Backend IV: “Authentication, Authorization and Sanitization”
The Internet is a scary place
Sony Pictures Entertainment hack

From Wikipedia, the free encyclopedia

The Sony Pictures Entertainment cyberhack was a release of confidential data belonging to Sony Pictures on November 24, 2014. The data included personal information about Sony Pictures employees and their family members, information about executive salaries at the company, copies of unreleased Sony films, and other sensitive material. The hackers called themselves the "Guardians of Peace" or "GOP" and demanded the cancellation of the planned film The Interview, a comedy about a plot to assassinate North Korean leader Kim Jong-un.[1][2][3] United States government cybersecurity agencies were evaluating the software, techniques, and network sources used in the hack, allege that the hackers were from North Korea.[4] North Korea has denied all responsibility, and some cybersecurity experts have proposed that current or former Sony Pictures employees may have been involved in the hacking.

Contents

1 Hack and perpetrators
2 Information obtained
3 Threats surrounding The Interview
4 U.S. accusations towards North Korea
   4.1 Doubts towards North Korea
5 Reactions
6 See also

The Interview poster with images of SETH ROGEN and JAMES FRANCO
Hackers crack more than 60% of breached LinkedIn passwords

Speed of hackers to crack passwords shows weakness of security scheme used by LinkedIn, researchers say
eBay is asking users to pick new passwords following a data breach earlier this year that exposed the personal information of an untold number of the auction giant’s 145 million customers.

In a blog post published this morning, eBay said it had “no evidence of the compromise resulting in unauthorized activity for eBay users, and no evidence of any unauthorized access to financial or credit card information, which is stored separately in encrypted formats. However, changing passwords is a best practice and will help enhance security for eBay users.”
JPMorgan Employee Password Was Key in Hack Hitting 76 Million Homes

By Hugh Son | Oct 2, 2014 7:39 PM ET | 665 Comments  Email  Print

**JPMorgan Chase & Co.**, the biggest U.S. bank, said a previously disclosed data breach affected 76 million households and 7 million small businesses.

Customer names, addresses, phone numbers and e-mail addresses were taken, the New York-based bank said today in a regulatory filing. Hackers also obtained internal data identifying customers by category, such as whether they are clients of the private-bank, mortgage, auto or credit-card divisions, said a person briefed on the matter.

Oct. 3 (Bloomberg) -- Bloomberg's Hans Nichols examines the type of information exposed in the latest data breach at JPMorgan, which affected 76 million households. He speaks on “The Pulse.”
A vulnerability in a web-based graphics system led to a breach of The Wall Street Journal’s network by a hacker, the newspaper acknowledged late Tuesday.

The system was taken offline, and the intrusion did not affect customers or customers’ data, according to a story published by the paper.

A hacker going by the handle “w0rm” posted a screenshot on Twitter on Tuesday showing a database from the newspaper. W0rm offered to sell the data for 1 bitcoin, or about US$620.
Profile summary

CyberCaliphate

Profile summary

CyberCaliphate

I love you isis

U.S. Central Command
@CENTCOM


MacDill AFB, Tampa, FL · centcom.mil

Followed by Lars Schwetje, Karie Fugett, BG Kathleen Cook and 100+ others.

U.S. Central Command @CENTCOM · 5m
pic.twitter.com/SdaoKO6Zkr
Details

U.S. Central Command @CENTCOM · 9m
AMERICAN SOLDIERS,
WE ARE COMING, WATCH YOUR BACK. ISIS.
Security is a big deal!
TODAY

• What is security?
• How will we try to break your site?
• Authentication, authorization, sanitization
• Common attacks
What today is not

• OS and browser security / lots of low level code
• Cryptography / lots of math
• Teaching you how to hack other countries
• MIT courses 6.857, 6.858, 6.875
You might not become l33t h4x0rz, but your site won’t do this.
security

the degree or resistance to, or protection from, harm; it applies to any vulnerable and valuable asset, such as a person, community, dwelling, ...

(from Wikipedia)
What is security in the context of computer systems?
What is security in the context of computer systems?

Protecting yourself
Protecting your users
What is security in the context of computer systems?

*Proper functionality, in the presence of adversaries*
What is security in the context of computer systems?

*Proper functionality, in the presence of adversaries*

1. What is “proper functionality”?
2. What can our adversaries do?
Security Policy

• What is the “proper functionality”? 
• “Only the author may edit her own blog posts” 
• “Only followers can see my tweets”
• “No one but me can see my location history”
Threat Model

• What can our adversaries do?
• Gain access to the database?
• Spam or DDoS?
• Inject their own (malicious) code?
• Try a bunch of other URLs?
Your turn!

Turn to someone next to you. Say hi.
Your turn!

Turn to someone next to you. Say hi.

Pick an app:

Facebook
Reddit
Bank of America
Github

Come up with a security policy and threat model.
volunteers?
You might not have total control.

Depends on libraries, protocols, compilers, ...
Security is just as much about the users as it is about your system!

Today, we focus on the system.
Ways security can break

- Too much text
- Too little text
- Text when you expect a number
- Number when you expect text
- Any number
- A negative number
- Zero
- Ridiculously large numbers
- Purposefully malicious text
- Unintentionally malicious text
- Text with markup
- Empty strings
- Strings with foreign characters
- An unexpected upload
- An unexpected image
- Lazy users
- Careless users
- Users with bad memory
- Users who leave their computers open
- Users with a single password
- Users who click on random links
- Users without Adblock
- Bad libraries, used well
- Bad libraries, used poorly
- Good libraries, used poorly
- Passwords aren’t secure
- Passwords are secure, but database isn’t
- Bad crypto
- Unsecured Starbucks wifi
- The NSA
- Proximity to Russia or North Korea
- Charles from 6.148 being delirious at 2AM after grading 60 sites
- You not attending this lecture
Three Main Points

• Authentication
• Authorization
• Sanitization
authentication

Are you who you claim to be?
authentication

Are you who you claim to be?

What are some authentication schemes?
authentication

Are you who you claim to be?

What are some authentication schemes?

some secret only you know
something about you that can’t be copied
Authentication

- Username/password
- Email/phone confirmation
- Password recovery questions
- PIN

- Two-factor auth
- Certificates
- Google/FB
- Biometrics
- DNA testing
- ...probably more...
Authentication Threat Model

- Attacker can “brute force” passwords
- Attacker can change client-side code/data
- Attacker may be able to dump database
- Attacker cannot break cryptography (encryption, signatures, certificates, HTTPS)
- Attacker might physically get ahold of user’s computer, but we can’t tell the difference
What does this mean?

• With unlimited time and "computing power", the attacker will probably win

• How do we distinguish between the real person and an attacker?
Password Handling

- Attacker might get ahold of database
- If we store all passwords as plaintext, everyone will be very sad if the database is breached
Forms with Passwords

• Use `<input type="password">`

• Use POST not GET

• Use HTTPS
HTTPS

• Lots of interesting, fun, complicated crypto

• Guarantees encryption: no one else can see the data

• Guarantees authentication: the server is who it says it is

• USE IT!
Logged in?

- Everything stored and check server-side!
- Login failure limit
- Sign out user after a certain amount of time
Storing Passwords

• Database may get dumped!

• Hash function: \( \text{hash(pw)} \Rightarrow \text{random long string} \)

• Fast to compute, hard to reverse

• So if an attacker gets to your database, it’s (almost) impossible to get the actual passwords.
Storing Passwords

- **Create user:**
  - get username and password
  - store username and hash(password)

- **Login:**
  - get username and password
  - compute hash(password)
  - compare against database
Good enough?

<table>
<thead>
<tr>
<th>Rank</th>
<th>Password</th>
<th>Change from 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123456</td>
<td>Up 1</td>
</tr>
<tr>
<td>2</td>
<td>password</td>
<td>Down 1</td>
</tr>
<tr>
<td>3</td>
<td>12345678</td>
<td>Unchanged</td>
</tr>
<tr>
<td>4</td>
<td>qwerty</td>
<td>Up 1</td>
</tr>
<tr>
<td>5</td>
<td>abc123</td>
<td>Down 1</td>
</tr>
<tr>
<td>6</td>
<td>123456789</td>
<td>New</td>
</tr>
<tr>
<td>7</td>
<td>111111</td>
<td>Up 2</td>
</tr>
<tr>
<td>8</td>
<td>1234567</td>
<td>Up 5</td>
</tr>
<tr>
<td>9</td>
<td>iloveyou</td>
<td>Up 2</td>
</tr>
<tr>
<td>10</td>
<td>adobe123</td>
<td>New</td>
</tr>
</tbody>
</table>
Even Better Security

- Hash + salt: randomize each username/password combo -- same passwords look different!
- In Rails: Devise plugin, has_secure_password
- In Node: use bcrypt plugin; Passport.js
authorization

Are you allowed to do this?

a.k.a. access control
You need permission

Want in? Ask the owner for access, or switch to an account with permission. Learn more

You are signed in as liucharles04@gmail.com.

Request access  Switch accounts
Authorization Threat Model

- Attacker can try to poke around the website (e.g. try URLs)
- Attacker can try to construct requests even if the UI doesn’t allow it
Key Points in Authorization

• Check your URLs!
• Just because a user is logged in, doesn’t mean they should be able to do everything
• Classes of users: any user, friends list, admin, etc.
Authorization Server-side

- Authorization / access control checks need to be done on the server!
- Check EVERY request.
- Distinguish GET (viewing something) from POST (creating/editing something)
But these URLs are so complicated, no one will ever stumble upon them!

*Is that good enough?*
Node Authorization

- In the router:
  
  ```javascript
  router.all("*", requireLogin)
  router.post("*", requireOwner)
  ```
Rails Authorization

• In the controllers
  before_action :require_login
  before_action :require_creator, only:
  [:new, :create]
functions in **RED** must be defined by you!

don’t worry -- code examples to come
sanitization

Making untrusted input safe to store and show
What’s untrusted?

• Big forms?
• Images?
• Ask the user for code?
EVERYTHING
When you take any user input, **NEVER** assume it is safe
Javascript Injection

• User inputs some text
• Server returns the text on a page without checking
• What if the user’s input is:
  <script>alert(“hello”); really_evil_code()</script>

• DEMO
Preventing JS Injection

- Most frameworks do it for you!
- EJS: `<%= %>` vs. `<%- %>`
- Handlebars: `{{ }}` vs. `{{{ }}}`
- Rails: raw `@var` vs. `@var`
- Note: It’s harder to make it insecure!
HI, THIS IS YOUR SON’S SCHOOL. WE’RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR – DID HE BREAK SOMETHING? IN A WAY –

DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students; --?

OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE’VE LOST THIS YEAR’S STUDENT RECORDS. I HOPE YOU’RE HAPPY.

AND I HOPE YOU’VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.
SQL Injection

• SQL database queries are strings:
  e.g. “SELECT * FROM users WHERE ...”

• Often user input goes into these strings

• DEMO:
What happened?

$user = $_POST["user"];

$pass = $_POST["pass"];

$query = "SELECT * FROM users WHERE username='$user' AND password='$pass'"
What happened?

$user = "charlesliu";

$pass = "" OR '1=1"

$query = "SELECT * FROM users WHERE username='charlesliu' AND password=" OR '1=1""
Solution?

• Escape user strings: don’t let user type in risky characters like quotes

• Rails does all this automatically!
But I’m using MongoDB!

I’m safe right?
$('#inject').click(function() {
    var loginInfo = {
        "username": {"$gt" : ""},
        "password": {"$gt" : ""}
    };
    $.post("/sessions", loginInfo);
});
users.findOne({
    "username": req.body.username,
    "password": req.body.password
});
MongoDB Injection

users.findOne({
    "username": {"$gt" : ""},
    "password": {"$gt" : ""}
});

Find user where username and password are greater than ""
Solution?

• Objects are risky in MongoDB queries
• Forcibly cast to strings, or check if object and throw an error
We’re almost done

- That was a lot
- Don’t worry: for 6.148 we’re not going to spend ridiculous efforts trying to break things
- But that doesn’t mean we won’t try the very basic attacks (e.g. `<script>alert("hello")</script>`

Tuesday, January 13, 15
A Balancing Act

- Balancing security with *usability*
- Very secure: no features, no inputs, no login
- Security measures: expiring logins, 2-factor authentication make it less convenient
- Convenience measures: "remember me", more features have the potential to reduce security
That’s all for now

• Start coding!
• Code, guides, sample apps will go up in the next few days
• Hackathon on Thursday evening! Sign up! Lots of help.
• Sponsor lectures, GOOD FOOD :)

Tuesday, January 13, 15
GOOD LUCK