Simulated Threat Detection with AWS GuardDuty

Objective

To demonstrate applied knowledge of AWS GuardDuty by simulating cloud-based threat detection, interpreting critical alerts, and analyzing real-world security scenarios in a controlled environment.

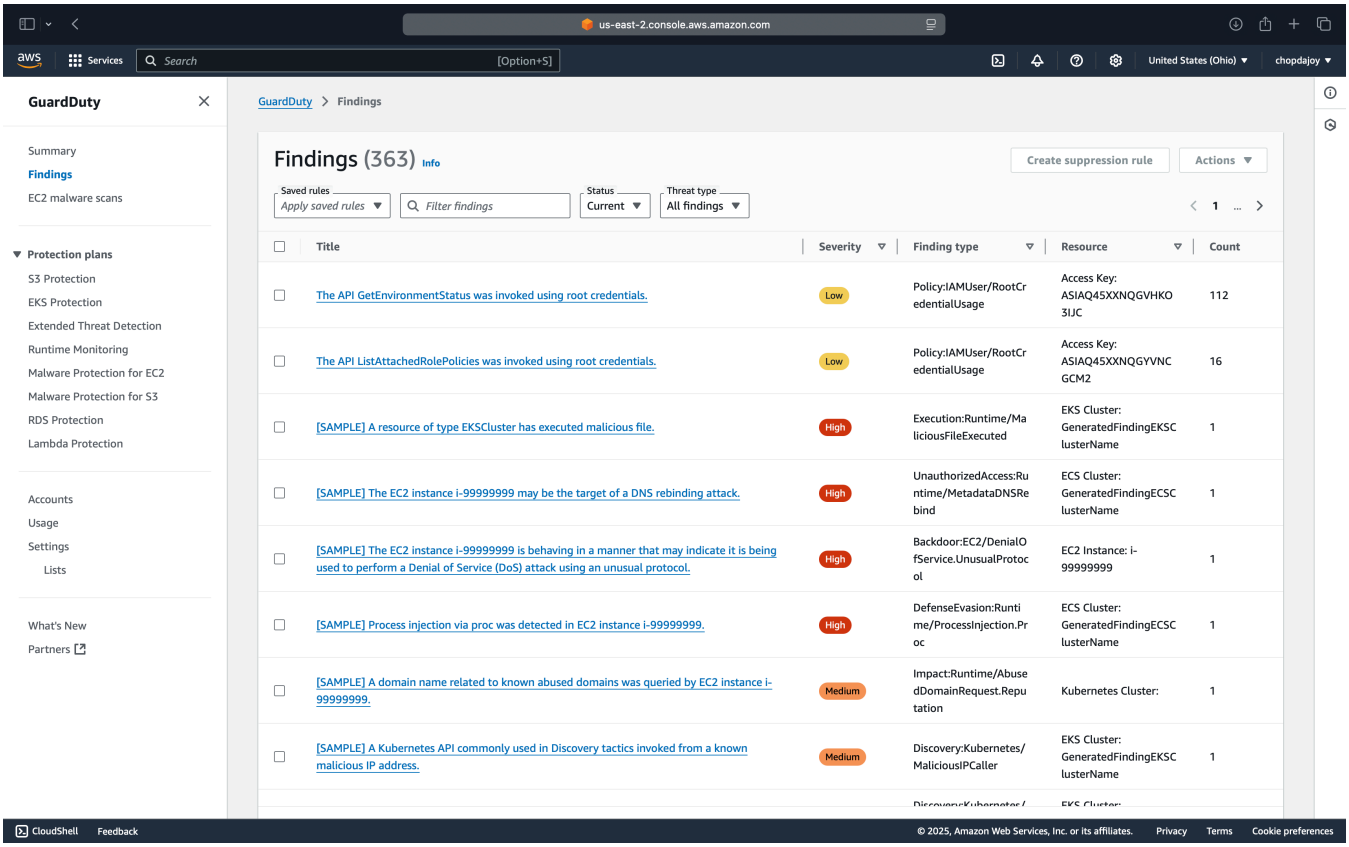
Actions Taken

1. Enabled AWS GuardDuty:

Activated GuardDuty in a fresh AWS Free Tier environment to monitor for potential security threats across EC2, IAM, S3, and RDS services.

2. Simulated Threat Data:

Used GuardDuty's built-in "Generate sample findings" feature to simulate common attack scenarios without risking live infrastructure or data.



3. Analyzed Multiple Threat Categories:

• Reconnaissance: Simulated port scanning and metadata scraping attempts

• Unauthorized Access: Sample alerts for brute-force login attempts

• Defense Evasion: Process injection and malware behavior detection

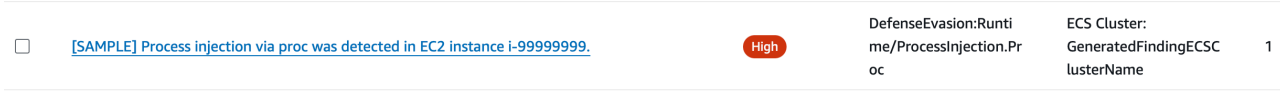
• Credential Abuse: Use of root account and malicious IP login attempts

4. Reviewed Severity and Recommendations:

Classified alerts by severity and reviewed GuardDuty's integrated remediation guidance and service-specific risks.

Deep Dive: High Severity Finding

Finding: DefenseEvasion\:Runtime/ProcessInjection.Proc



Severity: 8/8 (Critical)

Summary: Simulated detection of a process injection attempt within an EC2 instance, where one process attempts to modify another's memory via the `/proc` filesystem, a common tactic used for evading defenses, escalating privileges, or disabling security agents.

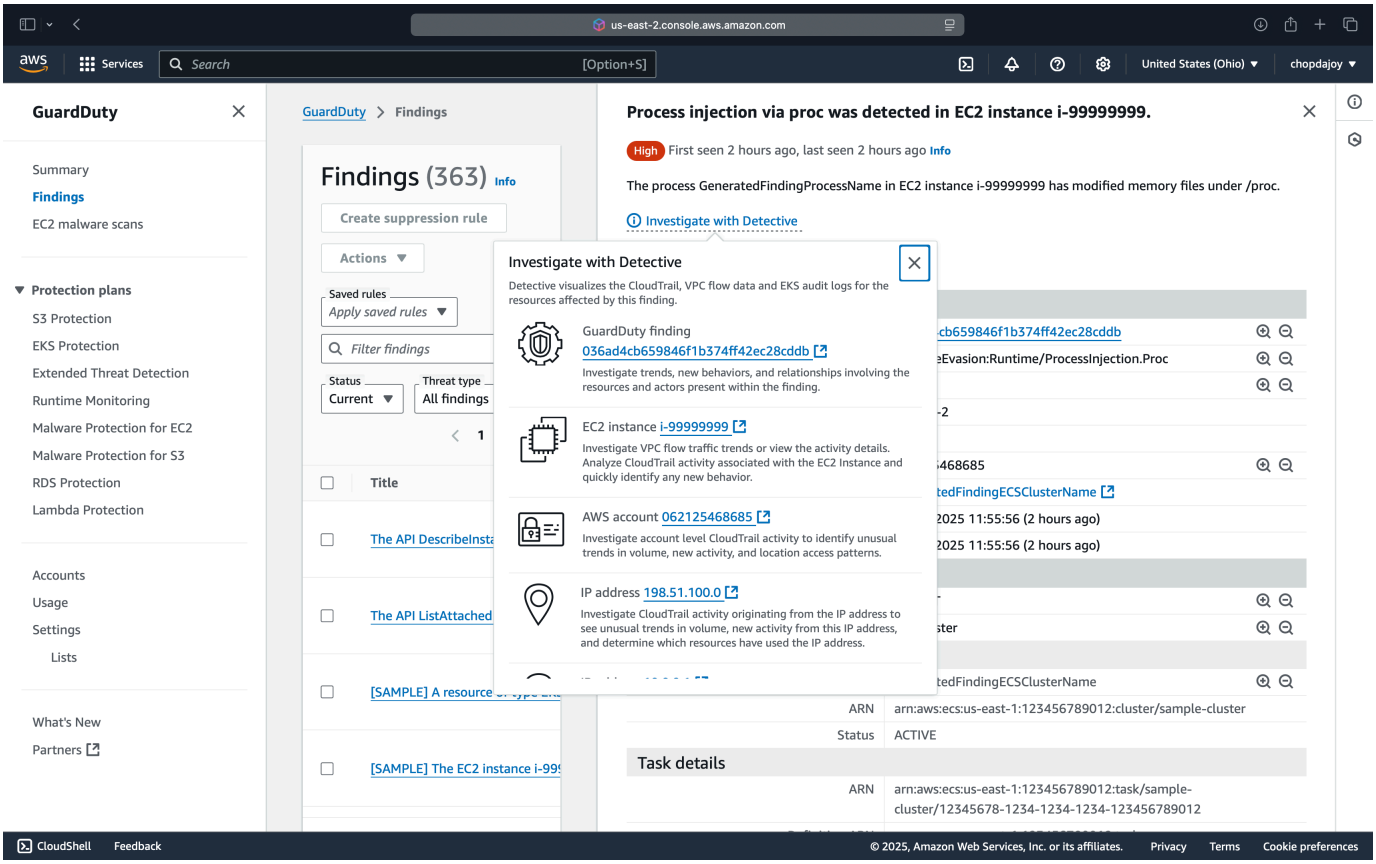
Why It Matters

Process injection is a high-risk evasion technique. If successful, attackers can hijack legitimate processes to run malicious code, bypass security controls, and maintain persistence on the instance, potentially compromising your AWS environment undetected.

How I Would Investigate

- Review VPC Flow Logs for unusual outbound traffic or connections from the compromised instance.
- Examine CloudTrail logs for unusual API activity tied to the instance or its IAM role.

- Snapshot the EC2 instance disk and memory for forensic analysis (use tools like Volatility).
- Audit the ECS cluster tasks and container images for unauthorized changes or rogue deployments.
- Check for recent process creations, hidden processes, or unexpected open ports on the instance.



How I Would Remediate

- Immediately quarantine the EC2 instance ***(i-99999999)*** by applying a restrictive security group or isolating it from the network.
- Terminate the instance after collecting forensic evidence.
- Rotate any associated IAM roles and credentials.

- Patch and redeploy a clean, hardened instance from a trusted AMI.
- Enforce runtime security tools (e.g., AWS Inspector, Falco).
- Tighten IAM permissions following least privilege principles.
- Set up GuardDuty-to-SNS or EventBridge alerts for automated response on similar findings.

Key Takeaways

- Learned how AWS GuardDuty detects sophisticated runtime evasion tactics like process injection without requiring an agent.
- Gained hands-on skills in analyzing defense evasion scenarios using AWS-native services and logs.
- Developed a structured investigation and remediation process for critical cloud runtime threats.
- Strengthened incident response readiness for advanced attack techniques targeting EC2 and containerized workloads.

As part of this exercise, I also exported the full set of simulated GuardDuty findings in .json format for offline analysis and potential integration with third-party SIEM or SOAR platforms. This artifact demonstrates my familiarity with handling threat data in structured formats, enabling advanced analysis, reporting, and automated response workflows.

This simulation reinforced my capability to operationalize AWS-native security service, interpret critical findings, and craft actionable response strategies under realistic cloud attack scenarios. Beyond technical execution, it highlights my proactive approach to security engineering, combining detection, investigation, and remediation into a cohesive incident response workflow.