RYAN VILLARREAL / MARCH 21, 2019

Let's talk about Terraform for a minute. Terraform is a tool for building, changing, and versioning cloud or local infrastructure safely and efficiently. This allows system administrators to construct infrastructure as code, which means you can create customized scripts to spin up entire environments in AWS, Google Cloud Console, Azure, and Digital Ocean.

This is very helpful for pentesting or red teaming exercises because it allows infrastructure to be stood up and torn down within matter of minutes in the instance an IP address or domain is burned on an engagement. The term burned would be defined here as a blue team or incident response team that has categorized and/or blocked a particular IP deemed a threat to the company. With Terraform we can quickly acquire new dynamic IP addresses and continue our attack vectors.

There is a particular open source project that has been developed by byt3bl33d3r known as Red-Baron. Red-Baron attempts to automate resilient, disposable, secure, and agile

infrastructure for Red Teams. The code on GitHub is a set of modules and custom/third-party providers for Terraform. The examples developed by byt3bl33d3r provides the ability to stand up command and control servers, and basic phishing sites on the more popular cloud services. However, I decided to take this one step further and write a custom script that would allow the automated deployment of a GoPhish server in AWS. This would allow for rapid deployment of GoPhish in AWS which would significantly cut down the amount of time required for the setup phase of a Social Engineering assessment. Additionally, it would limit the connections to the administrative portal of the GoPhish server to the external IP where the server was deployed.

Terraform, unfortunately is not cloud agnostic, and all scripts must be tailored to the environment you want to deploy in. For this blog post, and all of my current GitHub projects revolve around AWS. In order to run through the examples detailed in this blog post you will need a basic free-tier AWS account with an IAM account in order to deploy the necessary resources.

Setup

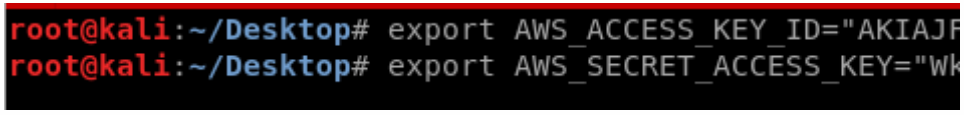
First step is to make sure that Terraform is installed on our host platform. The software developers make it very easy to get this setup.

```
#~ wget
https://releases.hashicorp.com/terraform/0.11.13/terraform_0.11.13_lin
```

```
ux_amd64.zip
#~ unzip terraform_0.11.13_linux_amd64.zip
#~ mv terraform /usr/local/bin/
```

That's all it takes to setup Terraform on Kali or any other Linux x64 distro. In order to deploy the GoPhish server the code base provided on my GitHub account will help assist with that process. The following steps detail how you can pull the code down, and export the AWS keys to your $PATH.

```
#~ git clone https://github.com/ryanvillarreal/TerraformFun && cd
TerraformFun
#~ cp examples/gophish.tf .
#~ export AWS_ACCESS_KEY_ID="accesskey"
#~ export AWS_SECRET_ACCESS_KEY="secretkey"
#~ export AWS_DEFAULT_REGION="us-east-1"
```



```
root@kali:~/Desktop# git clone https://github.com/ryanvillarreal/TerraformFun
Cloning into 'TerraformFun'...
remote: Enumerating objects: 88, done.
remote: Counting objects: 100% (88/88), done.
remote: Compressing objects: 100% (70/70), done.
remote: Total 88 (delta 29), reused 69 (delta 18), pack-reused 0
Unpacking objects: 100% (88/88), done.
root@kali:~/Desktop# cd TerraformFun/
root@kali:~/Desktop/TerraformFun# ls
examples  LICENSE  modules  README.md  scripts
root@kali:~/Desktop/TerraformFun# cp examples/gophish.tf .
root@kali:~/Desktop/TerraformFun# ls
examples  gophish.tf  LICENSE  modules  README.md  scripts
root@kali:~/Desktop/TerraformFun#
```

Stand Up

Terraform has three main stages or commands which it has to go through in order to deploy cloud resources. The `terraform init` command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. Also, it is safe to run this command multiple times. The second phase is the `terraform plan` command. This command is a convenient way to check whether the execution plan for a set of changes matches your expectations without making any changes to real resources or to the state. Finally, the `apply` phase. The `terraform apply` command is used to apply the

changes required to reach the desired state of the configuration, or the predetermined set of actions generated by a `terraform plan` execution plan.

```
#~ terraform init
#~ terraform plan
#~ terraform apply
```

The screenshot below shows Terraform performing the init phase and pulling down the required configuration files.

```
root@kali:~/Downloads/TerraformFun# terraform init
Initializing modules...
- module.create_vpc
- module.phishing_server

Initializing provider plugins...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = "~> 2.3"
* provider.external: version = "~> 1.1"
* provider.random: version = "~> 2.1"
* provider.tls: version = "~> 1.2"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
root@kali:~/Downloads/TerraformFun#
```

For demonstration purposes I have no included any screenshots for the `plan` phase, because it was not beneficial to the post. The screenshot below shows the beginning stages of the `apply` phase. Terraform lists out all of the actions that will be performed, along with all the information that is returned from AWS.

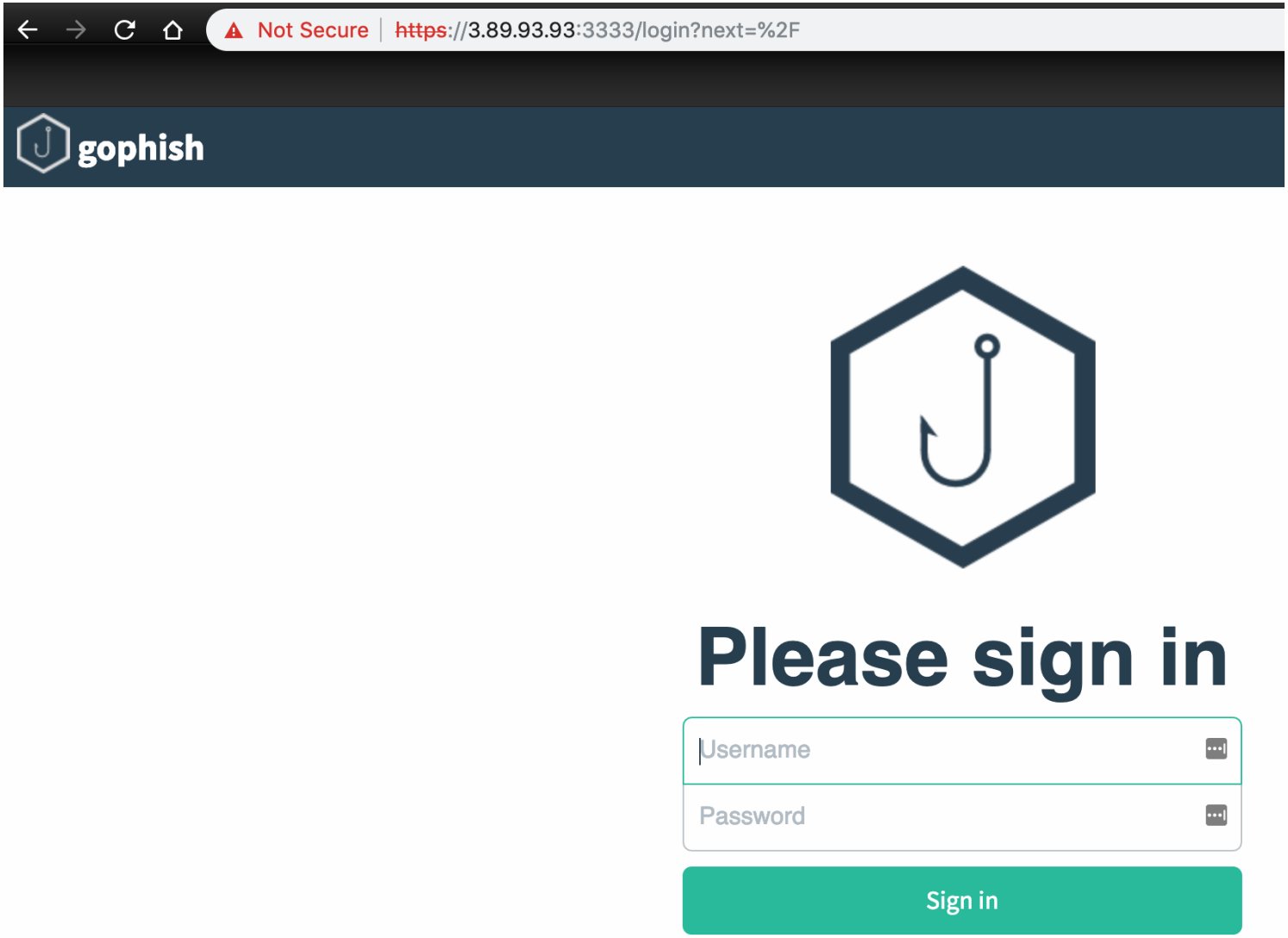
```
root@kali:~/Downloads/TerraformFun# terraform apply
provider.aws.region
  The region where AWS operations will take place. Examples
  are us-east-1, us-west-2, etc.

  Default: us-east-1
  Enter a value:

data.external.get_public_ip: Refreshing state...
data.aws_region.current: Refreshing state...

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + module.create_vpc.aws_internet_gateway.default
      id:                                       <computed>
      owner_id:                                 <computed>
      vpc_id:                                   "${aws_vpc.default.id}"

  + module.create_vpc.aws_route_table.default
      id:                                       <computed>
      owner_id:                                 <computed>
      propagating_vgws.#:                       <computed>
      route.#:                                  "1"
      route.~570683813.cidr_block:              "0.0.0.0/0"
      route.~570683813.egress_only_gateway_id:  ""
      route.~570683813.gateway_id:              "${aws_internet_gateway.default.id}"
      route.~570683813.instance_id:             ""
      route.~570683813.ipv6_cidr_block:         ""
      route.~570683813.nat_gateway_id:          ""
```

The screenshot below is apart of my custom GoPhish setup script that will automatically change the password for the default admin account on the GoPhish server. This password will be randomly created every time the setup script is run, therefore storing it some where safe is

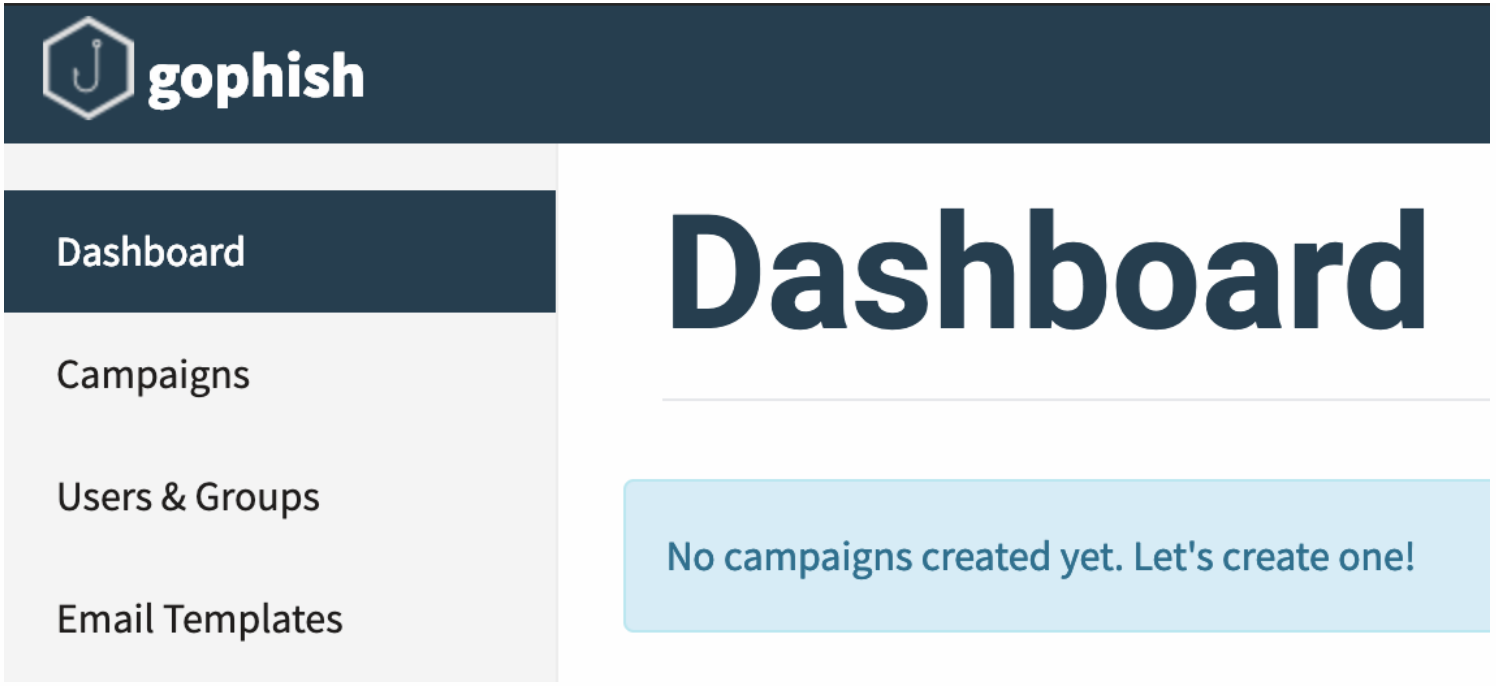
imperative. This helps ensure that our tools are safe. We do not want to get owned while trying to own others. In the future I would like to write this out to a file or somewhere that is more accessible than just dumping to the logs.

```
(remote-exec): fbkyKGtVqjxWZ
 Provisioning with 'local-exec'...
```

Now if we navigate to the external IP address on port 3333, as long as all of the AWS routes and firewall rules were implemented correctly you should be prompted with the GoPhish administrative login page.



Using our randomly created password with the username `admin` we successfully login.



To demonstrate the power of the Terraform tool, I have logged into my AWS dashboard to verify all of the running instances. As you can see in the screenshot below, in fact we have the `phishing-server` instance up and running.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
phishing-ser…	i-00d837a01c3077997	t2.micro	us-east-1b	running	2/2 checks …

By browsing to the Security Group page, we can see the firewall security group that was setup during `terraform apply` phase.

Security Group: sg-08b992e3327b88030

Description | **Inbound** | Outbound | Tags

Edit

Type	Protocol	Port Range	S
HTTP	TCP	80	0.
SSH	TCP	22	67
Custom TCP Rule	TCP	3333	67
HTTPS	TCP	443	0.

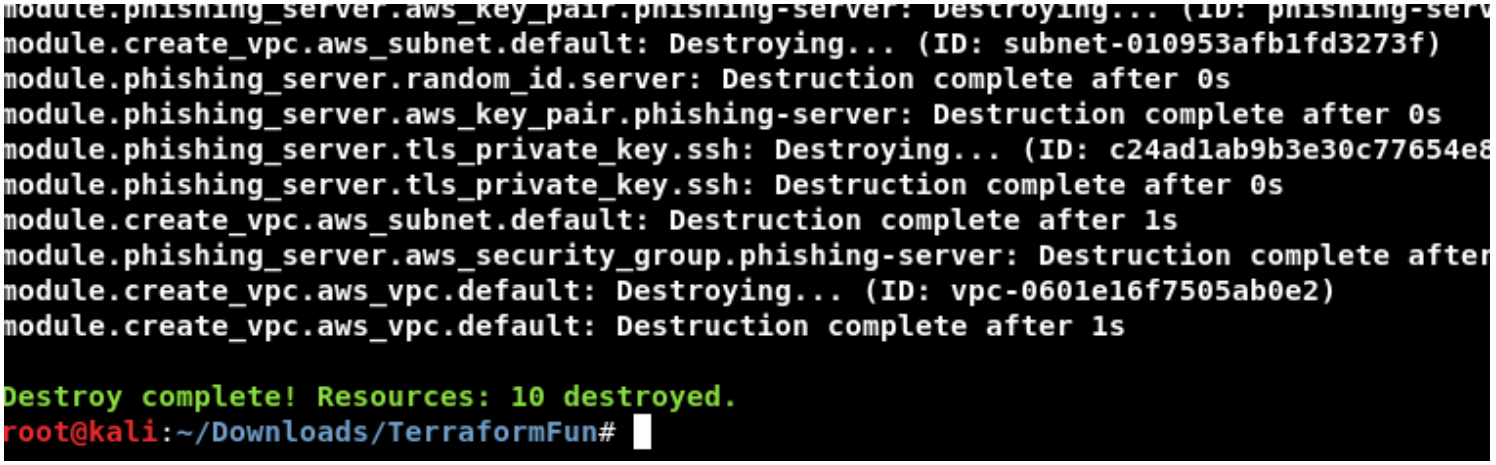
Tear Down

Now that the engagement is over, or your IP address has been burned we need to quickly tear down everything we created. Well Terraform makes that simplistic as well. To tear down the Terraform environment that you have stood up, make sure you are in the root TerraformFun directory and run the following:

```
#~ terraform destroy
```

```
Terraform will perform the following actions:

  - module.create_vpc.aws_internet_gateway.default

  - module.create_vpc.aws_route_table.default

  - module.create_vpc.aws_route_table_association.default

  - module.create_vpc.aws_subnet.default

  - module.create_vpc.aws_vpc.default

  - module.phishing_server.aws_instance.phishing-server

  - module.phishing_server.aws_key_pair.phishing-server

  - module.phishing_server.aws_security_group.phishing-server

  - module.phishing_server.random_id.server

  - module.phishing_server.tls_private_key.ssh


Plan: 0 to add, 0 to change, 10 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value:
```

Once the `terraform destroy` command is run it should report back that all Resources have been removed. I will say from experience though if for some reason your ssh keys or .terraform.state files have been corrupted or deleted it will be impossible for Terraform to know which resources to destroy, and will require the user to login to the AWS portal to destroy the created resources.



To verify once more, I have logged into the AWS portal and determined that all resources have successfully been destroyed.

Resources

You are using the following Amazon EC2 resources in the US East (N. Virginia) region:

- 0 Running Instances
- 0 Dedicated Hosts
- 0 Volumes
- 0 Key Pairs
- 0 Placement Groups
- 0 Elastic IPs
- 0 Snapshots
- 0 Load Balancers
- 2 Security Groups

Hopefully this blog post and the code base hosted on GitHub will help in your next phishing or red-teaming exercise. Future plans for this GitHub project is to add in Ansible support for more advanced management over all nodes, and utilization of AWS' Route53 to deploy Domain names to the GoPhish server. Until next time, keep on hacking!

SHARE

TAGS: CLOUD AWS TERRAFORM RED-BARON TERRAFORMFUN GOOGLE CLOUD GOPHISH AUTOMATE REDTEAMING PHISHING

— ABOUT RYAN VILLARREAL

DENVER, COLORADO TWITTER

NEXT

Bestest Red Team

MARCH 28, 2019

PREVIOUS

7 Tools For Malicious Document Creation

MARCH 19, 2019

— ABOUT —

Two cybersecurity professionals trying to get better at all things security.

— LATEST POSTS —

Information Gathering With Cobalt Strike
AUGUST 16, 2019

Navigating To A Web Site Step By Step
AUGUST 01, 2019

Atomic Red Team
JULY 30, 2019

— AUTHORS —

- Ryan Smith
- Bestest RedTeam
- Ryan Villarreal

— TAGS —

802.11 802.1X ACTIVE DIRECTORY ANTI-CSRF AUTOMATE AUTOMATION AWS BETA BETTERCAP BGP BITCOIN BLOODHOUND BLUE TEAM BURPSUITE BYPASS BYT3BL33D3R C2 CA CAPTURE THE FLAG CERTIFICATES CLOUD CLUSTER CME COBALT STRIKE COMMAND AND CONTROL

COMMAND LINE CONTAINER CORS CRACKMAPEXEC CSRF CTF CYBERSECURITY DETECTION DOCKER DOMAIN ADMIN DOMAIN CONTROLLER

DVWA ELEARNSECURITY ELK ELKSTACK ENUMERATION EWPT EXECUTIONPOLICY FREERADIUS GHOST GNU RADIO GOOGLE CLOUD GOPHISH

GRAPH THEORY HACKING HACKRF HASHCAT HIJACKING HTTP HTTP/2 IMPACKET INFORMATION GATHERING INTERNAL NETWORK

INTERNET OF THINGS JAVASCRIPT JUICESHOP JWT KALI LINUX KALI TOOLS KERBEROS LATERAL MOVEMENT LINUX MERLIN MICROSOFT

MICROSOFT OFFICE MINING NE0ND0G NEO4J NETWORKING NULL SESSION OFFENSIVE SECURITY OFFSEC OPEN REDIRECT OSWP OWASP

PASSWORD CRACKING PENETRATION TEST PENTEST PHISHING PHP PINEAPPLE PIXEL TRACKING PORTAINER POST EXPLOITATION POWERSHELL

PROTOCOLS PYTHON RADIO FREQUENCY RECON RED TEAMING RED-BARON REDTEAMING REPORTING REVIEW RF RFC RTL-SDR S3 SAMBA

SCANS SCAPY SCRIPTING SERVICE PRINCIPAL NAME SERVICES SHODAN SMB SMBCLIENT SOCIAL ENGINEERING SOFTWARE DEFINED RADIO SPN

SWARM SYSADMIN TERRAFORM TERRAFORMFUN TRAINING UUID VULNERABILITY SCANNING WARDRIVING WEB APP WEB APPLICATION

WEB TOKEN WEBAPP WIFI WIFU WIGLE WINDOWS WIRELESS WPA XSS

OPINIONS EXPRESSED ARE SOLELY OUR OWN AND DO NOT EXPRESS THE VIEWS OR OPINIONS OF OUR EMPLOYERS.