Scalable and lightweight CTF infrastructures using application containers

Arvind S Raj, Bithin Alangot, Seshagiri Prabhu and Krishnashree Achuthan

(arvindsraj,bithina,seshagiriprabhu)@am.amrita.edu, krishna@amrita.edu
Amrita Center for Cybersecurity Systems and Networks
Amrita Vishwa Vidyapeetham

Abstract

Attack-defence Capture The Flag (CTF) competitions are effective pedagogic platforms to teach secure coding practices due to the interactive and real-world experiences they provide to the contest participants. Two of the key challenges that prevent widespread adoption of such contests are: 1) The game infrastructure is highly resource intensive requiring dedication of significant hardware resources and monitoring by organizers during the contest and 2) the participants find the gameplay to be complicated, requiring performance of multiple tasks that overwhelms inexperienced players. In order to address these, we propose a novel attack-defence CTF game infrastructure which uses application containers. The results of our work showcase effectiveness of these containers and supporting tools in not only reducing the resources organizers need but also simplifying the game infrastructure. The work also demonstrates how the supporting tools can be leveraged to help participants focus more on playing the game i.e. attacking and defending services and less on administrative tasks. The results from this work indicate that our architecture can accommodate over 150 teams with 15 times fewer resources when compared to existing infrastructures of most contests today.

1 Introduction

In 2015 alone, the cybersecurity world saw more than 6000 security bugs being reported [1] in various kinds of software ranging from website content management systems and music players to web browsers and the OS kernel. Many of these were caused by vulnerabilities well studied for over a decade now and have documented fixes [2–5]. Thus, educating software developers about secure coding practices is of paramount criticality to mitigate losses and maintain both organizational and civilian safety. Capture The Flag(CTF) competitions provide a game based learning approach to understanding computer security principles and secure coding practices. In this contests, teams are provided with identical challenges that require the application of computer security concepts towards solving them. Their practical and hands-on nature makes them particularly promising as pedagogic tools.

There are two popular formats for CTF contests:

- **Jeopardy style**: Teams are provided with several challenges in multiple areas such as cryptography, digital forensics, system security and web application security. They are free to solve challenges in any order of preference and work in isolation from other teams.

- **Attack-defence**: Teams are provided a virtual machine consisting of identical vulnerable services. The overall objective is to discover the existing security issues, fix them and attack services of other teams using the vulnerabilities discovered.

It is believed that among the two formats, attack-defence CTFs provide greater learning outcomes due to their interactive and real-world nature [6]. However, very few attack-defence CTFs are organized worldwide every year [figure 1] and the number of participants is significantly lower in comparison to Jeopardy style CTFs [figure 2]. In addition, most attack-defense CTFs are in-person events - at the most 3 events were run online every year for the past 5 years.

Based on our experience participating in several online attack-defence CTFs and organizing the last 5 editions of InCTF [7], we believe the reasons for these trends are:

- **Complex infrastructure**: Attack-defence CTF infrastructures require high amounts of resources and engineering to ensure smooth execution of the contest. This can be particularly challenging for organizers, who also need to design services and run other peripheral activities as part of the entire contest.

- **Multi-tasking**: In addition to finding, fixing and

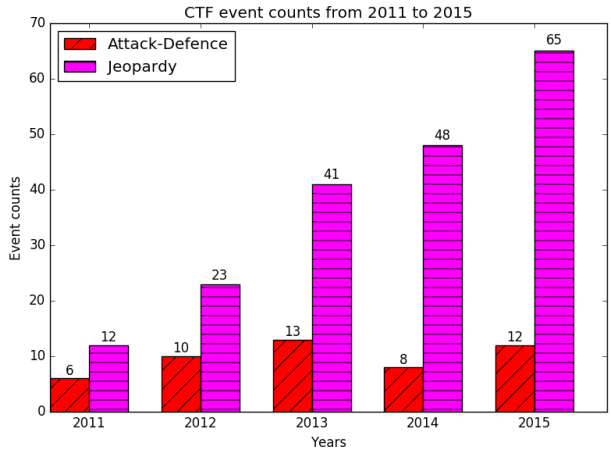

Figure 1: Number of events of both formats over past 5 years based on data from ctftime.org

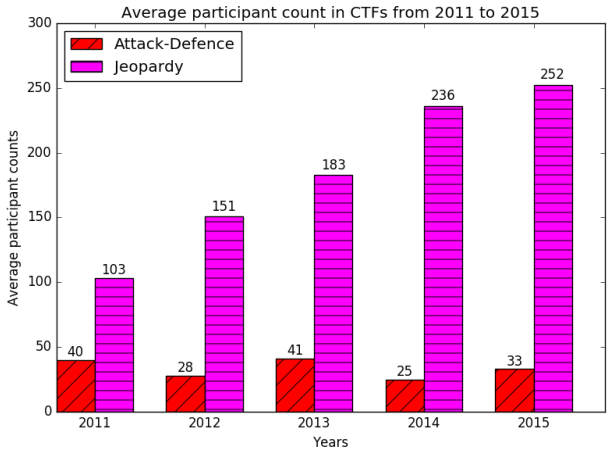

Figure 2: Average participation counts in events of two formats based on data from ctftime.org

exploiting vulnerabilities, participant teams are required to carry out several system administration tasks. This includes ensuring that services and network connections are functioning correctly, maintaining backups of services and monitor incoming traffic for attacks and so on. These can be challenging, especially for small teams and inexperienced players.

In this paper, we introduce a novel CTF infrastructure that uses Docker containers [8] instead of virtual machines. The use of Docker containers helps organizers focus on creating a good quality contest since the amount of resources and engineering effort is significantly lower. In addition, participants can focus on finding, fixing and exploiting vulnerabilities since much less system administration is necessary. The use of additional tools to manage Docker containers and associated components enhances the overall game experience and simplifies organizing an attack-defence CTF.

The rest of the paper is structured as follows. Section 2 provides an overview of use virtual machines in secure coding education, section 3 provides an overview of attack-defence CTFs and Docker, section 4 describes existing attack-defence CTF infrastructures and their shortcomings, section 5 describes the container based attack-defence CTF infrastructure, section 6 describes how we evaluated the system and section 7 concludes.

2 Related work

The infrastructure of most attack-defence CTFs is derived from the design pioneered by iCTF [6] over several editions. All services are packaged into a virtual machine which is run by the participants on their hardware or by the organizers on their server. In the former case, teams have direct access to their virtual machines while in the latter case, teams can access them via SSH. iCTF is also well known to experiment with contest design and gameplay but almost all editions feature virtual machines as a core component of the infrastructure [6, 9, 10]. Virtual machines are also popular in other game based approaches to teaching computer and network security and other computer science courses as well as focused training programs [11–17].

There have been few attack-defence CTFs which used application containers or similar approaches. Until few years ago, DefCon CTF finals used FreeBSD jails to isolate all services of all teams from each other. While FreeBSD jails are lightweight and secure, they are not as flexible as application containers. More recently, iCTF 2008 [18] and RuCTFe 2011 used OpenVZ containers in their contest infrastructure. However, iCTF 2008 involved participants trying to break into an isolated corporate network consisting of services hosted inside OpenVZ containers rather than attacking each other. In RuCTFe 2011 too, the services ran inside OpenVZ containers. However, these containers existed inside the virtual machine distributed to participants. To the best of our knowledge, this is the first attempt to construct a CTF game infrastructure that doesn't feature virtual machines as a core component, requires fewer resources due to the use of application containers to reduce resource requirements and is more participant friendly than existing infrastructures.

3 Background

3.1 Attack Defence CTFs

3.1.1 Overview

In attack-defence CTFs, teams are provided identical virtual machines consisting of custom written vulner-

able applications("services"). Teams analyze the services, discover the security issues, fix them and attack other teams using the same security bugs to earn points. Attack-defence CTFs test offensive and defensive skills in application as well as system and network security areas. As a consequence, attack-defence CTFs require specialized game infrastructures which are difficult to engineer and maintain reliably. The challenges are further compounded if the game is played online since participants can be widely distributed geographically around the world.

3.1.2 Services

A service is a custom written vulnerable application provided by the organizers. They can be simple network applications such as a chat server for connected clients to highly complex applications such as banking and social networking applications. All services provide the ability to store private information (called a "flag") in them, which can be retrieved only by providing a particular secret (e.g.: username and password). These services also contain one or more vulnerabilities such as SQL injection, buffer overflows and use of weak encryption keys deliberately introduced within them. The services are designed carefully such that the flag can be retrieved only by supplying the corresponding secret or exploiting the vulnerability. Since the flag and secret cannot be brute forced quickly, submitting a flag of another team is proof that the team successfully exploited the corresponding service of the other team.

3.1.3 Gameplay and scoring

Attack-defence CTFs typically last 8 to 12 hours in duration. During the first phase of the contest, no flags are stored in any of the services. Thus, teams devote this period for analyzing services and the system, fixing any security issues they discover and writing exploits for these security bugs for scoring points in the following phase. The second or scoring phase consists of several, approximately equal time periods called rounds. At the start of each round, scripts run by the organizers store flags in all services and later retrieve them towards the end of the round. Occasionally, organizers may introduce updates to services which either adds new or modifies existing service functionality. Teams continue to analyze the services and monitor network connections for new exploits and work towards discovering any vulnerabilities they may have missed. There are three types of scorable points a team can avail of during a round:

1. Offence points obtained by stealing flags from other teams.
2. Availability points awarded for ensuring services are online.
3. Defence points obtained by preventing other teams from stealing flags.

The gameplay and scoring mechanisms resemble real world scenarios closely:

1. Applications work as per the intended design for legitimate input yet can fail gracefully with malicious input and without revealing the secret information.
2. Applications may be attacked and compromised at any time. Thus, constant monitoring and swift remedy are required.

In addition, the offensive aspect helps participants think from attacker's perspective which is useful when designing secure software and developing security fixes.

3.2 Docker

Docker[8] is an operating system level virtualization technology. Similar to hardware virtualization, Docker allows executing processes in isolated sandboxes called containers and restricting their resource usage. These containers are created using a container image similar to how virtual machines are spawned from virtual disk images. However, unlike in hardware virtualization, Docker virtualizes the kernel of the host device. This reduces resource usage, image sizes and startup times of containers significantly compared to virtual machines while providing similar levels of isolation, security and performance [19]. Thus, Docker containers are a viable alternative for virtual machines in attack-defence CTF infrastructures.

Along with Docker, we use Docker Distribution [20] and Portus [21] to manage exploits and patches to services uploaded by teams. Docker Distribution is a container image management system which simplifies storing, updating and distributing images. Portus is a front end for Docker distribution that implements role-based access control to the images stored in Distribution and other resources. Users can create teams, private and shared image namespaces and provide different access levels to different users. This makes Portus an ideal solution for managing the container images of all teams.

4 Existing attack-defence CTF infrastructures

In this section, we describe the existing attack-defence CTF infrastructure designs commonly used today and their limitations.

4.1 Decentralized architectures

Almost all online attack-defence CTF events use the decentralized architecture[figure 3]. The infrastructure consists of several geographically distributed team networks and an organizer network inter-connected using a virtual private network(VPN). Since all team networks

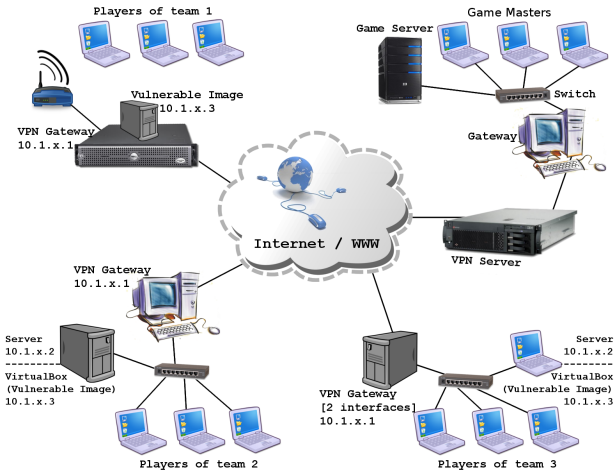

Figure 3: Organization of VPN based decentralized CTF infrastructure

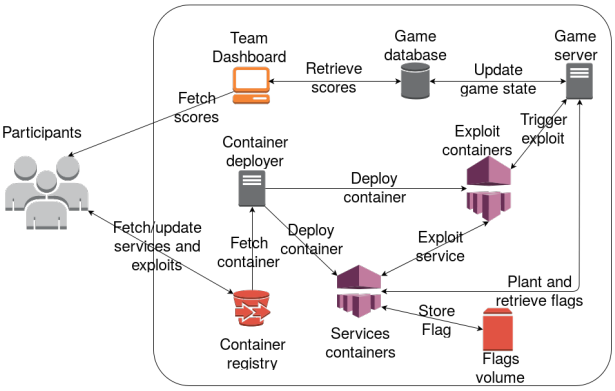

Figure 4: Container based attack defence CTF infrastructure

are identical in structure, teams can quickly locate and attempt to steal flags from services of other teams. Teams submit these flags for points, which are awarded instantly and displayed on the team dashboard. Simultaneously, teams monitor their network for incoming attacks to discover unknown vulnerabilities and implement appropriate remedies.

4.2 Centralized architectures

iCTF introduced the centralized architecture in 2012 and used it in 2013 as well [6]. Here, the organizers host the virtual machines of all teams on their servers and provide SSH access to teams. This approach reduces the amount of technical setup teams need to perform. Organizers decide if teams can run exploits on their own or to execute the exploits on behalf of the teams. The remainder of the infrastructure components and gameplay are similar to the decentralized architecture.

4.3 Limitations of existing infrastructures

The amount of resources, engineering and monitoring required in the decentralized infrastructure design can be quite high. For instance, rwthCTF 2012 ran 8 OpenVPN processes on a 16GB server with an 8 core Intel i7 processor to handle the VPN traffic load [22]. Also, teams struggle to setup the VPN and team network due to issues such as lack of technical expertise, lack of hardware to run the virtual machine, lack of networking equipment and potentially restrictive IT security policies at location from where they participate.

The centralized approach reduces the amount of hardware and infrastructure setup required by teams. However, organizers need exponentially higher amounts of resources to run the game infrastructure and virtual machines. In addition, teams geographically further away from the hosting site experience proportional network latencies, which could negatively impact their game experience. Also, if organizers end up running the exploits, the teams cannot debug exploits if they fail and rely on organizers to install any packages and libraries used by their exploits.

Also, regardless of architecture style used, inexperienced teams struggle to perform well in attack-defence CTFs. We believe this is because teams have to perform several additional tasks such as administering the network, keep services running, maintain backups of services and much more. Some teams are not aware of these, leading to greater frustration when service stops functioning due to destructive attacks performed on them or for other reasons. These easily distract inexperienced teams from the goal of learning secure coding practices.

5 Application container based infrastructure

We modified the iCTF contest infrastructure[6] to work with Docker containers. In this section, we describe the modified and newly introduced components, how service and exploit containers are created and executed and provide an overview of a round in a game using the enhanced infrastructure. Figure 4 shows the container based infrastructure design.

5.1 Components of infrastructure

We introduce four new components: *container image registry*, *service containers host*, *exploit containers host* and *flag storage volume*. In addition, we modified the gameserver to synchronize the containers periodically with their corresponding images stored in image registry and *vmcreator* to create containers instead of VMs as well as removed the *router* component since teams do not have access to the live containers. The remaining components are exactly similar to their counterparts in the iCTF contest infrastructure.

The container image registry provides an easy and secure means to distribute and manage container images by leveraging capabilities of Docker distribution and Portus. Docker distribution handles image storage and retrieval while Portus handles user authentication, request authorization and access control. Every team is assigned a namespace exclusively accessible to them and the gameserver. Teams can upload exploit containers and changes to service containers to their namespace similar to how they push changes to version controlled repositories hosted on Github and Bitbucket. Periodically, the gameserver synchronizes the containers with their images, as described later in the section. This decoupling of where teams update services and where the services execute helps work around the SSH latency issue in the centralized CTF architecture.

The container hosts are compute servers which run all service and exploit containers. In our infrastructure, we run both the service and exploit containers on a single server due to resource limitations. However, the design of the contest infrastructure makes it possible to run containers on multiple machines in order to distribute the load and improve performance. In addition, tools such as Docker Swarm [23] and Docker Universal Control Plane(UCP) [24] simplify scaling out to multiple physical machines and cloud compute servers such as Amazon AWS and Microsoft Azure. An alternative is to use Docker Cloud [25], a hosted service offering the capabilities of Docker Swarm and Docker UCP to manage multiple Docker containers. The container hosts run Docker daemons listening on a TCP socket which enables creating, running and destroying containers remotely using the Docker remote API. The Docker daemons are isolated from public internet and can be configured to accept requests from clients providing a valid certificate, thus restricting who can communicate with the daemons.

In order to synchronize containers with their images, we configure the Docker distribution to notify the gameserver when an image is modified. During the next round of synchronization(section 5.4), we simply delete the existing container and create a new container based on the latest image. Since creating and deleting containers are fast operations, they do not impose a a significant time penalty. Another approach is to create a new container using the updated image which "replaces" the old container in the game. The old container can be removed later by a garbage container collector process running in the background, further reducing the time penalty. However, we chose not to implement this approach since the former method is simpler and does not impose a very significant time penalty.

A side effect of deleting and recreating containers is that all flags stored by the service are also deleted. This can cause issues if flags are stored in databases, which would need to be recreated on every update. In such cases, Docker recommends provisioning a data only container for running the databases or creating a data volume to store the database files.

5.2 Building, distributing and executing service container images

Similar to the iCTF approach, we first create Debian packages of all services to simplify creating container images. We then proceed to create a Docker container image using the Debian packages and a minimal cloud image of Ubuntu. The entire process is fully automated and driven by few template files that can be easily modified depending on requirements. This process can be easily extended to any operating system that has a standard packaging format with a few operating system specific changes.

In order to distribute the images, we first create accounts and namespaces for all teams in Portus. After this, we push copies of the images to namespaces for teams to download. An alternative is to export each service container image to an archive, which can be encrypted and distributed prior to the contest. This can be useful since network bandwidth could become a bottleneck if everyone starts downloading the images at the start of the contest.

Every service binds to a particular port which is mapped to a port on container host machine. The service port is specified in the configuration file of the service and the host machine port is automatically assigned using the range of ports specified in the game configuration, ensuring there are no collisions. At the start of the game, all service container images are marked as "requiring update" and thus the containers are started by the gameserver as described in section 5.1. This process is fully automated and its duration depends on the speed of container host machine and the number of services and teams.

5.3 Building, uploading and executing exploit containers

Similar to service containers, exploit containers can be created using an image from Docker hub or a minimal cloud image matching the architecture of the exploit container host machine. Teams are free to choose the OS and setup the exploitation environment by installing custom libraries, tools, frameworks, programming languages and more as long as they correctly specify the command to run the exploit when the container is started. This is a notable improvement over iCTF where teams were provided a standard environment to write exploits and organizers had to install necessary additional packages requested by the teams. After building and testing the exploit container locally, teams push the image to

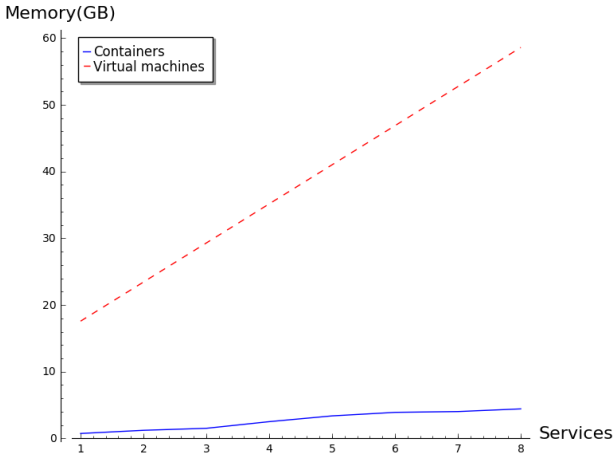

Figure 5: Main memory usage with 30 teams and varying services counts in container and virtual machine based infrastructure

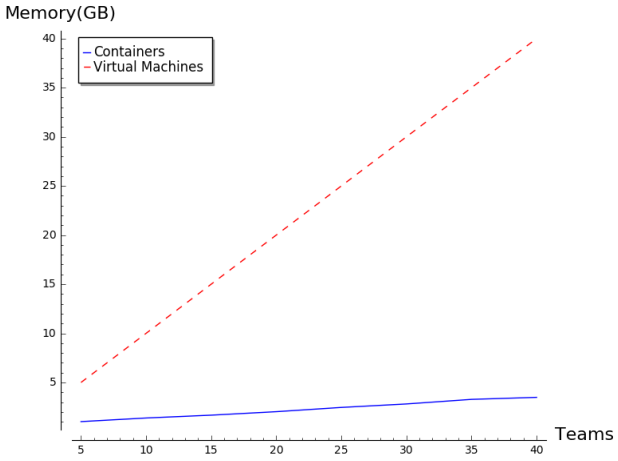

Figure 6: Main memory usage with 3 services and varying team counts in container and virtual machine based infrastructure

their namespace in the registry, which notifies the gameserver about the new image uploaded. The gameserver stores an entry for the exploit container in database and marks it as requiring update during the next synchronization phase.

The targets for exploits are passed as a JSON array of key-value pairs via an environment variable. Each element of the array consists of 3 key-value pairs - the IP address and port number of the target and the identifier of the flag to be retrieved. Using this target information, the exploits captures all possible flags and prints them to stdout. After the container has exited, flags are extracted and submitted for points and the container is deleted. Deletion is necessary because the only way to provide new targets via environment variables is to delete and recreate the container. However, this doesn't impose a very high penalty because creating and deleting containers are very fast processes.

5.4 Game Round Overview

The entire game is divided into several rounds of approximately similar and configurable duration. Each round consists of 5 sequentially occurring phases:

1. **Synchronize service containers**: All services containers whose images have been updated are recreated to pull in the latest changes.

2. **Synchronize exploit container images**: All updated exploit container images are synchronized with copies on the exploit containers host.

3. **Store flags**: The gameserver starts evaluating the services by storing flags in them. If flag planting fails, the service is marked as not functioning properly.

4. **Run exploits**: The gameserver runs all available exploit containers against all valid and currently active targets. Teams are awarded points for all valid flags stolen by their exploit.

5. **Retrieve flags**: The gameserver retrieves the flag planted in step 3 and updates state of service depending on the result of the retrieve operation.

5.5 Benefits for participants and organizers

The gameplay closely resembles bug fixing in software development: identifying issues, fixing them, testing thoroughly and committing changes/releasing patches and hence places minimal effort on the participants side. Additionally, version controlling images is part of the workflow and thus, participants need not worry about losing or breaking the service and not being able to restore it to a working state quickly. Another significant benefit is that services can be run locally in an environment identical to remote setup for analysis; thus working around network latencies. Other benefits for participants include no need to run exploits themselves, obtain networking equipment and machines or setup team network and VPN. Organizers can also easily scale the infrastructure to multiple machines using additional tools such as Docker Swarm and Docker Cloud. Docker also has a very active community which could lead to the development of tools, in future, that can be useful for hosting and organizing the infrastructure.

6 Performance evaluation of system

We compare the performance of container based and virtual machine based infrastructure by measuring resource usage in two scenarios:

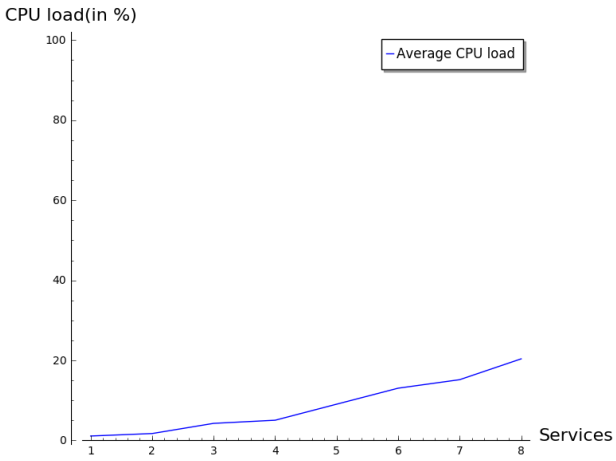

Figure 7: Average CPU usage with 30 teams and varying services counts in container based infrastructure

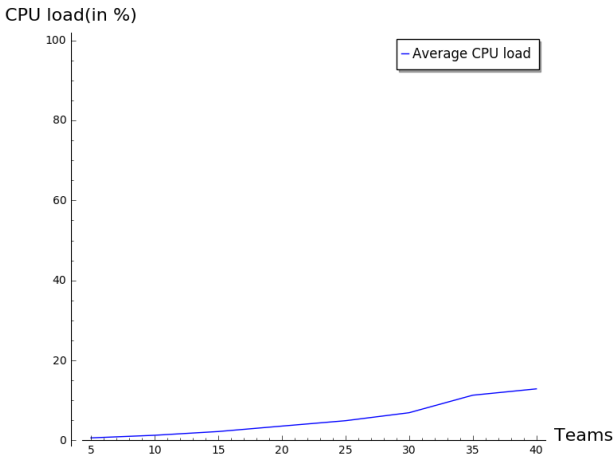

Figure 8: Average CPU usage with 3 services and varying team counts in container based infrastructure

1. We vary the number of teams while keeping the number of services constant.

2. We vary the number of services while keeping the number of teams constant.

We experimentally determine the resource usage of container based infrastructure and compare it with the estimated resource usage for virtual machine based infrastructure. We choose to estimate usage based on InCTF recommendations in latter case due to lack of resources for simulating the scenario. Figures 5 and 6 display observed and estimated average memory usage in both infrastructures while figures 7 and 8 display the measured CPU usage in container based infrastructures. In all simulations, we ran exploits "written" by all teams against all services of other teams.

6.1 Resource utilization in container based infrastructure

We measure the resource usage by instantiating several cases for the two scenarios described earlier:

1. We simulate 8 contests with 3 services in each. We start with 5 teams and keep adding 5 teams in every subsequent contest.

2. We simulate 8 contests with 30 teams in each. We start with 1 service and keep adding a service in every subsequent contest.

All containers ran on a server with 16GB memory and an 8 core Intel Core i5 2600 processor. We measure the CPU utilization and the memory usage for a 10 minute long round of game described in section 5.4.

6.2 Resource utilization in virtual machine based infrastructure

We estimate the resource usage based on the requirements for InCTF's attack-defence round. Typically, there are 3 vulnerable services and we recommend allocating 1GB RAM and 2 CPU cores for the virtual machine: 200MB for each service and remainder for the operating system. In nearly all editions of InCTF, these specifications were used and performance was observed to be sufficient. Based on these values, we compute the expected resource usage for all contests instantiated in section 6.1.

6.3 Discussion

Based on figures 5, 6, 7, and 8, we believe it is possible to run a contest with a similar number of teams and services as other attack-defence CTFs using the container based infrastructure and significantly fewer resources. However, we were unable to verify this experimentally since the spawning of too many exploit containers simultaneously overwhelmed the sole Docker daemon. We believe this can be resolved by carefully scheduling the execution of exploit containers on the container host machine. Alternatively, scaling out can help distribute the load among multiple daemons and machines. In addition, use of multiple machines makes more ports available, which can help accommodate more service containers and hence more teams. The use of clustering technologies such as Docker Swarm can further ease managing containers running on multiple machines. Another possibility is to execute multiple Docker daemons on the same machine which would require manually bridging the containers controlled by different daemons, which is not a trivial task.

7 Conclusion and future work

Attack-defence CTFs are considered to have better learning outcomes due to their interactive and real-world gaming environments. However, complex infrastructure setup and gameplay have been a deterrent to its widespread adoption. We propose a novel CTF infrastructure that uses Docker containers instead of virtual machines that significantly reduces the resource requirements and handles certain system administration tasks. The resource efficient infrastructure and several available third party utilities greatly simplify organizing such CTFs and scaling to several teams and helps participants focus more on learning secure coding practices.

One of the significant limitations of the system is that teams cannot capture exploits from the network and reverse engineer them to identify new vulnerabilities. Another challenge is that the exploit scheduling algorithm should dynamically adapt as more exploits are uploaded to ensure the infrastructure and exploits function correctly. We hope to address these issues as well as obtain a user evaluation of the system as part of our future work.

8 Availability

The container based attack-defence framework is available for download at http://github.com/inctf/inctf-framework.

References

[1] National Vulnerability Database, https://nvd.nist.gov/.

[2] Common Weakness Enumeration, https://cwe.mitre.org/.

[3] Theodoor Scholte, Davide Balzarotti, and Engin Kirda. Have things changed now? An Empirical Study on Input Validation Vulnerabilities in Web Applications. *Computers & Security*, 31 (3):344–356, 2012.

[4] Victor van der Veen, Nitish dutt Sharma, Lorenzo Cavallaro, and Herbert Bos. Memory Errors: The Past, the Present, and the Future. *Research in Attacks, Intrusions, and Defenses*, pages 86–106, 2012.

[5] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. SoK: Eternal War in Memory. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 48–62. IEEE, 2013.

[6] Giovanni Vigna, Kevin Borgolte, Jacopo Corbetta, Adam Doupe, Yanick Fratantonio, Luca Invernizzi, Dhilung Kirat, and Yan Shoshitaishvili. Ten Years of iCTF: The Good, The Bad, and The Ugly. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, 2014.

[7] InCTF contest, https://inctf.in/.

[8] Docker Inc, https://www.docker.com/what-docker/.

[9] Adam Doupé, Manuel Egele, Benjamin Caillat, Gianluca Stringhini, Gorkem Yakin, Ali Zand, Ludovico Cavedon, and Giovanni Vigna. Hit'em Where it Hurts: A Live Security Exercise on Cyber Situational Awareness. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 51–61. ACM, 2011.

[10] Yan Shoshitaishvili, Luca Invernizzi, Adam Doupe, and Giovanni Vigna. Do You Feel Lucky?: A Large-Scale Analysis of Risk-Rewards Trade-Offs in Cyber Security. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1649–1656. ACM, 2014.

[11] Elie Bursztein, Baptiste Gourdin, Celine Fabry, Jason Bau, Gustav Rydstedt, Hristo Bojinov, Dan Boneh, and John C Mitchell. Webseclab Security Education Workbench. In *CSET*, 2010.

[12] Andy Davis, Tim Leek, Michael Zhivich, Kyle Gwinnup, and William Leonard. The Fun and Future of CTF. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, 2014.

[13] Jelena Mirkovic, Aimee Tabor, Simon Woo, and Portia Pusey. Engaging Novices in Cybersecurity Competitions: A Vision and Lessons Learned at ACM Tapia 2015. In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*, 2015.

[14] Z Cliffe Schreuders and Lewis Ardern. Generating randomised virtualised scenarios for ethical hacking and computer security education: Secgen implementation and deployment. 2015.

[15] Tom Chothia and Chris Novakovic. An Offline Capture The Flag-Style Virtual Machine and an Assessment of its Value for Cybersecurity Education. In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*, 2015.

[16] Martin Carlisle, Michael Chiaramonte, and David Caswell. Using CTFs for an Undergraduate Cyber Education. In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*, 2015.

[17] Adrian Dabrowski, Markus Kammerstetter, Eduard Thamm, Edgar Weippl, and Wolfgang Kastner. Leveraging Competitive Gamification for Sustainable Fun and Profit in Security Education. In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*, 2015.

[18] Nicholas Childers, Bryce Boe, Lorenzo Cavallaro, Ludovico Cavedon, Marco Cova, Manuel Egele, and Giovanni Vigna. Organizing Large Scale Hacking Competitions. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 132–152. Springer, 2010.

[19] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An Updated Performance Comparison of Virtual Machines and Linux Containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*, pages 171–172. IEEE, 2015.

[20] Docker Inc, https://docs.docker.com/registry/.

[21] SUSE community, http://port.us.org/.

[22] 0ldEur0pe CTF Team. https://github.com/oldeurope/rwthctf2012/blob/master/vpn/README.md.

[23] Docker Inc, https://docs.docker.com/swarm/overview/.

[24] Docker Inc, https://docs.docker.com/ucp/overview/.

[25] Docker Inc, https://docs.docker.com/docker-cloud/.