Contents

Projects Overview

The following document is a description of project ideas we feel are doable in a reasonable amount of time. You may pick from any of the ideas, with the option to modify them in any way. We will review any changes and OK them once we verify they are in scope. In addition, if you have a project not on the list, you may write up description for us to review and verify. The deliverables are not an exhaustive list of deliverables, but the deliverables specific to your project that we must see.

Evaluating combinations of security tooling for containers

Below are some tools which can help secure containerized applications, some of which we have covered in class:

- Open Policy Agent: https://www.openpolicyagent.org/
- Anchore: https://anchore.com/
- Secret Stores, such as Vault: https://www.vaultproject.io/
- AppArmor/SELinux (Mandatory Access Control):
  - https://wiki.ubuntu.com/AppArmor
  - https://selinuxproject.org/page/Main_Page

For this project, choose 2 or more of the above technologies (or other related technologies that implement similar functions) and deploy all of them on a distributed cloud application. You will analyze how well/not well these tools work in combination. Frequently, these tools are demoed in isolation, so it will be interesting to see how well they compose.

You will identify 2 or more vulnerabilities that you will use to test your system.

Deliverables

- In your report, address the following:
  - Which 2+ technologies you are deploying.
  - Why have you chosen these tools? What types of attacks do you hope to defend against with this deployment?
  - Which vulnerabilities have you chosen and why?
  - Do you expect to be able to defend against these vulnerabilities with your chosen defense mechanisms? Why or why not?
  - Execute your exploits. Did you successfully defend against them? What would be visible to a system administrator when the attack is underway which would help them defend the system?

Securing credentials using secret store

To store user credentials in a microservice, secret stores are used such as Hashicorp Vault: https://www.vaultproject.io/

For this project, you will create a deployment that uses Vault or another secret store to store user credentials. You should demonstrate a vulnerability or two that are patched with the use of a proper secret store. In your report, describe how secret stores are able to secure credentials and other secret data. For your related work, please discuss papers that analyze storage of secrets in the cloud, proposals for new secret store technologies, and other works that are related to secret stores.

Deliverables

- In your report, address the following:
  - How do secret stores work?
  - In what cases would you recommend using a secret store?
  - What sorts of information are typically stored in a secret store?
  - Describe the exploits your deployment is vulnerable to prior to incorporating a secret store.
  - Why does the use of a secret store patch these vulnerabilities?

Demonstrate side-channel attack between co-located microservices

Side-channel attacks rely on information leaking through a side-channel, an information channel whose existence is an artifact of the design of the system, rather than an oversight in the design. An example is a shared cache in a multiprocess computer. The time it takes to read addresses can give the program information about whether or not that address may have been loaded into the cache by a separate program.

For this project, you will implement a side-channel attack between two microservices co-located on the same network node (i.e. the same virtual machine). In your report, describe what aspects of the system topology your attack relies on, how the attack works, and what exploits are possible this attack. When discussing related work, focus on papers related to side channel attacks in the cloud, mitigations for these sorts of attacks, analyses of how common these attacks are in practice, etc.

Deliverables

- In your report, address the following:
  - Describe the two microservices involved in your side-channel attack.
  - Describe the deployment containing these two services.
  - Describe the side-channel attack you have crafted and what aspects of the system it relies on to work. Why should your attack be considered a side-channel attack?
  - How might the system administrator detect this side-channel attack? How might they defend against it?

Detecting bugs in microservices

Many software developers employ continuous integration wherein microservices are continually updated and deployed. One pitfall of this approach is that the latest code for a microservice may introduce a bug. This motivates the use of tools that detect "regressions", or new bugs that do not plague previous versions. Detecting such regressions can help prevent deploying regressions widely.

For this project, you will use one or more tools to detect bugs in development versions of a few different microservices. These microservices can be real-world examples, or contrived toy microservices. Please review the research literature in the area of bug detection in the cloud. Then, select one or more tools to implement some of the recommendations you read about in these papers.

An example

OpenDiffy is an interesting tool that is advertised to catch bugs in microservices without the need to write tests. It works by running three instances of the microservice of interest:

- Two that are known-good, identical code.
- One that may be buggy (say, a new version of the app still in development).

It mirrors requests made to the microservice to all instances, and then compares their responses. The two known-good instances are used to filter out non-deterministic differences (i.e. any differences between the two identical apps are ignored). Any remaining differences in the third app are flagged as bugs.

Diffy source code: https://github.com/opendiffy/diffy

Deliverables

- What is the related work published in the research community?

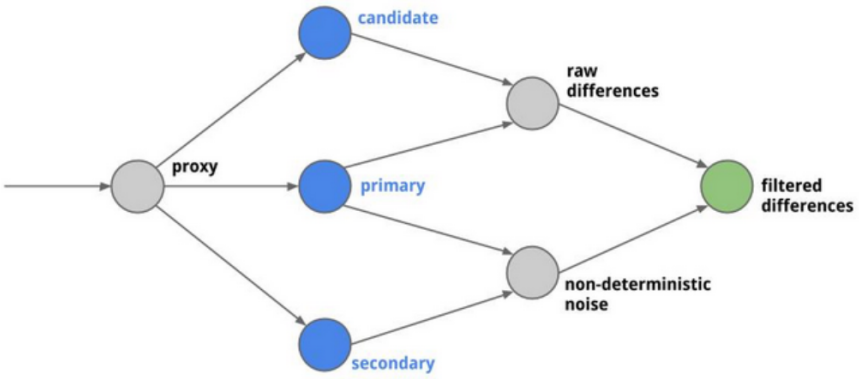

Figure 1: OpenDiffy system

  - Describe existing approaches.
- How the tool you have selected work (eg OpenDiffy).
- The pros/cons of using your tool as opposed to other testing infrastructure.
- Which microservices you evaluated and why.
- Which bugs you tested, whether or not you were successful in detecting them, and why or why not.
  - Ideally these are real bugs, but it is okay if you deliberately create bugs.

Network filtering in a distributed app

For this project, you will investigate how security can be improved by filtering network traffic using a Pod Security Policy or Data Plane filters.

More information on Pod Security Policies: https://kubernetes.io/docs/concepts/policy/pod-security-policy/

More information on Envoy, a popular proxy which can be used for filtering: https://www.envoyproxy.io/docs/envoy/latest/

In your discussion of related work, you can discuss recent developments in network security, analysis of the effectiveness of cloud microservice security practices, vulnerabilities exploited in the cloud, and more.

Deliverables

- In your report, do the following:
  - Discuss common vulnerabilities in networks and how they can be mitigated with filtering.
  - Discuss the similarities and differences between these two implementation schemes: Pod Security Policies and Network Proxies such as Envoy.
    * Are they meant to defend against the same threats? Different threats?

Scan the Internet for vulnerable devices

Shodan is a tool that collects data from IPs across the Internet: https://www.shodan.io/ For this project find CVEs that can be detected remotely (kernel based, application based, *etc*) and use shodan to discover the number of machines on the Internet that are vulnerable. Survey what the machines have in common. Create your own scan for the same CVEs and scan the Internet. Compare your results:

Deliverables

- A self written scanning tool.
- A comparison of your results with shodan.
- Figures and tables.
- A description of the CVEs.

Cilium

Cilium is a API aware network security filtering for container frameworks like Dockers and Kubernetes. Cilium uses Berkeley Packet Filters to provide simple and efficient ways to define and enforce network and app level security policies. It can be used for providing/restricting access for various containers in the same org/namespace to avoid unwanted flow of information.

About Cilium: https://cilium.io/

Source code: https://github.com/cilium/cilium

Deliverables:

- For this project you will report the following:

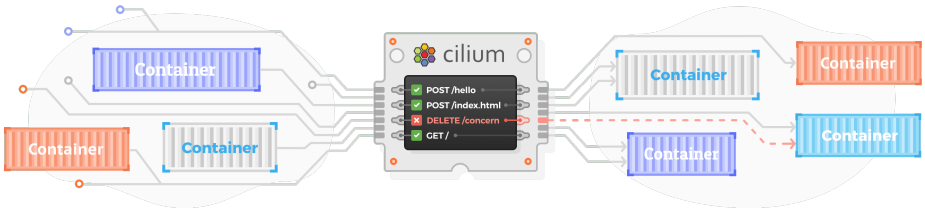

Figure 2: Cilium design

- Setup cilium with a sample microservice application (you are free to use kubernetes or independent docker containers).
- Show how cilium works for API filtering.
- How is cilium different from traditional network filtering like using iptables.
- Show API security using a demo multi-container application.

Find security vulnerabilities using kube-hunter

The tool was developed to increase awareness and visibility for security issues in Kubernetes environments. You can run kube-hunter on any self-owned clusters to scan open ports, IP addresses and gives a attackers-view of your kubernetes setup. It also indicated how exposed your cluster would be if one of the application pods is compromised.

Source Code: https://github.com/aquasecurity/kube-hunter

Deliverables

For this project you need to deploy a kubernetes cluster and a service mesh (Istio/Linkerd). Service meshes can be installed on top of microk8s that we used in class by `$microk8s.enable linkerd` or `$microk8s.enable istio`. You can deploy a simple application on top of the service mesh. Checkout meshery project to easily install service mesh and deploy applications. You need to run kube-hunter and report on existing vulnerabilities found by the tool and propose solutions to fix them.

- Report the followng
  - Application and service mesh used for the project.
  - How does the tool find vulnerabilities.
  - What are the points of vulnerabilities in the given cluster.
  - How did you fix these vulnerabilities.

DOS attack on kubernetes

Recently kubernetes was exposed to DoS attack using malicious configuration/payloads. Since then, the issue has been fixed. You can find more about the bug at https://discuss.kubernetes.io/t/announce-cve-2019-11253-denial-of-service-vulnerability-from-malicious-yaml-or-json-payloads/8349. This issue then plagued numerous projects using kubernetes like envoy and ambassador: https://blog.getambassador.io/denial-of-service-vulnerabilities-in-envoy-proxy-267c60ffc0a.

The github issue link with a sample PoC code is available at: CVE-2019-11253 which shows how intelligently crafted yaml files could leave a serious vulnerability in pro-grade systems.

Deliverables

As a part of the project, you need to pick a vulnerable version of kubernetes to demo this bug.

- You will report the following:
  - Sample application you chose to demonstrate the bug.
  - How does the bug leverage the existing vulnerability.
  - How was this bug fixed? What changes were incorporated?
  - How can these bugs be found in future/ Other vulnerabilities possible in the system?

MayaStor

MayaStor is a cloud-native declarative data plane with the goal to abstract storage resources and their differences through the data plane such that users only need to supply the what and do not have to worry about the how so that individual teams stay in control. For most use cases, microservices are designed to be stateless. But for stateful services, the microservie needs to talk to a storage service which is easy to scale and maintain isolation of data coming from various other services.

Source code: https://github.com/openebs/MayaStor

Deliverables

- For this project you will use a sample application in kubernetes along with persistence layer and use MayaStor to abstract storage resources and write directly to block devices instead of syscalls.

- Report the:
  - Working of MayaStor.
  - How does it go around using syscall to persist data from application layer.
  - What could be the benefits of this.
  - Quantify these benefits.

Evade detection

DVWA is a vulnerable web application you have exploited in previous labs. Set up DVWA with a monitoring tool, such as sysdig, and generate data of both benign and malicious activity. Use ML techniques to train a tool that can detect the malicious activity. Be sure to try many different methods and determine what sysdig output that should be used as features for training. Once a model is built attempt to create an evasion technique that will subvert your ML system.

Deliverables

- Describing the following:
  - What ML methods you tried?
  - What features you used?
  - What was your source of FPs and FNs.
  - Was ML necessary or can your detector use a simple heuristic?
  - Where you able to successfully subvert your ML model?

SQL generator

In the spirit of bashfuscator, create an SQL generator that can be used on both benign and malicious data. Create a library of SQL injection calls and feed them as input into your SQL generator. Your SQL generator must have a random aspect such that you can run it many times on the same input and receive different output.

Deliverables

- A working SQL generator.
- Documentation of the transformations your SQL generator makes.
  - Be very specific.
- Verify your generated SQL works.
- Use ML to detect malicious obfuscated vs non malicious obfuscated SQL