

# **RSA**®Conference2015

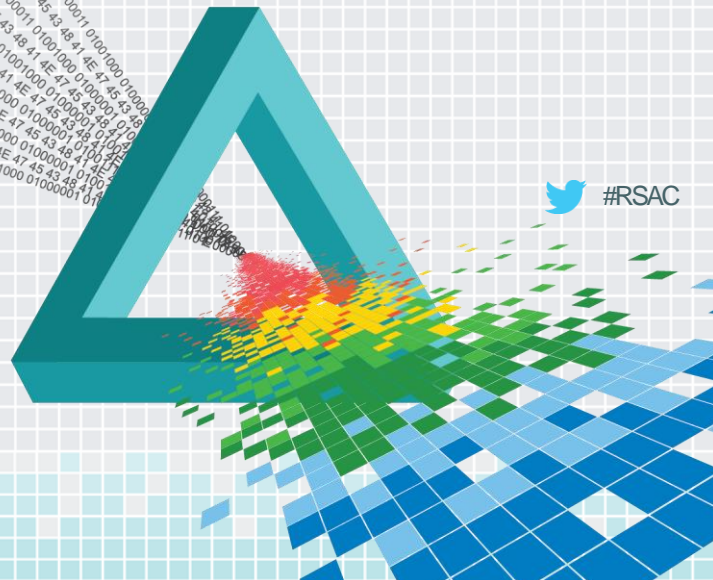
San Francisco | April 20-24 | Moscone Center

SESSION ID: ASD-W03

## Game of Hacks: The Mother of All Honeydumps

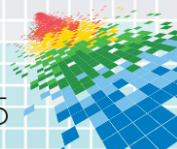
**Maty Siman**

CTO, Founder at Checkmarx



# Agenda

- ◆ What is Game Of Hacks?
- ◆ And now really...
- ◆ Discovery process
- ◆ What vulnerabilities existed in the code
- ◆ Which ones were detected
- ◆ Conclusions



# CISO Concerns



## OWASP CISO Survey 2013

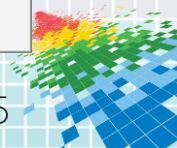
(<https://www.owasp.org/images/2/28/Owasp-ciso-report-2013-1.0.pdf>)

### *Top 5 CISO Priorities*

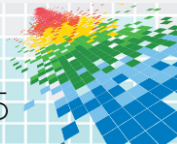
1. Security awareness and training for developers
2. Secure development lifecycle processes (e.g., secure coding, QA process)
3. Security testing of applications (dynamic analysis, runtime observation)
4. Application layer vulnerability management technologies and processes
5. Code review (static analysis of source code)

### *Top 5 CISO Challenges to effectively delivering your organization's application security initiatives*

1. Availability of skilled resources
2. Level of security awareness by the developers
3. Management awareness and sponsorship
4. Adequate budget
5. Organizational change



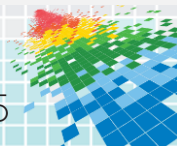
# Game of Hacks



# Game of Hacks

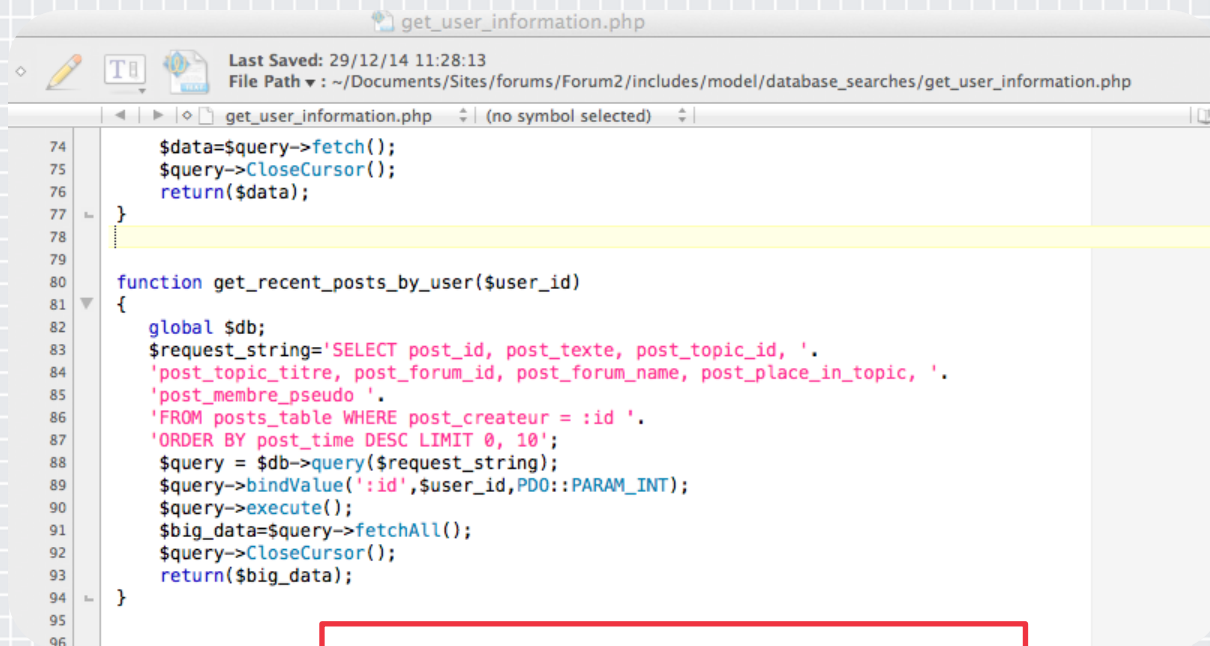


It all started when we noticed a group of geeks staring at a wall at Blackhat 2014



# Game of Hacks

Something like that is what they were looking at

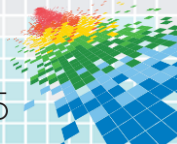


```

74  $data=$query->fetch();
75  $query->CloseCursor();
76  return($data);
77  }
78  .....
79
80  function get_recent_posts_by_user($user_id)
81  {
82      global $db;
83      $request_string='SELECT post_id, post_texte, post_topic_id, '
84      'post_topic_titre, post_forum_id, post_forum_name, post_place_in_topic, '
85      'post_membre_pseudo '
86      'FROM posts_table WHERE post_createur = :id '
87      'ORDER BY post_time DESC LIMIT 0, 10';
88      $query = $db->query($request_string);
89      $query->bindValue(':id',$user_id,PDO::PARAM_INT);
90      $query->execute();
91      $big_data=$query->fetchAll();
92      $query->CloseCursor();
93      return($big_data);
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

```

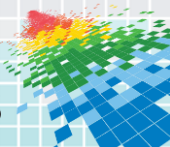
**Find the vulnerability !**





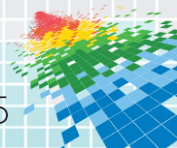
# Game of Hacks

That got us thinking...



# Game of Hacks

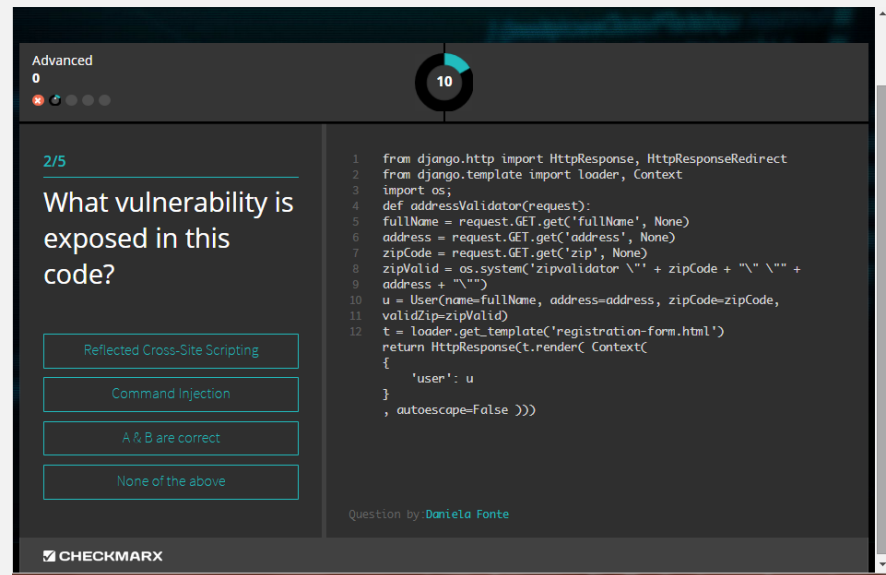
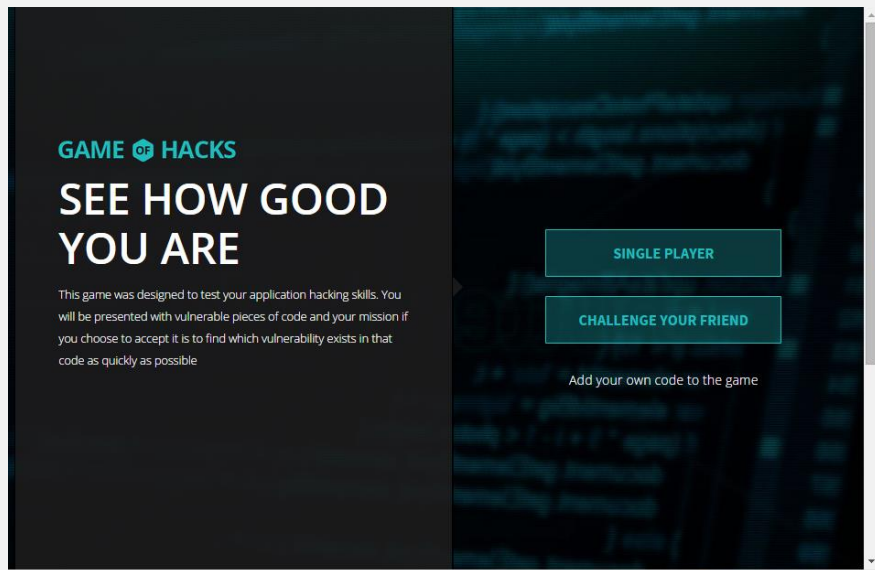
How can we leverage this behavior?





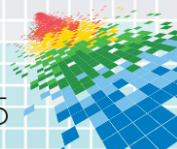
# 1+1=?

## An online “spot the vulnerability” game, AKA – “Game Of Hacks”

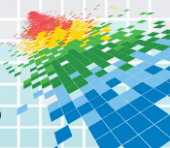


# Game of Hacks (GoH)

- ◆ Launched on August 2014
- ◆ 80,000 games were played since



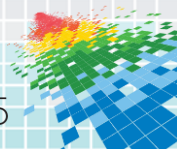
# What's behind GoH?



It was fairly safe to assume that people would try to hack the game.

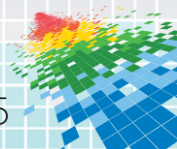


<https://news.ycombinator.com/item?id=8134699>

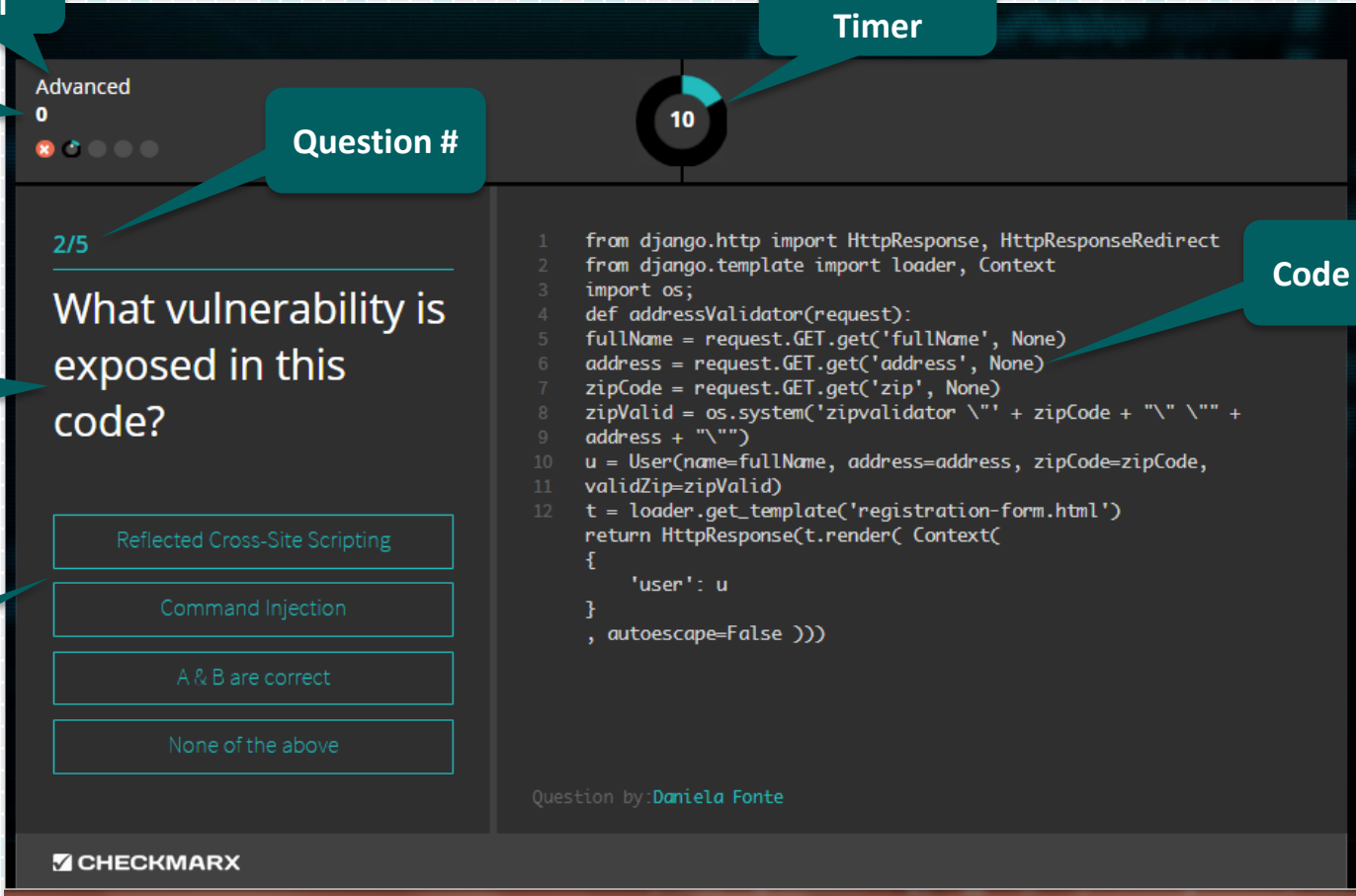


# Honeypot

- ◆ We might as well learn from it
- ◆ Various vulnerabilities with different degrees of complexity were left open on purpose
- ◆ Each vulnerability was patched once discovered







**Difficulty Level**

**Score**

Advanced  
0

**Question #**

10

**60-Second Timer**

**Code Snippet**

2/5

**Question**

What vulnerability is exposed in this code?

**Answers**

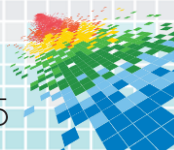
- Reflected Cross-Site Scripting
- Command Injection
- A & B are correct
- None of the above

```

1  from django.http import HttpResponse, HttpResponseRedirect
2  from django.template import loader, Context
3  import os;
4  def addressValidator(request):
5      fullName = request.GET.get('fullName', None)
6      address = request.GET.get('address', None)
7      zipCode = request.GET.get('zip', None)
8      zipValid = os.system('zipvalidator \'' + zipCode + "\" \"'\" +
9      address + "\"\"')
10     u = User(name=fullName, address=address, zipCode=zipCode,
11     validZip=zipValid)
12     t = loader.get_template('registration-form.html')
13     return HttpResponse(t.render( Context(
14     {
15         'user': u
16     }
17     , autoescape=False )))
  
```

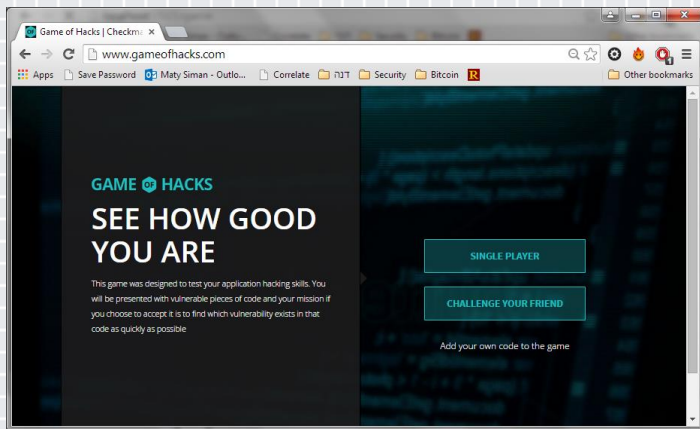
Question by: Daniela Fonte

**CHECKMARX**

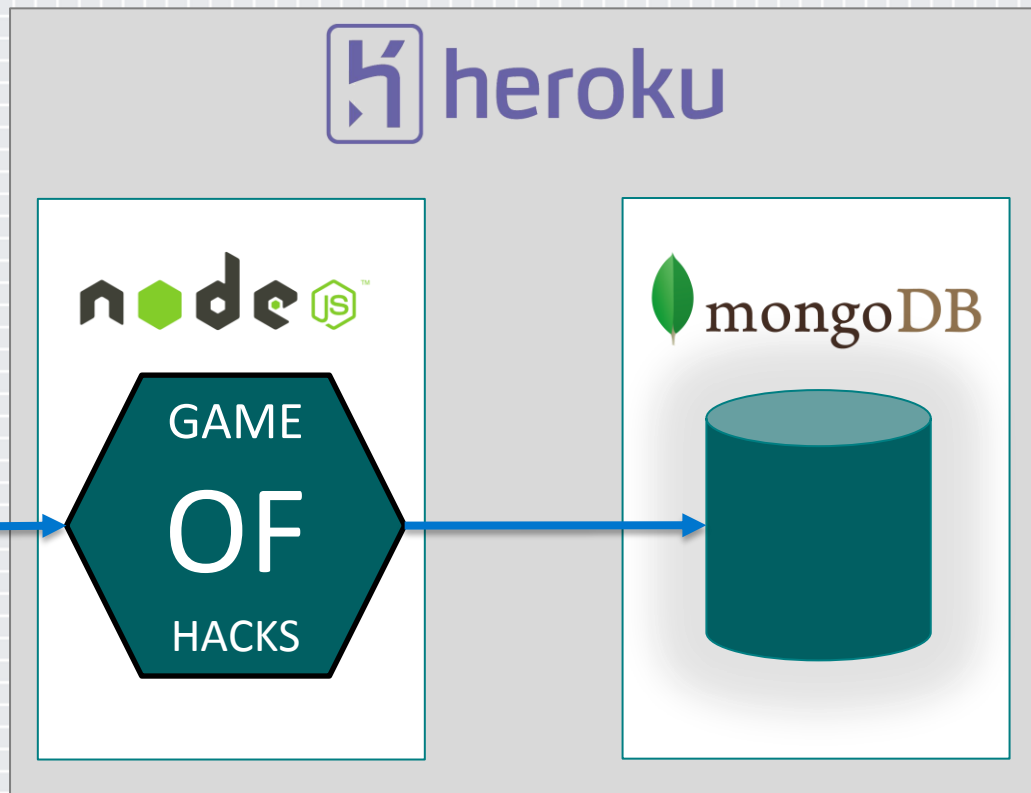


# GoH Architecture

## CLIENT



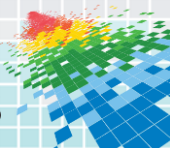
## SERVER



# Vulnerability Planting Process

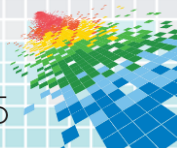
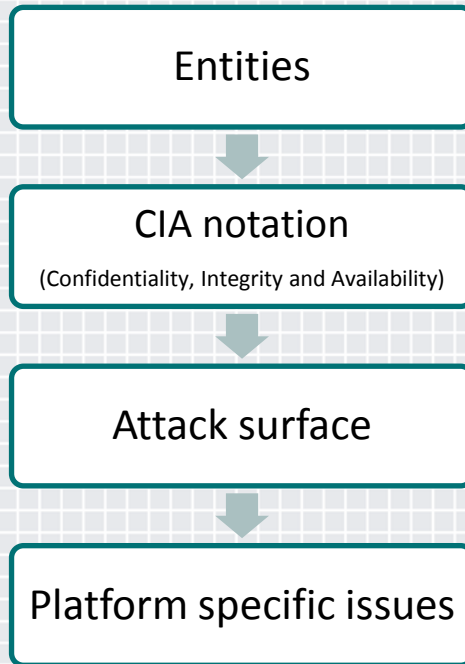


Why should you care?



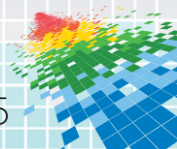
# Vulnerability Planting Process

- ◆ Use our Vulnerability planting process to detect weak points in your application.



# Entities

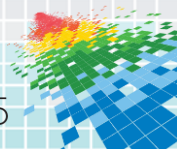
- ◆ Quiz Questions
- ◆ Answers
- ◆ Score
- ◆ Timer
- ◆ User





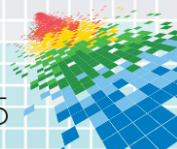
# Quiz Questions

- ◆ Client
  - ◆ How to retrieve the questions from the server?
  - ◆ How to send the answers back to the server?
  - ◆ Who is responsible for randomizing the order of questions?
- ◆ Server
  - ◆ How to validate the “real” client questions from a forged one?



# Answers

- ◆ Client
  - ◆ How to validate the answer?
- ◆ Server
  - ◆ How to mark a question as “answered”?



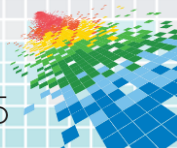
# Answered Question

- ◆ At first, users could initiate "app.sendAnswers" multiple times until they got a "correct answer" response.
- ◆ This allowed malicious users to systematically locate the correct answer – and to gain points over and over for the same question.

▲ mctx 202 days ago

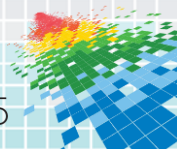
alex here. After getting a question correct, you can submit the same POST request with the same answer, and a very large negative number for the time. I imagine they're just adding your newly calculated score ( $30? - \text{time}$ ) to your session's previous score. A lesson in sanitising inputs!

- ◆ Solution:
  - ◆ "Question Already Answered" flag added



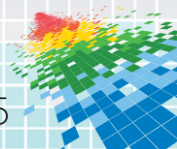
# Score

- ◆ Client
  - ◆ How to manage the score?
  - ◆ How it is calculated?
  - ◆ Can it be manipulated?
- ◆ Server
  - ◆ How to validate the various components used to compute the score?
  - ◆ Where is the score is computed?
  - ◆ Where is the score stored?



# Timer

- ◆ Client
  - ◆ How to enforce the timer?
  - ◆ What if someone answers too quickly?
- ◆ Server
  - ◆ Who should enforce and validate the timer?

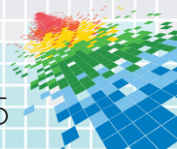




# Timer

The timer has two purposes:

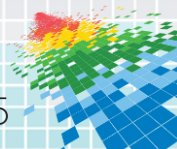
- ◆ Limit the time user is allowed to respond
  - ◆ This raises a question about how to enforce time limits
  - ◆ Does this enforcement take place at the client side or server side?
- ◆ Score calculation
  - ◆ The time it took the user to answer is part of the formula used to compute the score
  - ◆ How do you transmit the time from client to server and how do you validate the values?



# Timer

- ◆ On the first version of GoH, the timer was handled by the client.
- ◆ If the timer went off, the user was forced to go to the next question.
- ◆ The client sent the server the time it took to answer alongside the selected answer

**So what...?**



# Timer

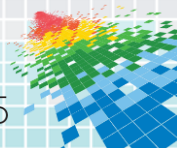
- ◆ As a result players were able to easily stop the timer by modifying the JS code
- ◆ We didn't expect the following:

▲ Sonicmouse 203 days ago

Play it on an iPhone. You can pause the timer by holding your finger on the iPhone's screen.  
Yeah, it's cheating, but isn't that what it's all about?

▲ CaRDiaK 203 days ago

Hacking the hack!



# Timer

- ◆ Score computation was  $(60 - \text{TimeToAnswer}) * \text{DifficultyLevel}$
- ◆ So they sent 0 to get max value
- ◆ Or even negative values

▲ granttimmerman 203 days ago

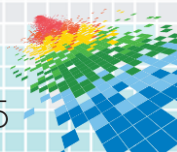
Here's how to hack the hacking game. Pretty simple (in your console):

```
app.sendAnswer({answer: 1,time: -999999999999})
```

(I added the instructions on the leaderboard itself)

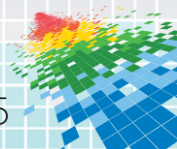
▲ byerley 203 days ago

Somewhat ironic that the high scores have already been hacked, though a little inevitable since the game is client side I guess.



# Timer

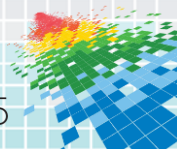
- ◆ In order to fix the vulnerability, the time is now computed at the server
- ◆ Yes – traffic time influences that a bit





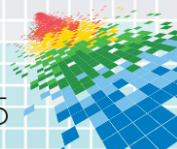
# User

- ◆ Client
  - ◆ How to validate user names?
- ◆ Server
  - ◆ How to enforce valid user names?
  - ◆ What are valid user names?



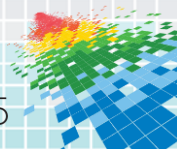
# Client side – Platform security considerations

- ◆ Client side was developed in Javascript
  - ◆ XSS
    - ◆ Dom based
  - ◆ Command Injection (Eval)



# Server Side – Platform security considerations

- ◆ Node.js apps might be vulnerable to countless types of attacks
  - ◆ Plain SQL Injection
  - ◆ JSON Based SQL Injection
  - ◆ Traceless Routing Hijacking
  - ◆ SSJS Injection
  - ◆ Weak Client Side Crypto
  - ◆ Etc.



# Meta-Meta-Game

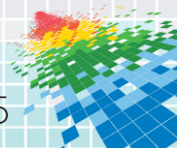
- ◆ Can you spot the vulnerabilities in the code?

```
SELECT * FROM users WHERE username = '$username' AND password = '$password'
```

- ◆ Fix:

```
db.users.find({username: username, password: password});
```

**wrong**



# JSON-base SQL Injection

- ◆ Node.JS, being a JSON based language, can accept JSON values for the .find method:

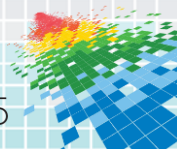
```
db.users.find({username: username, password: password});
```

- ◆ A user can bypass it by sending

```
POST http://target/ HTTP/1.1
Content-Type: application/json

{
  "username": {"$gt": ""},
  "password": {"$gt": ""}
}
```

- <http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html>



# JSON-base SQL Injection

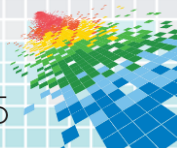
- ◆ You can use the following:

```
db.users.find({username: username});
```

Then

```
bcrypt.compare(candidateWord, password, cb);
```

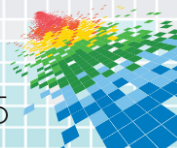
**wrong**



# JSON-base SQL Injection

```
db.users.find({username: username});
```

- ◆ This can lead to Regular Expression Denial of Service through the `{"username": {"$regex": "....."}}`
- ◆ So always validate the input as a string, not JSON
- ◆ This vulnerability was NOT detected by GoH users

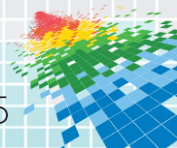






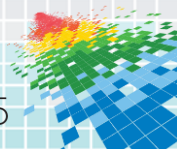
# Weak Crypto

- ◆ We used a weak crypto library.
- ◆ Google knows it's weak.
- ◆ Since it is used only at the client side, it is considered as less risky.
- ◆ Interesting... Node.js is server side JS based on Google Chrome (V8)
- ◆ The random generator is predictable – three consecutive values reveal past and future.
- ◆ Think of it as a quadric equation. With three data points you can draw the full graph.
- ◆ There wasn't any specific vulnerability. Just weaknesses. We were interested to see how they would be exploited. We are not aware if any were actually exploited.



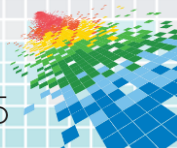
# It wasn't all a walk in the park

- ◆ There were some “attacks” we didn't expect and we had to handle in real time.
- ◆ Some were not technical but more “branding” oriented.



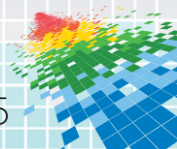
# Bots

- ◆ After we fixed the issues we discussed earlier, people developed bots that just guessed answers
  - ◆ At the end of the day, there are  $4^5 = 1024$  options, so there is a chance of  $1023/1024$  of getting at least one answer wrong. (or  $1/1024$  to get it all right)
  - ◆ There is 50% chance to get an “all-good” game if you try 708 times
    - ◆  $(1023/1024)^{708} \approx 50\%$
- ◆ The bots answered questions in a fraction of a second, so they always achieved the highest score possible



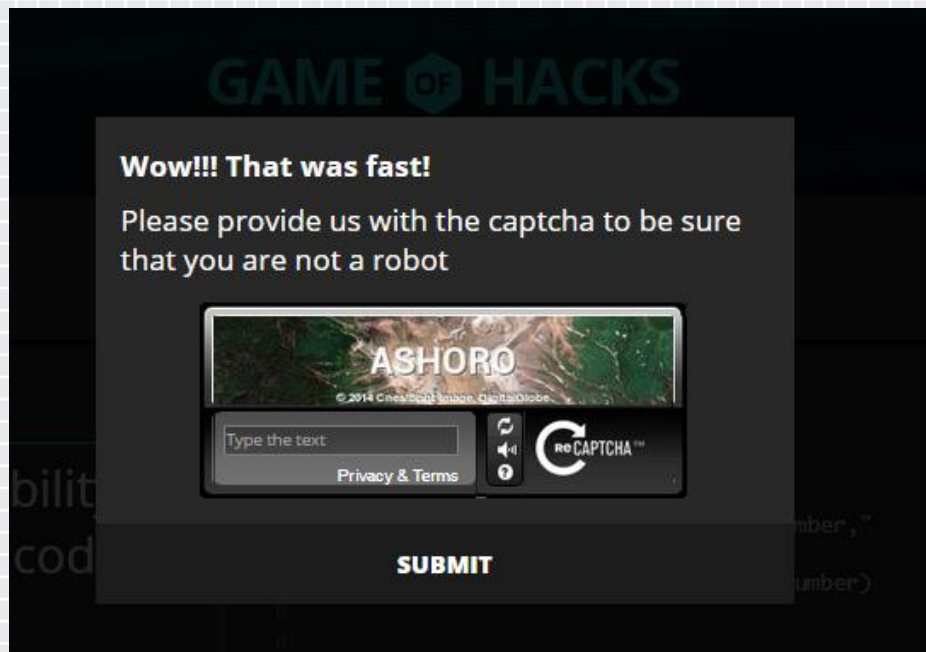
# Bots

- ◆ Some taught their bots the correct answers

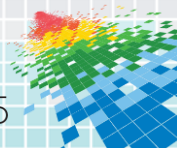




# Bots



We added a single captcha if a question was answered in less than a second



# Offensive Language

- ◆ People played a lot to eventually get to the leader board and share their life philosophy
- ◆ We had to manually approve every leaderboard entry

LEADERBOARD		
1	WE ARE KKK, SHOT [REDACTED]	420000
2	SEE HOW STUPID YOU ARE!!!	420000
3	ONE SHOT FINISH A [REDACTED]	420000
4	KILL ALL [REDACTED] !!!!!!!!!!!	420000
5	KILL [REDACTED] FOR YOUR COUNTRY	420000
6	KILL [REDACTED] !!! ☒	420000
7	□□□+□/☒☒☒□□	418600
8	☒	418600
9	☒☒☒☒☒	418600
10	☒☒☒☒☒	417200

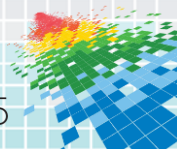
CLAIM YOUR SEAT ON THE IRON THRONE

PLAY ANOTHER GAME



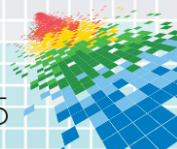
# Conclusions

- ◆ Make your own custom made Honeypot to learn more about hacker techniques
- ◆ Plan well and design what exactly you are interested in watching
- ◆ Prepare for the unexpected. Super-fast response.



# Apply

- ◆ The methodology we used to analyze the application can be used by you to protect your assets
- ◆ In your next project – follow this process to find weak points in your application
  - ◆ Build a top-down chart of the various components
  - ◆ Describe the entities managed by the components
    - ◆ For each entity, describe the relevant risks. Break down by CIA (Confidentially, Integrity and Availability)
  - ◆ Describe the inputs and outputs of each component
  - ◆ Describe unique risks each of the used technologies exposes
- ◆ Scan your code regularly!



# Thank you

Maty Siman | Founder & CTO at Checkmarx

Maty at Checkmarx dot com

