I J C T A, 9(16), 2016, pp. 8275-8285

Solving security issues in Docker using Stackelberg Games

Nimisha Sharath*, Vishal Kumar* and K. Chandrasekaran*

ABSTRACT

Container technology has taken the world by storm and is replacing virtual machines rapidly. Docker is an open source tool that has implemented containers based on the linux ecosystem. Being lightweight and scalable, Docker has made data scaling very simple. Despite the obvious success of Dockerized applications, certain faults hold it back when it comes to usage in organizations that work with distributed teams. From a security standpoint, it is essential to be aware of scenarios that could cause the downfall of a company's entire network ecosystem with a single command. The study begins with a deep dive into Docker's code base and stress tests that have unearthed some more security issues. Taking into account one such issue related to networking in a multi-container environment gives rise to play on the trade off between efficiency of resource usage in a container, to its security. Using the backward induction technique of a stackelberg competition model, the efficiency of the distributed system can be made a function of its security and functionality. Solutions for both the scenarios where the attacker's type is known and unknown have been proposed using game theoretic approaches. A comparison on Machine Learning and Linear Programming based approaches give rise to the most optimal method for adopting a defense strategy in case of attacks in such distributed systems.

1. INTRODUCTION

Docker is a new and upcoming technology that is becoming a rage in the technology world. Based on the Linux container technology, this is poised to revolutionize the way applications are developed. It makes use of the new containers technology to create isolated workspaces, similar to what a VM does, but without the need of a guest operating system. Your only need is Linux kernel and Docker engine.

When you are using a Virtual Machine, you have a Physical Machine and a hypervisor, which is similar to Docker, a physical machine and a Host Linux OS. Changes can be seen above these layers. VM based environment has a very heavy guest OS for every instance that VM is running. This gives a very large memory overhead compared to a Docker based environment in which you have a common Docker engine which hosts many different containers which use the same kernel of the host. Over the past few years, data storage in huge organizations has undergone changes right from moving data to the cloud, to virtualization and now to containers. With the advantage that containers have over VMs, they are rapidly replacing VMs throughout. Docker, an organization that has built this container technology using Go language has become the most container technology till date. Being open source, with the number of contributors to its code base increasing day by day, Docker is a constantly improving software.

Despite its successes in the field of data storage, it's usage conditions in industry could create scenarios that undermine the security of the data being stored within the containers. In this paper, we have deep dived into one such scenario, that involves multiple containers on a network. In earlier articles and papers about Docker, some issues that had been brought up were:

* National Institute of Technology Karnataka Surathkal, with Bachelor of Technology from Department of Computer Science and Engineering

Large attack surface of the docker daemon,

Unprivileged access to devices connected on host systems, via containers,

Using SSH to enter into containers, etc.

The issue demonstrated in this study is a result of dabbling of certain network settings inside a container. The solution to improving security in a software is to usually find the attack vector, attack target and increase level of security correspondingly. The issue with increasing security levels in this scenario would be, losing a certain amount of functionality that Docker intends on providing. This is obviously a situation that requires a trade off between functionality and security of the software..Using this concept of game theory, the study uses linear programming to provide an optimal solution to the de-fender, in terms of which containers(targets) to secure fully and which containers to allow full functionality. This combination will ensure the highest payoff/utility for the defender, given a certain attack scenario. This section also deals with providing solutions when the nature of the attacker is unknown and has to be determined without wasting any stages of the game. Section V provides details of the simulation of the proposed approach and the results. To validate this solution, we have compared how our approach using linear programming will provide a more efficient solution when compared to machine learning methods.

Section II gives a basic preview of networking in Docker and Section III explains networking in Docker. Section IV explains the practical issue about how the attack works in detail. Section V mathematically models this scenario to be a stackelberg game, with 2 players This has been explained in section VI. The last few sections of the paper are the conclusions drawn from this study and references used for the formulation of the paper.

2. RELATED WORKS

Docker containers are basically LXC containers[9], and they have the same security features. Extra layer of security is possible by enabling Apparmour or SELinux. To expand Docker security and adaptability, Enrico Bacis et al[1] pro-posed an expansion to the Dockerfile format. Image maintain-ers will be able to ship a particular SELinux policy for the procedures that keep running in a Docker image, upgrading the security of containers.

Docker’s systems are at present configured to give the “best effort” to all the traffic; and parameters, for example, transfer speed, dependability, and packets every second for a particular application can’t be ensured. Consequently a single bandwidth-intensive application results in poor or unsatisfactory execution for some other application sharing the Docker system. This problem was solved by A. Dusia, Yang Yang and M. Taufer [2], they introduced quality of service (QoS) mechanisms that provide preferential treatment to traffic and applications.

The Docker group has recently put genuine efforts into enhancing its security. This incorporates investigating how mandatory access control can be implemented using Apparmor policies and SELinux, system call filters such as seccomp [4] for contained applications [5]. These endeavors ensure a fundamental platform (and running different containers on it) if, for example, a containerized application is traded off by exploiting an obscure vulnerability. Likewise, a more com-pelled environment has been presented for a few operations, for example, docker pull [6]. Image signing and verification has been recently incorporated in Docker which ensures that image comes from the right maintainer and is has not altered with anything[7]. These functionality minimizes the external threats.

M.Zhang, D.Marino made a framework (Harbormaster)[10] which implements fine grained policies on container man-agement and protects container hosts. It makes proxy for all commands which are destined for the Docker daemon and then evaluates them according to the policies before passing it along.

Avrim Blum, Nika Haghtalab, Ariel D. Procaccia [11], in their paper have spoken of playing stackelberg games which model attacks in terms of security purposes. The above references have dealt with security issues in Docker and have provided solutions for the same.

In our study,we have not only unearthed an issue that has not been discussed earlier, but also provided an efficient game theoretic solution to adopt a security strategy to provide optimal protection to the data stored in the containers.

3. BACKGROUND TO NETWORKING IN DOCKER

This section explains the basics of networking in docker, following which the attack scenario will be discussed.When Docker is installed, it creates 3 default networks: bridge, none and host. The bridge network represents the docker0 network present in all Docker installations. The none network adds a container to a container-specific network stack. That container lacks a network interface. The host network adds a container on the hosts network stack. Youll find the network configuration inside the container is identical to the host. The easiest network to create is a bridge network. The containers you launch into this network must reside on the same Docker host. Each container in the network can immediately communicate with other containers in the network. Though, the network itself isolates the containers from external networks.

Iptables are present in most linux distributions as default firewall. It is a front-end solution to modifying the network stack of the linux system. The working of the iptable is ac-cording to matching of the packets moving across the network, according to the rules specified in the iptable directory.

Since Docker is a container technology based on linux host systems, all containers have access to iptables.

The rules present in these iptables will match the protocol type of the packet, the port's source or destination address, inter packet relation, etc.

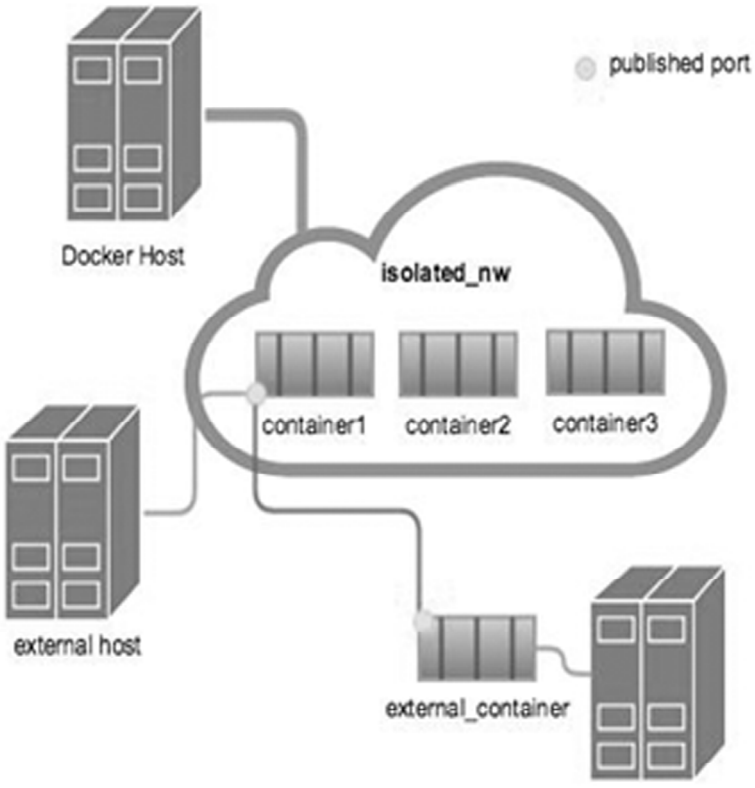

Figure 1: Docker Network[8]

Going with figure 1, within a user-defined bridge network, it can be seen how connections can be made. You can expose and publish container ports on containers in this network. This is useful if you want to make a portion of the bridge network available to an outside network.

Because another machine on the local network of the host can now forward packets through this machine to the outside world and also to the containers. There are no iptable rules in place to stop this.

4. A PRACTICAL MULTI-CONTAINER ISSUE

One of the issues one faces while dabbling with the network settings in Docker is that, it is possible to create a situation where a container can be exposed to the world as a medium to connect to other containers without even root access. In such situations, this container can result in allowing malicious users accessing other containers and using or modifying the data present there. Using Docker Swarm on a Dell Inspiron 15 hardware platform, 20 containers with a few text files with read write activated rights were created. One of the containers were made 'transparent', or through which other containers could be accessed. All the other 19 containers connected were accessible and text files could be modified. We figured that one way to work around this would be to modify IPtable rules on the 'transparent' container in order to put restrictions on its behaviour. As predicted, this results in issues with usage of a number of features. All the published ports in the container do not work and additional connections cannot be made which defeats the purpose of creating such an environment in order to scale and share data effectively.

5. MATHEMATICAL MODELLING OF ATTACK SCENARIO

5.1. Game Theory and Stackelberg Games

The scenario described above can be considered a conflict between keeping containers secure and achieving full function-ality in terms of networking with other containers. On careful observation, it can be modelled into a game with rational players. The players of the game would be the defender, or the person designing the operations for a company who wants to keep the environment safe while the attacker will be the malicious user who wants to feed on the data via the transparent container. The defender has to decide on whether to

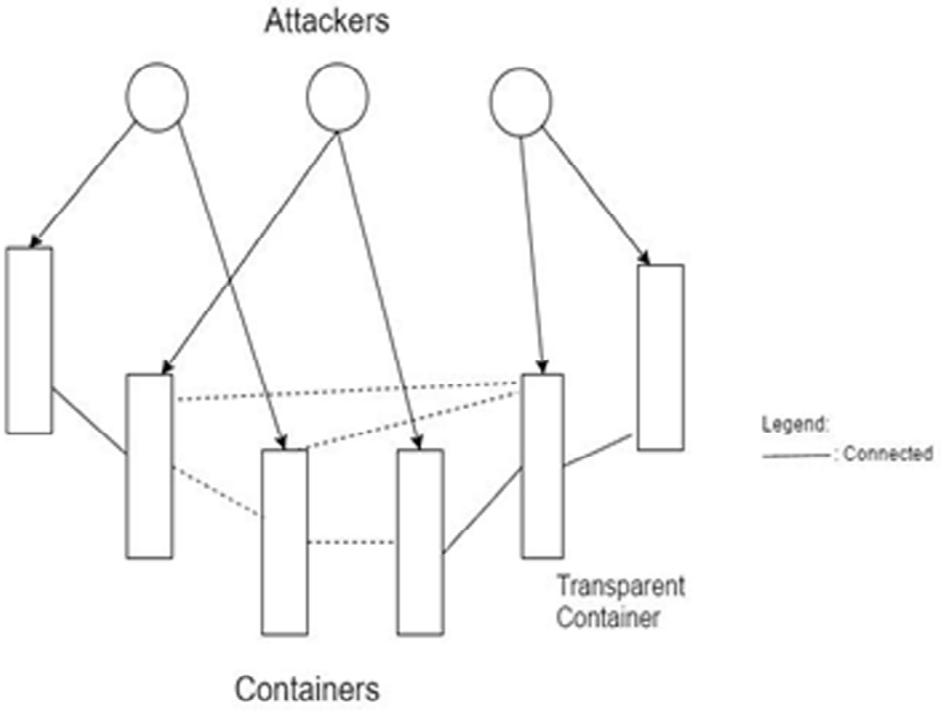

Figure 2: Multi-container attack scenario

'defend' a particular container or allow full access and retain full functionality. By 'defend', it is implied that the published ports are inaccessible.

Before the optimal solution to this scenario is derived, it is essential to understand this class of games, that deal with security, or commonly called 'Stackelberg Games'.

In a Stackelberg Game, player one commits to a strategy, upon which player two will selfishly optimize to get his/her reward. Practically, malicious users who attack organizations for their data carefully plan their actions to bypass the security measures taken by the organization. This puts the stackelberg system into perspective in our scenario.

Assuming that the defender is the leader with respect to adopting a security strategy first, the attacker will be the fol-lower in terms of bypassing the measures taken by the leader. Thus, the game that will be played is not a simultaneous, but a sequential one where the follower has complete information of the leader's initial strategy. It would appear that this puts the follower(attacker) at an advantage, since he/she will react to the leader's(defender's) actions, but it is not so. Stackelberg games ensure that the first player has an advantage. The entire study of the paper can be seen in stages in figure 3.

This can be explained with the 'backward induction'. Using backward induction, the follower's best response is taken as a function of the leader's strategy.Thus, given the follower's best response, the leader chooses his/her strategy. The equilibrium that is obtained now, is called the Stackelberg-Nash Equilibrium. Thus, working backwards, the defender is at an advantage. The following section illustrates the above with a simple example.

5.2. Analysis of Backward Induction

The game being considered has 2 players, both decide how much to produce. The product of the number of units each player produces and price of the unit will determine overall cost. The number of units produced decides the outcome. Too many in production will increase costs while too little will cause no profit at all.

At each stage, the table is checked and the leader has to pick an amount to produce. Following this, the next player will select the amount he/she has to produce. The profits of both the players is shown in the table.

Table 1
Example of a 2 firm Stackelberg Game

	Follower Ouput = 7	*Follower Ouput = 10*
Leader Output = 6	66,67	48,80
Leader Output = 9	72,56	45,50

Some notations for calculation of payoff:

qL: Quantity that leader will produce.

qF: Quantity that follower will produce.

P= 24- qF- qL: The unit price.

P_L: Leader's profit.

P_L: Follower's profit.

Subgame Perfect Equilibrium: The leader will play with 9. If he goes with 6, then follower will go with 10. If leader goes with 9, follower will go with 7.

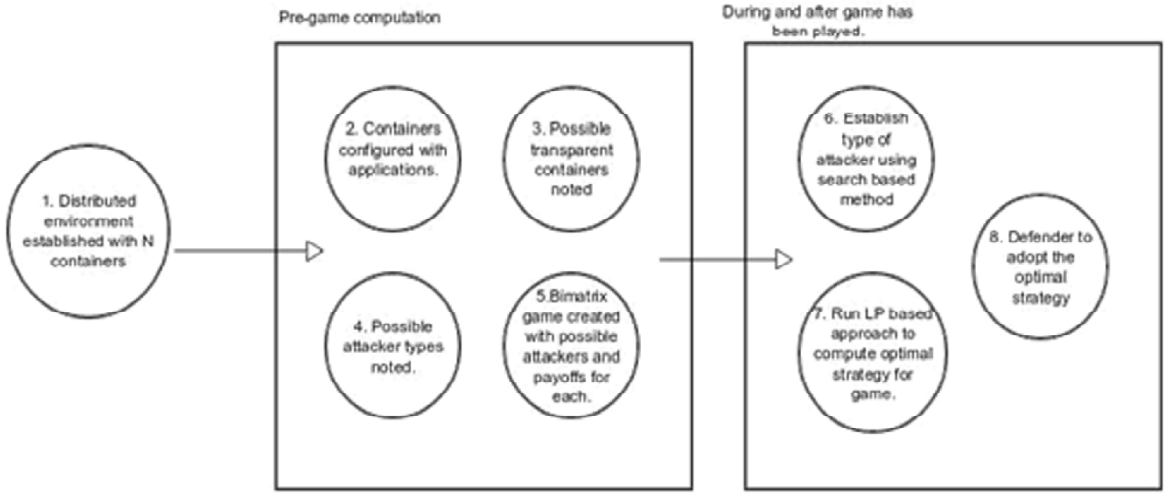

Figure 3: Step by step approach to computing Defender's optimal strategy

If all that the players want to do is maximize the amount of money they have, then they would analyze the situation as follows. Examining the game tree given as one of the figures, the output choice of 6 units would leave the Leader with 48 points whereas a choice of 9 units would result in Leader's own payoff of 72 points. Liking 72 points more than 48 points Leader chooses to produce 9 units. This is the idea behind backward induction reasoning.

5.3. Multi-Container Stackelberg Game

The above game explained the concept of best response to a game as well as backward induction. Now, let us consider a scenario where there are 2 containers, which are likely to get attacked. As explained in previous sections, the defender can choose to make the container secure or choose to leave it as it is. By doing the former, the container is secure but functionality is reduced and by doing the latter the functionality remains unchanged, but container can get attacked. For the sake of simplicity in this example, the functionality hasn't been taken into consideration while computing payoffs. For diversity, we consider 2 types of attackers. $\hat{o}_1$ and $\hat{o}_2$.

Table 2
An example of a security game with one resource and two targets, with two possible utilities for the attacker.

i	$u_d(.)$	$u_u(.)$	$u_a(.)\ \hat{o}_1$	$u_a(.)\ \hat{o}_1$	$u_a^c(.)\ \hat{o}_2$	$u_a^c(.)\ \hat{o}_2$
1	0	-1	0	1	0	1
2	0	-1	0	1	0	0

As described in [10]: $u_c^d(i)$: Defender's payoff when attacker hits covered target i

$u_c^u(i)$: Defender's payoff when attacker hits uncovered target i

$u_a^c(i)$: Attacker's payoff when attacker hits covered target i

$u_a^u(i)$: Attacker's payoff when attacker hits uncovered target i

$$U_d(i, p) = u_c^d(i)p_i + u_d^u(i)\ (1-p_i) \quad (1)$$

The above uses coverage probability as p and gives expected utility for the defender

$$U_a(i, p) = u_a^c(i)p_i + u_a^u(i)\ (1-p_i) \quad (2)$$

The above uses coverage probability as p and gives expected utility for the attacker. Note that U(i, p) is a linear function in p_i . Since U(i, p) only depends on Using U(i, p_i) during the remaining parts of the paper,

as U is a linear function in p. Every strategy s, adopted by the defender will be adopted with a probability p. The attacker hence chooses a target b(s) such that, b(s) belongs to $argmax_{i \in N}$ Ua(i,p). This approach used by the attacker doesn't depend on the mixed strategy adopted and hence doesn't depend on the probability p either.

When the attacker's utility or rather, attacker's type is not known, finding the optimal strategy for the leader is not possible with zero error. Say there is a situation with 2 players such that the defender's payoff for having been attacked at either of the unprotected targets is -1. If protected, then 0. Consider two possible utilities for the attacker. In the first case (type 1), the attacker values both targets equally, with payoff of 1 for attacking a target that is not protected and 0 for attacking a target that is protected. In the second case (type 2), the attackers utility for target 1 is the same as in case 1, however, there is 0 payoff for attacking target 2 whether or not it is protected (See Figure 1). The attackers' type is not known beforehand.

Say the security strategy is averaged over 3 rounds and the probability vector for having covered targets 1 and 2 are (2/3,1/3).

An attacker of type 1 would respond to this by attacking target 2, whereas an attacker of type 2 would respond to p by attacking target 1. So, by observing the attackers response to p, we can detect the type of the attacker in play and use the optimal strategy for that attacker type.

5.4. Search based solution to finding Stackelberg Equilibrium in cases where attacker's strategy is known

In the above formulation, we chose to maximize the utility of the defender by manually calculating the probability vector of attacks and used backward induction. In cases where the number of targets and attackers can cause the multitude of combinations for picking the optimal strategy, we choose to model the game as a linear program with a cost function to be maximized under certain constraints. This Linear Program, as shown in the below equation, finds the optimal defender mixed strategy among all strategies that cause an attack on target i. The best strategy among all the solutions, the one giving highest payoff with be adopted by the defender.

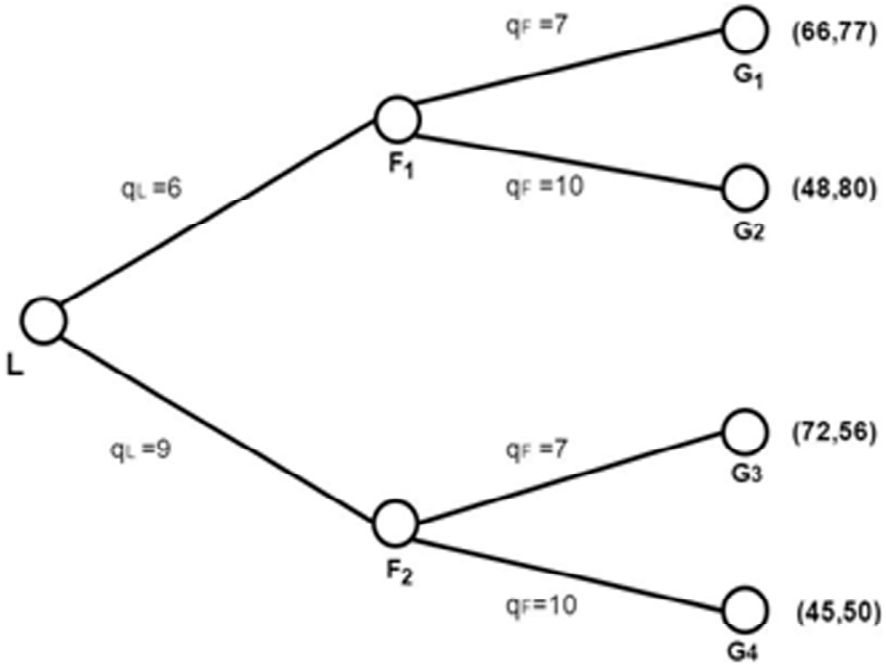

Figure 4: Game Tree to pick optimal solution

$$maximize\ U_d(i, \sum_{j:M\,i\,j=1} S_j)$$

such that. for i', $U_a(i', \sum_{j:M\,i'\,j=1} S_j) <= U_d(i, \sum_{j:M\,i\,j=1} S_j)$

for all j, $S_j >= 0$ and $(\sum_{j=1}$ to m $S_j) = 1$ (3)

Again, when the type of attacker is unknown, the missing Linear Programs can be constructed by testing with the kind of response each attacker gives for each and every combination of targets left protected or unprotected. This can cause exponential complexity in terms of computation.

5.5. Defender's best strategy using knowledge of attacker's type

The previous section covered a linear programming based approach for the leader to adopt the best strategy in cases where the follower adopts attacks that are unpredictable. Most of the times, the patterns are not predicted as a result of a violent DDos attack or stochastic network attacks.

In this game, includes only two attacker types, and the attackers identity can be determined in a single round by playing (3/5; 2/5): if target 1 is attacked then the attacker is of type 1 , and if target 2 is attacked then he is of type 2 . The optimal policy of the defender is to always protect target 1, 2 out of 3 times. Else, if attacker is of type 1, defender will lose out a lot, since defender has calculated equal probability of attack to either target. Indeed, in that case the attacker would attack target 1, causing the defender to incur a huge expected loss (at least L/3). In other words, an optimal policy would essentially instruct the defender to play it safe by assuming the attacker is of type 1, instead of using a single best response query to learn the attackers type and play optimally thereafter.

The entire procedure can be followed step by step according to the diagram shown.

6. EXPERIMENTAL RESULTS AND ANALYSIS

Using Docker Swarm[8] on a Dell Inspiron 15 hardware platform, 20 containers with a few text files with read write activated rights were created. One of the containers were made 'transparent', or through which other containers could be accessed. All the other 19 containers connected were accessible and text files could be modified. Starting off with 2 types of attackers and having a maximum of 10 types, an attack on an uncovered container causes a loss of 10 points whereas an attack on a covered one causes 0 damage. Since the damage also depends on whether the functionality of the container is compromised, we assume that the functionality f is related to the security of the container s in the following way. If efficiency, another measure of payoff is E, then

$$E = \alpha f + \beta s \quad (4)$$

6.1. Validation of Results

A common norm is to utilize machine learning techniques to observe the behaviour of the attacker, classify them into types and then use knowledge of the classification to adopt a strategy. But machine learning methods fail here, as backwards induction cannot work if the leader is not ready with a strategy at the beginning of the game. If the leader waits to learn the strategy of the follower at any stage of the game, the game seizes to be a stackelberg model and ends up giving a negative payoff for that stage to the defender. We ran 2 simulations, calculating the defender's efficiency along the way. First, using the LP based approach, using backward induction. Next, using a curve fitting technique in python scikit to predict the attacker type and then adopt a best response based approach using rationality. The below graphs give us a clear indication of how the LP based approach works in comparison with the Machine Learning based approach in providing the optimal covering spots for the defender to achieve maximum efficiency whilst ensuring both security and functionality of the distributed system.

6.2. Few more issues with Docker

In the previous section, we simulated a scenario which is relevant to modern day organizations in terms of ensuring security and functionality of their data storage methods using containers. In the following section, we dive into Docker's code and perform stress tests that unearth a number of security issues.

1) Setup: This study was performed using a fuzzy software tester called Trinity. This tool simulates randomized calls to the kernel to make an excess of system calls that puts the host under test. We ran Trinity in an unprivileged container. To do this we wrote the Dockerfile script with all the dependency required for running trinity. To run trinity simultaneously in multiple containers we worked on small bash script.
2) Issues noticed in Docker: Running simultaneously 20 unprivileged trinity containers consumes most of the CPU resources and causes other deployed application containers to starve. Also, running more than 20 containers does not increase the CPU usage and it can be noticed that 20 running trinity containers can hung up a 4GB machine.

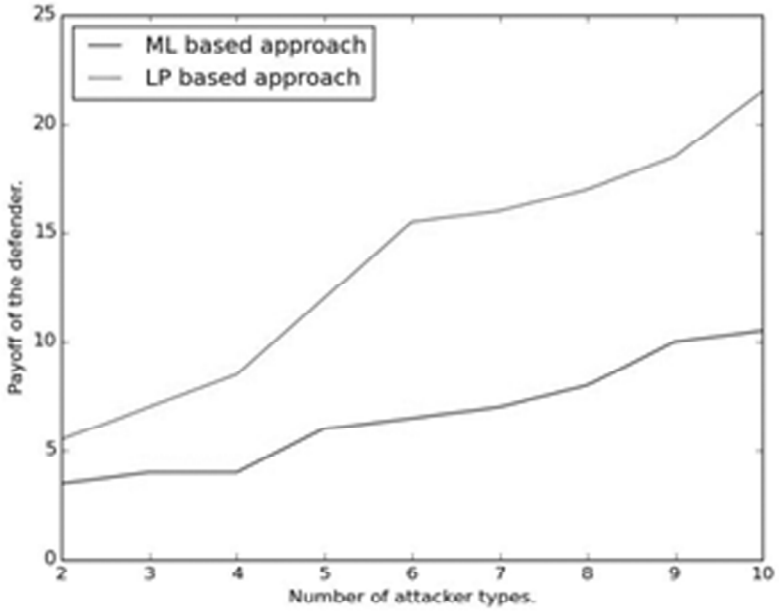

Figure 5: Kernel before using Trinity

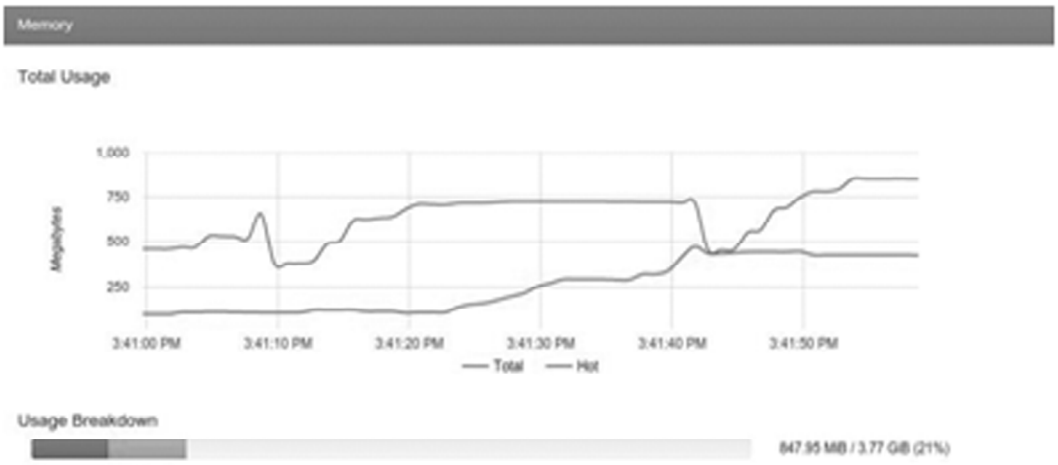

Figure 6: Memory before using Trinity

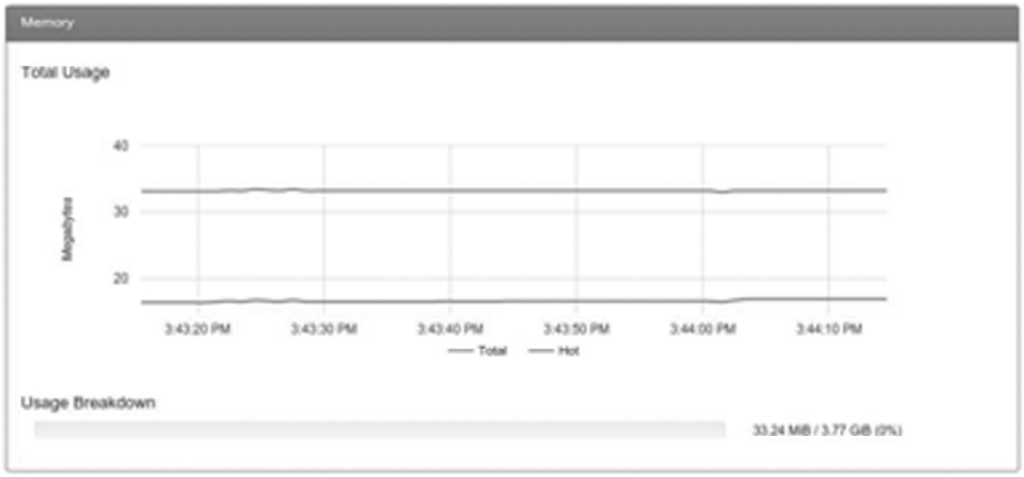

Figure 7: Kernel after using Trinity

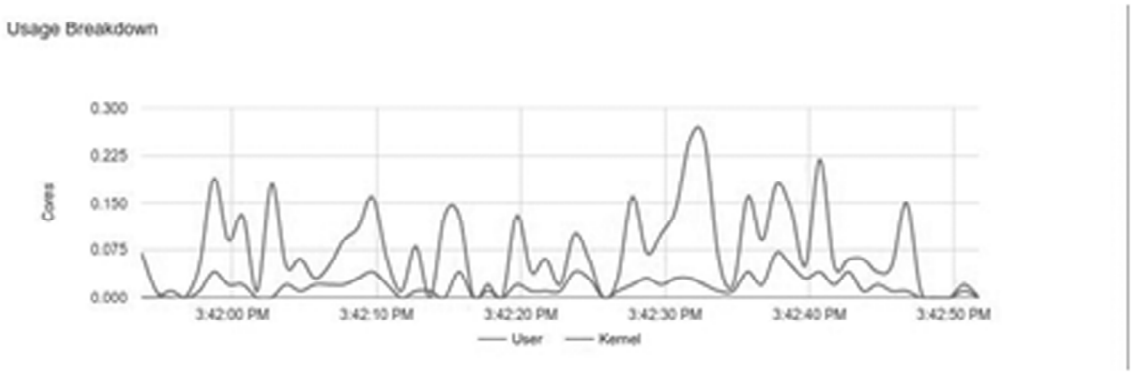

Figure 8: Memory after using Trinity

3) Inference from plots: The trinity program performs various initializations, opening the file descriptors and creating memory mappings. Then it start off a number of child pro-cesses that perform the system call tests. High usage of Kernel can be seen in Fig.7 which is a cadvisor[3] plot after running a trinity container over a docker engine. The Red line shows the kernel panicking when given totally random system calls. Blue line shows the user usage of kernel. Trinity containers almost use up the memory causes other process to starve which may cause system crash. The excess memory usage of kernel is shown in Fig. 8. Red line(hot) shows that the extra memory is being consumed by the kernel.

7. CONCLUSION

Docker is a tool that has revolutionized data storage. The obvious benefits of using Docker have certainly made up for the small issues that have surfaced in this study. Apart from unearthing issues related to Docker's security, we also provided a solution to one of the most important and commonly faced issues related to Docker's networking.

On simulating an environment resembling a distributed data storage environment of an organization, we formulated an attack based on the transparency of a container. This gave rise to a situation where there could be multiple types attackers who have targets on various publicly exposed containers. First, we found a mathematical relation between efficiency, functionality and security of a container. Using this, we formulated a stackelberg game with 2 players. Using the backward induction model, the defender(security maintainer) could adopt a strategy that can cause least damage whilst maintaining the functional-ity of the distributed system. In cases where the type of attacker or his pattern is unknown, a search based approach

for the same has been proposed. A common methodology to classify attackers would be using machine learning. The futility of such an approach has also been proposed by comparing our method to a curve fitting based approach. In conclusion, the methods proposed in this study have ensured a high payoff to the defender. In further study, we must dive into exhaustive search based methods in cases where attacker's type is unknown. Perhaps utilizing a monte carlo based model would be the way to go.

REFERENCES

[1] Bacis, E., Mutti, S., Capelli, S., & Paraboschi, S. (2015). DockerPolicy-Modules: mandatory access control for docker containers. IEEE CNS.

[2] A. Dusia, Yang Yang and M. Taufer, "Network Quality of Service in Docker Containers," Cluster Computing (CLUSTER), 2015 IEEE International Conference on, Chicago, IL, 2015, pp. 527-528. doi: 10.1109/CLUSTER.2015.96

[3] Google, Kubernetes. http://kubernetes.io/.

[4] W. Drewry, dynamic seccomp policies (using bpf filters), https://lwn. net/Articles/475019/.

[5] Docker, Docker security: Linux kernel capabilities, http://bit.ly/ 1d3gWQV

[6] Advancing docker security: Docker 1.4.0 and 1.3.3 releases, Novem-ber 2014, https://blog.docker.com/2014/12/ advancing-docker-security-docker-1-4-0-and-1-3-3-releases.

[7] Introducing docker content trust, August 2015, https://blog. docker.com/2015/08/content-trust-docker-1-8/.

[8] Docker swarm, https://docs.docker.com/swarm/

[9] D. Bernstein, Containers and cloud: From lxc to docker to kubernetes, Cloud Computing, IEEE, vol. 1, no. 3, pp. 8184, Sept 2014

[10] M. Zhang, D. Marino and P. Efstathopoulos, "Harbormaster: Policy Enforcement for Containers," 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), Vancouver, BC, 2015, pp. 355-362.

[11] Avrim Blum, Nika Haghtalab, Ariel D. Procaccia, "Learning to Play Stackelberg Security Games" Carnegie Mellon University, vol. 1, 2015.

[12] Namespaces(7) - Linux manual page. (n.d.). Retrieved March 3, 2016, from http://man7.org/linux/man-pages/man7/ namespaces.7.html