVE to a crontab location
  - Shell returned

1. PUT /fileserver/1.txt HTTP/1.1
2. Host: localhost:port#

   …

7. Content-Length: 247
8. { *crontab command to initiate socket connection to shell* }

1. **MOVE /fileserver/1.txt HTTP/1.1**
2. Destination: file:///etc/cron.d/root
3. Host: localhost:port#

   …

8. Content-Length: 0

Case Study: ActiveMQ (CVE-2016-3088)

- Sysdig log snippet

```
10:14:04.999140525 3 java (10306) > switch next=0 pgft_maj=0 pgft_min=33...
10:14:05.049191227 3 java (10306) < futex res=-110(ETIMEDOUT)
10:14:05.049194706 3 java (10306) > futex addr=7F20F0236928 op=129... val=1
10:14:05.049195721 3 java (10306) < futex res=0
10:14:05.049202973 3 java (10306) > futex addr=7F20F0236954 op=137... val=1
...
10:14:05.089969920 3 java (10340) > getsockname
10:14:05.089971658 3 java (10340) < getsockname
10:14:05.089976977 3 java (10340) > getsockname
10:14:05.089978207 3 java (10340) < getsockname
...
10:14:05.099234302 3 java (10306) < futex res=-110(ETIMEDOUT)
10:14:05.099237402 3 java (10306) > futex addr=7F20F0236928 op=129...
10:14:05.099238268 3 java (10306) < futex res=0
10:14:05.099244128 3 java (10306) > futex addr=7F20F0236954 op=137...
10:14:05.099250022 3 java (10306) > switch next=0 pgft_maj=0 pgft_min=33...
10:14:05.128901808 2 java (10346) > write fd=137(<p>pipe:[1141873]) size=1
```

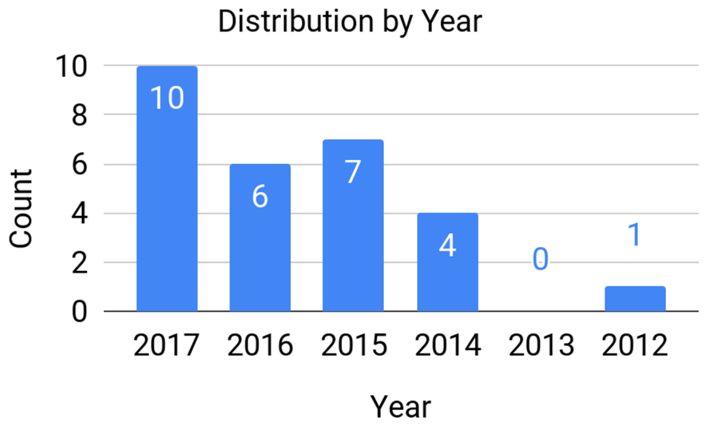
Case Study: ActiveMQ (CVE-2016-3088)

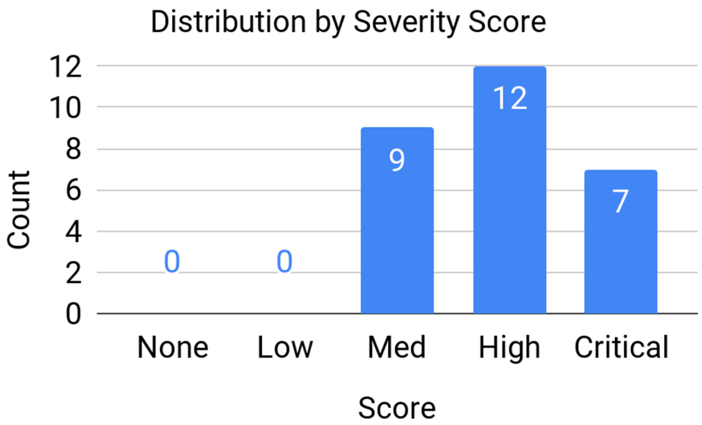
- Processed log snippet

timestamp	read	futex	accept	fcntl	getsockname
1528884844803	0	4	0	0	0
1528884844903	0	4	0	0	0
1528884845003	0	4	1	3	2
1528884845103	258	599	0	200	0
1528884845203	531	1542	0	436	0

Studied Vulnerabilities

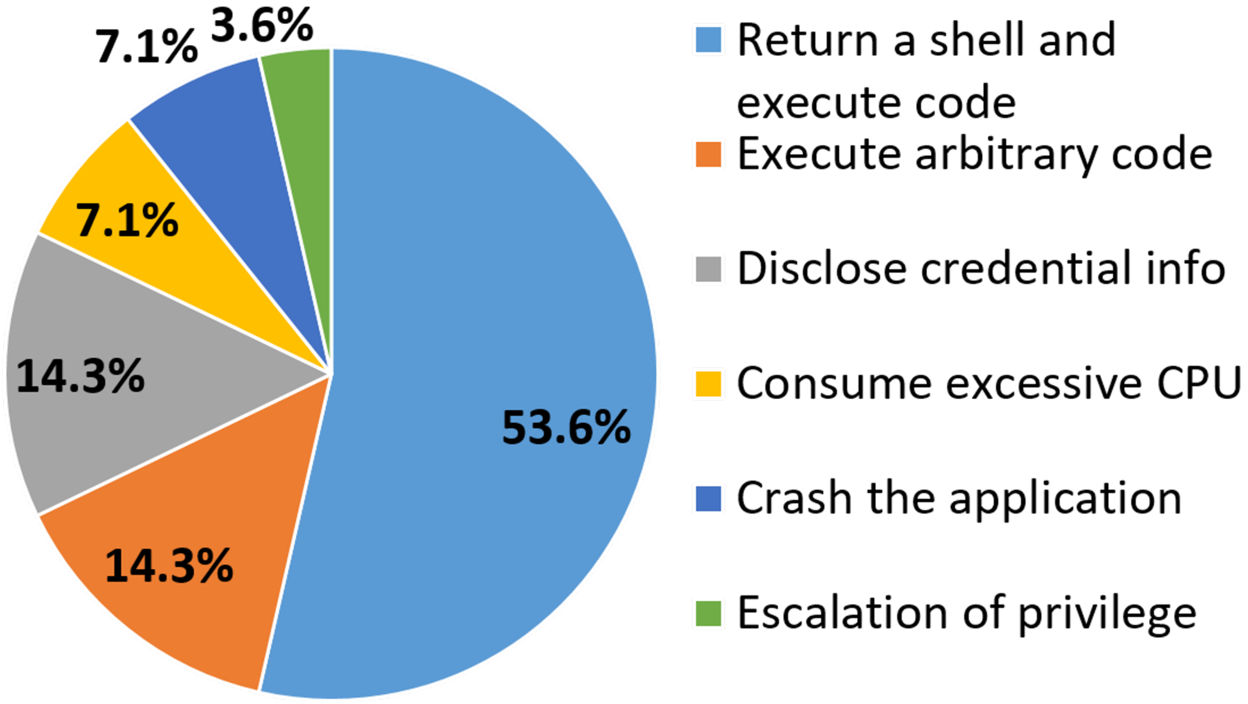
- 28 recent vulnerabilities of moderate to high severity
- Variety of applications
  - Web, file services





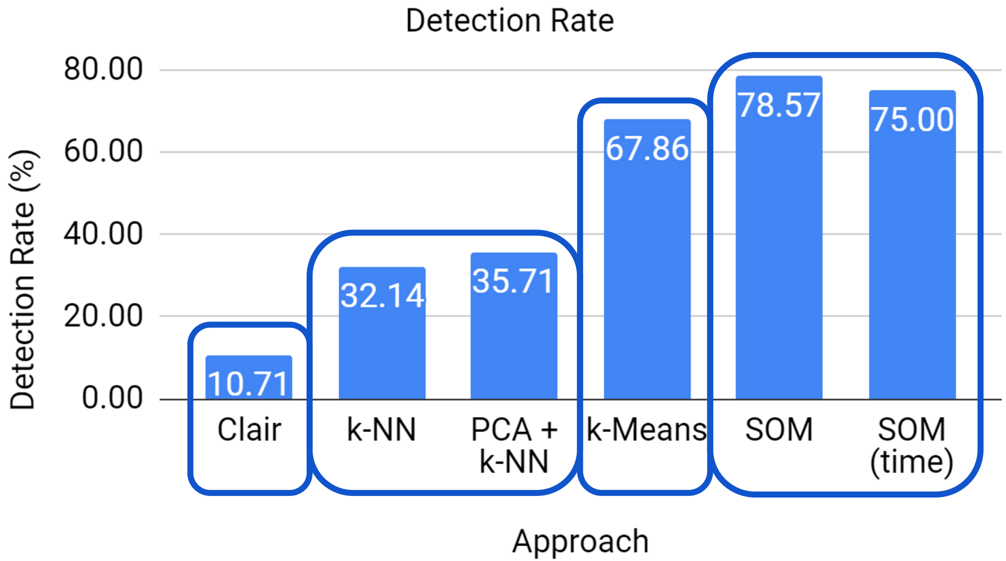
Studied Vulnerabilities

- Vulnerability categories



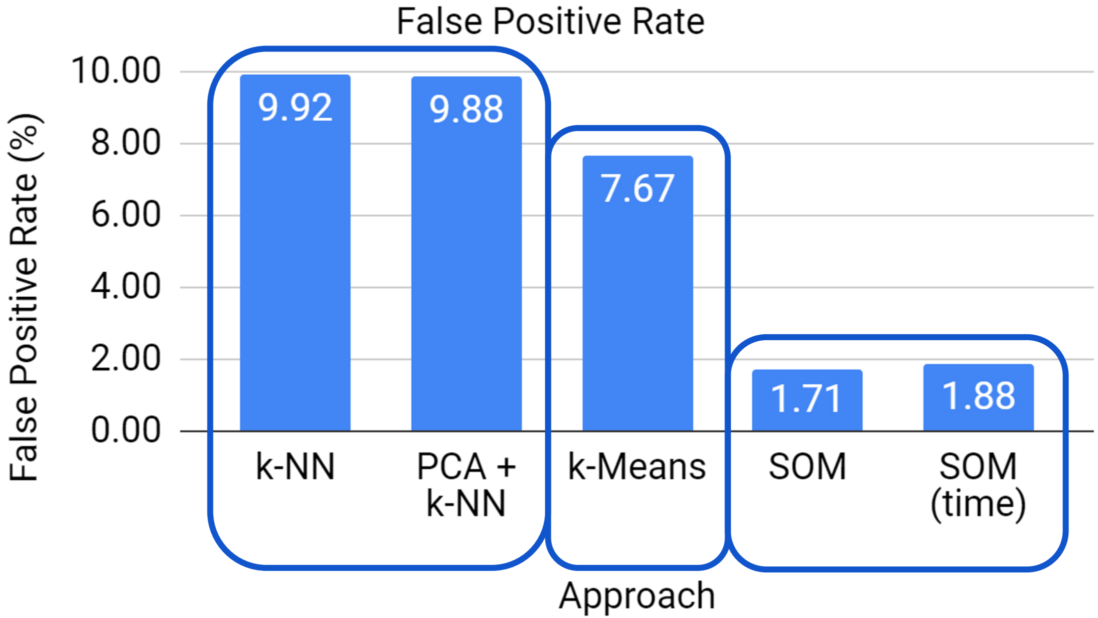
Results - Detection Rate

- Percentage of attacks in which an alarm is raised during exploitation



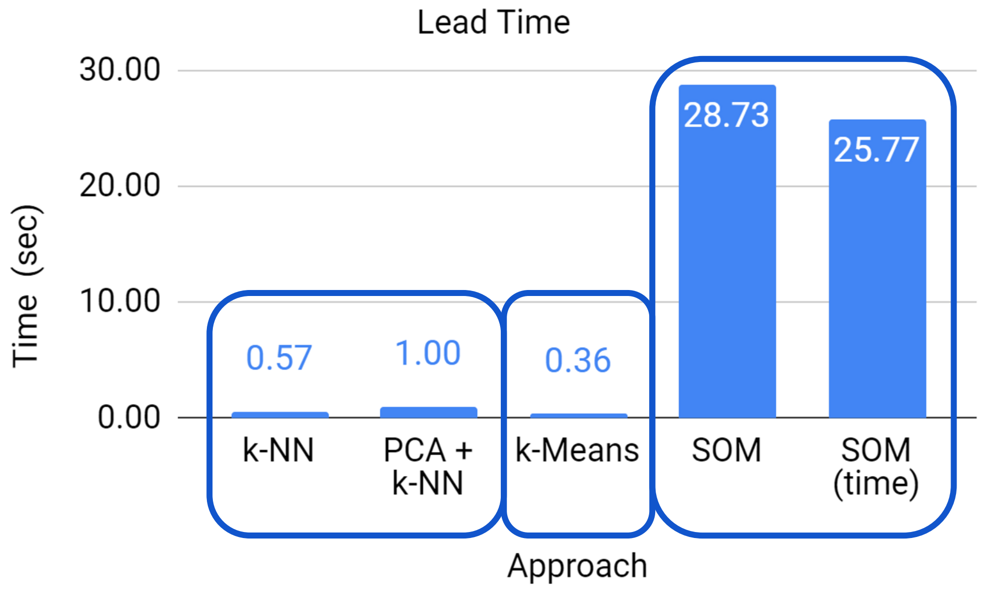
Results - False Positive Rate (FPR)

- False alarm rate



Results - Lead Time

- Time between the alarm notice and attack completion
- For the category of exploits that return a shell



Results - Summary

- Self Organizing Map (SOM) shows the most promising results
  - Detection rate
  - False positive rate
  - Lead time
- Detection over frequency vectors yields improved results over that of time vectors

Future Work

- Investigate more vulnerability case studies
- Improve detection accuracy of the studied schemes

Conclusion

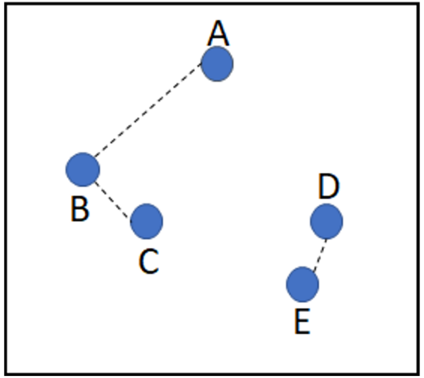
- Studied 28 real world vulnerabilities in 24 common containerized applications
- 24 of 28 vulnerability exploits detected (**85.7%**)
  - Static alone detects 3 of 28 exploits (**10.7%**)
  - Dynamic alone detects 22 of 28 exploits (**78.6%**)

Thank you!

Backup slides

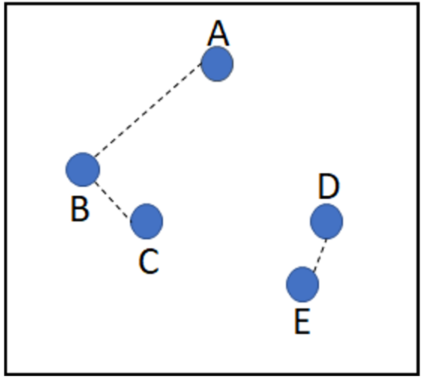
k-Nearest Neighbors

- *Principal Component Analysis (PCA) + k-Nearest Neighbors (k-NN)*
- Example
  - 1-nearest neighbor
  - Largest 20% of nearest neighbor distance



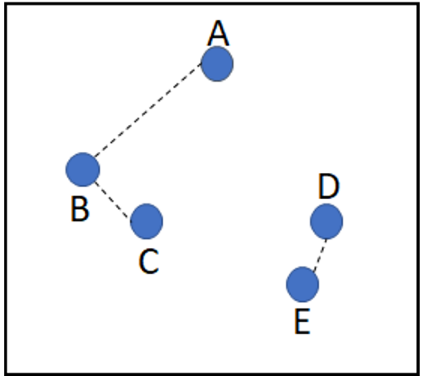
k-Nearest Neighbors

- *Principal Component Analysis (PCA) + k-Nearest Neighbors (k-NN)*
- Example
  - 1-nearest neighbor
  - Largest 20% of nearest neighbor distance



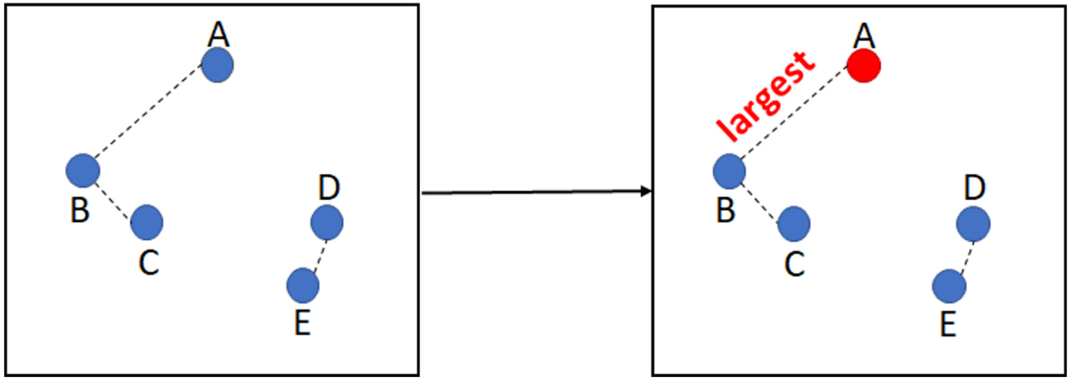
k-Nearest Neighbors

- *Principal Component Analysis (PCA) + k-Nearest Neighbors (k-NN)*
- Example
  - 1-nearest neighbor
  - Largest 20% of nearest neighbor distance



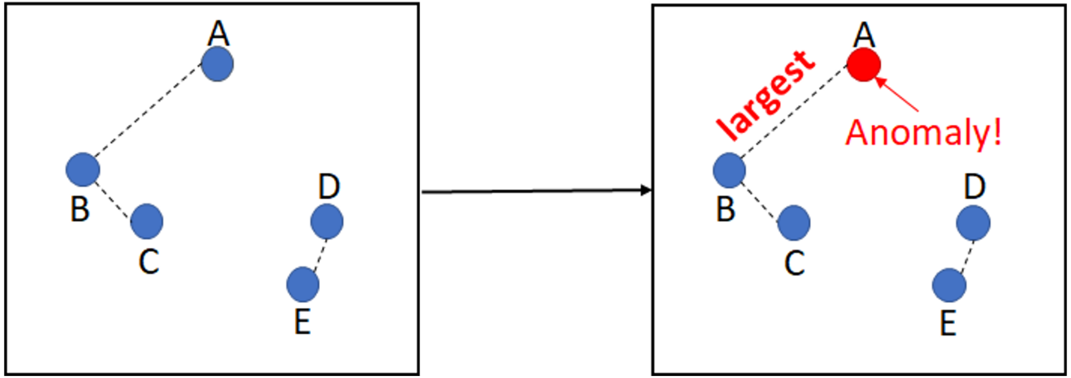
k-Nearest Neighbors

- *Principal Component Analysis (PCA) + k-Nearest Neighbors (k-NN)*
- Example
  - 1-nearest neighbor
  - Largest 20% of nearest neighbor distance



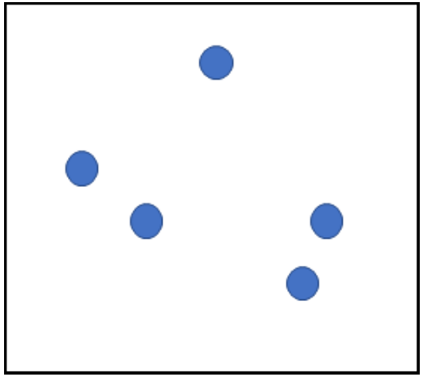
k-Nearest Neighbors

- *Principal Component Analysis (PCA) + k-Nearest Neighbors (k-NN)*
- Example
  - 1-nearest neighbor
  - Largest 20% of nearest neighbor distance



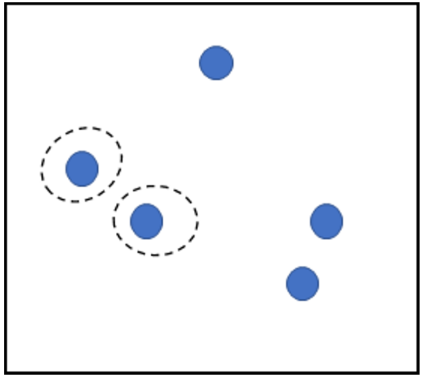
k-Means

- *k-Means*
- Example
  - 2 clusters



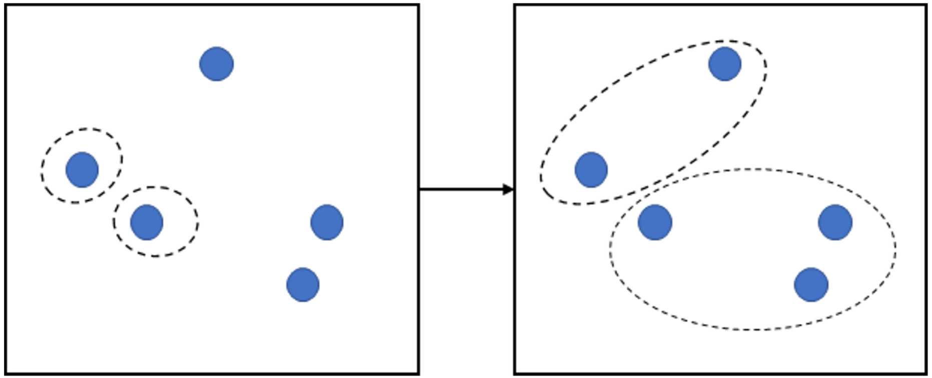
k-Means

- *k-Means*
- Example
  - 2 clusters



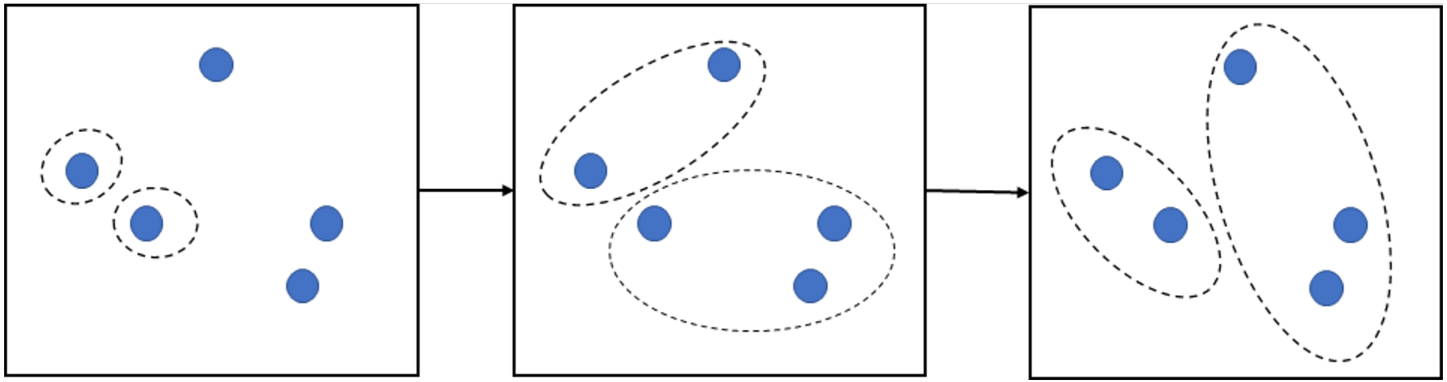
k-Means

- *k-Means*
- Example
  - 2 clusters



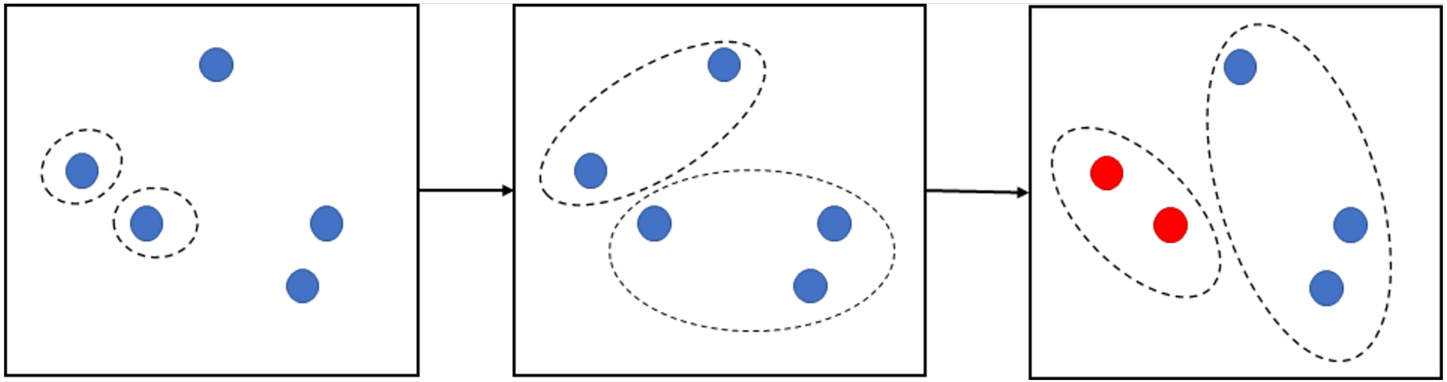
k-Means

- *k-Means*
- Example
  - 2 clusters
  - Cluster size threshold of 2



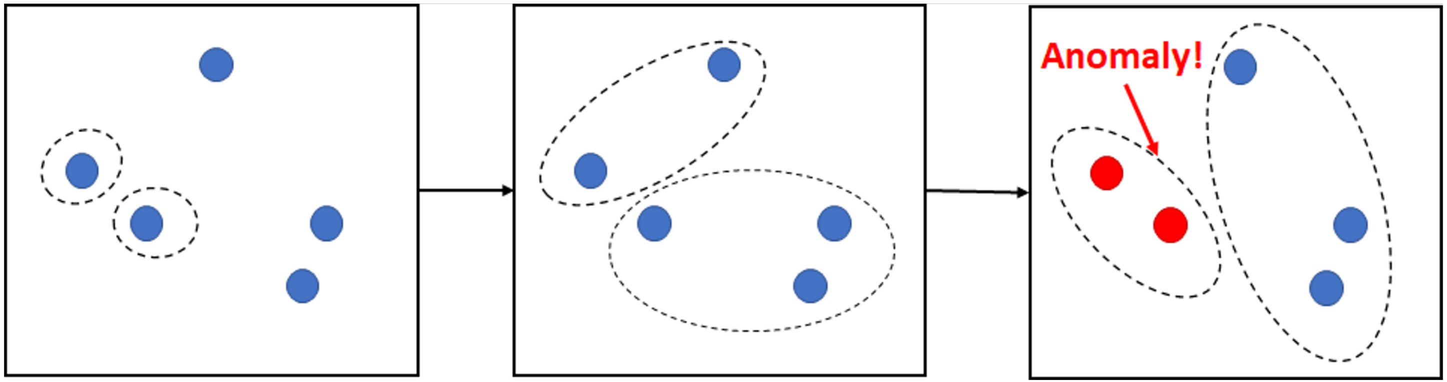
k-Means

- *k-Means*
- Example
  - 2 clusters
  - Cluster size threshold of 2



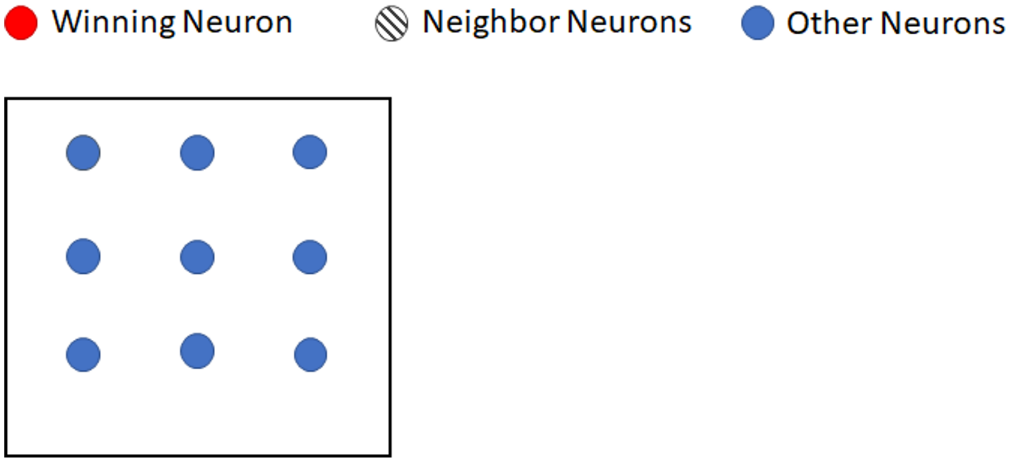
k-Means

- *k-Means*
- Example
  - 2 clusters
  - Cluster size threshold of 2



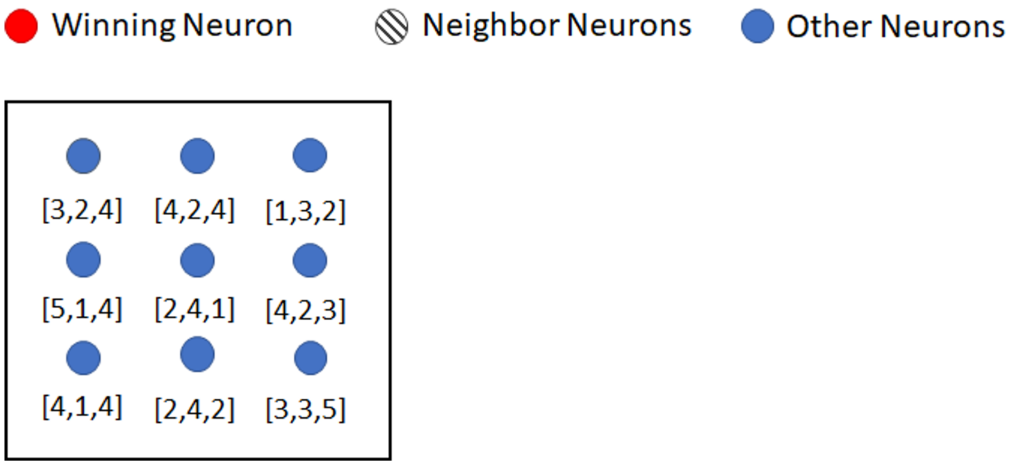
Self Organizing Map

- *Self Organizing Map (SOM) learning phase*
- Example
  - $W(t + 1) = W(t) + N(t)L(t)(V(t) - W(t))$
  - $W(t + 1) = W(t) + (1)(0.5)(V(t) - W(t))$



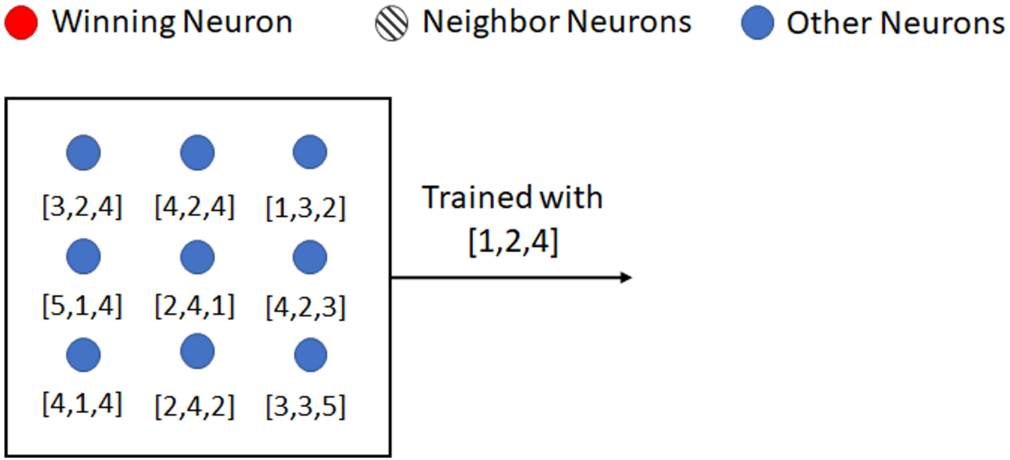
Self Organizing Map

- *Self Organizing Map (SOM) learning phase*
- Example
  - $W(t + 1) = W(t) + (1)(0.5)(V(t) - W(t))$
  - $W(t + 1) = W(t) + (0.5)(V(t) - W(t))$



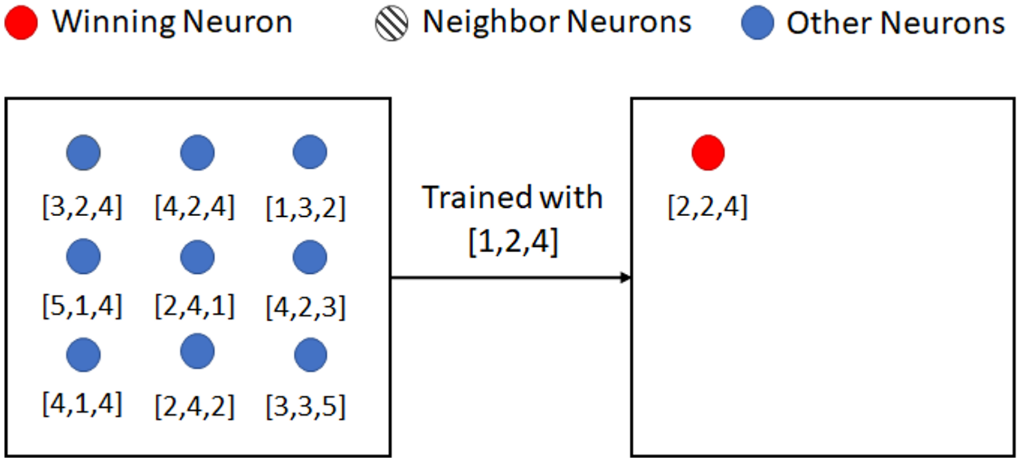
Self Organizing Map

- *Self Organizing Map (SOM) learning phase*
- Example
  - W(t + 1) = W(t) + (0.5)(V(t) - W(t))
  - [1, 2, 4] closest to [3, 2, 4]



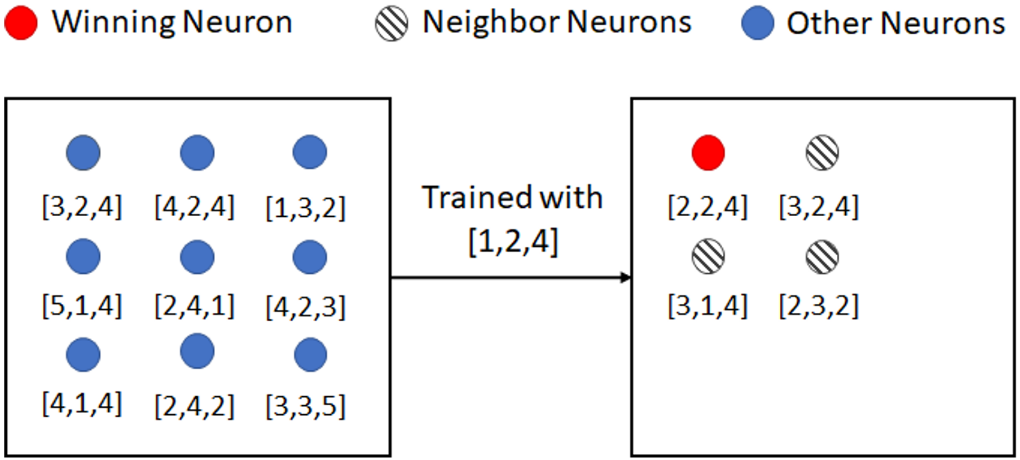
Self Organizing Map

- *Self Organizing Map (SOM) learning phase*
- Example
  - $W(t + 1) = W(t) + (0.5)(V(t) - W(t))$
  - $W(t + 1) = [3, 2, 4] + (0.5)[-2, 0, 0] = [2, 2, 4]$



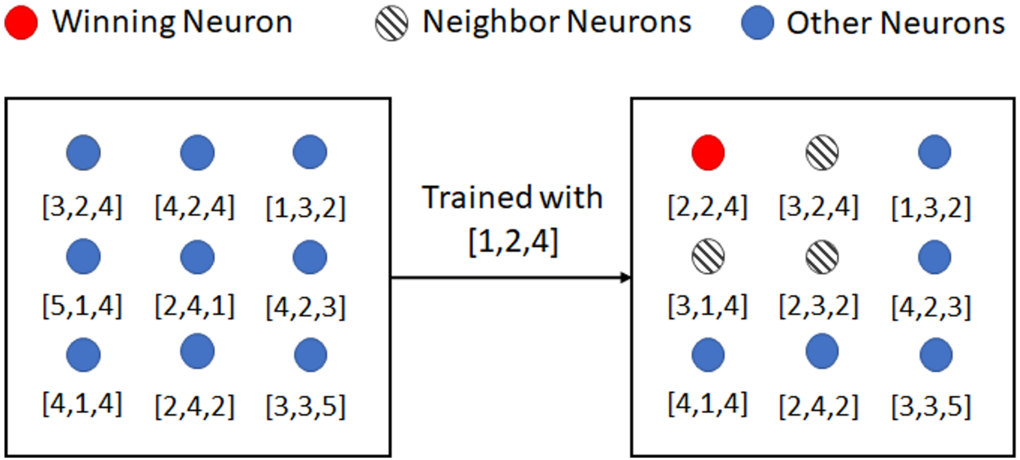
Self Organizing Map

- *Self Organizing Map (SOM) learning phase*
- Example
  - $W(t + 1) = W(t) + N(t)L(t)(V(t) - W(t))$
  - $W(t + 1) = W(t) + (1)(0.5)(V(t) - W(t))$



Self Organizing Map

- *Self Organizing Map (SOM) learning phase*
- Example
  - W(t + 1) = W(t) + N(t)L(t)(V(t) - W(t))
  - W(t + 1) = W(t) + (1)(0.5)(V(t) - W(t))



Self Organizing Map

- *Self Organizing Map (SOM) sample mapping*
- For each test sample
  - Sum distances between the winning neuron and its neighbors (neighborhood area size)
  - Anomalies are in the largest percentile of sizes