Red + Blue = Purple: Our Journey Building a Dedicated Purple Team at Meta

Jayson Grace
Cedric Owens
SANS Hackfest 2022

Meta

whoami

Jayson Grace

Offsec Professional since 2012 (Pentester, Red Team Operator, Tool Developer)

Death Metal Vocalist

Built and led a Red Team in the government space

One of the first pentesters on Splunk's ProdSec team

Former pentesting consultant

whoami

Cedric Owens @cedowens

Started as a blue teamer

IR -> Vuln Mgt -> Red Team -> Purple Team

I enjoy hiking and traveling with the fam!

Passion Projects: macOS research

Meta Purple Team!

Agenda

Purple Teaming at Meta

Engagements

Tooling & Automation

Measuring Effectiveness

Purple Teaming at Meta

Meta

What is Purple Teaming?

- Traditionally a cybersecurity testing exercise in which Blue Team (defenders) and Red Team (attackers) members work closely together to:
  - Test effectiveness of existing defenses in a non-discreet way
  - Discover and fix gaps in detection and prevention capabilities
  - Test organization’s ability to detect threat actors’ Tactics, Techniques, and Procedures (TTPs)
  - Rapidly prototype and implement new detections and/or preventions

Red Team vs. Purple Team

Red Team	Purple Team
Realistic objective-based exercise with a focus on stealth and emulating what a determined adversary would do	Executing pre-determined attacks with a focus on generating signal and testing defenses
Unearth new ways to challenge/bypass defenses, uncover systemic issues, and demonstrate how attackers leverage known attack vectors	Break issues into atomic tests, execute them to generate signal for defenders, provide scripts for security regression testing of TTPs
On average, 3 months (“attack” occurs in final 1-3 weeks)	3-5 weeks
Outputs: • Detailed report covering the entire attack • Tactical and strategic recommendations • Workplace group for ongoing collaboration • Thorough presentation for all affected stakeholders • Tangible insights into how an organization can respond to various types of attackers.	Outputs: • Creation of new detection/prevention POCs • Validation of existing detections/preventions • Report that both teams collaborate on writing to outline the attacks executed, defenses created, signal gathered, and instructions for replicating and validating the exercise.
Sparring partner to prepare for a fight with the heavyweight champion.	Running drills to level up the capabilities and resilience of a fighter. Practice makes perfect!

Operationalizing the Practice

- The Purple Team was built as an internal consultancy
- Uphold quality and consistency
- Created a diverse "menu" of offerings that fit different team's needs
- Variety is the spice of life and keeps people happy

Prioritization

- Our intake request form drives all of the work that we do
- Oncall to triage new requests so they don't sit in our intake queue forever
- Not all requests are a good fit for Purple Teaming
- Priority queue to determine execution timelines

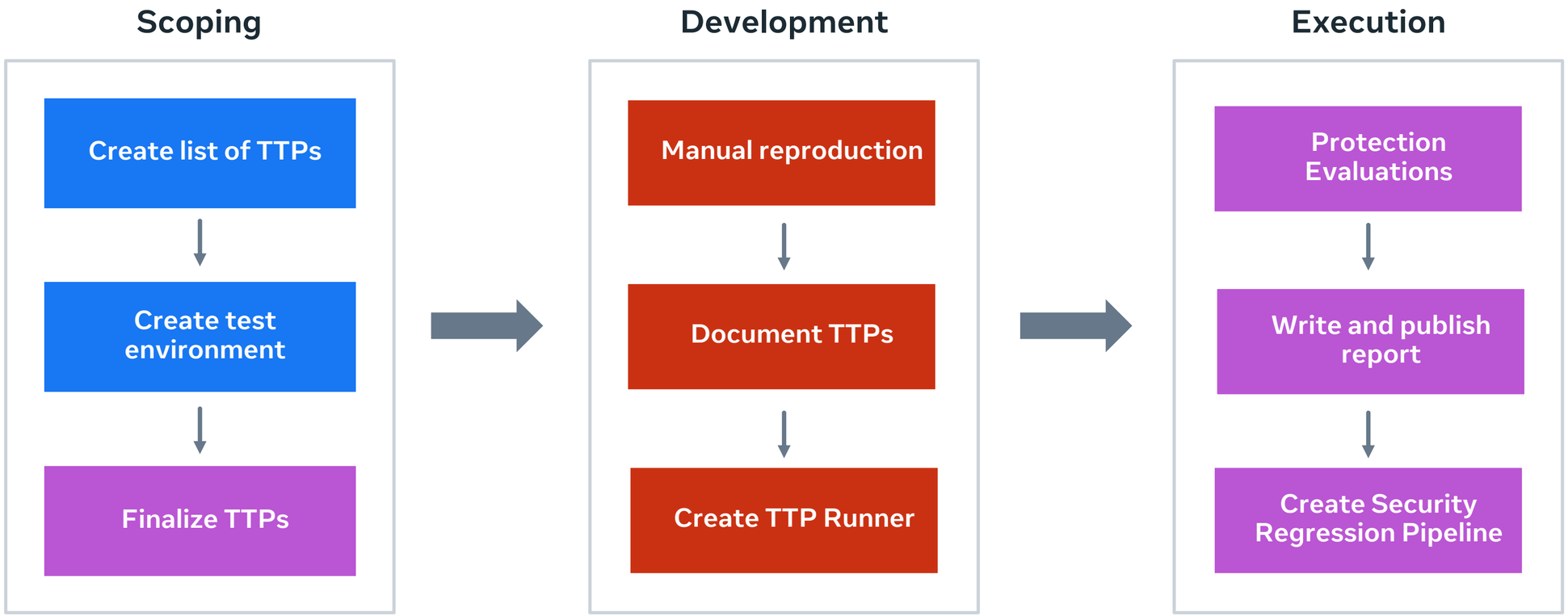
Engagements

Meta

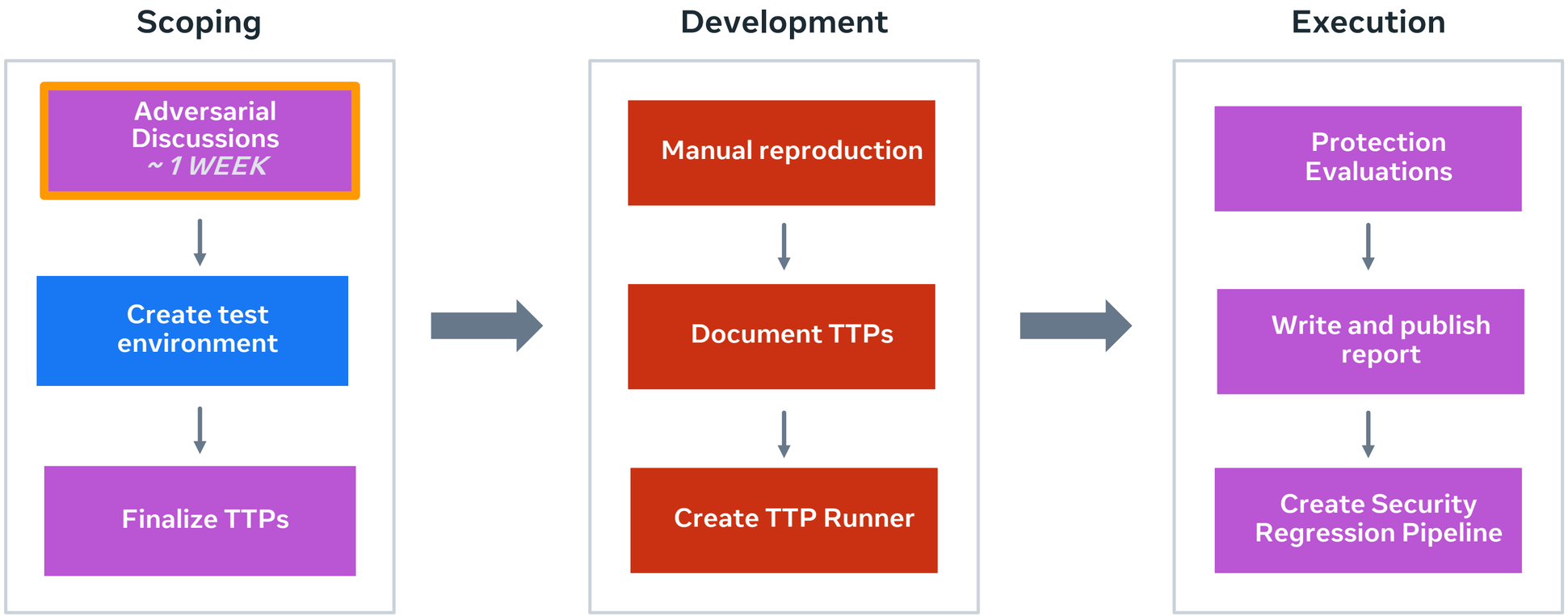
High Level

- The customer provides a test environment
- Manually reproduce and document TTPs
- Automate TTPs using our TTP Runner
- Run the attacks with defenders in protection evaluation sessions
- Create Security Regression Pipeline (SRP) to track TTP lifespan

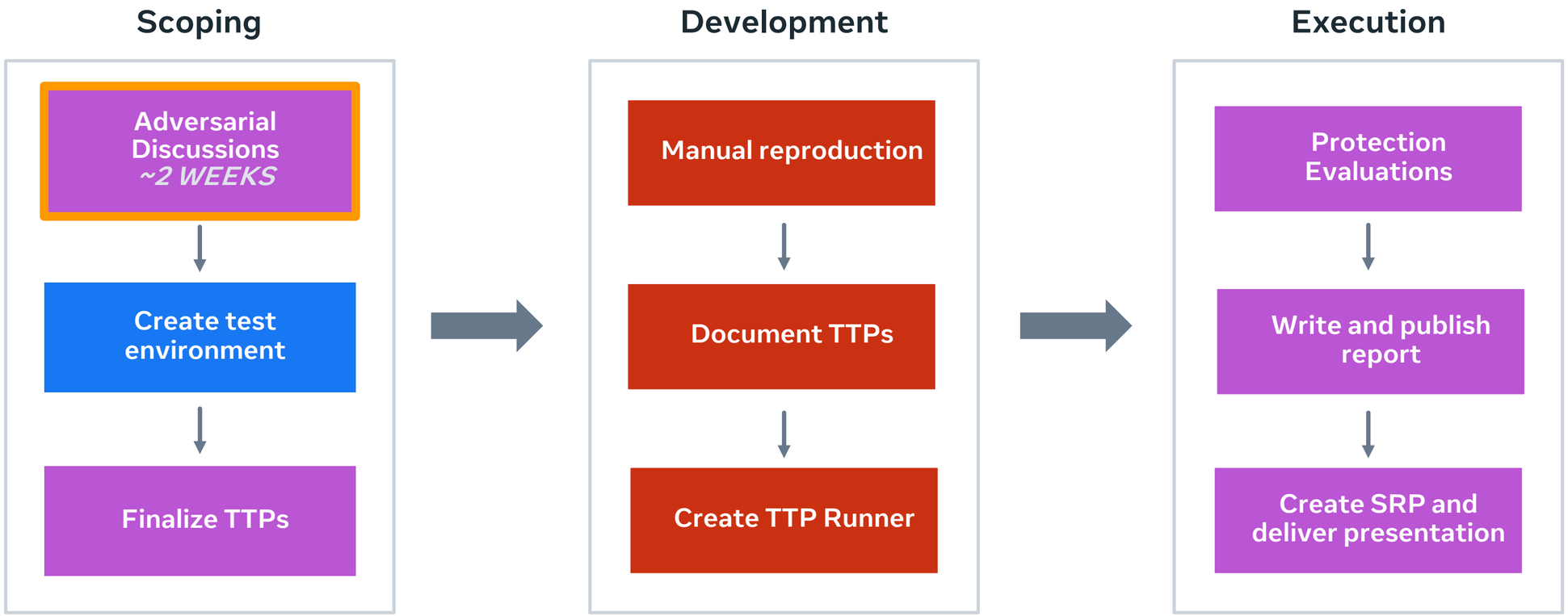
Traditional Purple Team (~3-4 weeks)



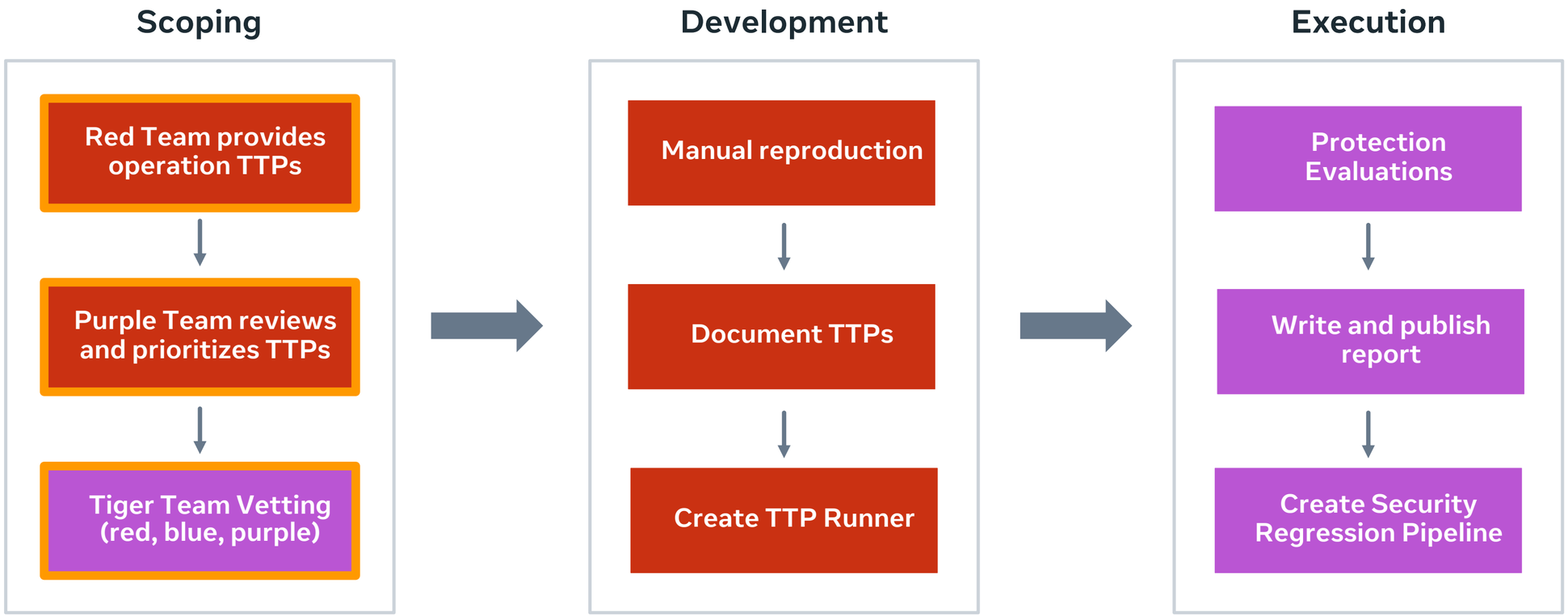
Surface Expansion Lite (~4-5 weeks)



Surface Expansion (~5-6 weeks)



Op Follow-Up Engagement (~4-5 weeks)



Tooling & Automation

Meta

Ephemeral Workloads FTW!

“Cattle Not Pets” philosophy

- Automated provisioning
- Ansible, Terraform, Terragrunt, GitHub Actions, Cloud Native Services
- C2 Servers, SRP target hosts, test environments, etc.
- Automated teardown
- Reduces risk of “test infra” hanging around
- “On-demand” malware sandboxes for blue team to investigate

Ansible

```
- name: Purple Team Kali Provisioner
  hosts: all
  environment:
    DEBIAN_FRONTEND: noninteractive
  roles:
    # Installing docker from this role
    # requires a hack to make it think
    # it's a debian system:
    - role: geerlingguy.docker
      become: yes
      ansible_distribution: "debian"
      ansible_distribution_release: "buster"
      ansible_os_family: "Debian"
      docker_users: ["kali"]
    - role: golang
    - role: l50.sliver
      become: yes
    - role: mimipenguin
      become: yes
    - role: linpeas
      become: yes
    - role: linenum
      become: yes

  tasks:
    - name: Create PT directory
      ansible.builtin.file:
        path: "{{ lookup('env','HOME') }}/.pt"
        state: directory
        mode: "0755"

    - name: Install packages
      become: yes
      ansible.builtin.apt:
        name: "{{ install_packages }}"
        state: present
        update_cache: yes
      when: ansible_distribution_release == "kali-rolling"

    - name: Install pip packages
      ansible.builtin.pip:
        name: "{{ item }}"
        state: present
      when: ansible_distribution_release == "kali-rolling"
      loop: "{{ pip_packages }}"

    - name: Install gems
      become: yes
      community.general.gem:
        name: "{{ item }}"
        user_install: no
```

Terraform && Terragrunt

```
inputs = {
  ami_name               = local.name
  aws_account_id         = local.aws_account_id
  aws_accounts           = [local.aws_account_id]
  base_ami_id            = dependency.kali-base-ami-builder.outputs.ami_id
  build_artifacts_bucket = dependency.bucket.outputs.s3_bucket_id
  component_name         = "provision-${local.name}"
  component_description  = "Provision ${local.name} golden image"
  component_version      = local.version
  env                    = local.env
  image_recipe_version   = local.version
  install_deps_cmd       = <<-EOT
  sudo apt-get update && sudo apt-get install -y jq curl
  EOT
  kms_key_id             = dependency.kms.outputs.key_id
  provision_user       = "kali"
  provision_user_home  = "/home/kali"
  provisioning_repo    = "github.com/project/provisioning-repo-goes-here"
  region               = local.region
  subnet_id            = dependency.aws-infra.outputs.private_subnet_ids[0]
  vol_size             = local.vol_size
  vpc_id               = dependency.aws-infra.outputs.vpc_id
}
```

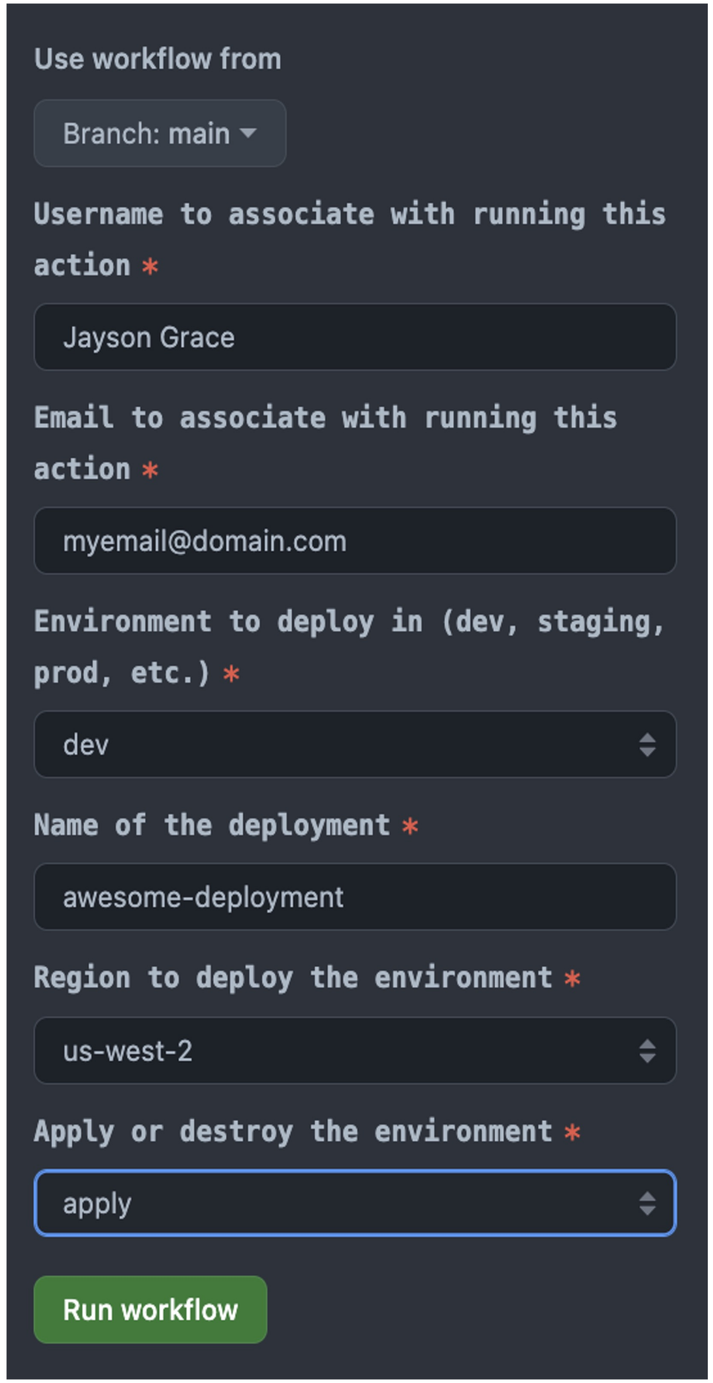
Github Actions

```
- name: build
  run: |
    mkdir -p bin
    go build -o bin/admiralctl
- name: deploy-instance-and-execute-runner
  shell: bash
  run: |
    set -e

    WORKLOAD_NAME="srp-linux-canary-$(date +%s)"
    ./bin/admiralctl deploy terraform/workloads/security-regression-pipeline/linux/canary "${WORKLOAD_NAME}"
    sleep 120
    INSTANCE_ID=$(aws ec2 describe-instances --filter "Name=tag:Name,Values=${WORKLOAD_NAME}" | jq -r '.Reservations[0].Instances[0].Insta
    COMMAND_ID=$(aws ssm send-command --instance-ids "${INSTANCE_ID}" --document-name "AWS-RunShellScript" --parameters commands='sudo /pu
    aws ssm wait command-executed --command-id "${COMMAND_ID}" --instance-id "${INSTANCE_ID}"
    RUNNER_OUTPUT=$(aws ssm get-command-invocation --command-id "${COMMAND_ID}" --instance-id "${INSTANCE_ID}" | jq -r .StandardOutputCont
    echo -e "${RUNNER_OUTPUT}"
    SUCCESS=$(echo -e "${RUNNER_OUTPUT}" | grep 'ttp runner completed successfully')

    ./bin/admiralctl destroy "${WORKLOAD_NAME}"

    if [[ ! -z "${SUCCESS}" ]]
    then
      echo "Runner executed successfully!"
      exit 0
    else
      echo "Runner failed!"
      exit 1
    fi
```

Spinach

- Deploy minimal infra required for workloads
- Easy to use with or without cloud knowledge
- Web UI or CLI
- Functionality can be extended through Terraform modules
  - Ex. Golden Image as a Service
- Currently AWS-focused (can be expanded to other providers)
- Leverages Terraform and Terragrunt

Spinach Web UI



Spinach CLI

Create new deployment

```
./spinach-"${OS}" newdeploy \
    -n "${DEPLOY_NAME}" \
    -e "${TF_ENV}"
```

Note: Deployments created outside of github actions will not persist as part of the repo unless you add, commit, and push them.

Run terragrunt apply

```
ACTION=apply
./spinach-"${OS}" "${ACTION}" \
    -n "${DEPLOY_NAME}" \
    -e "${TF_ENV}"
```

Run terragrunt destroy

```
ACTION=destroy
./spinach-"${OS}" "${ACTION}" \
    -n "${DEPLOY_NAME}" \
    -e "${TF_ENV}"
```

Add Terraform module to an existing deployment

```
tf_modules:
  - terraform-aws-kms
  - terraform-aws-ec2-image-builder
```

```
./spinach-"${OS}" apply \
    -n "${DEPLOY_NAME}" \
    -e "${TF_ENV}" \
    --config deployments/golden-image-builder/config.yaml
```

Fleet Admiral

- Deploy and manage security-oriented on-demand workloads
- Leverages Terraform and Terragrunt
- Easily stand up pre-provisioned AMIs for various operating systems
- Beneficial for purple, blue, and red teams. Ex:
  - Purple: deploying and managing ephemeral purple team infra
  - Blue: deploying and managing sandboxed ephemeral instances to detonate malware or TTPs for analysis
  - Red: deploying and managing ephemeral infrastructure for testing

Fleet Admiral

```
./bin/admiralctl deploy terraform/workloads/pt-kali $(whoami)-kali-dev
```

If you want a larger-than-default instance, simply add an appropriate var:

```
./bin/admiralctl deploy terraform/workloads/pt-kali $(whoami)-kali-dev -- -var instance_type=t3.large
```

```
./bin/admiralctl deploy terraform/workloads/rt-kali "operator-rt-kali-1"
./bin/admiralctl deploy terraform/workloads/rt-kali "operator-rt-kali-2"
```

TTP Runner

- Framework created to facilitate the development, automation, and execution of TTPs.
- Written in Go
- TTPs can be written in Go, Bash, Swift, or Python
- Build once… edit yaml configurations as many times as needed
- Logs TTP success/failure state
- Module system to facilitate integrations for tools like MDE, Splunk, etc.
- TTP Runner is delivered per Purple Team engagement
- Leverages Fleet Admiral and automation libraries we've written over time
- Ability to spin up and drive C2 servers, automate browser-based workflows, auto enroll new accounts, move laterally, and much more.

TTP Runner

README.md

Load TTPs and Ship Runner to a Target

1. Load the TTPs from a specific repo. For example:

```
./bin/magefile loadTTPs https://github.com/metaredteam/ws-ttps.git
```

2. Update `ttpStubs` in `cmd/runTTP.go` to reflect the number of ttps you are loading. For example, if you have five ttps:

```
// mapping of TTP ID to function names
ttpStubs = map[string]interface{}{
   "1":  ttps.TTP1,
   "2":  ttps.TTP2,
   "3":  ttps.TTP3,
   "4":  ttps.TTP4,
   "5":  ttps.TTP5,
```

   Each TTP must belong to the `ttps` package and the entry function for each TTP must start with `TTP` followed by a number, i.e. `func TTP1() error`

3. Compile `ttp-runner` with the newly loaded ttps:

```
# If you want to name the binary something specifically:
./bin/magefile compile engagementname-ttps

# If you want the current directory to be used for the binary name:
./bin/magefile compile ''
```

4. Transfer `ttp-runner` to the target server:

```
./bin/magefile ship engagementname-ttps $USER@targetserver /path/for/ttp-runner/on/server
```

README.md

Usage

- Login to the target system and use this command to list all TTPs:

```
./ttp-runner listTTPs
```

- To run a specific TTP:

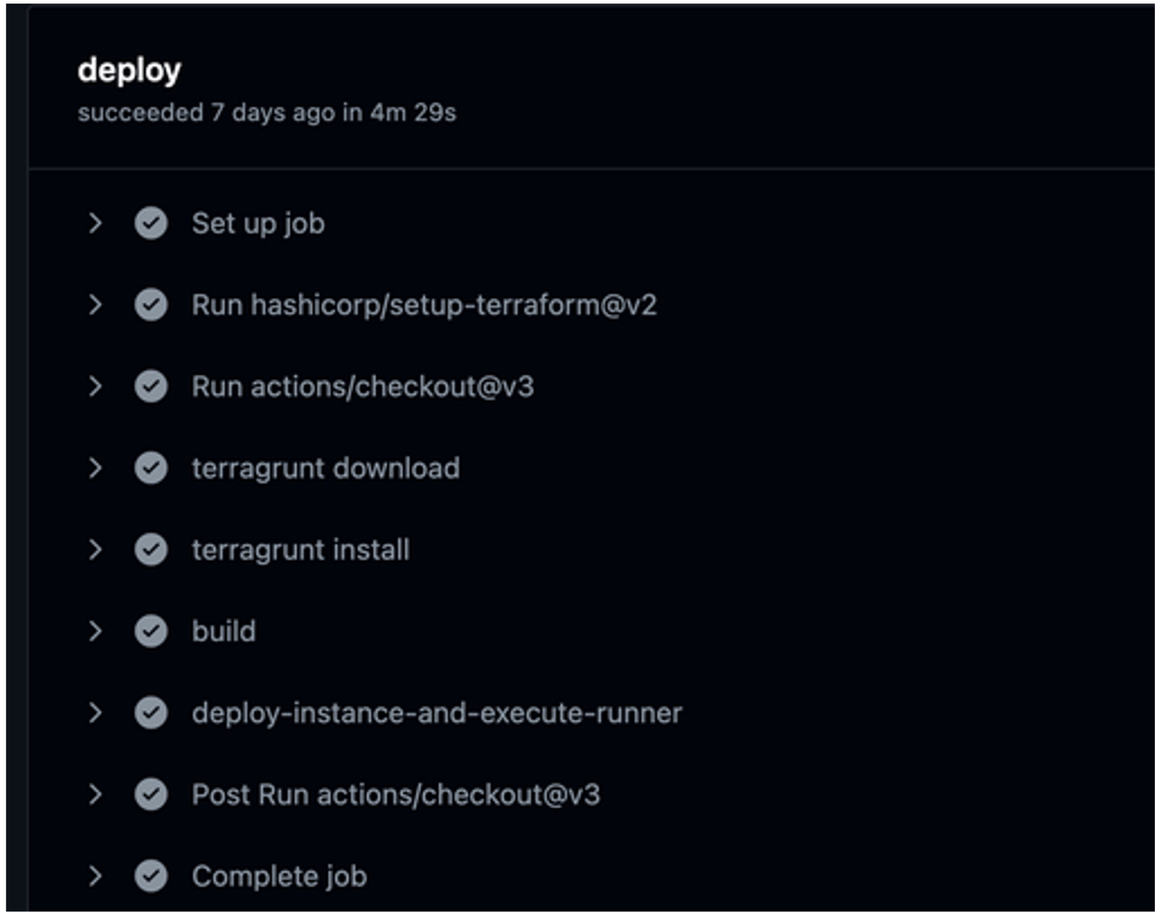
```
TTP_ID=1
./ttp-runner runTTP -t $TTP_ID
```

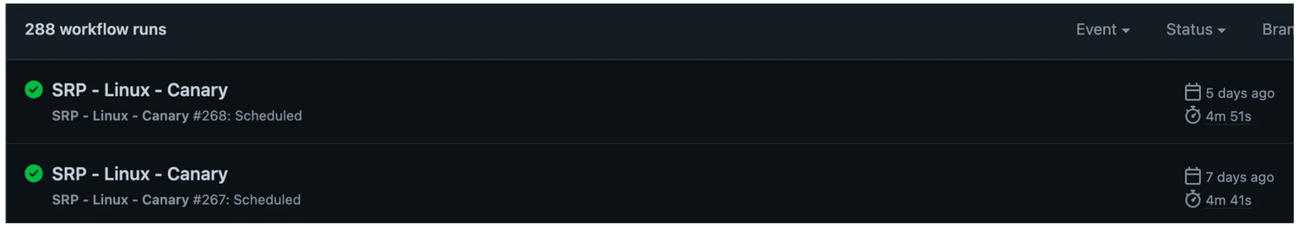
CLI Options for ttp-runner

```
Run TTPs for a Purple Team Engagement.

Usage:
  ttp-runner [command]

Available Commands:
  cleanArtifacts Clean up all artifacts generated from running ws-ttp.
  completion     Generate the autocompletion script for the specified shell
  help           Help about any command
  listTTPs       List all TTPs.
  runTTP         Run a specified TTP.

Flags:
      --config string   Config file (default is config/config.yaml)
  -d, --debug           Show debug messages.
  -h, --help            help for ttp-runner
  -t, --toggle          Help message for toggle

Use "ttp-runner [command] --help" for more information about a command.
```

Security Regression Pipeline (SRP)

Security Regression Pipeline (SRP)





MITRE CALDERA 0-days

- Found three 0-days in MITRE CALDERA
  - Writeups: https://github.com/metaredteam/external-disclosures/security/advisories?state=published
  - CVE-2022-40605
  - CVE-2022-40606
  - CVE-2022-41139
- Worked with MITRE to do collaborative vuln disclosures
  - Understand systemic issues and provide evidence

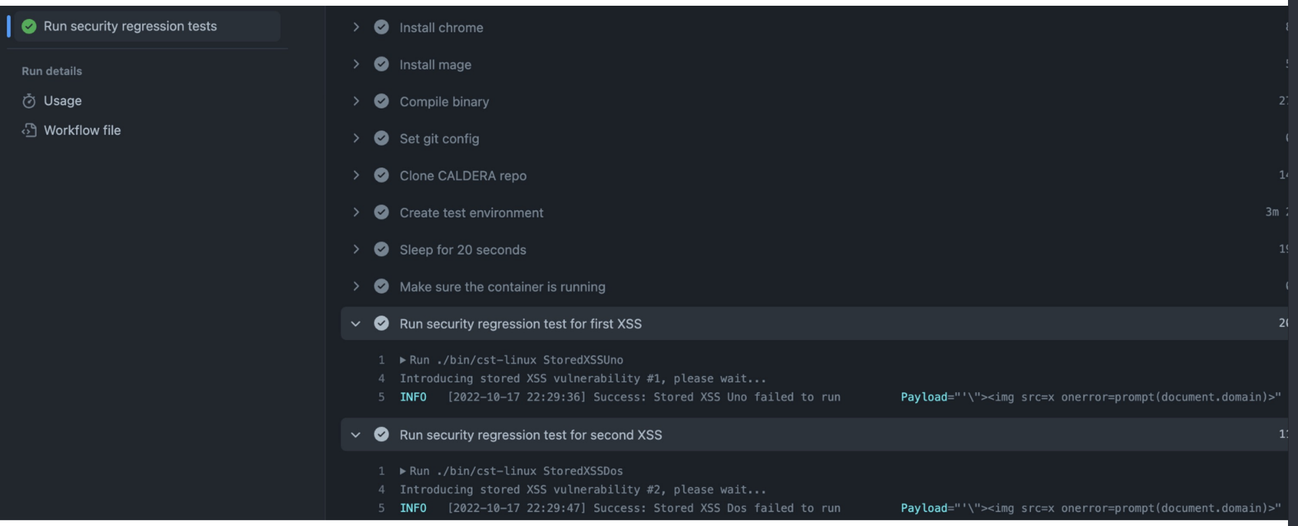
MITRE CALDERA TTP Runner

```
▶ Run ./bin/cst-linux StoredXSSUno
Introducing stored XSS vulnerability #1, please wait...
ERROR  [2022-10-17 22:31:56] failure: Stored XSS Uno ran successfully      Payload="'\"><img src=x onerror=prompt(document.domain)>"
error="failure: Stored XSS Uno ran successfully"
▶ Run ./bin/cst-linux StoredXSSDos
Introducing stored XSS vulnerability #2, please wait...
ERROR  [2022-10-17 22:32:08] failure: Stored XSS Dos ran successfully      Payload="'\"><img src=x onerror=prompt(document.domain)>"
error="failure: Stored XSS Dos ran successfully"
```

▾ Assets 16

caldera-security-tests_1.0.0_checksums.txt	1.47 KB	5 days ago
caldera-security-tests_1.0.0_darwin_amd64v2.tar.gz	6.1 MB	5 days ago
caldera-security-tests_1.0.0_darwin_amd64v3.tar.gz	6.1 MB	5 days ago
caldera-security-tests_1.0.0_darwin_arm64.tar.gz	5.86 MB	5 days ago
caldera-security-tests_1.0.0_linux_amd64v2.tar.gz	5.87 MB	5 days ago
caldera-security-tests_1.0.0_linux_amd64v3.tar.gz	5.87 MB	5 days ago
caldera-security-tests_1.0.0_linux_arm64.tar.gz	5.37 MB	5 days ago
caldera-security-tests_1.0.0_linux_armv6.tar.gz	5.54 MB	5 days ago
caldera-security-tests_1.0.0_linux_armv7.tar.gz	5.54 MB	5 days ago
caldera-security-tests_1.0.0_windows_amd64v2.tar.gz	5.95 MB	5 days ago
caldera-security-tests_1.0.0_windows_amd64v3.tar.gz	5.95 MB	5 days ago
caldera-security-tests_1.0.0_windows_arm64.tar.gz	5.45 MB	5 days ago
caldera-security-tests_1.0.0_windows_armv6.tar.gz	5.68 MB	5 days ago
caldera-security-tests_1.0.0_windows_armv7.tar.gz	5.67 MB	5 days ago
Source code (zip)		5 days ago
Source code (tar.gz)		5 days ago

MITRE CALDERA SRP




```yaml
name: CALDERA Security Regression Pipeline
on:
  pull_request:
  push:
    branches: [master]

  # Run once a week (see https://crontab.guru)
  schedule:
    - cron: "0 0 * * 0"

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

jobs:
  tests:
    uses: fbsamples/caldera-security-tests/.github/workflows/srp.yaml@main
```

Coming Soon

- TTP Runner
- Spinach
- Fleet Admiral
- Several general purpose Security Regression Pipelines
- Ansible playbooks and roles
- Terraform modules

Measuring Effectiveness

Meta

Helping The Organization

- Empower other teams to run Purple Team TTPs via automation
  - Automate with consistency in mind (same test = same network signal over time)
  - No need to wait for Purple Team to be available to do this
- Tracking detection status against TTPs over time
  - Goal is to show improvement over time in TTP detection
- Build allies across the organization
  - Sharing offensive knowledge and techniques with defensive teams
  - Purple team is not here to “give you more work”, but to help via automation
- Tracking detections created from Purple Team Engagements
  - Tagging new detections in a unique way
  - Tracking how often those purple team tagged detections fire and the fidelity

Metrics Metrics Metrics!

TTP Tracking!

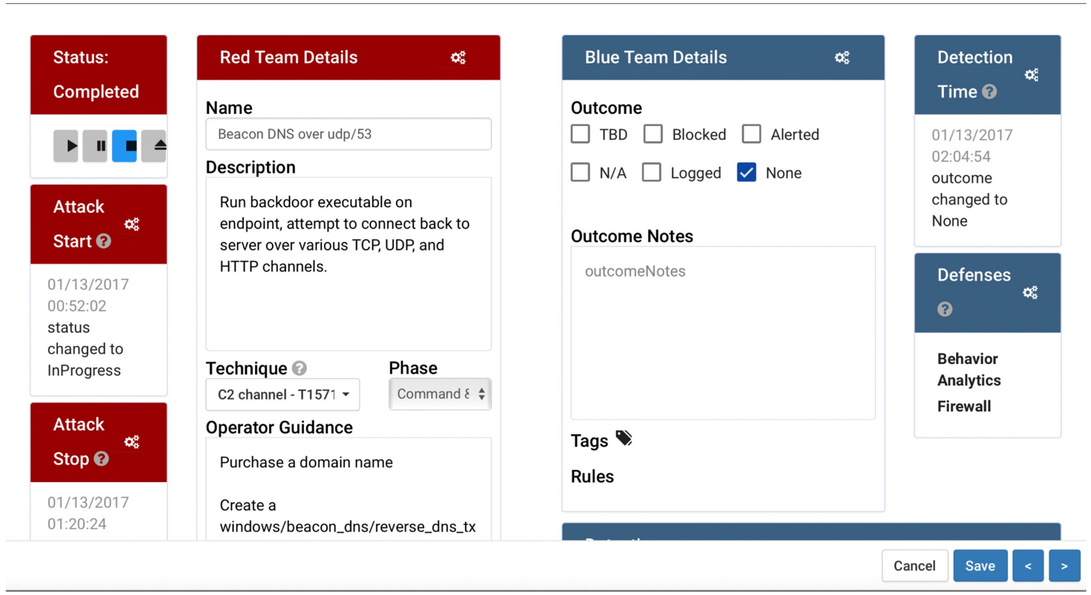
- TTP Detection Trends Over Time
- Tracking TTP Regressions
- Mapping Purple Team Exercise Derived Detections To Incident Detections
- Shift Left: Tracking usage of Purple Team Automation Tools

TTP Mapping/Tracking

- Leveraging MITRE ATT&CK
  - Lots of relevant TTPs for Enterprise Environments
    - Windows
    - macOS
    - Linux
    - Network
    - Cloud
    - Containers
  - Can begin testing these TTPs and mapping detection coverages
- Tracking TTPs for Custom Apps/Environments/Infrastructure
  - Create custom layers in your tracking tool
  - Can still track those TTPs over time

Helpful Frameworks

- VECTR (by Security Risk Advisors)
  - Freeware
  - Easily facilitates collaborative TTP testing and tracking
  - Has some neat built in reporting/metrics
  - API functionality
- ATT&CK Navigator
  - Freeware
  - Can be really useful with threat actor TTP tracking/mapping
  - Visualize defensive coverage
- Commercial Tools
  - May be a good option if limited in resources/time



Vectr Test Case

In Summary

- Purple teams bring more to the table than TTP testing
- Solve the “small team, big company” problem: Automation!
- Lots of skill sets at play: software development, adversarial mindset, knowledge of modern tech stacks, empathy, communication and collaboration
- Shift Left: Self Service Opportunities Through Automation and Collaboration
- Meaningful metrics is the key to measuring impact

Thanks for listening! Any Questions?

Meta