CI/CD For Hackers or Not

JENKINS ATTACK VECTOR

HADESS

WWW.HADESS.IO

INTRODUCTION

In the realm of continuous integration and continuous delivery (CI/CD), Jenkins has established itself as a pivotal tool, automating various phases in the software development lifecycle. While it streamlines development processes, ensuring the security of Jenkins instances and its plugins is paramount to safeguarding the integrity and confidentiality of the software being developed and the environments in which it is developed.

From a **Red Team** perspective, Jenkins can be viewed as a lucrative target. Red Team operations, which are simulated cyber-attacks conducted by ethical hackers, aim to evaluate and enhance the security posture of an organization by identifying vulnerabilities and weaknesses just as a real attacker would. Jenkins instances, often being rich in sensitive data and integrations with other critical systems, can provide a wealth of opportunities for attackers if not secured appropriately.

On the other hand, **Plugin Security Review** is crucial to ensuring that the extensions and additional functionalities provided through plugins do not introduce vulnerabilities into the Jenkins environment. Plugins, being developed by various authors and organizations, can vary significantly in terms of their security robustness. A meticulous security review of plugins is vital to preventing the introduction of vulnerabilities that could be exploited by malicious actors.

In this discourse, we will delve into the intricate details of Jenkins from a security viewpoint, exploring how Red Team operations can identify and help mitigate potential threats, and how conducting a thorough security review of plugins is pivotal in sustaining the security of the Jenkins environment. Both perspectives are integral to fortifying the security framework surrounding Jenkins and ensuring that it remains a secure and efficient tool in the software development pipeline.

DOCUMENT INFO

HADESS

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.

At Hadess, our mission is twofold: to unleash the power of white hat hacking in punishing black hat hackers and to fortify the digital defenses of our clients. We are committed to employing our elite team of expert cybersecurity professionals to identify, neutralize, and bring to justice those who seek to exploit vulnerabilities. Simultaneously, we provide comprehensive solutions and services to protect our client's digital assets, ensuring their resilience against cyber attacks. With an unwavering focus on integrity, innovation, and client satisfaction, we strive to be the guardian of trust and security in the digital realm.

Security Analyst:
Negin Nourbakhsh
Fazel Mohammad Ali Pour
Parsa Momeni
Surya Dev Singh

TABLE OF CONTENT

Executive Summary

Jenkins, a cornerstone in the CI/CD landscape, is not only pivotal in automating development pipelines but also emerges as a significant target for Advanced Persistent Threat (APT) actors. The security of Jenkins is multifaceted, involving understanding and mitigating potential attack vectors and surfaces, which is crucial for protecting the CI/CD pipeline and, consequently, the organizational assets.
Attack Vectors in Jenkins

- **Credential Exploits:** Involving tactics like credential stuffing and API key exposure.
- **Shell Exploits:** Including remote code execution and script manipulation.
- **Plugin Vulnerabilities:** Stemming from the use of outdated or misconfigured plugins.

Attack Surfaces in Jenkins

- **User Interface:** Encompassing the web interface and API endpoints.
- **Log Recorders:** Ensuring logs and tasks do not expose sensitive data.
- **Script Console:** Managing script execution and access.
- **Jenkins CLI:** Overseeing command execution and access control.
- **Credentials:** Safeguarding storage and transmission of credentials.
- **Shell:** Securing command and script execution.
- **Plugins:** Ensuring security and access control of plugins.

APT Report: Jenkins in the Crosshairs APTs targeting Jenkins exploit various vectors to gain unauthorized access, exfiltrate data, or establish a foothold within an organization's network. A typical APT scenario might involve initial access through credential stuffing, establishing a foothold via malicious plugins, privilege escalation through misconfigurations, data exfiltration, and causing impactful disruptions.

Case Study
A high-profile e-commerce platform fell victim to an attack exploiting a known vulnerability in a Jenkins plugin, leading to unauthorized access and data exfiltration.
Tools
Shodan, a search engine for internet-connected devices, can be utilized to locate Jenkins servers using specific dorks, such as searching for web pages with a title that includes "Dashboard [Jenkins]" or Jenkins servers based on the hash of the favicon.

Jenkins Important Files: A Red Teaming Perspective Red teaming, simulating cyber-attacks, identifies vulnerabilities in systems like Jenkins. Files and directories, such as **/bitnami/Jenkins/home/users.xml** (storing user data) or **/bitnami/Jenkins/home/credentials** (storing encrypted credentials), can be exploited at different stages of a red team operation, from reconnaissance to cleanup, to gain unauthorized access, decrypt sensitive data, or manipulate processes.

Jenkins: Critical Paths and API Endpoints in Red Teaming
Understanding critical paths and API endpoints in Jenkins is vital for both attackers and defenders in red teaming. Paths like **/bitnami/Jenkins/home/users/** and API endpoints like **/whoAmI/api/json** can be exploited across various stages, from reconnaissance to impact, to extract secrets, automate login attempts, initiate malicious builds, or exploit vulnerabilities in plugins.

Jenkins Plugin Security and Development Guidelines
Ensuring the security of Jenkins plugins is paramount. Developers must adopt a security-first approach in plugin development and usage to prevent vulnerabilities and maintain the stability and security of the Jenkins environment. This involves adhering to best practices and guidelines that prioritize security in the development, deployment, and management of plugins.

Key Findings

This technical summary provides a succinct overview of various aspects of Jenkins security, from understanding and mitigating attack vectors and surfaces to exploring critical paths and API endpoints from a red teaming perspective, and ensuring the secure development and management of Jenkins plugins. The insights and scenarios presented underscore the importance of a robust security posture in managing and utilizing Jenkins in CI/CD pipelines.

- Attack Vectors in Jenkins
- Jenkins Important Files: A Red Teaming Perspective
- Jenkins: Critical Paths and API Endpoints in Red Teaming
- Jenkins Plugin Security and Development Guidelines
- 10 Jenkins Plugins for White Box Testing

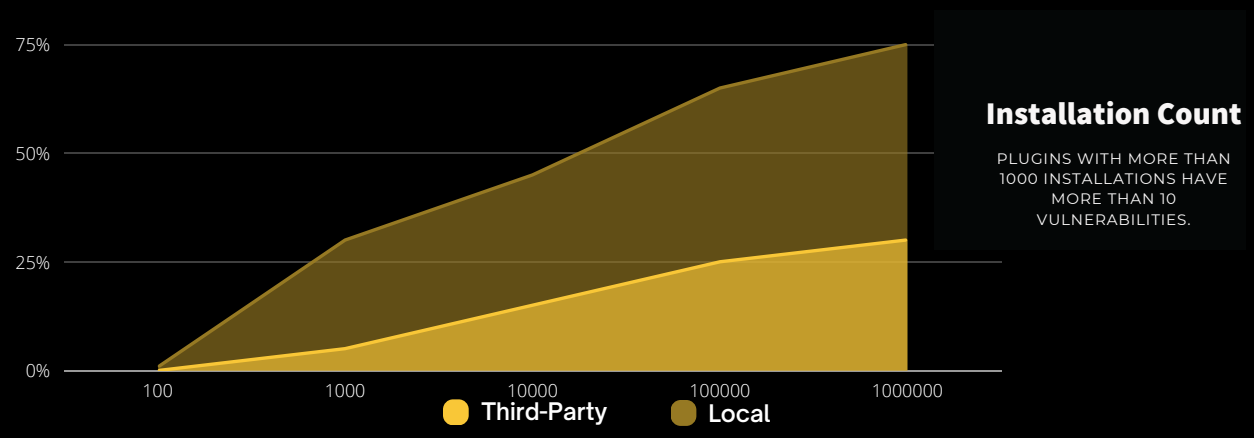
Abstract

In the intricate web of Continuous Integration and Continuous Delivery (CI/CD) pipelines, Jenkins emerges as a quintessential automation server, facilitating the seamless building, deploying, and automating of projects across various domains. While Jenkins propels development endeavors with its versatile capabilities, it inadvertently presents a lucrative attack surface for malicious entities, particularly Advanced Persistent Threat (APT) actors. This document embarks on a meticulous exploration of the security perspective of Jenkins, intertwining the realms of Red Team operations and Plugin Security Review. Red Team operations, characterized by simulated cyber-attacks, unveil potential vulnerabilities and weaknesses within Jenkins, providing a roadmap towards fortified security protocols. Concurrently, a thorough examination of Jenkins plugin security is imperative to ensure that the extensions and additional functionalities do not inadvertently become conduits for cyber threats. Through a dual lens of proactive threat simulation and meticulous plugin security review, this discourse aims to weave a tapestry of strategies, insights, and best practices that underpin a robust security framework for Jenkins, safeguarding organizational assets and ensuring the integrity of the CI/CD pipeline amidst the ever-evolving cyber threat landscape.

Checked plugins are divided based on the type of functionality and application of the plugin, as well as the amount of installation and the amount of vulnerabilities detected in each package.

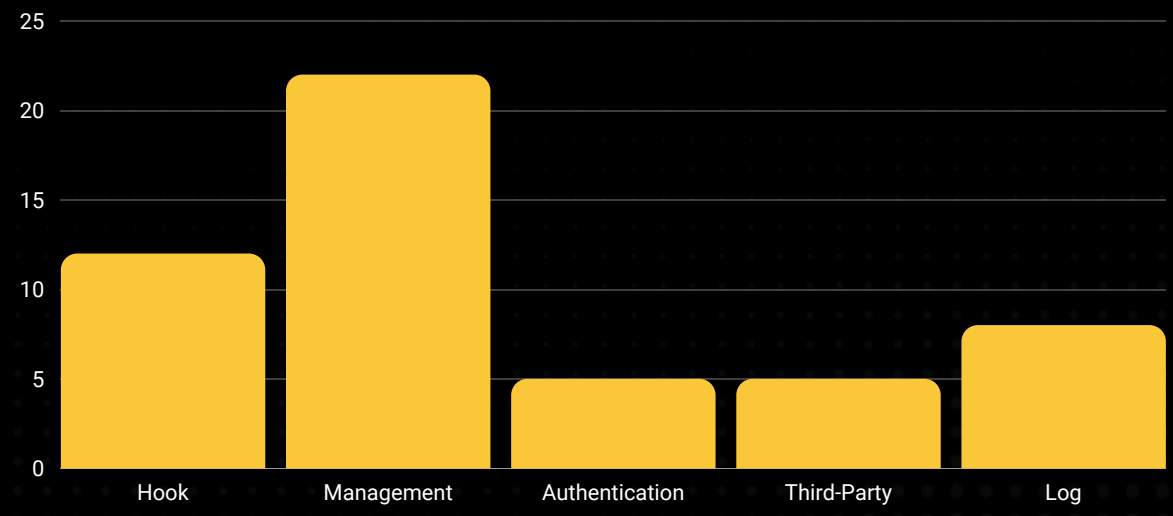
Vulnerabilities Type

The type of vulnerability detected in plugins by the number of plugins installed



Plugin Type

A vulnerability has been found for each type of plugin

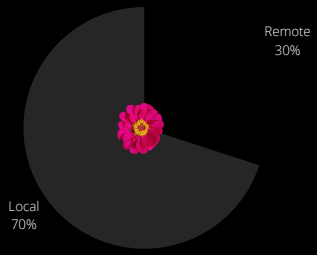


Plugin List:

No	Plugin
1	Display URL API
2	Script Security
3	Pipeline: Supporting APIs
4	LDAP
5	SnakeYAML API
6	Pipeline Graph Analysis
7	Pipeline: Input Step
8	Pipeline: Stage Step
9	Pipeline: Job
10	JUnit
11	Telegram
12	Google Metadata

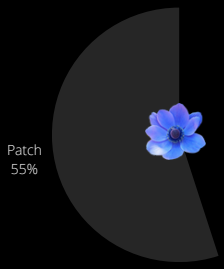
No	Plugin
13	Linkedin
14	Krypotwire
15	warrior
16	Compressed File Viewer
17	CIFS
18	Git Forensics
19	apk size Viewer
20	ZOOM

Jenkins, a widely-adopted automation server that facilitates continuous integration and continuous delivery (CI/CD), is enriched by a vast ecosystem of plugins that extend its functionality and integration capabilities. However, this pluggable architecture, while instrumental in enhancing Jenkins' utility, also introduces a myriad of attack surfaces that can be exploited by adversaries. From an offensive security standpoint, understanding, and exploring these plugin-related attack surfaces is pivotal to identifying vulnerabilities and fortifying defenses.



Impact

Any restriction on their side could be removed per vulnerability



Authentication/Authorization

A plugin with +10000 installations was found in plugin list to be vulnerable to broken authentication and authorization

HADESS.IO

Jenkins Attack Vector

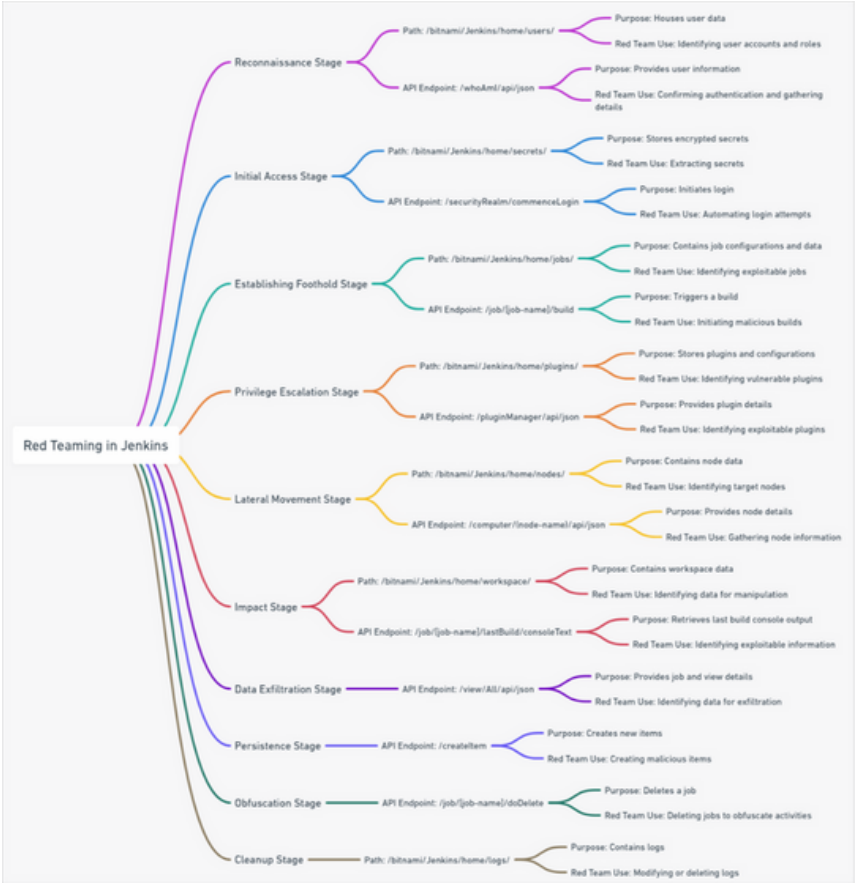
Jenkins

01

ATTACKS

The plugin architecture, while amplifying Jenkins' capabilities, introduces a spectrum of attack surfaces that can be exploited in offensive security strategies. Understanding these attack vectors and implementing robust defensive mechanisms is crucial to safeguarding Jenkins environments against potential threats and ensuring the security and integrity of CI/CD pipelines. This exploration into plugin attack surfaces and offensive security strategies aims to equip security professionals with the insights needed to protect Jenkins instances against plugin-related threats and vulnerabilities.

Attack Vectors in Jenkins



🛡️ Credential Exploits

- **Credential Stuffing:** Leveraging automated login requests to illicitly gain access.
- **API Key Exposure:** Unintentionally revealing API keys in public repositories or logs.

🐚 Shell Exploits

- **Remote Code Execution (RCE):** Enabling attackers to execute malicious code on remote servers.
- **Script Manipulation:** Injecting malicious scripts to alter processes.

🛠️ Plugin Vulnerabilities

- **Outdated Plugins:** Employing plugins that are outdated or contain vulnerabilities, rendering them exploitable.
- **Misconfigured Plugins:** Utilizing plugins that are incorrectly configured, thereby revealing sensitive information.

🎯 Attack Surfaces in Jenkins

User Interface (UI)

- **Web Interface:** Serving as the primary interaction point for both users and administrators.
- **API Endpoints:** Facilitating interaction points for automated bots and integrations.

Log Recorders

- **Logs:** Managing and storing logs in a manner that prevents the exposure of sensitive information.
- **Tasks:** Ensuring that logs of tasks do not inadvertently reveal sensitive data.

Stages of Attacks

Privilege Escalation
Initial Access
Lateral Movement
Credential Access
Impact

Script Console:
Script Execution: Ensuring only authorized scripts are executed.
Script Access: Limiting access to the script console to authorized personnel.

Jenkins CLI:
Command Execution: Ensuring only authorized commands are executed.
Access Control: Limiting CLI access to authorized personnel.
Credentials:

Storage: Ensuring credentials are stored securely.
Transmission: Ensuring credentials are transmitted securely.
Shell:

Command Execution: Ensuring shell commands are executed securely.
Script Execution: Ensuring shell scripts are executed securely.
Plugins:

Plugin Security: Ensuring plugins do not introduce vulnerabilities.
Plugin Access: Ensuring only authorized personnel can manage plugins.
APT Report: Jenkins in the Crosshairs
Advanced Persistent Threats (APTs) targeting Jenkins aim to exploit the aforementioned vectors to gain unauthorized access, exfiltrate data, or establish a foothold within an organization's network.

APT Scenario:
Initial Access: Utilizing credential stuffing or exploiting vulnerabilities.
Establishing Foothold: Installing malicious plugins or scripts.
Privilege Escalation: Exploiting misconfigurations or vulnerabilities.
Data Exfiltration: Utilizing outbound connections to transfer data.
Impact: Disrupting CI/CD pipelines, data theft, or deploying malicious code.
Case Study:

Target: A high-profile e-commerce platform.
Attack Vector: Exploited a known vulnerability in a plugin.
Impact: Unauthorized access to source code and data exfiltration.

Tools
below are some example Shodan dorks that can be used to find Jenkins servers:

1. `http.title:"Dashboard [Jenkins]"`

   - This dork searches for Jenkins servers by looking for web pages with a title that includes "Dashboard [Jenkins]".
2. `http.favicon.hash:-1372880220`

   - This dork searches for Jenkins servers based on the hash of the favicon (the small icon displayed on the tab in a web browser).

3. `port:"8080" product:"Jenkins"`

   - This dork searches for Jenkins servers running on port 8080.
4. `jenkins`

   - A simple dork that searches for instances of Jenkins on the web.

Jenkins Important Files: A Red Teaming Perspective

Red teaming involves simulating cyber-attacks to identify vulnerabilities and weaknesses in systems before actual attackers exploit them. In the context of Jenkins, various files and directories can be of particular interest during different stages of a red team operation. Let's explore how the mentioned files might be leveraged and additional files that might be of interest.

1. Reconnaissance Stage:
/bitnami/Jenkins/home/users.xml
Purpose: Stores user data, including usernames and potentially information about their roles.
Red Team Use: Identifying potential user accounts to target for access.
/bitnami/Jenkins/home/updates/defaults.json
Purpose: Contains information about default update sites and update centers.
Red Team Use: Identifying outdated plugins or configurations that can be exploited.

2. Initial Access Stage:
/bitnami/Jenkins/home/credentials
Purpose: Stores encrypted credentials used by Jenkins.
Red Team Use: Decrypting or leveraging credentials to gain unauthorized access.

3. Establishing Foothold Stage:
/bitnami/Jenkins/home/secret.key
Purpose: Used for encrypting sensitive data in Jenkins.
Red Team Use: Potentially decrypting sensitive data or impersonating the Jenkins instance.

4. Privilege Escalation Stage:
/bitnami/Jenkins/home/node-monitor.xml
Purpose: Stores configuration and status information about node monitoring.
Red Team Use: Identifying misconfigurations or vulnerabilities in node monitoring that can be exploited for privilege escalation.

5. Lateral Movement Stage:
/bitnami/Jenkins/home/logs/tasks
Purpose: Contains logs of tasks performed by Jenkins.
Red Team Use: Identifying tasks that could be manipulated or identifying additional systems to target.

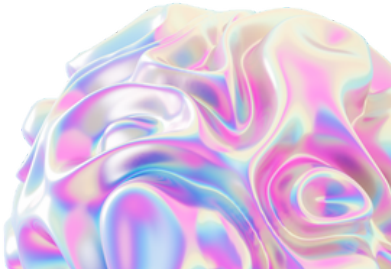
6. Impact Stage:
/bitnami/Jenkins/home/jobs/[job-name]/builds/
Purpose: Stores build histories and configurations for specific Jenkins jobs.
Red Team Use: Manipulating or deleting build histories to disrupt CI/CD pipelines and operations.
Additional Files of Interest:

7. Data Exfiltration Stage:
/bitnami/Jenkins/home/workspace/
Purpose: Contains data related to the workspace of each job.
Red Team Use: Identifying sensitive data or artifacts that can be exfiltrated.

8. Persistence Stage:
/bitnami/Jenkins/home/plugins/
Purpose: Stores Jenkins plugins and their configurations.
Red Team Use: Installing malicious plugins or manipulating existing ones to maintain access.

9. Obfuscation Stage:
/bitnami/Jenkins/home/config.xml
Purpose: Contains global configuration options for Jenkins.
Red Team Use: Modifying configurations to obfuscate malicious activities or disable security features.

10. Cleanup Stage:
/bitnami/Jenkins/home/logs/
Purpose: Contains various logs related to Jenkins operations.
Red Team Use: Deleting or modifying logs to erase traces of the attack.



Jenkins: Critical Paths and API Endpoints in Red Teaming

1. Reconnaissance Stage:
API Endpoint: /whoAmI/api/json
Purpose: Provides information about the authenticated user.
Red Team Use: Confirming successful authentication and gathering user details.

2. Initial Access Stage:
API Endpoint: /securityRealm/commenceLogin
Purpose: Initiates user login.
Red Team Use: Automating login attempts for credential stuffing or brute-force attacks.

3. Establishing Foothold Stage:
API Endpoint: /job/[job-name]/build
Purpose: Triggers a build for a specific job.
Red Team Use: Initiating malicious builds or disrupting CI/CD pipelines.

4. Privilege Escalation Stage:
API Endpoint: /pluginManager/api/json
Purpose: Provides details about installed plugins.
Red Team Use: Identifying outdated or vulnerable plugins for exploitation.

5. Lateral Movement Stage:
API Endpoint: /computer/(node-name)/api/json
Purpose: Provides details about a specific node.
Red Team Use: Gathering information about nodes for targeted attacks.

6. Impact Stage:
API Endpoint: /job/[job-name]/lastBuild/consoleText
Purpose: Retrieves the console output from the last build of a job.
Red Team Use: Identifying errors or information to further exploit the environment.
Additional Paths and API Endpoints:

7. Data Exfiltration Stage:
API Endpoint: /view/All/api/json
Purpose: Provides details about all jobs and views.
Red Team Use: Identifying valuable data or configurations to exfiltrate.

8. Persistence Stage:
API Endpoint: /createItem
Purpose: Endpoint to create new items (e.g., jobs).
Red Team Use: Creating malicious jobs or items to maintain access.

9. Obfuscation Stage:
API Endpoint: /job/[job-name]/doDelete
Purpose: Deletes a specified job.
Red Team Use: Deleting jobs to obfuscate activities and erase traces.

Jenkins Plugin Security and Development Guidelines

Jenkins, a widely-used CI/CD tool, allows developers to extend its functionality through plugins. However, the development and use of plugins must be approached with a security-first mindset to prevent vulnerabilities and ensure the stability of the Jenkins environment.

Jenkins Plugin Security:
Dependency Scanning:

Ensure all dependencies of your plugin are free from known vulnerabilities.
Use tools like OWASP Dependency-Check to identify and fix issues.
Code Review:

Conduct regular code reviews to identify potential security issues.
Ensure that no secrets or sensitive data are hardcoded in the plugin code.
Access Control:

Implement strict access controls and ensure that only authorized users can configure or use the plugin.
Use Jenkins' built-in functions like Jenkins.checkPermission to enforce permissions.

```
public void doSomeAction(StaplerRequest req, StaplerResponse rsp) throws IOException, ServletException {
    Jenkins.get().checkPermission(Jenkins.ADMINISTER);
    // Action code here
}
```

Input Validation:

Validate and sanitize all inputs to prevent injection attacks.
Use allow-lists and regular expressions to validate data.

```
public FormValidation doCheckName(@QueryParameter String value) {
    if (value.matches("^[a-zA-Z0-9_]+$")) {
        return FormValidation.ok();
    } else {
        return FormValidation.error("Invalid name");
    }
}
```

Output Encoding:

Ensure all outputs are properly encoded to prevent XSS attacks.
Use functions like Util.escape to encode data.

String safeOutput = Util.escape(inputString);

Jenkins Plugin Development Guidelines:
Follow the MVC Architecture:

Separate the Model, View, and Controller to ensure clean and maintainable code.
Use Jenkins API:

Leverage Jenkins API for accessing built-in functionalities and objects.
Implement Descriptors:

Descriptors (like BuildStepDescriptor or DescriptorImpl) help in defining global configurations and settings.

PowerShell is a powerful scripting language that is also a part of the Windows operating system. Attackers often use PowerShell to masquerade as legitimate users or processes on a Windows machine. They may use PowerShell to download and execute malicious code or perform other malicious actions.
Here's an example of a simple PowerShell command that an attacker might use to download and execute a malicious script from the internet:

```
@Extension
public static final class DescriptorImpl extends BuildStepDescriptor<Builder> {
    public boolean isApplicable(Class<? extends AbstractProject> aClass) {
        return true;
    }

    public String getDisplayName() {
        return "My Plugin Name";
    }
}
```

Define Configurations:
Use config.jelly to define the configuration options in the UI.

```
<j:jelly xmlns:j="jelly:core" xmlns:f="/lib/form">
    <f:entry title="Parameter" field="parameter">
        <f:textbox />
    </f:entry>
</j:jelly>
```

Handle Build Steps:

Implement Builder class to define actions to be taken during a build step.

```
public class MyBuilder extends Builder {
    private final String parameter;

    @DataBoundConstructor
    public MyBuilder(String parameter) {
        this.parameter = parameter;
    }

    @Override
     public boolean perform(AbstractBuild<?, ?> build, Launcher launcher, BuildListener listener)
{
        // Build step actions here
        return true;
    }
}
```

Manage Plugin Dependencies:

Ensure your pom.xml correctly defines all dependencies and Jenkins version.

```
<dependencies>
  <dependency>
    <groupId>org.jenkins-ci.plugins</groupId>
    <artifactId>some-plugin</artifactId>
    <version>1.2.3</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

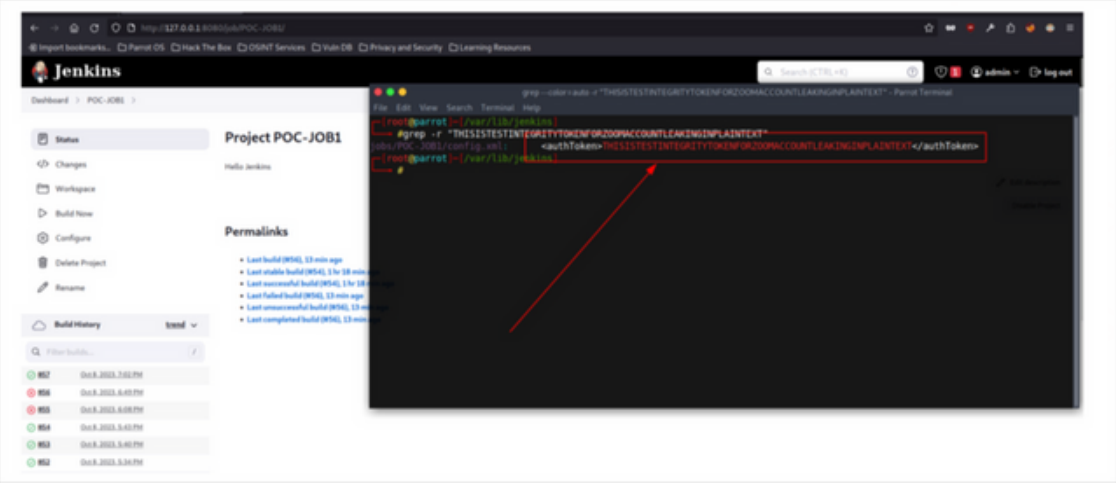
Vulnerability Testing on Jenkins Plugins

https://github.com/jenkinsci/telegram-notifications-plugin : Telegram
https://github.com/jenkinsci/google-metadata-plugin : Google Metadata
https://github.com/jenkinsci/label-linked-jobs-plugin : Linkedin
https://github.com/jenkinsci/kryptowire-plugin/tree/master/docs : Krypotwire
https://github.com/jenkinsci/warrior-plugin.git : warrior
https://github.com/jenkinsci/compressed_files_viewer-plugin.git : Compressed File Viewer
https://github.com/jenkinsci/publish-over-cifs-plugin : CIFS
https://plugins.jenkins.io/git-forensics/ : Git Forensics
https://github.com/jenkinsci/android-apk-size-watcher-plugin : apk size Viewer
https://github.com/jenkinsci/zoom-plugin : ZOOM

Possible Findings

Zoom Meeting Plugin

Source Code : https://github.com/jenkinsci/zoom-plugin.git



The Jenkins server stores the Auth Token/Integrity Token of Zoom in unencrypted manner , where they can be viewed by users with access to the master file. This file can be viewed by Overall/Manage permission on Jenkins.

Integration tokens are typically generated by the application that you are connecting to. For example, to generate a Zoom channel integration token, you would go to your Zoom account settings and navigate to the Integrations page. Once you have generated the integration token, you would need to provide it to the other application that you are connecting to.

Integration tokens can be sensitive because they allow the two applications to access each other's data. For example, if you connect your Zoom account to a CRM (customer relationship managements) system using an integration token, the CRM system would be able to access your Zoom contact list and meeting schedule.

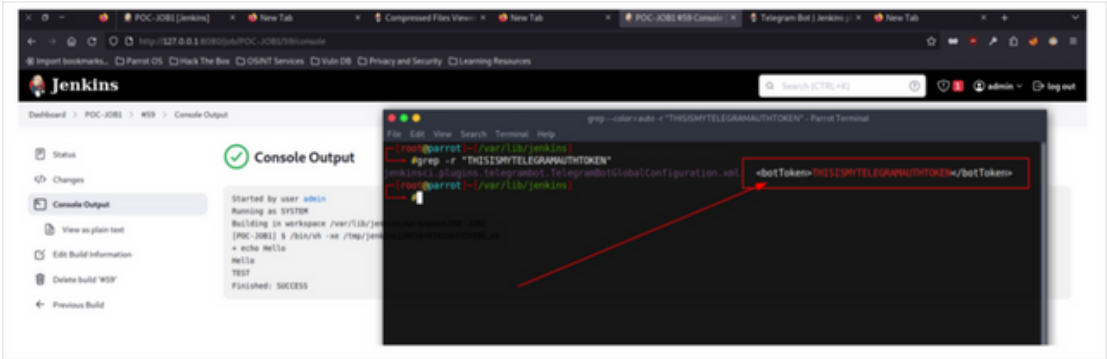
Findings

Auth Token stored in plain text

Impact

The impact of this can lead to complete zoom account takeover / hijacking

Telegram Notification Bot Plugin



Source Code : https://github.com/jenkinsci/telegram-notifications-plugin

Telegram Plugin stores the Bot Token in plain test in unencrypted way in its global configuration file on the Jenkins master, where they can be viewed by users with access to the master file system. which can lead to Telegram Bot Hijacking.

Findings

Bot token stored in plain text (unencrypted)

impact

Telegram Bot Takeover

CompressFileViewer Plugin

Full Source code : https://github.com/jenkinsci/compressed_files_viewer-plugin/blob/master/src/main/webapp/js/index.js

There seems to be one potential XSS vulnerability in web interface of the plugin :

```
const url = document.getElementsByClassName('hiddenData')[1].innerText
```

This code retrieves the URL of the compressed file list from the hiddenData element on the web page. If the hiddenData element contains malicious JavaScript code, that code will be executed when the openAndInsertExtractedFiles() function is called.


```
var xhr = new XMLHttpRequest();

xhr.open('GET', url, true);

xhr.responseType = 'blob';

xhr.onload = function(e) {

if (this.status == 200) { openAndInsertExtractedFiles(this.response)

}

};

xhr.onerror = function(e) {

alert("Error " + e.target.status + " occurred while receiving the

compressed file.");

};

xhr.send();
```

CIFS

Full Source Code :
https://github.com/jenkinsci/publish-over-cifs-plugin/blob/main/src/main/java/jenkins/plugins/publish_over_cifs/CifsClient.java

```
package jenkins.plugins.publish_over_cifs;


public boolean changeDirectory(final String directory) { final String newLocation =
createUrlForSubDir(directory); final SmbFile dir = createFile(newLocation);

if (helper.exists(dir, newLocation) && helper.canRead(dir, newLocation)) { context = newLocation;

return true;

} else {

return false;

}

}

private String createUrlForSubDir(final String directory) {

return directory.endsWith("/") ? context + directory : context + directory

+ '/';

}

...
```

there is a one potential security issue in the code. The createFile() method does not properly validate the input URL, which could allow an attacker to inject malicious code. example, an attacker could pass a URL that contains a PowerShell script, which would then be executed when the createFile() method is called.

Google Metadata

Source Code :
https://github.com/jenkinsci/google-metadata-plugin/blob/develop/src/main/java/com/google/jenkins/plugins/metadata/MetadataContainer.java

```
package com.google.jenkins.plugins.metadata;

public static synchronized MetadataContainer of(Run<?, ?> build) { MetadataContainer container =
build.getAction(MetadataContainer.class); if (container == null) {

container = new MetadataContainer();

build.addAction(container);

}

return container;

}

/**

* @param metadataValue

*      the metadata value to be serialized.

* @return serialized form of the given {@link MetadataValue}.

* @throws MetadataSerializationException

*       when serialization runs into problem.

*/

public String serialize(MetadataValue metadataValue)

throws MetadataSerializationException {

return listSerialize(ImmutableList.of(metadataValue));

}

public <T extends MetadataValue> String listSerialize(Iterable<T> values) { try {

return getObjectMapper()

.writerFor(new TypeReference<Iterable<MetadataValue>>() {})

.writeValueAsString(values);

} catch (JsonProcessingException ex) {

throw new MetadataSerializationException(ex);

}

}


public <T extends MetadataValue> T deserialize(Class<T> clazz, String serialized) {

return Iterables.getOnlyElement(listDeserialize(clazz, serialized));

}

...
```

There are a few potential security vulnerabilities in the above code :

Insecure deserialization: The deserialize() and listDeserialize() methods do not properly validate the input string before deserializing it. This could allow an attacker to inject malicious code into Jenkins application.
cross-Site Scripting (XSS): The serialize() and listSerialize() methods serialize the MetadataValue objects to JSON. If the MetadataValue objects contain malicious JavaScript code, then the XSS vulnerability could be exploited when the serialized JSON is rendered in a web browser.
Remote Code Execution (RCE): The deserialize() and listDeserialize() methods deserialize the JSON strings to MetadataValue objects. If the serialized JSON contains malicious code, then the RCE vulnerability could be exploited when the MetadataValue objects are executed.

02

CONCLUSION

Jenkins, while pivotal in automating and streamlining operations, can be susceptible to various attack vectors if not secured meticulously. Organizations must prioritize securing Jenkins by understanding potential attack vectors and surfaces, implementing robust security practices, and continuously monitoring for anomalous activities to safeguard their CI/CD pipelines against APTs and other cyber threats.

HADESS

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.

Website:

WWW.HADESS.IO

Email

MARKETING@HADESS.IO

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.