RSAConference2015

San Francisco | April 20-24 | Moscone Center

SESSION ID: ASD-W03

Game of Hacks: The Mother of All Honeypots

Maty Siman

CTO, Founder at Checkmarx

#RSAC

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 t

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

get_user_information.php

Last Saved: 29/12/14 11:28:13
File Path ▾ : ~/Documents/Sites/forums/Forum2/includes/model/database_searches/get_user_information.php

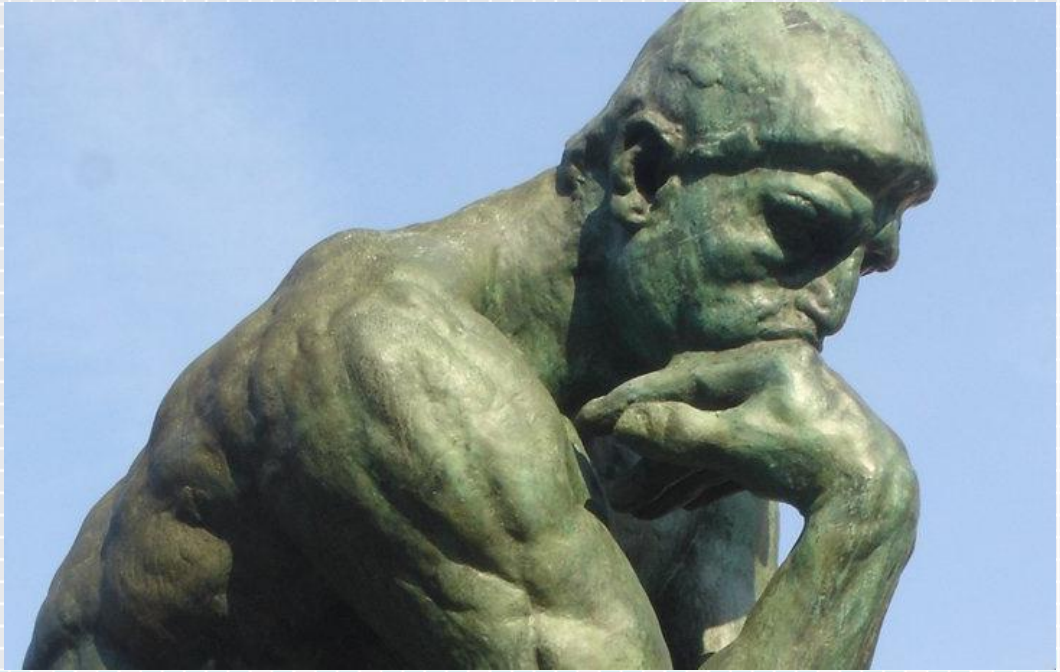
get_user_information.php (no symbol selected)

```
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
```

Find the vulnerability !

Game of Hacks

That got us thinking…

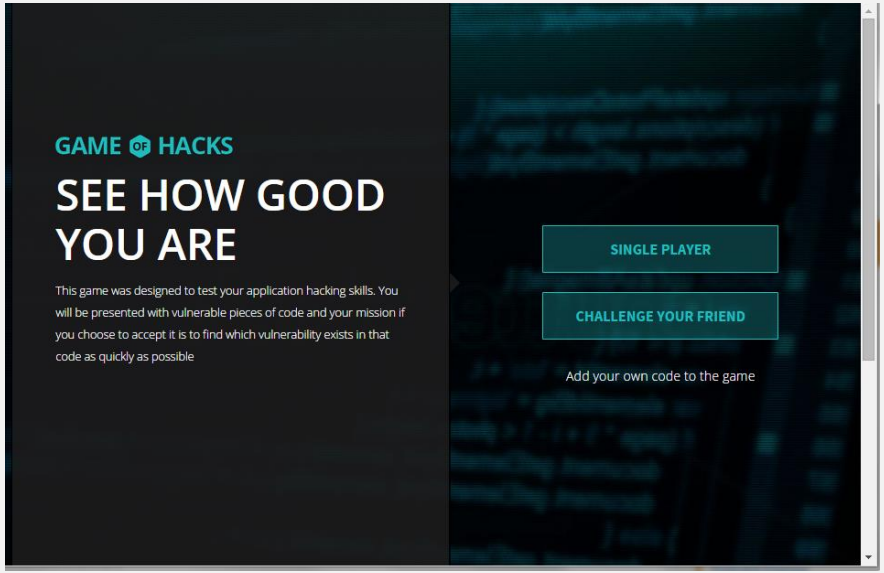


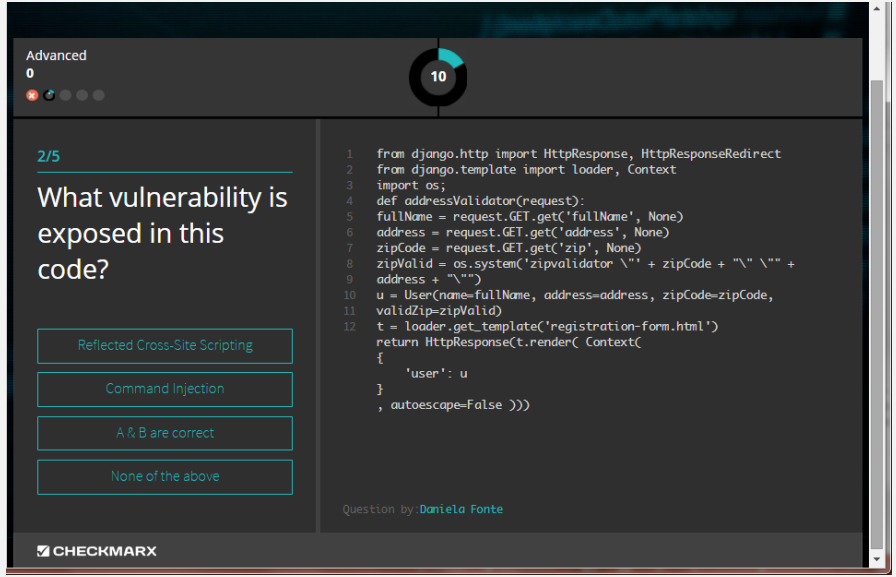
Game of Hacks

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?

Honeypot

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

▲ ZirconCode 203 days ago

The best part of this game is the leaderboard:

7e+177 - maplesyrupguy

1e+73 - Game itself was harder than to hack it

1.3999999999999998e+26 - I learned at least one thing...

1.555555682012021e+24 - Tries to teach security, fails at security.

1.4e+22 - jon

1e+22 - idiots

123456789212345710000 - Ironically it's easy to hack a game about finding security bugs!

100100000000000000000 - did i do

70000000000000130000 - alex

2800000000000251000 - frankwins

▲ kybernetyk 202 days ago

Maybe that was the meta 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

Question #

60-Second Timer

Question

Code Snippet

Answers

Advanced

0

10

2/5

What vulnerability is exposed in this code?

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

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader, Context
import os;
def addressValidator(request):
fullName = request.GET.get('fullName', None)
address = request.GET.get('address', None)
zipCode = request.GET.get('zip', None)
zipValid = os.system('zipvalidator \"' + zipCode + "\" \"" +
address + "\"")
u = User(name=fullName, address=address, zipCode=zipCode,
validZip=zipValid)
t = loader.get_template('registration-form.html')
return HttpResponse(t.render( Context(
{
    'user': u
}
, autoescape=False )))
```

Question by: Daniela Fonte

CHECKMARX

GoH Architecture

SERVER

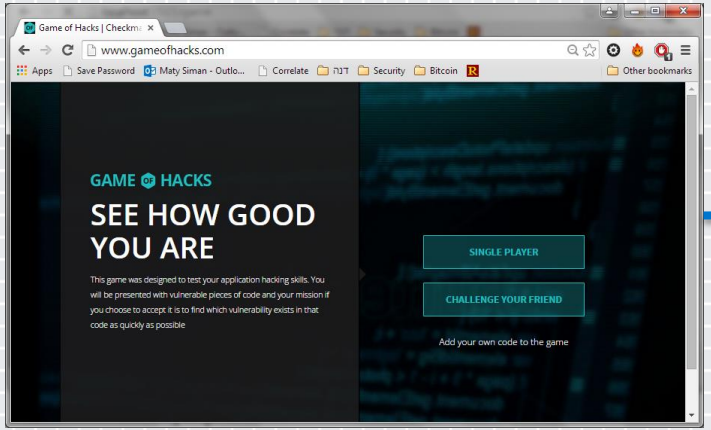
heroku

node.js

GAME OF HACKS

mongoDB

CLIENT

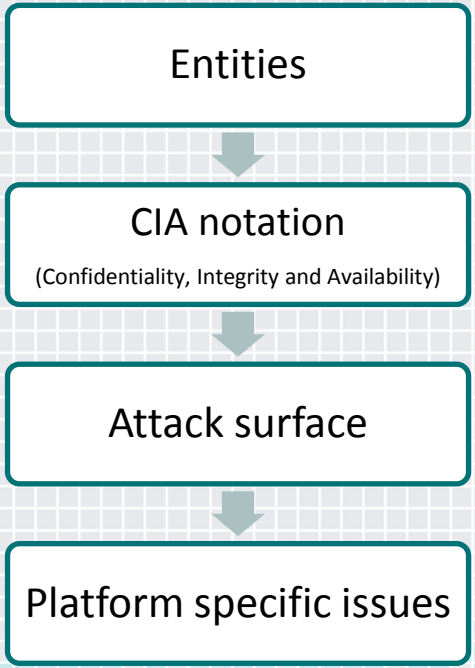


Vulnerability Planting Process

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?-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 – TimeToAnswer) * 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

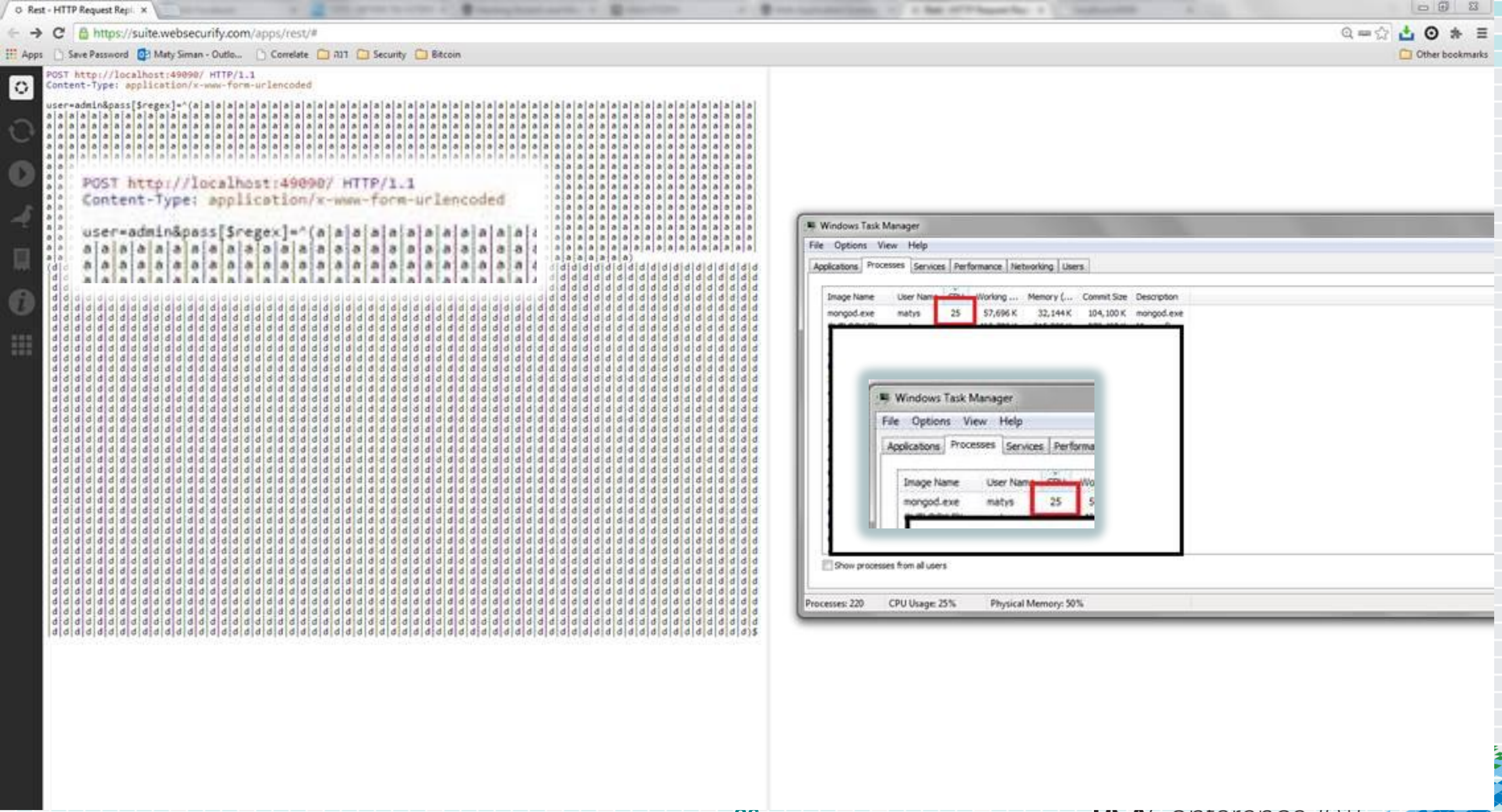
```
bcrypt.compare(can[illegible]word, password, cb);
```

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 <u>NOT</u> detected by GoH users



```
POST http://localhost:49090/ HTTP/1.1
Content-Type: application/x-www-form-urlencoded

user=admin&pass[$regex]=^(a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a
```

Image Name	User Name	CPU	Working ...	Memory (...	Commit Size	Description
mongod.exe	matys	25	57,696 K	32,144 K	104,100 K	mongod.exe

Processes: 220 CPU Usage: 25% Physical Memory: 50%

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.

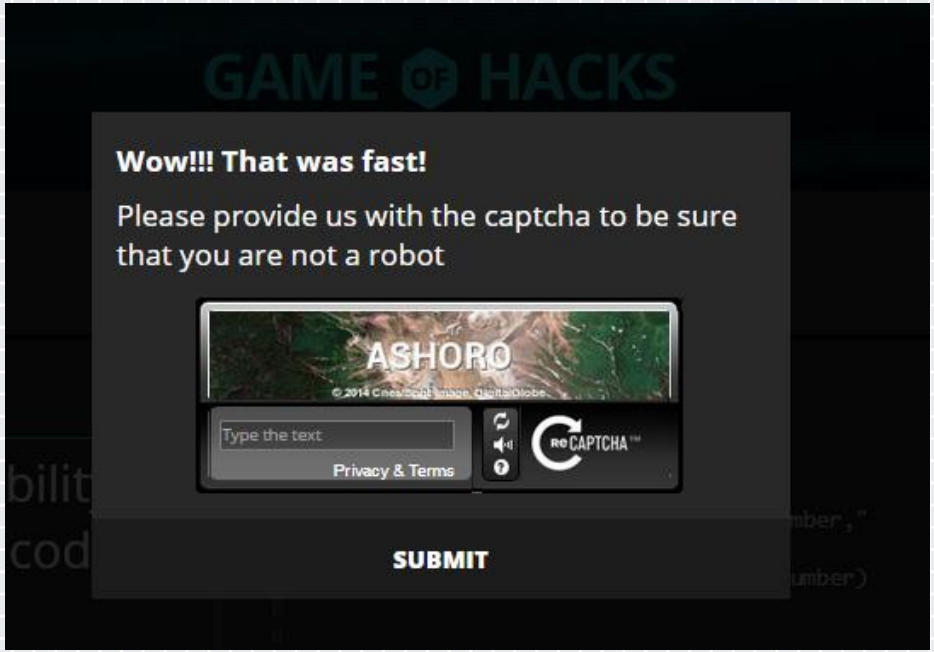
Bots

- 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 ~= %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

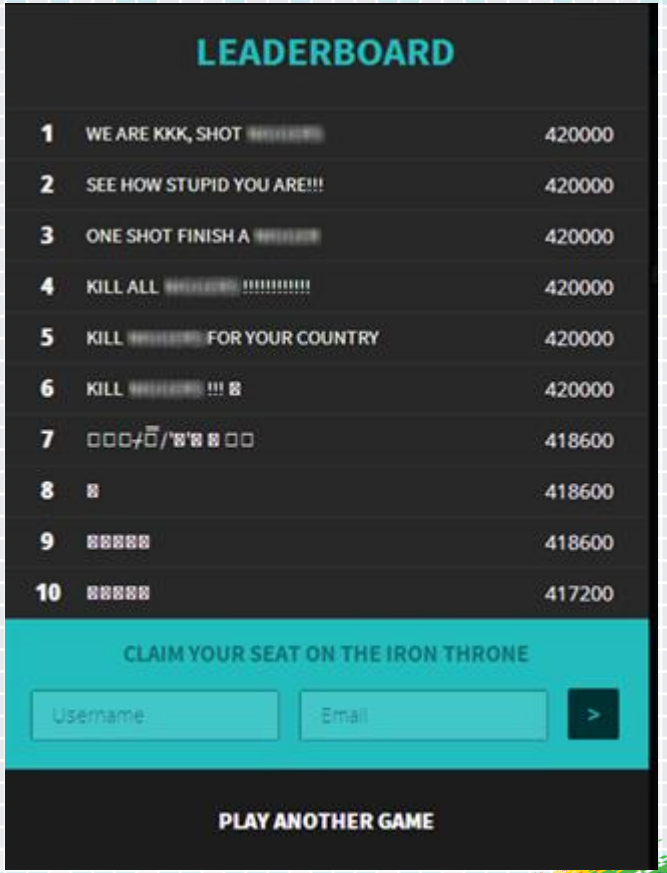
Bots



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



Conclusions

- Make you 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