Hello Everyone,

My name is Md. Olid Hasan Bhuiyan and I'm thrilled to be here today to share with you some insights from my research paper titled "On Detecting Malicious Code Injection by Monitoring Multi-level Container Activities." This is a joint work between “Bangladesh University of Engineering and Technology” and “Concordia University, Canada”.

In the next few minutes, I will take you through exploring the existing works of container security and unveiling our innovative approach to enhance malicious code injection detection. Here we have listed the topics that we’ll touch upon today. Let’s start with understanding the motivation behind our work.

Let's say you're hosting a large outdoor event, and you've set up a number of shipping containers to serve as temporary offices, storage spaces, and restrooms. One day, a group of thieves breaks into the event grounds and gains access to the shipping containers. They realize that one of the containers contains valuable equipment, but it's locked with a sturdy padlock.

Instead of trying to break the lock, the thieves use a cutting torch to cut a hole in the side of the container. Once inside, they quickly grab the equipment and make their escape. The hole in the container is small enough that it's not immediately noticeable, and the theft goes undiscovered until the event is over.

In this scenario, the shipping container is analogous to a container in a software environment, and the padlock represents security measures such as access controls and authentication. The thieves' use of a cutting torch to bypass the lock is similar to an attacker exploiting a vulnerability or weakness in a container's security measures to gain unauthorized access to the underlying system or data. The theft itself is equivalent to data theft or a breach of sensitive information in a software environment.

Now a days, container is highly used despite its weak security system

We target malicious code injection attack because this attack can result in many more severe attacks and it is quite hard to detect a malicious code injection attack. This is because it can be done in many ways, following different paths. We are going to talk about the paths that finally lead to malicious code injection in the later section of our presentation.

So lets say, why container technology has become so popular? Because they are portable, lightweight, isolated from each other (that means even if one of the container is down, the others can operate independently) and finally they offer multi-tenant service which means one container can provide service to multiple end users.

But container technology has some drawbacks as well. First of all, they have large attack surfaces. There are lots of vulnerable points in a container. As containers operates in an isolated way, communication between different containers is quite complex. Finally, container has thin protection layer.

Now let’s take a brief idea about container. Container is a lightweight OS-level virtualization method. It is a Stand-alone piece of executable software. This means this does not have any environmental dependency. All the dependencies come within a container. And remember container is not a virtual machine.

So what is the difference between a virtual machine and a container. The main difference is that container has its operating system. All the libraries and executables comes with a container. Whereas virtual machine depends on the host machine. As a result, for microservice architecture, container is a more suitable option than virtual machine.

Now, let's take a look at what previous research has contributed to this field. We'll discuss existing works, examining their strengths and limitations. By building upon the foundation laid by previous works, we aim to introduce innovative approaches that address gaps in current detection methods.

The first work mentioned here suggested a host-based intrusion detection mechanism. They used BoSC technique (Bag of system call technique for their work) and they used Strace tool for that.

The second work Proposed a container-level anomaly-based intrusion detection systems for multi-tenant applications. Mainly worked on four different vulnerabilities - Authentication Bypass, DoS Overflow, System User Privilege Escalation and Integer Overflow. Tool used: **sysdig**

The next one Discussed the possible security issues raised by the usage of containers and offers suggestions for resolving them. But they did not implement their proposal. Same goes for the next one as well.

The last work mentioned here Suggested a new security layer using the Most Privileged Container (MPC). This MPC layer implements privilege-based access control and grants permissions for resource access based on rules. They did not implement their proposal either.

So, as we can see previous works in this field has some limitations. Most of them used a general solution or detection mechanism for different types of attacks. They did not work on attack specific detection mechanism. Some of the research works only proposed a methodology, did not even test their suggested method. And most importantly, None of these research works are related to multi-level protection in containerized environment.

Next, let's define the problem we're addressing. We'll outline the specific challenges and objectives of our research. What is a malicious code injection attack - A type of cyber attack where an attacker inserts malicious code into a vulnerable application or system, with the intent of compromising the security or functionality of the target. There are several potential **attack paths** to insert malicious code into a container. For example, we can implement malicious code injection using leveraging XSS, CSRF, SQLi and RBAC attack. We have got these attack paths from CNCF. So our aim is to detect malicious code injection in container for all **potential attack paths.**

We can achieve this by

1. Monitoring different levels of container
2. Fetching different features(**name, sequence, frequency)** of system calls
3. Using different tools like **sysdig, strace and kubernetes** dashboard

This image summarize the main working principle of our proposed mechanism. As we can see, XSS and SQLi attack happens in application layer and container engine and operating system layer get affected as a of the attacks. So, we are going to monitor several layers our container. The operating system layer is going to be monitored by Strace and the whole container is monitored by strace and Kubernetes dashboard. These are all reverse engineering tools.

Moving on to our methodology, we'll provide an overview of the principles and techniques used to detect malicious code injection.

In the first step of our methodology, we are going to simulate both benign and malicious activity in a container. We are going to then collect these activities using different system call parameters like frequency, sequence and event logs. Now to reduce the overload and to make our detection system efficient, we will do some filtering to filter out unnecessary system calls and parameters. After that using the existing data, we are going to design a blueprint to detect malicious activities. Finally, we are going to evaluate our system against malicious and benign activities in containers.

This is our approach overview. There are two major stages here –

1. Pretraining phase
2. Runtime detection and monitoring phase

There are some section under each of these stages. We are going to discuss each of these section in more details.

To simulate normal user activity in container based application, we use 4 different applications. Our operating system is Seed Ubuntu. As local Kubernetes orchestrator, we use minikube and several reverse engineering tools like strace, sysdig and minikube dashboard are also used.

Next we simulate 4 different attacks following the attack paths from CNCF. Next step in our methodology is to Collect container activities through syscall parameters and event. We use minikube dashboard to collect event logs, Strace to collect system call sequence and Sysdig to collect system call frequency.

Then we do system call filtering. We do that for both frequency and sequence. In case of sequence, we drop unnecessary parameters like timestamp.

One of our major contributions is in filtering system call frequency. While working with frequency, we noticed that some system call does not change their frequency irrespective of what action we do in our container application. So, based on the frequency change, we find out some impactful system calls. This filtering is quite helpful to improve the efficiency and reduce the workload of our mechanism.

Our next step is to Design syscall parameter blueprint to detect malicious code injection. For event logs, we mainly focused on three aspects –

1. Type of API request
2. The service account token that makes the API request and
3. Content of the response.

In case of frequency, we set up a standard frequency for each of the system calls for benign activities. As a result, we build a threshold range for each impactful system calls. Finally, we use BoSC (Bag of system call) technique to build a standard sequence set for each action in our container application.

Now let’s take a look at Bag of system call technique. Suppose this is sequence of system calls. Here S1, S2… represent system calls. We get this sequence while doing benign activities in our application. For the simplicity of understanding, we take a Bag size of 3, in our case, S1-S2-S3. Now if this sequence does not exist in benign sequence dataset, we insert this sequence to this dataset otherwise increment its occurrence by 1. Then we slide by one system call and thus get another sequence (S2-S3-S4). We do same thing for this sequence as well. This is how, we build a standard sequence dataset for our application.

Last step of our methodology is to Malicious code injection detection using the benign behavior model. For event logs, if we notice that the API request and the content of the response are for privileged account but the service account token is from default account, we consider this request as a malicious action. In case of frequency, if a certain percentage of system calls crossed the limit of their standard threshold, we call this a malicious action in our system. Finally for sequence, if a sequence has 80% or more match with our standard sequence set, we call this a benign action otherwise malicious.

So based on our methodology, we can say our detection mechanism can be classified into two categories.

1. Log file based Detection
2. System call based Detection
   1. Frequency based
   2. Sequence based

Now if we compare our result with the existing work, we can say that our mechanism outperforms the previous works by a great margin.

From the graph of Frequency based detection, we can see that as the number of impacted field increases, our detection mechanism shows better performance. From the graph of sequence based detection, it is evident that bag size of 2 to 6 has the best result in our case.

As we have mentioned earlier, one of our key contribution is to identify key group of system calls. The X-axis of the graph on the left side represents the cumulative system calls and the Y-axis represents the performance metrics. As we can see, for 2 impacted fields, when we take into consideration nanosleep and munmap system calls performance of our detection mechanism drops by a significant amount. In this way, we identified the key group of system calls that is best suitable for our detection mechanism.

Now lets look at the major contribution of our research work

1. Multi-level monitoring of container using different monitoring tools
   1. Makes it feasible to identify an attack in a different level if an attacker manages to get past a single layer of a container without being detected
2. Path Specific detection mechanism
   1. Covers lots of potential ways to inject malicious code in a container
3. Identifying Key group of system calls
   1. Have increased the efficiency of our detection mechanism as we are no longer concerned about unnecessary system calls.

Finally, this is the list of the related works that were quite helpful throughout our research work. Thank you for your time and attention.

We appreciate the opportunity to share our research findings with you.