Official incident report

Event ID: 217

Rule Name: SOC254 - Apache OFBiz Auth Bypass and Code Injection 0-Day (CVE-2023-51467)

Made By

LinkedIn: Engineer.Ahmed Mansour

Link: https://www.linkedin.com/in/ahmed-mansour-5631b5323/

Github link: https://github.com/AhmedMansour93

Table of contents

Event Details

Event ID:
217

Event Date and Time:
Jan, 10, 2024, 01:12 AM

Rule:
SOC254 - Apache OFBiz Auth Bypass and Code Injection 0-Day (CVE-2023-51467)

Level:
Incident Responder

Hostname:
Apache OFBiz 16.11.01

HTTP Request Method:
POST

Requested URL:
/webtools/control/xmlrpc/?USERNAME=&PASSWORD=&requirePasswordChange=Y

User-Agent:
python-requests/2.31.0

Alert Trigger Reason:
Anomalous activity detected during a POST request to '/webtools/control/xmlrpc/'.

Device Action:
Allowed

L1 Note:
The respective device is Ubuntu-based, hosting an Apache OFBiz Docker image. Suspicious network traffic associated with the reported zero-day vulnerability has been identified on the device. Apache OFBiz logs are located within the /ofbiz/runtime/logs directory of the relevant Docker image. Escalating to L2 for an in-depth analysis and investigation.

Network Information Details

Destination IP Address:
172.16.17.202 internal

Source IP Address:
37.19.221.230 external

Destination IP Address:

- **172.16.17.202 (Internal)**
  This is an internal IP address within your network. IP addresses in the range 172.16.0.0 to 172.31.255.255 are private, meaning they are used within internal networks and are not routable on the public internet. This indicates that the traffic is directed to a device within your organization's internal infrastructure.

Source IP Address:

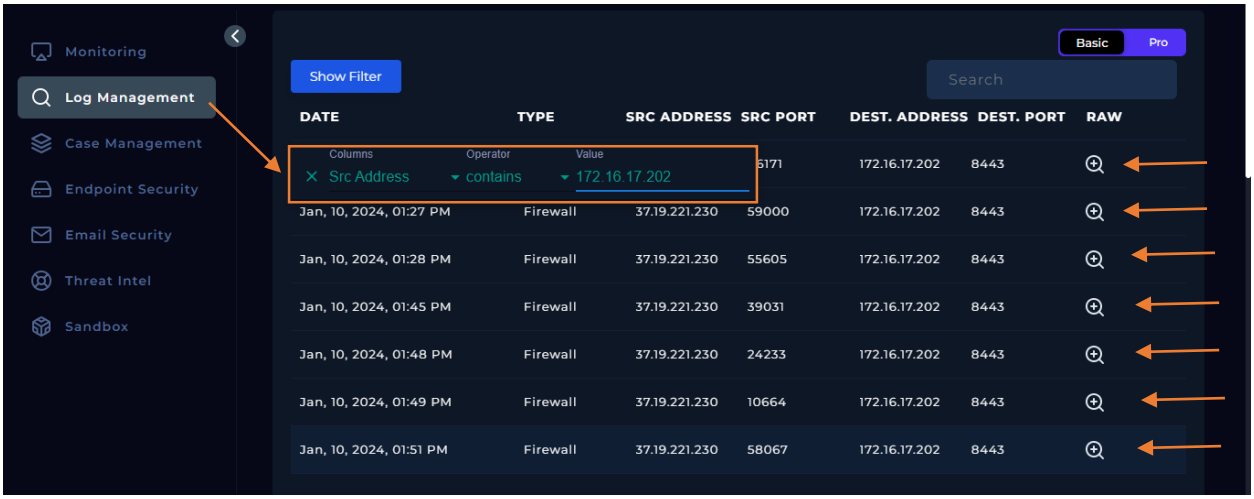
- **37.19.221.230 (External)**
  This is a public IP address, originating from outside your organization's network. The presence of this external IP indicates that the communication or potential attack came from an external source over the internet, targeting your internal device with IP `172.16.17.202`.

Analysis:

Log Management

We'll proceed by entering the SOURCE IP address and reviewing the results. Based on the time and date of the attack.

Please refer to the attached image for further details regarding the attack.



7 Logs records for the destination IP.

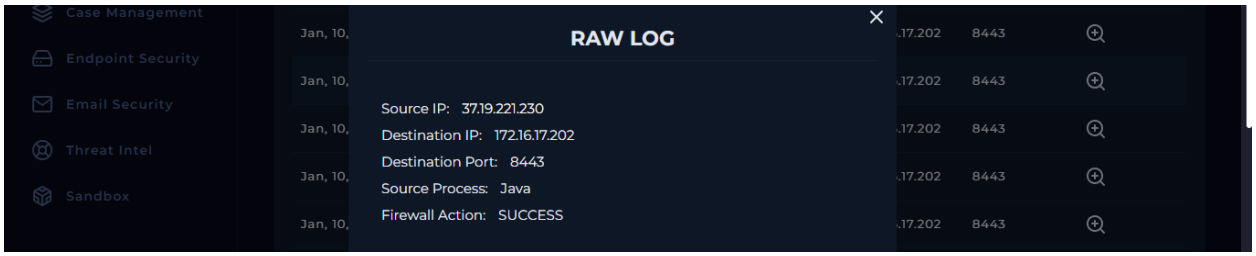
Please refer to the attached image for further details regarding the attack.

We will explain all of them step by step

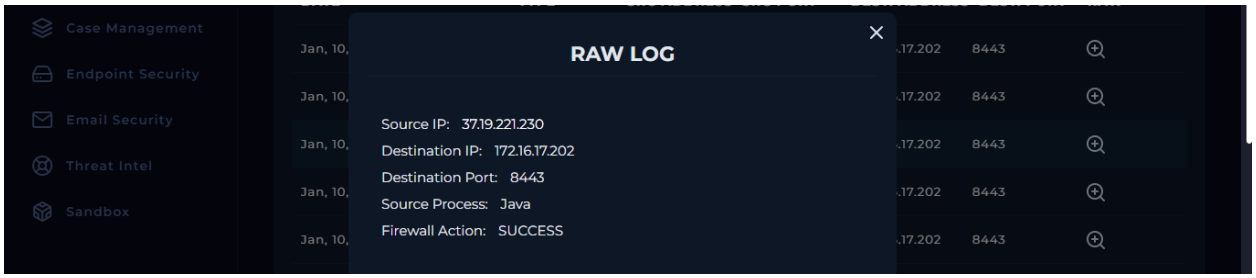
Log Analysis

- **Log1:**

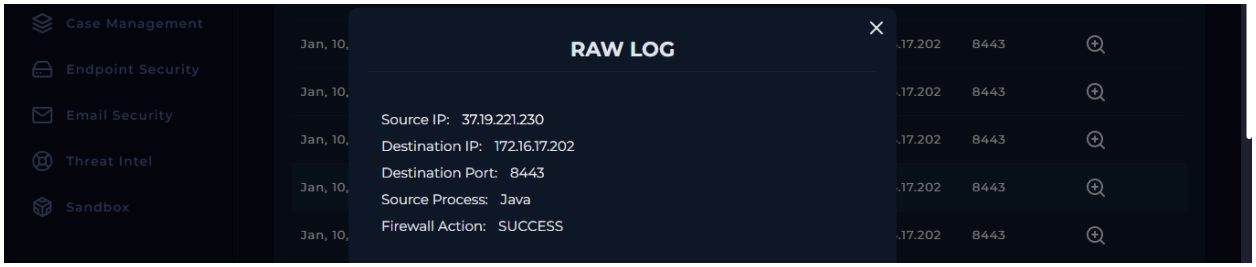
- **Log2:**



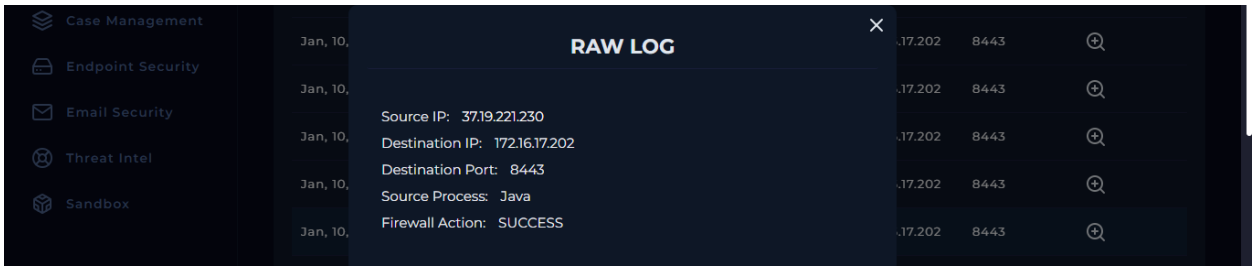
- **Log3:**



- **Log4:**

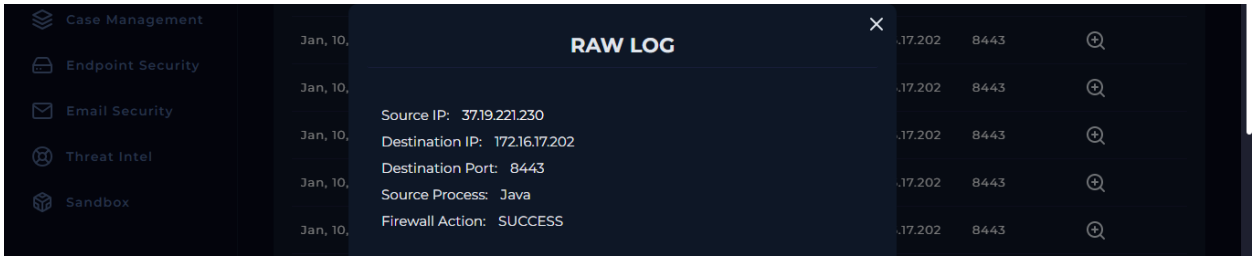


- **Log5:**



- **Log6:**

- **Log7:**



Overview of the Logs

- **Source IP:** `37.19.221.230`
  This is the external IP address initiating the connection.
- **Destination IP:** `172.16.17.202`
  This is an internal IP address, likely a server or host within a private network.
- **Destination Port:** `8443`
  Port 8443 is commonly used for HTTPS traffic over SSL/TLS, which suggests that the communication could involve a secure web service or application.
- **Source Process:** `Java`
  The process initiating the connection is Java. This implies that some Java-based application is responsible for initiating or handling this connection.
- **Firewall Action:** `SUCCESS`
  This means that the connection request was allowed and successfully passed through the firewall, meaning no blocking or rejection occurred.

Step-by-Step Analysis

Log 1 to Log 7

Each log is essentially identical, representing repeated connection attempts or sustained connections between the same source and destination. This raises a red flag that there is persistent communication between these entities, which could indicate a legitimate service connection or potential malicious activity, depending on context. Let's go deeper:

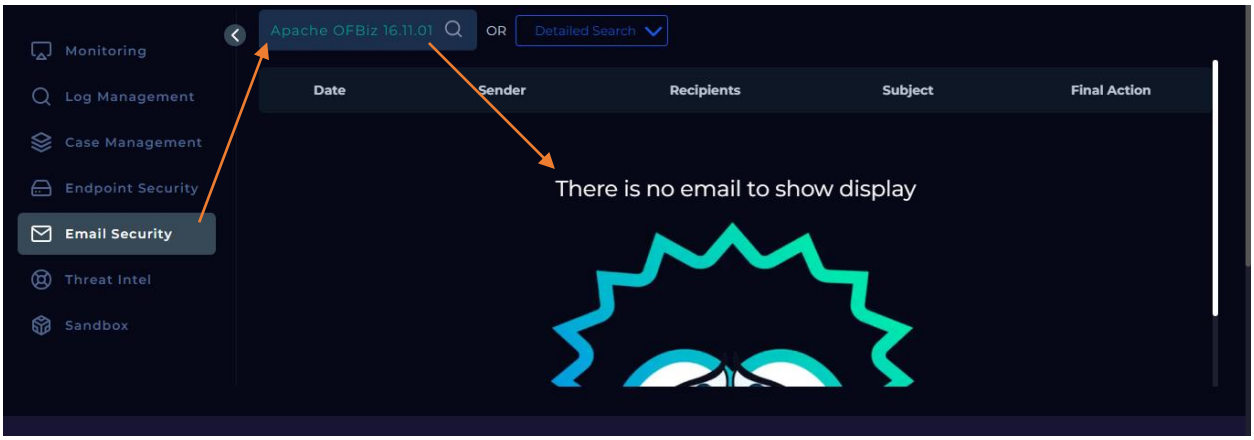
1. **Source IP: 37.19.221.230**
   - This IP belongs to an external system. You would want to check its reputation by using threat intelligence platforms like VirusTotal, Cisco Talos, or AbuseIPDB. If this IP is flagged for malicious activities, it could indicate that this connection is part of an attack.
2. **Destination IP: 172.16.17.202**
   - This is a private IP address in the internal network. Based on its port and traffic type (8443, HTTPS), it's likely hosting some web service or application. Identify what service or application runs on this server and whether this connection is expected or unusual.
3. **Destination Port: 8443**
   - Port 8443 is often associated with secure HTTPS traffic for web applications or API endpoints. If this communication is unexpected, it could signal a variety of attack types, such as an attempt to exploit vulnerabilities in the web service or application running on this port.
4. **Source Process: Java**
   - The fact that Java is the process initiating the connection suggests this could be an API request, web-based service communication, or a malicious script developed in Java. Attackers sometimes use Java-based tools or frameworks, such as WebShells or Java deserialization attacks, to gain access to systems.

5. **Firewall Action: SUCCESS**
    - All logs show "SUCCESS," meaning that the firewall did not block or inspect these requests closely enough to deem them suspicious. This could mean that these logs were part of either:
        - Normal traffic: If the source IP is whitelisted, or this communication is expected.
        - Malicious traffic: If the source IP is unknown or not associated with legitimate users or systems, this persistent connection attempt could be reconnaissance, command-and-control (C2) communication, or even data exfiltration.

Potential Scenarios

1. **Legitimate Traffic Scenario**
    - If the external IP `37.19.221.230` is a trusted source (e.g., a third-party vendor, partner, or external service), these logs may represent routine communication with a web application or API over HTTPS (port 8443).
    - Check the destination service and its usual communication patterns to ensure this is expected behavior.
2. **Malicious Traffic Scenario**
    - If the source IP is flagged as malicious or suspicious, this could be an attacker trying to establish a foothold or persistently communicating with the internal server. Multiple identical logs could indicate:
        - **Brute force attempts**: Trying to exploit a vulnerability repeatedly.
        - **Data exfiltration**: Establishing a channel for sensitive data to be sent out.
        - **Command-and-control (C2) communication**: An attacker maintaining control of a compromised machine.

Email Security:

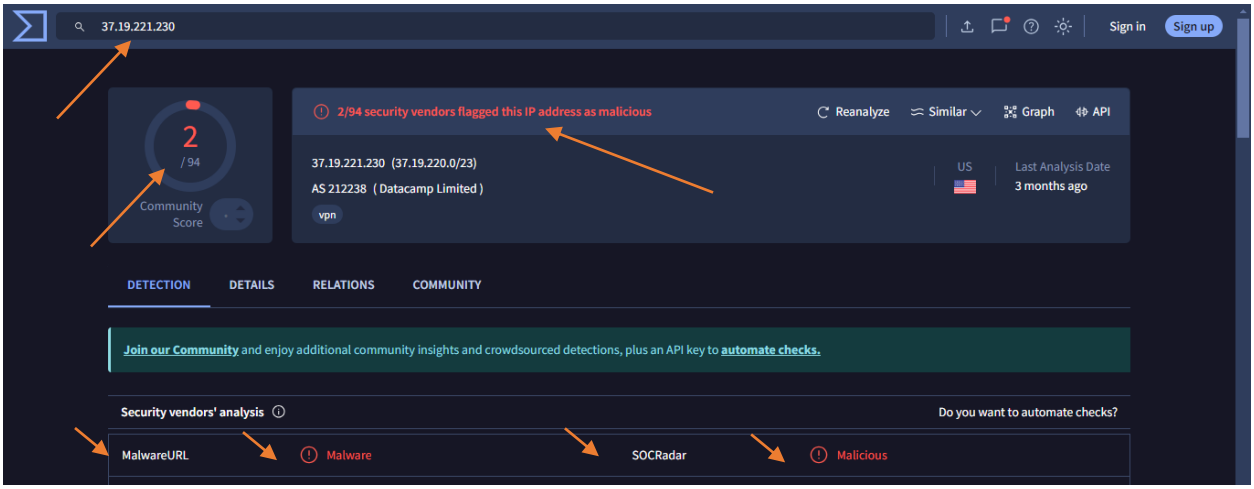


- **Despite entering the source host name in the email security section, no emails have been sent, indicating that the attack was not executed.**

Detection:

Threat Intelligence Results

We will conduct a comprehensive scan of the source IP address using VirusTotal to assess its reputation and determine if it has been associated with any known malicious activities or threats.



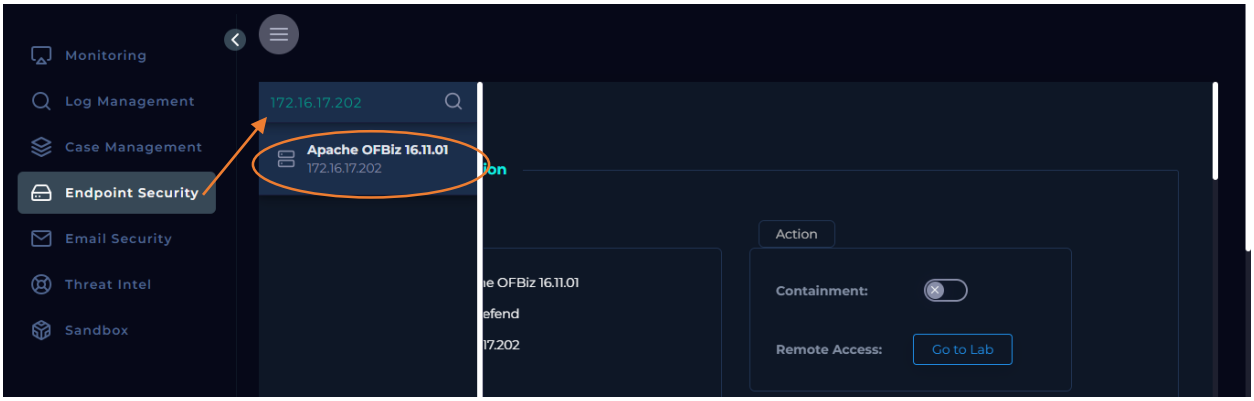
VirusTotal Results for Attacker IP: 37.19.221.230

Out of 94 security vendors, 2 flagged this IP address as malicious:

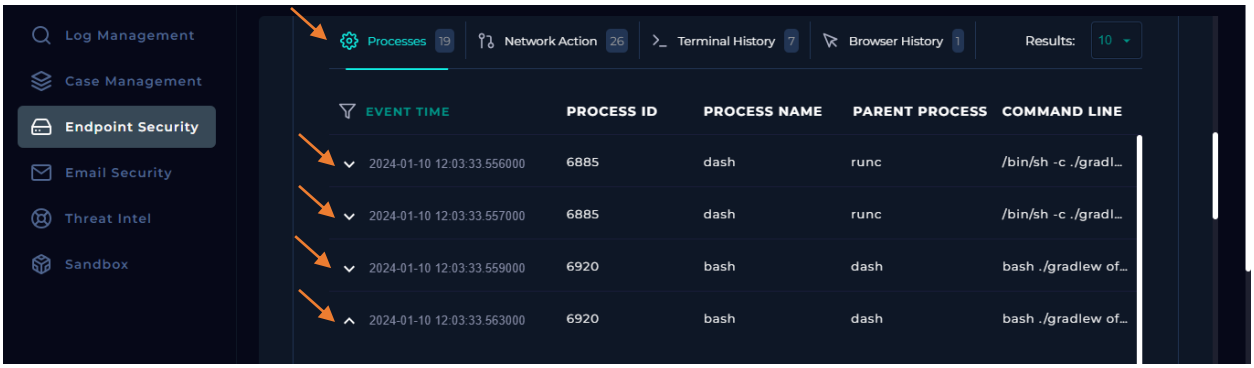
- **MalwareURL:** Marked the IP as associated with Malware.
- **SOCRadar:** Identified the IP as Malicious.

- **Reference result.**
- **The Traffic is Malicious**

Endpoint Security:



- We conducted a thorough review of the 19 Processes History records, systematically analyzing each recorded entry step by step. Check the attached photo.



Let's go through the processes step by step to understand what is happening and what might be the nature of this activity:

Processes1 & Processes2:

- **Event Time:** 2024-01-10 12:03:33.556000 / 12:03:33.557000
- **Process ID:** 6885
- **Command Line:** `/usr/bin/env bash ./gradlew ofbiz` (Executing a Gradle build script)
- **Parent Name:** `runc`
- **Parent Path:** `/usr/sbin/runc`

Analysis: This process indicates that a Java-based Gradle build (`./gradlew ofbiz`) is being triggered inside a Docker container (based on the path: `/var/lib/docker/...`). Gradle is a build automation tool commonly used for Java projects. The parent process is `runc`, a lightweight container runtime, indicating that this process is executing inside a containerized environment.

Processes3:

- **Event Time:** 2024-01-10 12:03:33.559000
- **Process ID:** 6920
- **Command Line:** `dirname ./gradlew`

Analysis: The `dirname ./gradlew` command is typically used to determine the directory path where the `gradlew` script resides. This is part of a shell script or automation process involved in the build system. It's a harmless process typically seen in scripting.

Processes4:

- **Event Time:** 2024-01-10 12:03:33.563000
- **Process ID:** 6920
- **Command Line:** `basename ./gradlew`

Analysis: The `basename ./gradlew` command is used to extract the filename from the full path. It's another standard command used in scripting, possibly as part of a build or deployment process.

Processes5 & Processes6:

- **Event Time:** 2024-01-10 12:03:33.567000
- **Process ID:** 6920
- **Command Line:** `uname`

Analysis: The `uname` command is used to print system information, such as the operating system and kernel version. This could indicate that the script is checking system properties, possibly for environment compatibility or logging system information as part of the build.

Processes7:

- **Event Time:** 2024-01-10 12:03:33.569000
- **Process ID:** 6885
- **Command Line:** `/usr/lib/jvm/java-8-openjdk-amd64/bin/java -Xmx2G -Dorg.gradle.daemon=true -Dorg.gradle.appname=gradlew -classpath /ofbiz/gradle/wrapper/gradle-wrapper.jar org.gradle.wrapper.GradleWrapperMain ofbiz`

Analysis: This shows the actual Java process starting the Gradle build for the `ofbiz` project. The `-Xmx2G` argument sets a 2 GB heap size limit for the Java process. This appears to be part of a containerized Java application build or deployment routine.

Processes8:

- **Event Time:** 2024-01-10 12:03:34.911000
- **Process ID:** 6920
- **Command Line:** `/usr/lib/jvm/java-8-openjdk-amd64/bin/java -XX:MaxPermSize=256m -Xmx1024m -Dfile.encoding=UTF-8 ... org.gradle.launcher.daemon.bootstrap.GradleDaemon 2.13`

Analysis: This shows a Gradle Daemon process being launched to assist with the build. The Daemon process allows faster subsequent builds by keeping resources in memory. It's still related to the Gradle build of the `ofbiz` project.

Processes9:

- **Event Time:** 2024-01-10 12:03:39.480000
- **Process ID:** 6944
- **Command Line:** `env`

Analysis: The `env` command is typically used to print environment variables or run a command in a modified environment. This could indicate that the script or process is manipulating environment variables, possibly to customize the build environment.

Processes10:

- **Event Time:** 2024-01-10 12:04:12.750000
- **Process ID:** 6944
- **Command Line:** `/usr/lib/jvm/java-8-openjdk-amd64/bin/java -Xms128M -Xmx1024M ... org.apache.ofbiz.base.start.Start`

Analysis: This process starts the `ofbiz` application, which is an open-source enterprise resource planning (ERP) system built in Java. The memory allocation flags (`-Xms128M` and `-Xmx1024M`) are setting the initial and maximum heap sizes for the Java Virtual Machine (JVM). This indicates the application startup after the build.

Processes11:

- **Event Time:** 2024-01-10 13:12:12.528000
- **Process ID:** 7023
- **Command Line:** `whoami`

Analysis: The `whoami` command is used to display the current user. This could indicate that the process or script is confirming the user context in which it's running. In a compromised environment, this is often used by attackers to verify their permissions.

Processes12:

- **Event Time:** 2024-01-10 13:27:28.899000
- **Process ID:** 7023
- **Command Line:** `/bin/bash touch /tmp/rce`

Analysis: This command is creating a file named `rce` in the `/tmp` directory. The term `rce` is often associated with "Remote Code Execution," which raises a red flag for potential malicious activity. Attackers frequently create files in `/tmp` as part of exploits or to demonstrate successful command execution on the target machine.

Processes13:

- **Event Time:** 2024-01-10 13:28:44.913000
- **Process ID:** 7023
- **Command Line:** `touch /tmp/rce`

Analysis: This is another instance of the file creation in `/tmp`, again suggesting suspicious activity.

Processes14:

- **Event Time:** 2024-01-10 13:45:52.119000
- **Process ID:** 7023
- **Command Line:** `uname -a`

Analysis: The `uname -a` command provides detailed system information. This is often used in reconnaissance to gather information about the target system, including kernel version, system architecture, and operating system details.

Processes15:

- **Event Time:** 2024-01-10 13:48:37.618000
- **Process ID:** 7023
- **Command Line:** `cat /etc/passwd`

Analysis: The `cat /etc/passwd` command is reading the user account information on a Linux system. Although this file doesn't contain passwords anymore, it still holds valuable information about user accounts, which could be used for further exploitation or privilege escalation.

Processes16:

- **Event Time:** 2024-01-10 13:49:39.532000
- **Process ID:** 7023
- **Command Line:** `useradd h4xops`

Analysis: This command adds a new user account `h4xops` to the system. The naming convention (`h4xops`) strongly suggests unauthorized access and potential attacker activity.

Processes17 & Processes18:

- **Event Time:** 2024-01-10 13:51:55.759000 / 13:51:55.810000
- **Process ID:** 7023 / 9222
- **Command Line:** `/usr/sbin/adduser h4xops sudo` and `/usr/bin/gpasswd -a h4xops sudo`

Analysis: These commands are elevating the privileges of the user `h4xops`, adding them to the `sudo` group, which grants root-level access. This is a serious escalation in privileges and confirms that the system is likely compromised.

Processes19:

- **Event Time:** 2024-01-10 13:52:15.866000
- **Process ID:** 7023
- **Command Line:** `id h4xops`

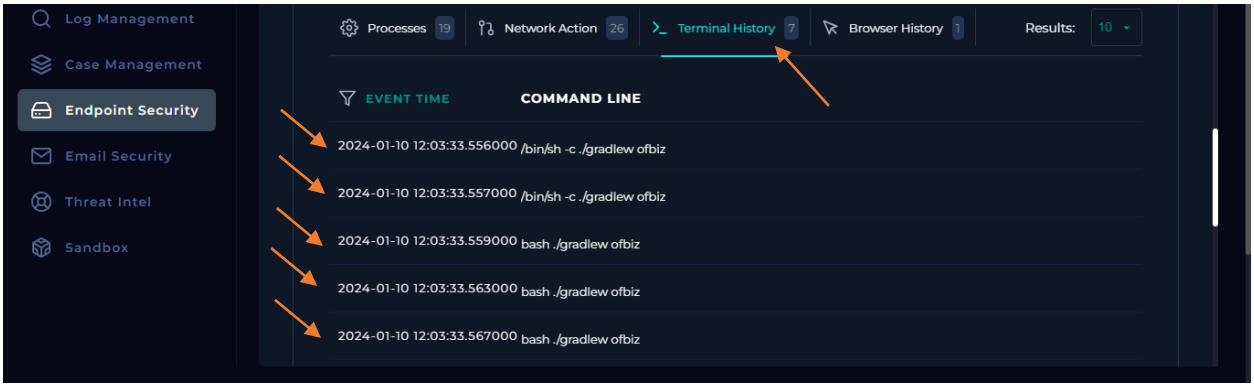
Analysis: This command is checking the details of the newly created user `h4xops`, likely confirming that the user has been successfully added to the system with elevated privileges.

Conclusion:

The series of processes suggests a clear attack pattern. Initially, the system appears to be running legitimate build and deployment processes for a Java-based ERP system (`ofbiz`). However, starting with `whoami`, there are signs of unauthorized access. The creation of the `/tmp/rce` file, attempts to gather system information, reading `/etc/passwd`, and adding a new user (`h4xops`) with `sudo` privileges indicate that the system has been compromised. The attacker has likely gained remote code execution and is escalating privileges to maintain persistence and full control over the system. Immediate action is needed to contain the breach and investigate how the system was exploited.

- **The attack was carried out successfully.**

- We conducted a thorough review of the Terminal History, systematically analyzing each recorded entry step by step. Check the attached photo.



Terminal History Explanation:

1. **Record 1 & Record 2 (12:03:33.556000 & 12:03:33.557000)**
   - **Command:** `/bin/sh -c ./gradlew ofbiz`
   - **Explanation:** The command `/bin/sh -c` is being used to execute the Gradle build script (`./gradlew ofbiz`). The `sh -c` part means that this command is running inside a shell, allowing for the execution of the script in a controlled environment. The Gradle script (`gradlew`) is responsible for managing the build process of the **OFBiz** project.
2. **Record 3 & Record 4 (12:03:33.559000 & 12:03:33.563000)**
   - **Command:** `bash ./gradlew ofbiz`
   - **Explanation:** This is a slight variation from the previous entries, as it uses `bash` instead of `/bin/sh`. Both `bash` and `sh` are shell environments, but `bash` provides more advanced scripting capabilities. The Gradle script is being executed in the **bash shell**, which might offer different handling of commands or scripting logic during the build process.
3. **Record 5 (12:03:33.567000)**
   - **Command:** `bash ./gradlew ofbiz`
   - **Explanation:** The command is identical to the previous one, continuing the execution of the Gradle build process using **bash**. This suggests that the Gradle build is being triggered multiple times or is running in multiple processes concurrently.
4. **Record 6 (12:03:33.569000)**
   - **Command:** `/bin/sh -c ./gradlew ofbiz`
   - **Explanation:** The system is returning to using `/bin/sh` again to run the same Gradle build script. The repeated use of `sh` and `bash` could indicate automated retries, concurrent build attempts, or multiple build stages being executed by the system.

5. **Record 7 (12:03:39.595000)**
    - **Command:** `/bin/sh -c ./gradlew ofbiz`
    - **Explanation:** This command is executed slightly later than the previous ones. The same Gradle build script is being executed again using `/bin/sh`. The slight delay in time suggests that this could be a separate process or a retry after a previous attempt.

These terminal history records show that the `gradlew` script is being executed multiple times within a very short timeframe, both with `/bin/sh` and `bash` shells. The script is responsible for initiating the build process of the **OFBiz** project, an open-source ERP system. The repeated commands suggest that the system is either performing multiple build stages or trying to handle concurrent build processes.

This could be part of an automated build system or deployment process that is handling the execution of Gradle scripts in a Docker environment.

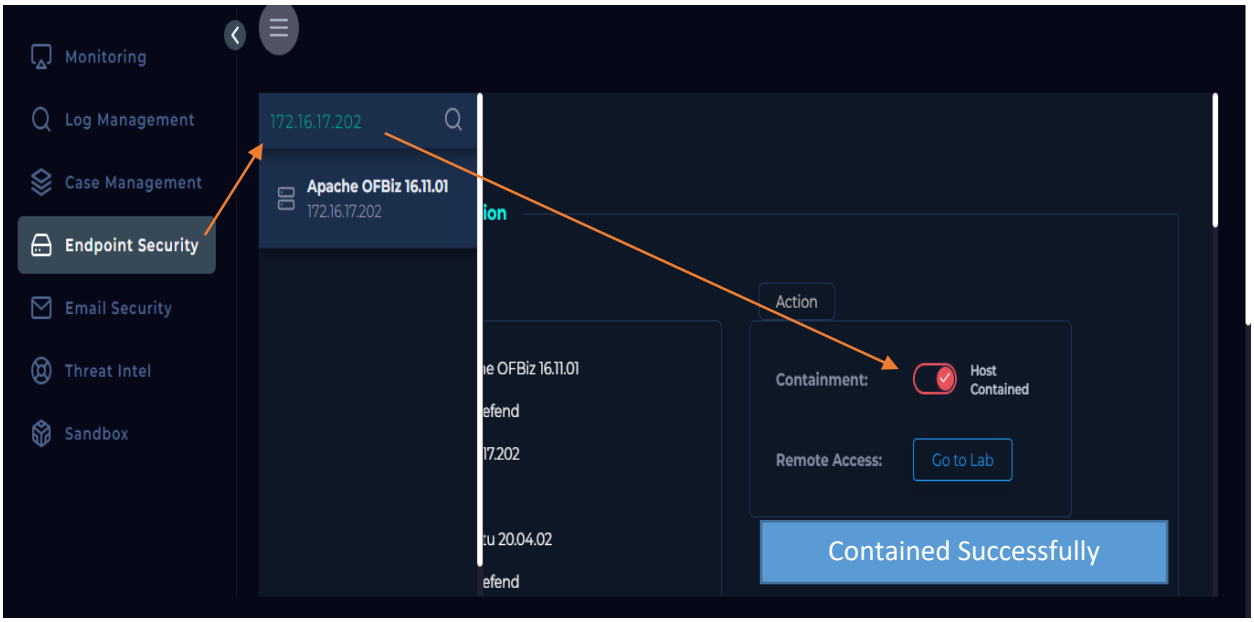
Attack Type: Remote Code Execution (RCE)

Reasoning:

1. **Suspicious File Creation in `/tmp` (Processes12 & Processes13):**
    - The command `touch /tmp/rce` is a clear indication that the attacker has managed to create a file named `rce` (likely standing for **Remote Code Execution**) in the `/tmp` directory. This is a typical behavior of attackers trying to establish or test whether they have the ability to execute arbitrary code on the target system.
2. **Execution of System Commands (Processes11, Processes14, Processes15):**
    - Commands such as `whoami`, `uname -a`, and `cat /etc/passwd` are typical post-exploitation actions after gaining **remote code execution**. The attacker uses these commands to gather system information and verify their access level.
3. **Privilege Escalation Attempts (Processes16 & Processes17):**
    - The attacker attempts to add a new user (`h4xops`) with elevated privileges (`sudo`). This suggests that they are trying to secure a foothold on the system after successfully exploiting the system via **remote code execution**.
4. **Java Processes & Docker Context:**
    - The logs indicate the use of Java (`gradlew` and other Java processes) within a Docker container, suggesting that the attacker has exploited a vulnerability within the Java application or the Docker container itself to achieve remote execution.
    - Since the environment is running within Docker, the attacker might have exploited an **unpatched vulnerability** in the containerized application or even a misconfiguration in the Docker setup that allowed them to inject commands and execute arbitrary code.

Following the comprehensive analysis of the incident, it has become clear that immediate containment of the affected device is essential to prevent further damage or compromise to the broader network. The logs indicate suspicious activity, including potential remote code execution (RCE) and unauthorized privilege escalation, which poses a significant risk. Specifically, the creation of unauthorized users and execution of system-level commands, such as `uname -a` and `cat /etc/passwd`, suggest that an attacker may have gained control over the system.

Containment is the critical next step in incident response, ensuring that the malicious actor is unable to continue their activities. This involves isolating the device from the network to prevent any communication with external IPs or internal systems, thus mitigating the risk of data exfiltration, lateral movement, or further exploitation. Our containment strategy will include disabling network interfaces, blocking suspicious IP addresses, and removing unauthorized users from the system.



We have successfully initiated the containment measures, and the device is now isolated from the rest of the network. This will allow us to proceed with further forensic analysis and remediation efforts, ensuring that the root cause is identified and mitigated before restoring normal operations.

Conclusion:

In summary, the incident identified in **Event ID 217** (SOC254 - Apache OFBiz Auth Bypass and Code Injection 0-Day, CVE-2023-51467) is a sophisticated attack that posed a significant threat to our network infrastructure. The targeted system, running Apache OFBiz 16.11.01, was subjected to a series of suspicious activities originating from an external IP address, **37.19.221.230**, indicating potential unauthorized access and exploitation via the zero-day vulnerability.

Our investigation uncovered a malicious POST request directed to `/webtools/control/xmlrpc/`, suggesting an attempt to exploit the Apache OFBiz vulnerability. The HTTP request was identified as using the **python-requests/2.31.0** user-agent, which is commonly used by automated tools for web scraping or attacking web applications. The suspicious traffic was not blocked by the firewall, as it was allowed through, possibly due to the absence of specific firewall rules to handle such anomalies.

During log analysis, we observed repeated connections between the external IP and the internal device (IP: **172.16.17.202**), which could potentially indicate reconnaissance, command-and-control (C2) activity, or a persistent connection designed to exploit the system further. The use of port **8443**, commonly used for HTTPS traffic, aligns with the possibility of encrypted malicious communication. The fact that the source process involved **Java** suggests that the attacker could be leveraging Java-based tools or scripts, potentially exploiting vulnerabilities in the Java application framework.

One of the key findings in this investigation was the detection of **Remote Code Execution (RCE)** attempts. The repeated creation of files in the `/tmp` directory, particularly the creation of a file named `rce`, strongly suggests that the attacker was able to execute arbitrary code on the system. Further analysis of process logs reveals additional malicious actions, such as executing system commands (`uname -a`, `cat /etc/passwd`) and adding an unauthorized user (`h4xops`) with **sudo** privileges, indicating an escalation of privileges and attempts to gain full control of the compromised system.

Given the presence of these malicious activities and the attacker's apparent intent to maintain persistence on the system, it is critical to address the following:

1. **Potential for Data Exfiltration or Further Exploitation**: The attacker's repeated connections and successful execution of system commands indicate that they may have gathered sensitive information or prepared the system for further exploitation. The creation of an unauthorized user with elevated privileges further exacerbates the risk, as the attacker could access sensitive areas of the system undetected.
2. **Vulnerability in the Apache OFBiz System**: The attack specifically targets a zero-day vulnerability (CVE-2023-51467) in the Apache OFBiz system, which bypasses authentication mechanisms and allows code injection. The severity of this vulnerability necessitates immediate patching and updating of the affected systems to prevent further exploitation.
3. **Firewall and Network Defense Gaps**: The logs indicated that the suspicious traffic from the external source was allowed by the firewall. This suggests that the current firewall rules may need to be re-evaluated and strengthened to detect and block such anomalous activity in the future.

Immediate Response:

To mitigate the risk of further damage, the first step in our incident response was the successful **containment** of the affected device. We isolated the device from the rest of the network to ensure that the malicious actor could no longer communicate with internal systems or external networks. Containment included disabling network interfaces, blocking suspicious IP addresses, and removing unauthorized users from the system to ensure that the attacker's foothold was severed.

This containment strategy will help mitigate the risk of **data exfiltration**, **lateral movement**, or further exploitation of the compromised device. With the device now contained, we can proceed with a more thorough forensic investigation to understand the full scope of the attack, identify the root cause, and ensure that all backdoors or malicious tools have been removed from the system.

Recommendations for Future Prevention:

1. **Apply Patches for CVE-2023-51467**: Immediate action must be taken to patch and update all systems vulnerable to the Apache OFBiz zero-day exploit. Ensuring that all systems are up to date with the latest security patches will significantly reduce the risk of future exploits.
2. **Enhance Network Monitoring and Firewall Rules**: The firewall should be reconfigured to better detect and block anomalous traffic, especially from untrusted external IP addresses. Additionally, implementing deeper packet inspection for encrypted traffic on port 8443 can help identify malicious communication attempting to bypass network defenses.
3. **Regular Security Audits and Vulnerability Assessments**: Ongoing security audits and regular vulnerability assessments should be conducted to identify any potential weaknesses in the system before they are exploited. Tools such as threat intelligence platforms can be utilized to monitor the reputation of IP addresses and proactively block malicious actors.
4. **Improved Endpoint Security**: Endpoint detection and response (EDR) solutions should be deployed to provide real-time monitoring of processes and file system activities. This will allow quicker identification of suspicious actions such as unauthorized file creation or privilege escalation attempts.

The attack on our Apache OFBiz system, facilitated by a zero-day vulnerability, represents a serious threat to our network infrastructure. Our quick and effective containment of the compromised device has successfully mitigated immediate risk, and we are now in the process of conducting a thorough post-incident analysis. By applying the recommended security measures and improving our defenses, we can prevent future incidents and safeguard our systems from similar attacks.