

인터넷응용보안 9주차 과제

202121556 곽지현

DOM Based XSS (1)

개발자도구 검색창에서 lesson/ 검색 -> js 파일 열기

Base route가 start.mvc#lesson/ 이므로 테스트용 route는 start.mvc#test/ 임을 유추

The screenshot shows a browser window with two tabs, both titled "WebGoat". The URL in the address bar is "127.0.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9". The main content area displays the "Cross Site Scripting" page from the WebGoat application. On the left, there's a sidebar with "Introduction" and "General" sections, and buttons for "Show hints" and "Reset lesson". Below the sidebar, the developer tools are open. The "Sources" tab is selected, and the "start.mvc" file is open. In the code editor, the "GoatRouter.js" file is shown with several routes defined. A search bar at the bottom of the devtools has "lesson/" typed into it. The right side of the devtools interface shows the "Scope" section, which is currently not paused. At the bottom, the console shows some search results related to "lesson/".

```
routes: {
    'welcome': 'welcomeRoute',
    'lesson/:name': 'lessonRoute',
    'lesson/:name/:pageNum': 'lessonPageRoute',
    'test/:param': 'testRoute',
    'reportCard': 'reportCard'
},  
lessonController: null,  
menuController : null,  
titleView: null,
```

lessonmenu.mvc — 127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
9 "link": "#lesson/WebGoatIntroduction.lesson",
16 "link": "#lesson/WebWolfIntroduction.lesson",
Search finished. Found 41 matching lines in 3 files.

빈칸에 start.mvc#test/ 넣고 Submit 버튼 클릭

WEBGOAT

Cross Site Scripting

Search lesson

Show hints Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 +

Identify potential for DOM-Based XSS

DOM-Based XSS can usually be found by looking for the route configurations in the client-side code. Look for a route that takes inputs that are "reflected" to the page.

For this example, you will want to look for some 'test' code in the route handlers (WebGoat uses backbone as its primary JavaScript library). Sometimes, test code gets left in production (and often test code is simple and lacks security or quality controls!).

Your objective is to find the route and exploit it. First though, what is the base route? As an example, look at the URL for this lesson ...it should look something like /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9. The 'base route' in this case is: start.mvc#lesson/ The **CrossSiteScripting.lesson/9** after that are parameters that are processed by the JavaScript route handler.

So, what is the route for the test code that stayed in the app during production? To answer this question, you have to check the JavaScript source.

start.mvc#test|

Correct! Now, see if you can send in an exploit to that route in the next assignment.

DOM Based XSS (1) 성공!

WEBGOAT

CROSS SITE SCRIPTING

Search lesson

Show hints Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 +

Identify potential for DOM-Based XSS

DOM-Based XSS can usually be found by looking for the route configurations in the client-side code. Look for a route that takes inputs that are "reflected" to the page.

For this example, you will want to look for some 'test' code in the route handlers (WebGoat uses backbone as its primary JavaScript library). Sometimes, test code gets left in production (and often test code is simple and lacks security or quality controls!).

Your objective is to find the route and exploit it. First though, what is the base route? As an example, look at the URL for this lesson ...it should look something like /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9. The 'base route' in this case is: start.mvc#lesson/ The **CrossSiteScripting.lesson/9** after that are parameters that are processed by the JavaScript route handler.

So, what is the route for the test code that stayed in the app during production? To answer this question, you have to check the JavaScript source.

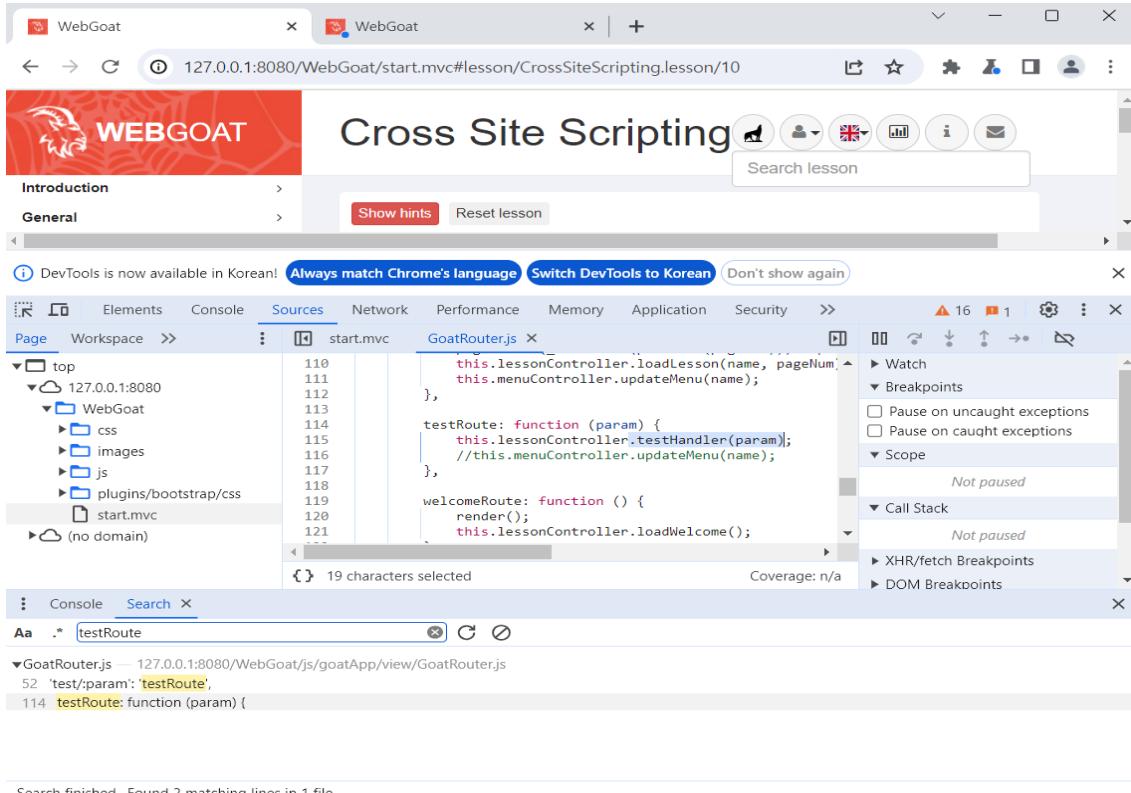
✓

Correct! Now, see if you can send in an exploit to that route in the next assignment.

DOM Based XSS (2)

개발자도구 검색창에서 testRoute 검색 -> js 파일 열기

testHandler에서 파라미터를 처리하고 있는 것을 확인

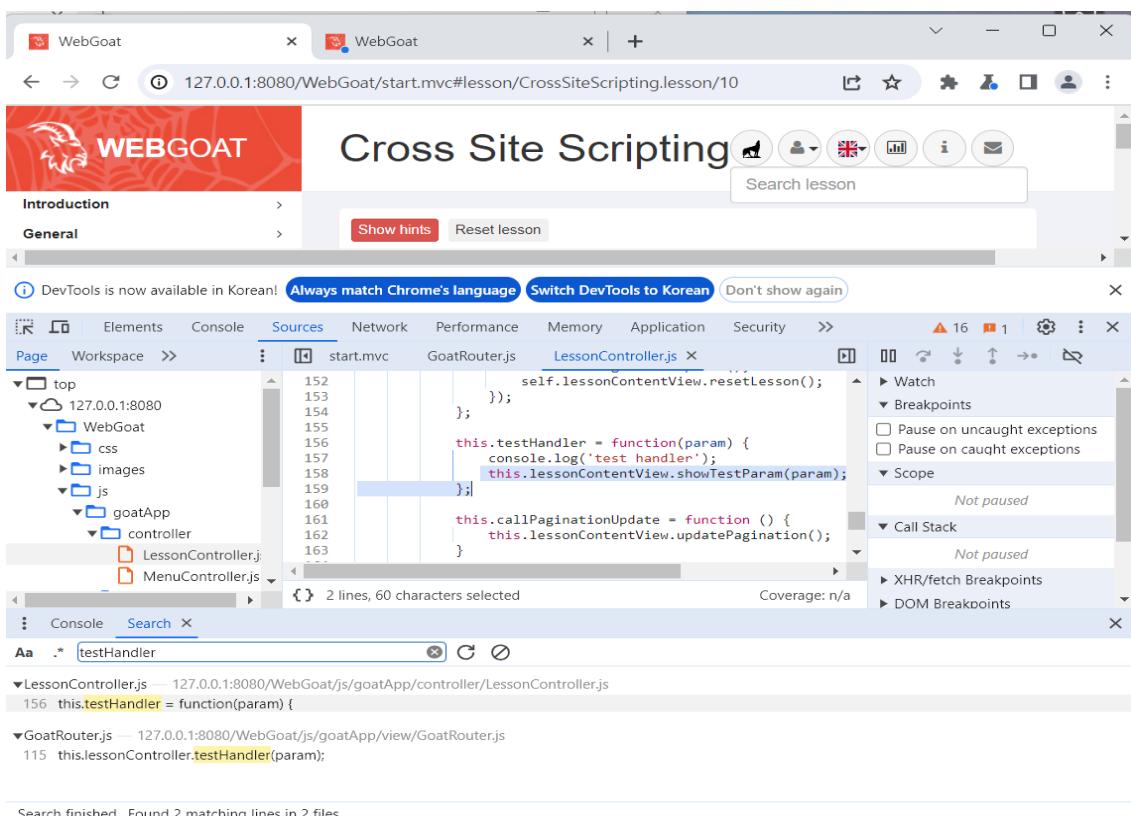


```
110     this.lessonController.loadLesson(name, pageNum);
111     this.menuController.updateMenu(name);
112 },
113
114 testRoute: function (param) {
115     this.lessonController.testHandler(param);
116     //this.menuController.updateMenu(name);
117 },
118
119 welcomeRoute: function () {
120     render();
121     this.lessonController.loadWelcome();
```

Search finished. Found 2 matching lines in 1 file.

다시 검색창에서 testHandler 검색 -> js 파일 열기

showTestParam()에서 파라미터를 처리하고 있는 것을 확인

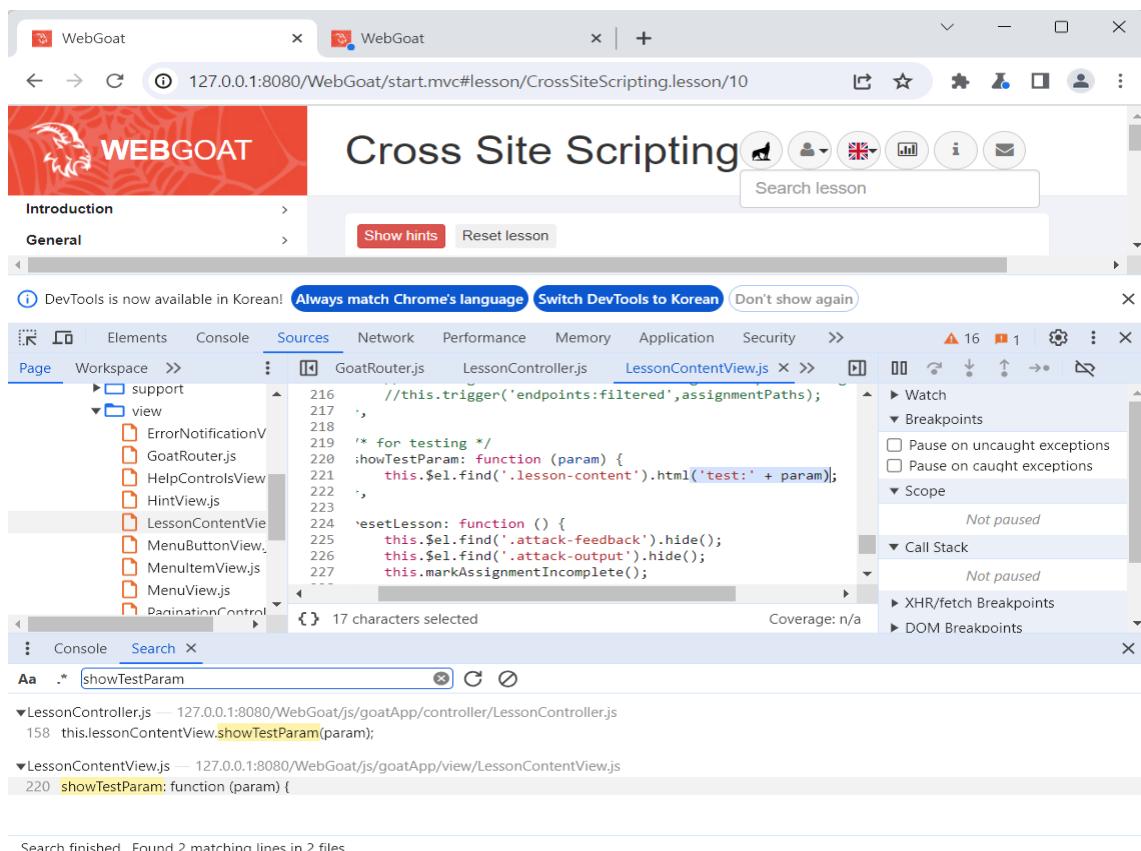


```
152         self.lessonContentView.resetLesson();
153     };
154
155     this.testHandler = function(param) {
156         console.log('test handler');
157         this.lessonContentView.showTestParam(param);
158     };
159
160     this.callPaginationUpdate = function () {
161         this.lessonContentView.updatePagination();
162     }
163 }
```

Search finished. Found 2 matching lines in 2 files.

다시 검색창에서 showTestParam 검색 -> js 파일 열기

showTestParam() 함수는 파라미터를 lesson-content 클래스를 통해 내장 HTML 코드로 Test 단어와 함께 추가한다는 것을 확인



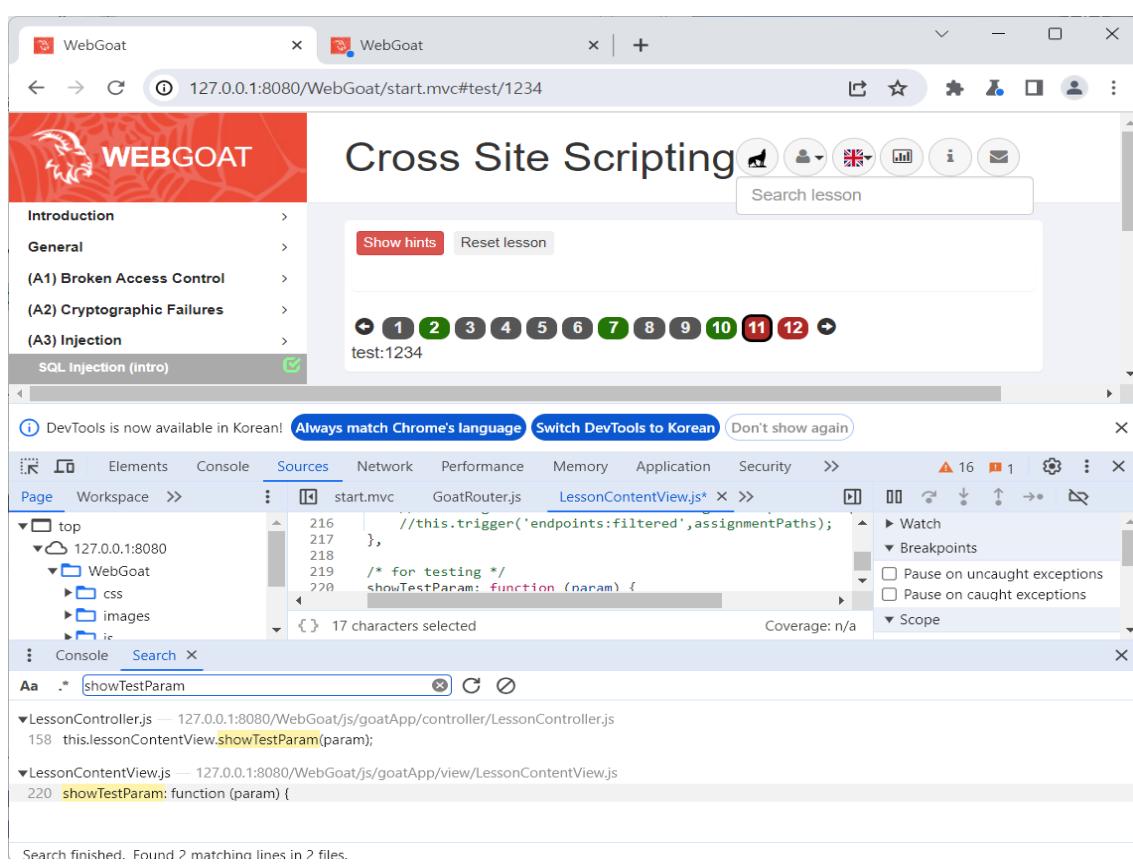
The screenshot shows the Chrome DevTools interface with the Sources tab selected. The left sidebar shows the project structure with files like GoatRouter.js, LessonController.js, and LessonContentView.js. The LessonContentView.js file is open, displaying the following code:

```
216 //this.trigger('endpoints:filtered',assignmentPaths);
217 },
218 /* for testing */
219 showTestParam: function (param) {
220     this.$el.find('.lesson-content').html('test:' + param);
221 },
222 },
223 resetLesson: function () {
224     this.$el.find('.attack-feedback').hide();
225     this.$el.find('.attack-output').hide();
226     this.markAssignmentIncomplete();
227 }
```

The search bar at the bottom shows "showTestParam". The console output below shows two matching lines from LessonController.js and LessonContentView.js.

```
158 this.lessonContentView.showTestParam(param);
220 showTestParam: function (param) {
```

주소창에 start.mvc#test/1234 넣고 수행 -> test: 1234가 화면에 표시

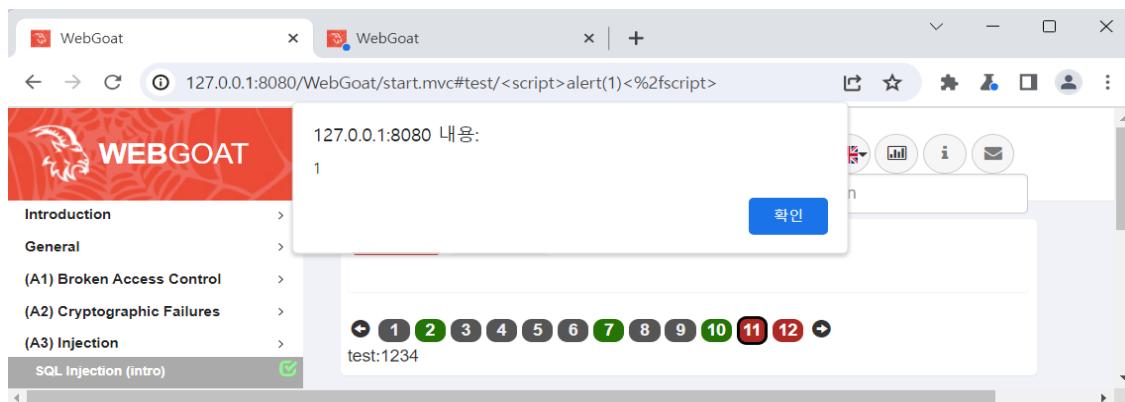


The screenshot shows the Chrome DevTools interface with the Sources tab selected. The left sidebar shows the project structure with files like start.mvc, GoatRouter.js, and LessonContentView.js. The LessonContentView.js file is open, displaying the same code as before:

```
216 //this.trigger('endpoints:filtered',assignmentPaths);
217 },
218 /* for testing */
219 showTestParam: function (param) {
220     this.$el.find('.lesson-content').html('test:' + param);
221 },
222 },
223 resetLesson: function () {
224     this.$el.find('.attack-feedback').hide();
225     this.$el.find('.attack-output').hide();
226     this.markAssignmentIncomplete();
227 }
```

The search bar at the bottom shows "showTestParam". The browser window shows the URL 127.0.0.1:8080/WebGoat/start.mvc#test/1234. The page displays the text "test:1234" in the center.

주소창에 start.mvc#test/<script>alert(1)<%2fscript> 넣고 수행 -> Alert 창이 뜨는 것을 확인



The screenshot shows a browser window with two tabs both titled "WebGoat". The URL in the address bar is "127.0.0.1:8080/WebGoat/start.mvc#test/<script>alert(1)<%2fscript>". A modal dialog box is displayed in the center, containing the Korean text "127.0.0.1:8080 내용:" followed by the number "1". A blue button labeled "확인" (Confirm) is at the bottom right of the modal. Below the modal, a navigation bar has items 1 through 12, with item 11 highlighted in red. The status bar at the bottom of the browser says "test:1234".

DevTools is open, showing the Elements tab with the HTML structure of the page. The Styles tab shows a rule for "body" with "font-family: inherit". The Console tab shows the following JavaScript code:

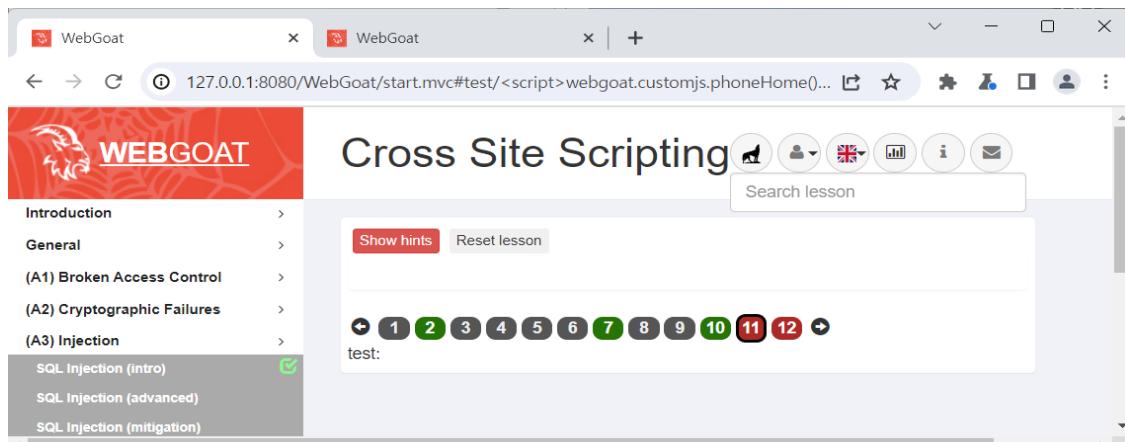
```
!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
... <head>... </head> == $0
<body>
  <section id="container"></section>
  <!-- About WebGoat Modal -->
  <div class="modal" id="about-modal" tabindex="-1" role="dialog" aria-labelledby="mvModalLabel" aria-...
html head
```

```
LessonController.js — 127.0.0.1:8080/WebGoat/js/goatApp/controller/LessonController.js
158 this.lessonContentView.showTestParam(param);

LessonContentView.js — 127.0.0.1:8080/WebGoat/js/goatApp/view/LessonContentView.js
220 showTestParam: function (param) {
```

Search finished. Found 2 matching lines in 2 files.

주소창에 start.mvc#test/<script>webgoat.customjs.phoneHome()<%2fscript> 넣고 수행 -> 콘솔창에서 생성된 난수를 확인 (난수 : -1542427723)



The screenshot shows a browser window with two tabs both titled "WebGoat". The URL in the address bar is "127.0.0.1:8080/WebGoat/start.mvc#test/<script>webgoat.customjs.phoneHome()<%2fscript>". A modal dialog box is displayed in the center, containing the text "Cross Site Scripting" and a search bar labeled "Search lesson". Below the modal, a navigation bar has items 1 through 12, with item 11 highlighted in red. The status bar at the bottom of the browser says "test:". The DevTools console tab is open, showing the following output:

```
phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.","output":"phoneHome Response is -1542427723","assignment":"DOMCrossSiteScripting","attemptWasMade":true}
```

Console tab content: "showTestParam"

뒤로 돌아가 빈칸에 생선된 난수 (-1542427723) 넣고 Submit 버튼 클릭

WEBGOAT

Cross Site Scripting

Introduction >

General >

(A1) Broken Access Control >

(A2) Cryptographic Failures >

(A3) Injection >

(A5) Security Misconfiguration >

(A6) Vuln & Outdated Components >

(A7) Identity & Auth Failure >

(A8) Software & Data Integrity >

(A9) Security Logging Failures >

(A10) Server-side Request Forgery >

Client side >

Challenges >

Show hints Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 +

Try It! DOM-Based XSS

Some attacks are "blind." Fortunately, you have the server running here, so you can tell if you are successful. Use the route you just found and see if you can use it to reflect a parameter from the route without encoding to execute an internal function in WebGoat. The function you want to execute is:

`webgoat.customjs.phoneHome()`

Sure, you could use console/debug to trigger it, but you need to trigger it via a URL in a new tab.

Once you trigger it, a subsequent response will come to your browser's console with a random number. Put that random number below.

-1542427723

DOM Based XSS (2) 성공!

WEBGOAT

Cross Site Scripting

Introduction >

General >

(A1) Broken Access Control >

(A2) Cryptographic Failures >

(A3) Injection >

(A5) Security Misconfiguration >

(A6) Vuln & Outdated Components >

(A7) Identity & Auth Failure >

(A8) Software & Data Integrity >

(A9) Security Logging Failures >

(A10) Server-side Request Forgery >

Client side >

Challenges >

Show hints Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 +

Try It! DOM-Based XSS

Some attacks are "blind." Fortunately, you have the server running here, so you can tell if you are successful. Use the route you just found and see if you can use it to reflect a parameter from the route without encoding to execute an internal function in WebGoat. The function you want to execute is:

`webgoat.customjs.phoneHome()`

Sure, you could use console/debug to trigger it, but you need to trigger it via a URL in a new tab.

Once you trigger it, a subsequent response will come to your browser's console with a random number. Put that random number below.

✓

Correct, I hope you did not cheat, using the console!

모든 문제 성공!

The screenshot shows a browser window with two tabs both titled "WebGoat". The active tab displays the URL 127.0.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/11. The page title is "Cross Site Scripting". On the left, there is a sidebar with a red header containing a goat logo and the word "WEBGOAT". The sidebar menu includes categories like "Introduction", "General", and various sub-categories under "(A1) Broken Access Control" through "(A10) Server-side Request Forgery", as well as "Client side" and "Challenges". On the right, the main content area has a "Search lesson" input field and a navigation bar with icons for user profile, help, and email. Below these are numbered buttons from 1 to 12. A text block below the buttons reads: "Now it is time for a quiz! It is recommended to check the OWASP Cross-Site Scripting explanations <https://owasp.org/www-community/attacks/xss/>. Answer all questions correctly to complete the assignment." A green box contains a question: "1. Are trusted websites immune to XSS attacks?" with four multiple-choice options, each preceded by a radio button. The options are: "Solution 1: Yes they are safe because the browser checks the code before executing.", "Solution 2: Yes because Google has got an algorithm that blocks malicious code.", "Solution 3: No because the script that is executed will break through the defense algorithm of the browser.", and "Solution 4: No because the browser trusts the website if it is acknowledged trusted, then the browser does not know that the script is malicious."