TRY HACK ME: Introduction to DevSecOps Write-Up

Task 1 Introduction-

Introduction

In this lab, you will get introduced to the different styles of Software Development and their evolution throughout the years. It's essential to understand how things have improved over the years and their impact on the security industry.

Learning Objectives

The history behind software development practices and how they've evolved over the years

The importance of this field and the concepts of what makes DevSecOps

DevSecOps culture and as a discipline

DevSecOps Learning Path

This is the first room in a new DevSecOps learning path being developed. The new path will cover:

Introduction to DevSecOps (Secure SDLC, Environments, and Tools)

Security of the Pipeline (Pipeline Automation, Source Code Security, Automated Code Testing, Dependency Management, CI/CD, and Environment Security)

Security in the Pipeline (Attacking the Pipeline, Exploiting Vulnerabilities in the Pipeline, Defending the Pipeline)

Infrastructure as Code (Cloud DevOps, Secret Management, Exploiting Terraform, Exploiting Vagrant, Exploiting Docker)

Answer to the questions of this section-

No Answer needed

Task 2 DevOps: A New Hope –

The story of DevOps

A long time ago, there were waterfalls in a galaxy far, far away.

Waterfall Model

This is the name given to how project management was approached back in the day (the 70s). The cycle constituted and relied on a hierarchy, where every member had a specific responsibility. For example, System admins worked tirelessly to keep everything running smoothly and afloat. Developers build and add as many features as possible, and finally, Quality Assurance (QA) engineers test the system's functionality, ensuring everything works as expected.

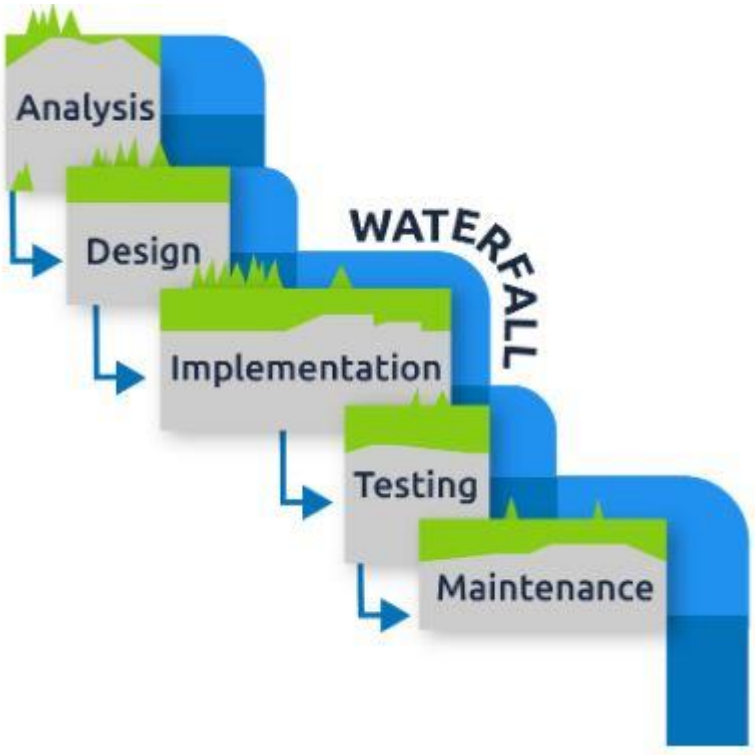


Diagram for Waterfall Software Development Model stages

If anything troubles the servers or something needs deployment, sysadmins will jump on it. If it's a code problem, devs will put out the fire. If there is anything to do with testing functionality and feedback, Quality Assurance teams will take care of it. But what if there is a flaw? A bug? Who fixes it? These situations led to many blame games and passing the baton around that created friction, things would get backlogged, and the symbiosis between teams would end up not working anymore. As the number of customer expectations grew, features and new releases increased. Responsibilities and tasks would end up being an accumulative, giant mess. Bugs and security flaws were backlogged, plenty of these unresolved, and more releases scheduled, which would be not scalable and messy. Excessive noise and pressure led to distrust, communication gaps, and friction between teams.

This popular problem-solving strategy and system became a root cause of ineffectiveness in flexibility and communication across teams.

Agile Model

With the challenges teams were facing with waterfall, businesses started developing ways that allowed more flexibility and adaptability. Somewhere in early 2000, The Agile Methodology was coined. Soon, a manifesto was released: Agile Manifesto, emphasising four values for agile development:

Diagram for Agile Software Development Model stages

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change vs following a plan

Companies now value team collaboration and rely on self-organising teams, focusing on clients and plenty of room for change and flexibility.

But something was still missing.

DevOps: A New Hope

In 2008, a conversation between Andrew Clay and Patrick Debois led to something quite revolutionary. While discussing the drawbacks of Agile, DevOps was born. After an event in Belgium the following year called "DevOpsDays," DevOps became the next buzzword, and its popularity increased.

DevOps is quite different from the previous methodologies because it focuses on driving "cultural change" to increase efficiency. It does this by uniting the magic of all teams working on a project, using integration and automation. With these ingredients, you get a cross-integration across all departments, QA+sysadmins+developers. For example, ensuring developers can now be involved in deployment and sysadmins can now write scripts, QA can figure out how to fix flaws vs constantly

testing for functionality. By introducing automation and integration of systems, these engineers can now have the same visibility at all times and interact accordingly. We will dive more into how DevOps does this in the latter rooms as we talk about pipelines, automation, Continuous Integration and Continuous Delivery (CI/CD).

Why is DevOps important?

DevOps builds a philosophy that emphasises building trust and better liaising between developers and other teams (sysadmins, QA, etc.). This helps the organisation align technological projects to business requirements, increasing the impact and value to the business as projects become more efficient and prioritised accordingly. Changes rolled out are usually small and reversible, visible to all teams involved. This ensures better contribution and communication that helps with the pace and an increased competency when delivering work.

In Summary:

Thanks to the advent of DevOps, today's development infrastructure is fully automated and operates on a self-service basis:

Developers can provide resources to public clouds without depending on IT to provision infrastructure, which in the past led to weeks to months of delays.

Continuous integration and deployment (CI/CD) processes automatically set up testing, staging, and production environments in the cloud or on-premises. They can be decommissioned, scaled, or re-configured as needed.

Infrastructure-as-Code (IaC) is widely used to deploy environments declaratively*, using tools like Terraform and Vagrant.

Organisations can now provision containerised workloads dynamically using automated, adaptive processes

*The declarative approach requires that users specify the end state of the infrastructure - for example, deploy these machines in a running state directly into an environment, automating the configuration choices throughout the workflow. The software builds it and releases it with no human interaction.

The imperative/procedural approach takes action to configure systems in a series of actionable steps. For example, you might declare to deploy a new version of the software and automate a series of steps to get a deployment-ready state. You choose when to apply those changes at the end by adding a "gate" this gate could be a button to release the changes, e.g. "deploy changes" button, after all the automated checks and new configurations pass.

In such a workflow, even a tiny problem could create a mess. Moreover, as the number of new releases increases (the actual case), the whole matter may turn disastrous. Things would surely go out of hand with an issue still unresolved and plenty of features scheduled to be released.

Read more at: https://www.appknox.com/blog/history-of-devops

Answer to the questions of this section-

Answer the questions below

What methodology relies on self-organising teams that focus on constructive collaboration?

agile — Correct Answer — Hint

What methodology relies on automation and integration to drive cultural change and unite teams?

devops — Correct Answer

What traditional approach to project management led to mistrust and poor communication between development teams?

waterfall — Correct Answer — Hint

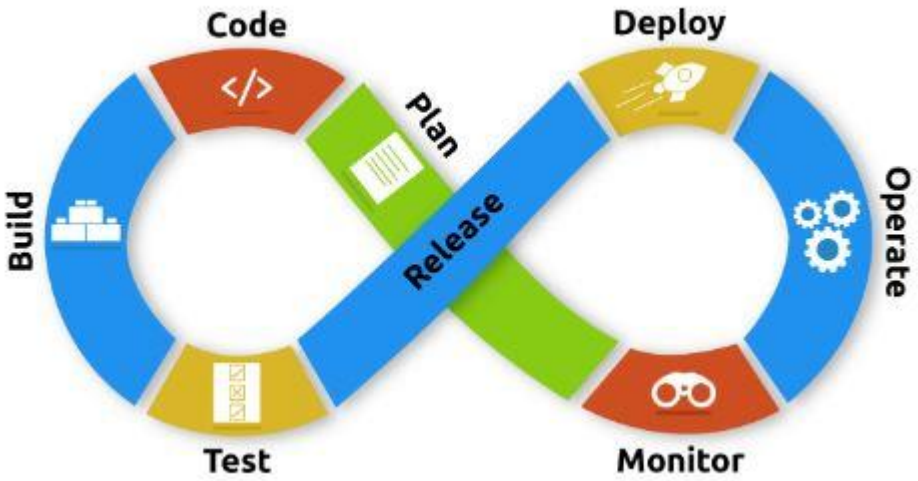
What does DevOps emphasize?

building trust — Correct Answer

Task 3 The Infinite Loop –

How does DevOps work?

DevOps is visualized as an infinite loop, describing all the comprising phases:



the infinite loop diagram describing a DevOps lifecycle

Following the infinite loop of the DevOps diagram, let's expand on some DevOps tools & processes that we'll look at as we follow the DevSecOps pathway and how they help an organization:

CI/ CD – In the previous task, we mentioned CI/CD (Continuous Integration and Continuous Deployment); CI/CD deals with the frequent merging of code and adding testing in an automated manner to perform checks as new code is pushed and merged. We can test code as we push and merge thanks to a new dynamic and routine in deployment, which takes the form of minor code changes systematically and routinely. Thanks to this change in dynamic, CI/CD helps detect bugs early and decreases the effort of maintaining modular code massively, which introduces reliable rollbacks of versions/code.

INFRASTRUCTURE AS CODE (IaC) – a way to manage and provision infrastructure through code and automation. Thanks to this approach, we can reuse code used to deploy infrastructure (for example, cloud instances), which helps inconsistent resource creation and management. Standard tools for

IaC are terraform, vagrant, etc. We will use these tools further in the pathway as we experiment with IaC security.

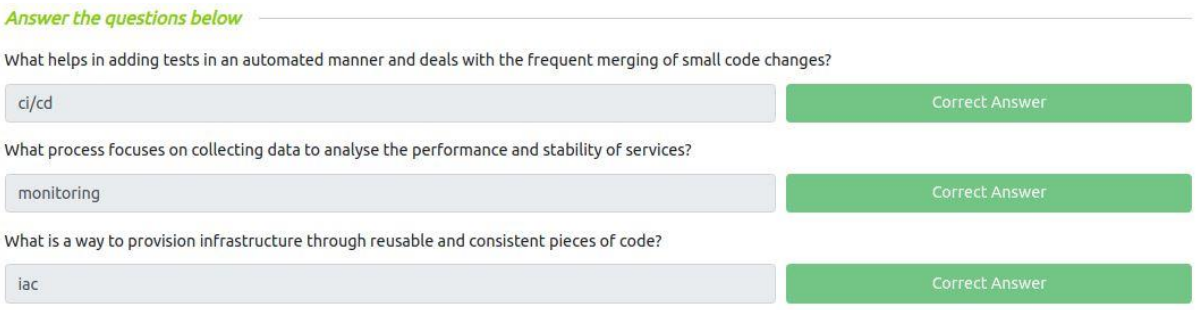
CONFIGURATION MANAGEMENT – This is where the state of infrastructure is managed constantly and applying changes efficiently, making it more maintainable. Thanks to this, lots of time is saved, and more visibility into how infrastructure is configured. You can use IaC for configuration management.

ORCHESTRATION – Orchestration is the automation of workflows. It helps achieve stability; for example, by automating the planning of resources, we can have fast responses whenever there is a problem (e.g., health checks failing); this can be achieved thanks to monitoring.

MONITORING – focuses on collecting data about the performance and stability of services and infrastructure. This enables faster recovery, helps with cross-team visibility, provides more data to analyze for better root-cause analysis, and also generates an automated response, as mentioned earlier.

MICROSERVICES – An architecture that breaks an application into many small services. This has several benefits, like flexibility if there is a need to scale, reduced complexity, and more options for choosing technology across microservices. We will look at these in more detail in the DevSecOps pathway.

Answer to the questions of this section-



Task 4 Shifting Left –

Introduction

Security can now be easily integrated because of the visibility and flexibility that DevOps introduces. You might have heard of the concept "Shifting Left." This means that DevOps teams focus on instilling security from the earliest stages in the development lifecycle and introducing a more collaborative culture between development and security.

Since security can now be introduced early, risks are reduced massively. In the past, you would find out about security flaws and bugs at the very late stages, even in production. They are leading to stress, rollbacks, and economic losses. Integrating code analysis tools and automated tests earlier in the process can now identify these security flaws during early development.

Shifting Left

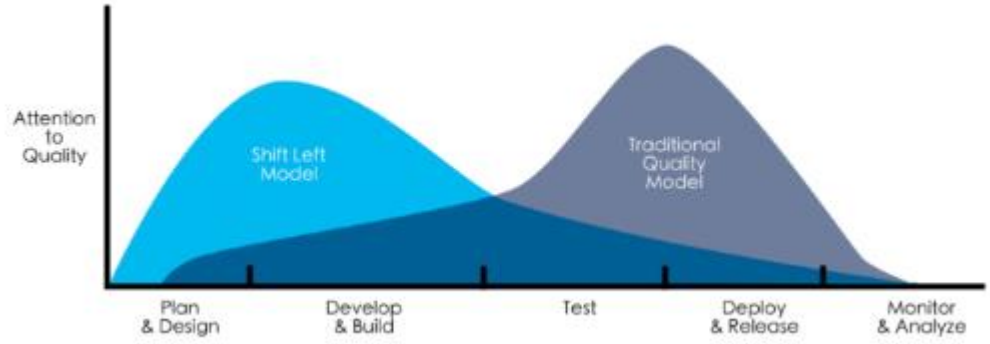
In the past, security testing was implemented at the end of the development cycle. As the industry evolved and security functions were introduced, security teams would perform various analyses and security testing in the final stages of the lifecycle.

Depending on the results of security testing, it would either permit the application to proceed for deployment into production or reject the application and pass it back to developers for remediating the flaws identified. This resulted in long delays in development and friction between teams.

Implementing security measures during all stages of the development lifecycle (shifting left) rather than at the end of the cycle will ensure the software is designed with security best practices built in. By detecting security flaws early in development, remediation costs are lower, and there would be no need to roll back changes as they are being addressed on time. This reduces cost, builds trust, and improves the security and quality of the product.

Why are we shifting left

Back in the day, before agile, developers would request infrastructure from IT and receive servers weeks or months later. Nowadays, this provisioning of infrastructure in the cloud is automated. This shift has improved development productivity and speed. However, this increased velocity can also spark security concerns and lead to flaws that can go unnoticed.



The Shift-Left approach ensures these flaws are caught early by introducing processes from the start. In this fast-paced environment, post-development security reviews of new software versions or analysis of cloud infrastructure configurations become a bottleneck. Even when problems are discovered, there is not enough time to remediate them before the next version or feature is introduced. To keep up with customer needs, they need a fast-paced environment for scaling and growth. Security is at risk of being left behind; instilling security in the beginning and adapting security testing to become flexible and adapted to the development lifecycle increases the chances of addressing things promptly.

This development approach to shifting left in DevOps can be referred to as DevSecOps.

With DevOps, security gets to be introduced early in the development cycle and this minimizes risks massively. Integrating code analysis tools and automated tests earlier in the process can lead to better identification and elimination of security loopholes. And as the software gets to the deployment stage, everything works smoothly as anticipated. Security is not an add-on. It's a must-have design feature. Blending security in DevOps would not only enhance the impact of DevOps but also eliminate a lot of other bottlenecks that could arise otherwise. With a rise in the frequency of

cyber threats and tightening regulations, adding security to DevOps is not a choice now, but certainly an obligation.

Read more at: https://www.appknox.com/blog/history-of-devops

Answer to the questions of this section-

Task 5 DevSecOps: Security Strikes Back –

DevSecOps is an approach that relies heavily on automation and platform design that integrates security as a shared responsibility. It is a culture-driven development style that normalises security as a day-to-day operation.

What is the value?

DevSecOps helps bring down vulnerabilities, maximises test coverage, and intensifies the automation of security frameworks. This reduces risk massively, assisting organisations in preventing brand reputation damage, and economic losses due to security flaws incidents, making life easier for auditing and monitoring.

How to implement this efficiently?

Culture is key. It does not work without open communication and trust. It only works with collective effort. DevSecOps should aim to bridge the security knowledge gaps between teams; for everyone to think and be accountable for security, they first need the tools and knowledge to drive this autonomy efficiently and confidently.

DevSecOps Challenges

Security Silos

It is common for many security teams to be left out of DevOps processes and portray security as a separate entity, where specialised people can only maintain and lead security practices. This situation creates a silo around security and prevents engineers from understanding the necessity of security or applying security measures from the beginning.

This is not scalable or flexible. Security should be a supportive function to help other teams scale and build security, without security teams being a blocker, but rather a ramp to promote secure solutions and decisions. The best practice is to share these responsibilities across all team members instead of having a specialised security engineer.

Lack of Visibility & Prioritisation

Aim to create a culture where security and other essential application components treat security as a regular aspect of the application. Developers can then focus on development with confidence about security instead of security departments playing police and the blame game. Trust should be built between teams, and security should promote the autonomy of teams by establishing processes that instil security.

Stringent Processes

Every new experiment or piece of software must not go through a complicated process and verification against security compliances before using developers. Procedures should be flexible to account for these scenarios, where lower-level tasks should be treated differently, and higher-risk tasks and changes are targeted for these more stringent processes.

Developers need environments to test new software without common security limitations. These environments are known as "SandBox," which are temporarily isolated environments. These environments have no connection to any internal network and have no customer data

Answer to the questions of this section-

Task 6 DevSecOps Culture –

DevSecOps Culture

Promote autonomy of teams

Whether it is a large organization or a start-up in hypergrowth, the only way to not leave security behind is by promoting the autonomy of teams. This can be done by automating processes that fit seamlessly with the development pipeline until security tests become just another type of test, like unit testing, smoke bombs, etc.

Leading by example and promoting education like creating playbooks / runbooks to spot these flaws and fix them, understand their risk, and build confidence in engineers to make the secure decision independently. The ratio of developers, platform, infrastructure engineers, etc., won't be the same as security engineers, and we must understand they can't be in every conversation. Security should act as a supporting function that focuses on building trust and creating as much overlap in knowledge between teams as possible.

Visibility and Transparency

For every tool being introduced or practised, there needs to be a supporting process that provides visibility and promotes transparency to other teams. This means that if we want to build autonomy in groups, as mentioned earlier, they need to have visibility on the security state of the service they own or maintain. For example, a dashboard visualizes the number of security flaws by the criticality of the service. This helps prioritize accordingly, so tasks don't get lost in the backlog or noise, and they can tackle flaws at the right time. The security state measure depends on the company, but it could be the number of high findings a service might or might not have which determine if it is in a good security state.

Transparency would refer to introducing tools and practices that are accessible to teams. For example, if you present a check before merging code, and the review doesn't pass and shows a message saying "signature of possible code injection flaw detected, please remediate," the developer or engineer should have access to the tool that is flagging that message. Usually, these analysis tools that flag those alerts have a UI that specifies the line in code where it's affected. They include a definition and a remediation suggestion with steps. In this example, a developer role can be created so that they have access to more information. This promotes education and autonomy by extending transparency that, traditionally, was only accessible by security teams.

Account for flexibility thanks to understanding and empathy

As mentioned earlier, instilling security in DevOps processes with visibility and transparency is no easy task. There is a factor that can determine success: the level of understanding and empathy. This means that the definition of risk for security teams is unequivocal, but for other teams, risk can be different and just as precise for them. This doesn't only apply to risk but to an umbrella of things; it ramifies into what they prioritize, how they work, and what they think is important enough to leave aside a project with a tight deadline to fix a bug.

There is no magic tool or process for everyone. It is essential to understand how developers/engineers work, what they know to be a risk, and what they prioritize. If you know their perspective, it's easier to build a process that finds common ground and has a higher chance to work vs adding another tool that creates more noise and stress for everyone. This understanding builds perspective, which accounts for empathy for how other teams work and builds a process that accounts for flexibility. This is needed because every situation might be different, deadlines might be different, and bandwidth can change over time.

As a DevSecOps engineer, suppose you took the time to understand how a team owns a service. In that case, that will have a security scanner added to its development process, worked and viewed priority; it will be easier to get their buy-in and demonstrate value. For example, if it's a platform team and owns an internal service but a core service, a risk would be a bug that disrupts the service, not a potential injection that lives behind a proxy. You would need internal credentials to exploit it. You can tune the scanners or add a triaging process that tackles the questions they would ask themselves, and this would, in turn, build trust vs crying wolf and security processes being questioned.

Answer to the questions of this section-

Answer the questions below

How can you make security scalable so it's not left behind when start ups face hypergrowth or in large corporations?

promote autonomy of teams — Correct Answer — Hint

How can you support teams in understanding risk and educating on security flaws?

visibility and transparency — Correct Answer

What are key factors to successfully instill security in the development process by accounting for flexibility?

understanding and empathy — Correct Answer — Hint

Task 7 Exercise: Fuel Trouble –

Software Development Models

SEC3PO, X Fighter Dev, Chewba-QA and S2-A2 have been assigned to discover minerals in nearby planets in the Galaxy. Watto, the Bug, is funding the mission by providing equipment and fuel to carry out this project. They are tasked to set a course and research these planets. Can you guess which Software Development Model they have used in each case to achieve the mission?

Click the View Site button on the top right, and look at the static site attached to this task. Can you figure out which approach was taken in each comic snippet?

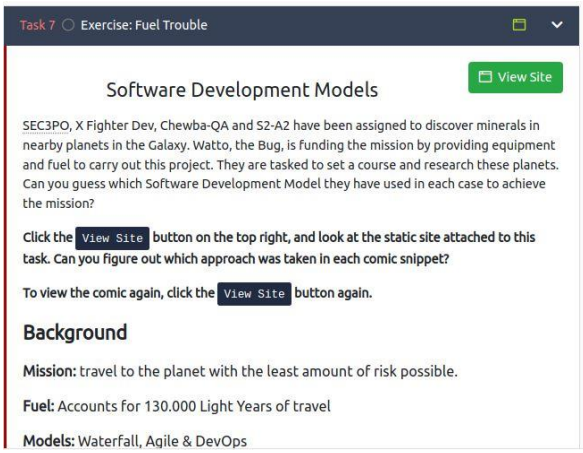
To view the comic again, click the View Site button again.

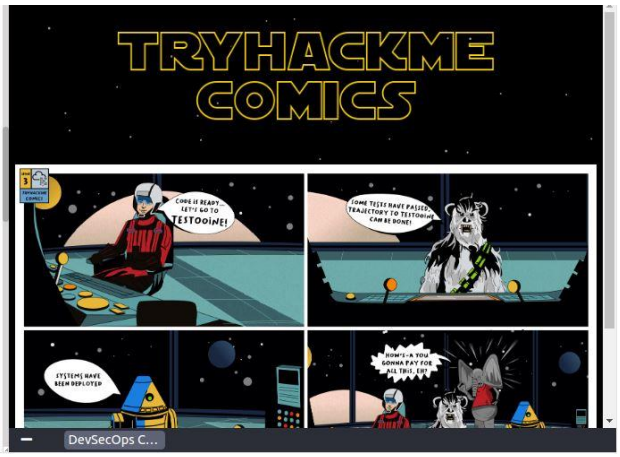
Background

Mission: travel to the planet with the least amount of risk possible.

Fuel: Accounts for 130.000 Light Years of travel

Models: Waterfall, Agile & DevOps





Planet	Distance from Spaceship
Testooine	125.000 Light Years
Hackboo	100.000 Light Years
TryHothMe	80.000 Light Years

Dagobug 60.000 Light Years

Comic 1

Some tests have passed to go to Tatooine, which was the initial decision by x fighter dev. It is decided that it is the next planet to visit.

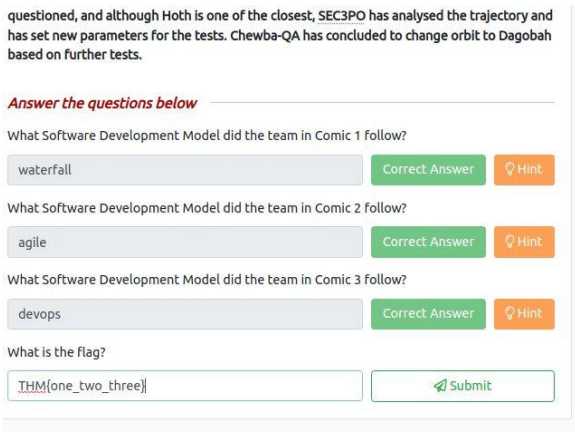
Comic 2

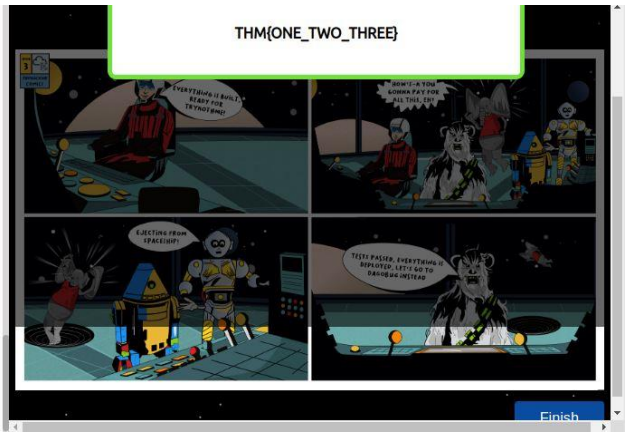
Some tests indicated that it is a high risk to travel to Tatooine; first, some tests have passed for Naboo, but S2-D2 has decided the course should be changed to Hoth.

Comic 3

The initial decision based on analysis by X fighter Dev is to travel to Hoth. Costs were questioned, and although Hoth is one of the closest, SEC3PO has analysed the trajectory and has set new parameters for the tests. Chewba-QA has concluded to change orbit to Dagobah based on further tests.

Answer to the questions of this section-





That is all for this Write-up, hoping this will help you in solving the challenges of Introduction to DevSecOps Room. Have Fun and Enjoy Hacking! Do visit other rooms and modules on TryHackMe for more learning.

-by Shefali Kumai

For more cyber security learning follow me here-

https://github.com/ctf-time

https://www.youtube.com/channel/UCf-F-eATCUXYaUVk8Xl7OOQ

https://www.instagram.com/cybersecurity.cyber_seek/

https://twitter.com/Shefali37920461