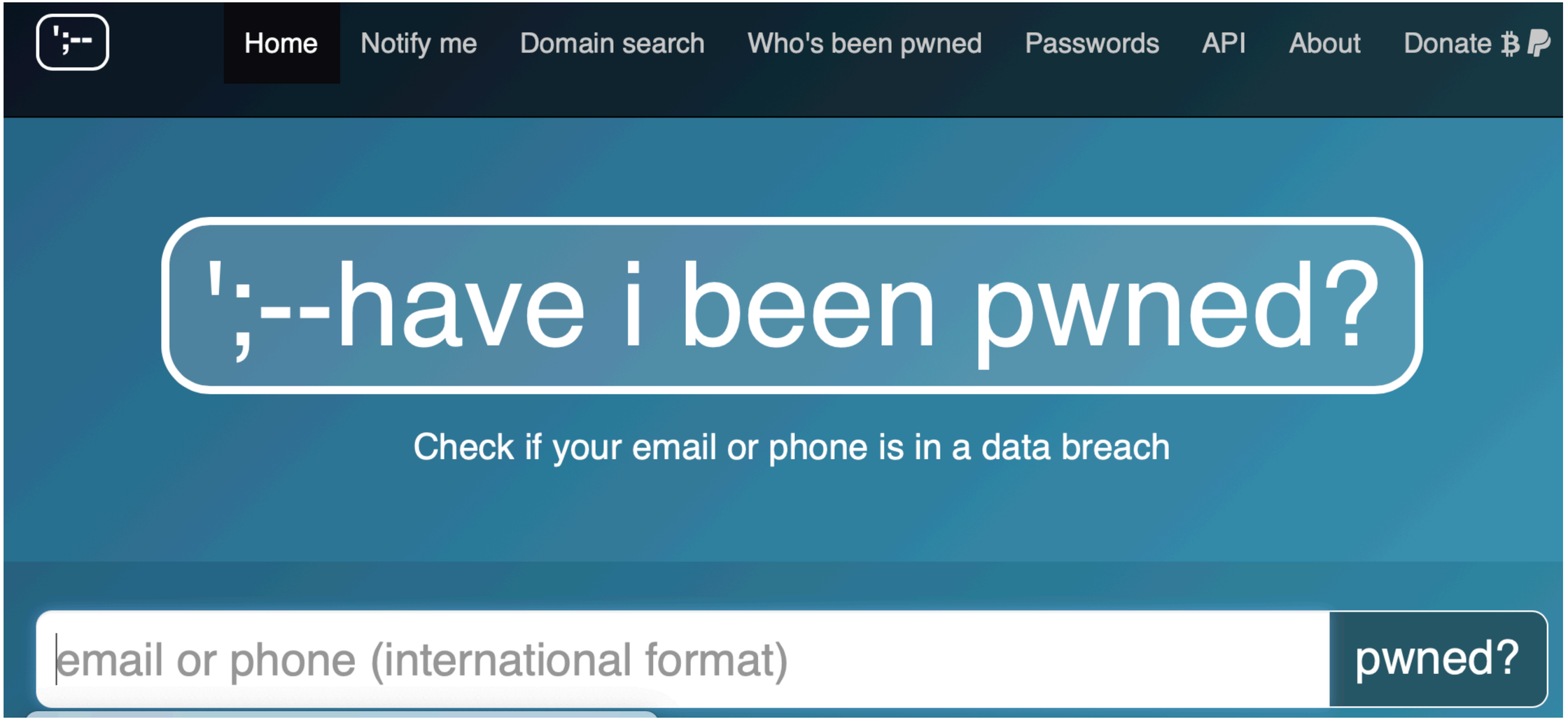
SQL Injection: A History' OR 1=1; --

BlueTeamCon

By: Will McCardell

'; UPDATE outline SET value = '

- Brief History of SQL Injection
- Defensive Guidance over the Ages
- Pentesters aren't seeing SQL Injection as often
- '; --


';--
Home
Notify me
Domain search
Who's been pwned
Passwords
API
About
Donate
';--have i been pwned?
Check if your email or phone is in a data breach
email or phone (international format)
pwned?

Probably

What is SQL Injection?

Impact

- Bypass authentication
- Extract all database data
- Wipe or modify data
- Reconnaissance
- Lateral movement behind defenses

TSA

- Administrative access to the Known Crew Member portal
- Free travel
- Pretend to be a pilot

ian carroll / Bypassing airport security via S...

Bypassing airport security via SQL injection

08/29/2024

Introduction

Like many, Sam Curry and I spend a lot of time waiting in airport

First Publication

Phrack

- Phrack, Volume 8, Issue 54
  - December 25th, 1998
- "NT Web Technology Vulnerabilities"

Microsoft's SQL Injection Definition

rain.forest.puppy's article

"According to them, what you're about to read is not a problem, so don't worry about doing anything to stop it."

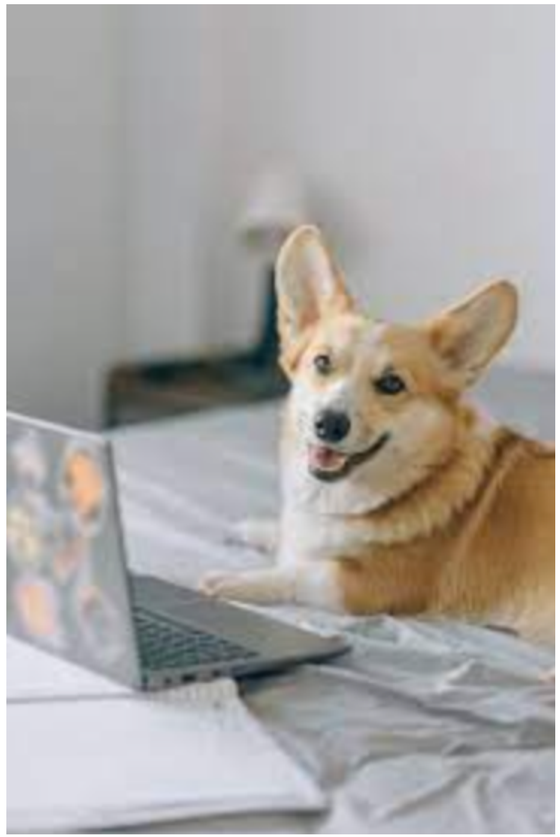
Phrack Article

First documented SQL Injection

```
- WHAT'S THAT REALLY MEAN?  People can possibly piggyback SQL commands into
your statements.  Let's say you have:

        SELECT * FROM table WHERE x=%%criteria from webpage user%%

Now, what if %%criteria from webpage user%% was equal to:

        SELECT * FROM sysobjects

It would translate to:

        SELECT * FROM table WHERE x=1 SELECT * FROM sysobjects

which would be valid SQL and execute (both commands).  But wait, there's more.
```

A Puppy's Guide to SQL Injection Defense

- Escape the user controlled input
- Use custom stored procedures



A Puppy's Guide to SQL Injection Defense

Dynamic Queries with Escaping User Input

A Puppy's Guide to SQL Injection Defense

Custom Stored Procedures

```
// This should REALLY be validated
String custname = request.getParameter("customerName");
try {
  CallableStatement cs = connection.prepareCall("{call sp_getAccountBalance(?)}");
  cs.setString(1, custname);
  ResultSet results = cs.executeQuery();
  // … result set handling
} catch (SQLException se) {
  // … logging and error handling
}
```


OWASP
®

OWASP Guidance, 2005

"A Guide to Building Secure Web Applications and Web Services"

- Parameterized stored procedures
- Prepared statements
- ORM (eg Hibernate)
- Dynamic queries

OWASP Guidance, 2005

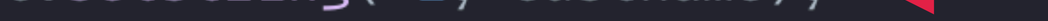
Parameterized Stored Procedures

```
// This should REALLY be validated
String custname = request.getParameter("customerName");
try {
  CallableStatement cs = connection.prepareCall("{call sp_getAccountBalance(?)}");
  cs.setString(1, custname);
  ResultSet results = cs.executeQuery();
  // … result set handling
} catch (SQLException se) {
  // … logging and error handling
}
```

OWASP Guidance, 2005

Prepared Statements

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

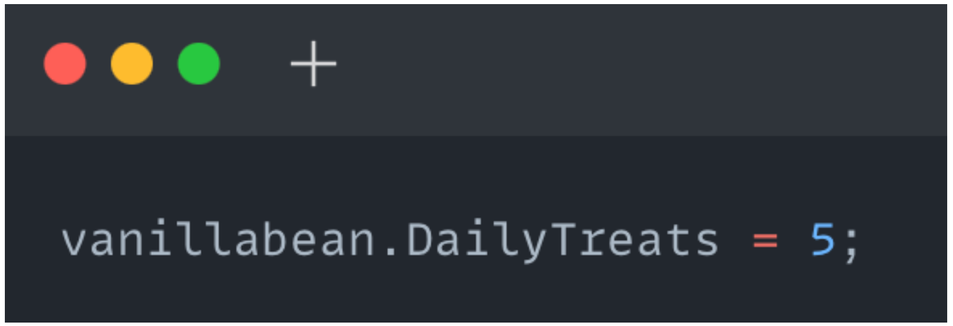


OWASP Guide, 2005

Object-Relational Mappings (ORM)

Automagically bind the application's object data to the database's representation of the same data

Using SQL

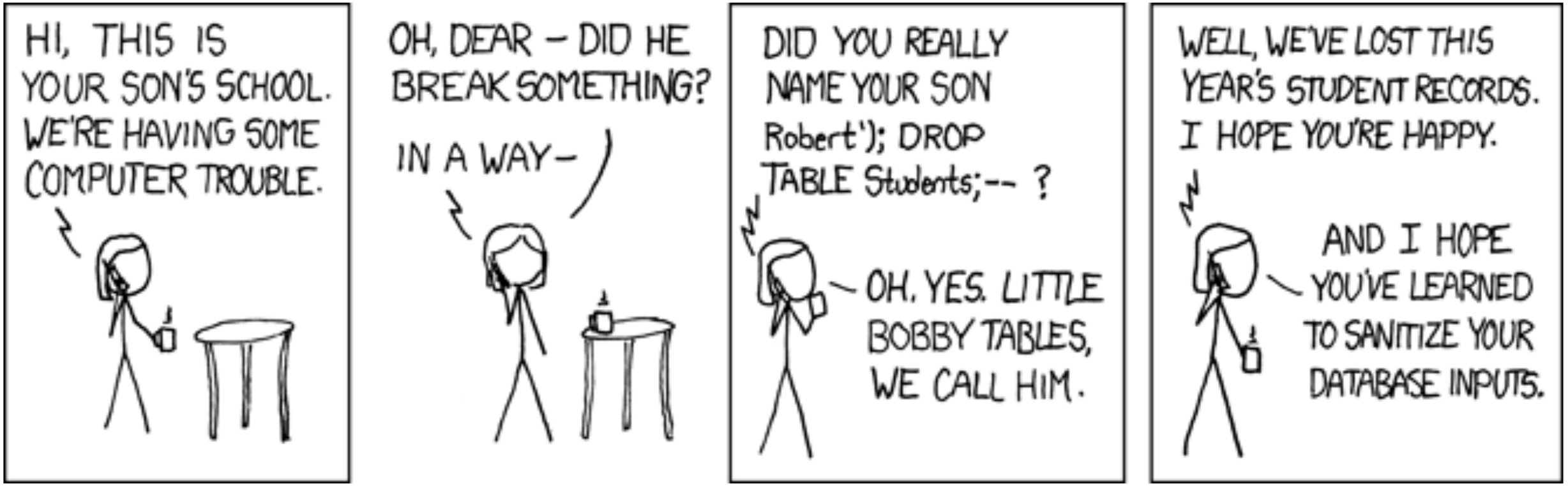


Using ORM

OWASP Guidance, 2005

Dynamic SQL Queries with Escaping User Input

```
string input = request.getParameter("CatDeservingTreat");
input = "\"" + input + "\"";
input = input.replace("'","''");
sqlCmd.ExecuteReader("SELECT DailyTreats FROM CatTreats WHERE CatName = '" + input +"';");
```

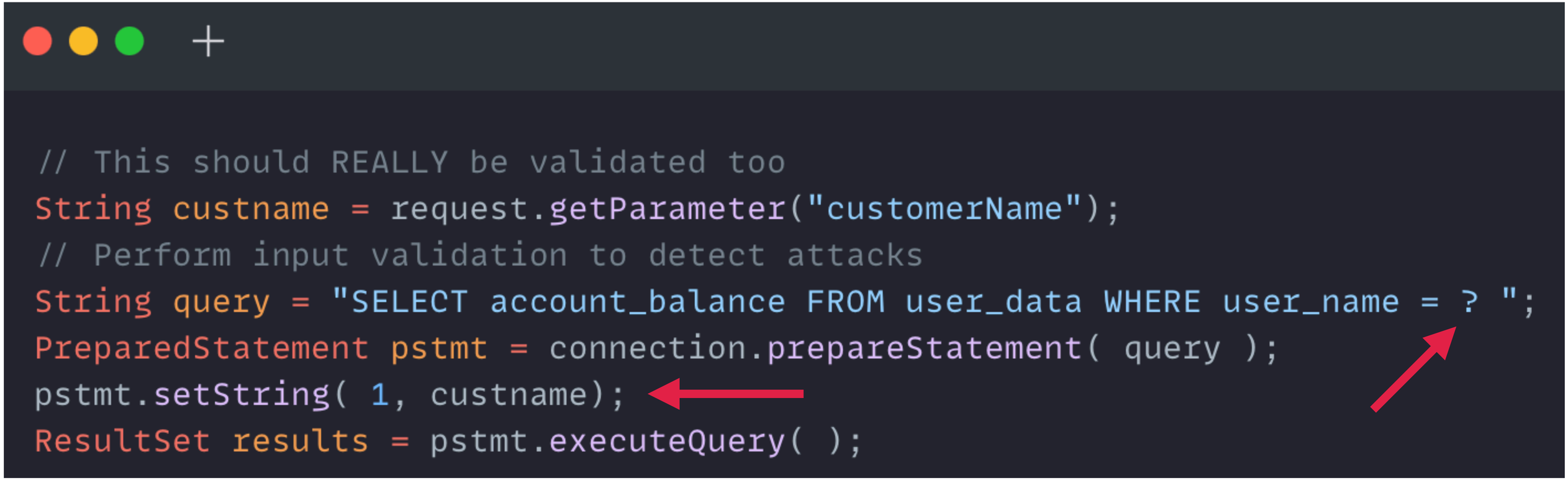

HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR - DID HE BREAK SOMETHING?
IN A WAY-
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

OWASP Guidance, 2023

Primary Defenses

- Option 1: Use of prepared statements (with parameterized queries)
- Option 2: Use of properly constructed stored procedures
- Option 3: Allow-list input validation
- Option 4: Escaping all user supplied input

Option 1: Prepared Statements



```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

Option 2: Properly Constructed Stored Procedures

```
// This should REALLY be validated
String custname = request.getParameter("customerName");
try {
  CallableStatement cs = connection.prepareCall("{call sp_getAccountBalance(?)}");
  cs.setString(1, custname);
  ResultSet results = cs.executeQuery();
  // … result set handling
} catch (SQLException se) {
  // … logging and error handling
}
```

Option 3: Allow-List Input Validation

```
String tableName;
switch(PARAM):
  case "Value1": tableName = "fooTable";
                 break;
  case "Value2": tableName = "barTable";
                 break;

  ...
  default      : throw new InputValidationException("unexpected value provided"
                                                    + " for table name");
```

Option 4: Dynamic Queries with Escaping Input

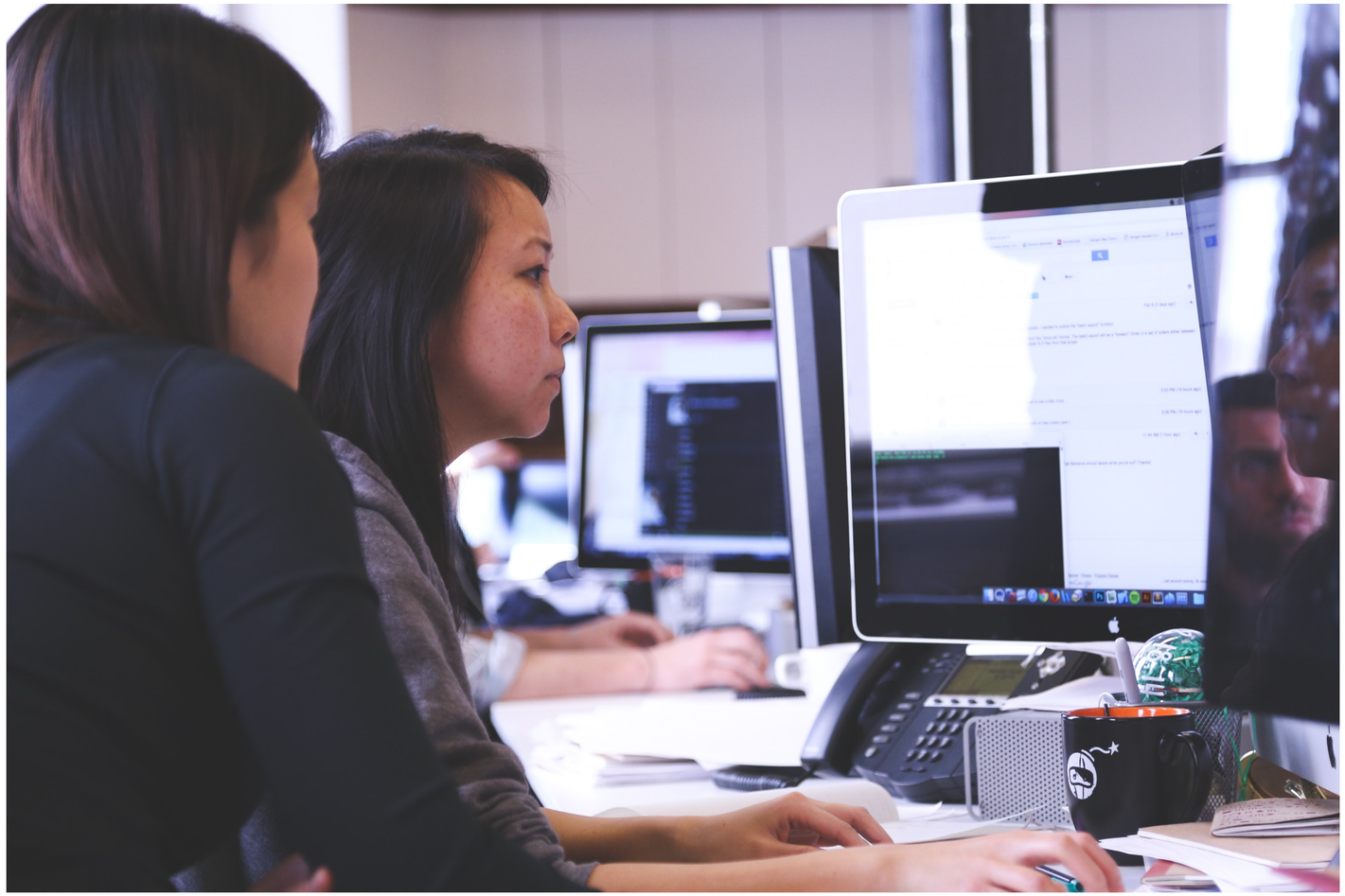
"This technique should only be used as a last resort, when none of the above are feasible. Input validation is probably a better choice as this methodology is frail compared to other defenses and *we cannot guarantee it will prevent all SQL Injections in all situations*."

```
string input = request.getParameter("CatDeservingTreat");
input = "\"" + input + "\"";
input = input.replace("'","''");
sqlCmd.ExecuteReader("SELECT DailyTreats FROM CatTreats WHERE CatName = '" + input +"';");
```

Where Did SQL Injections Go?

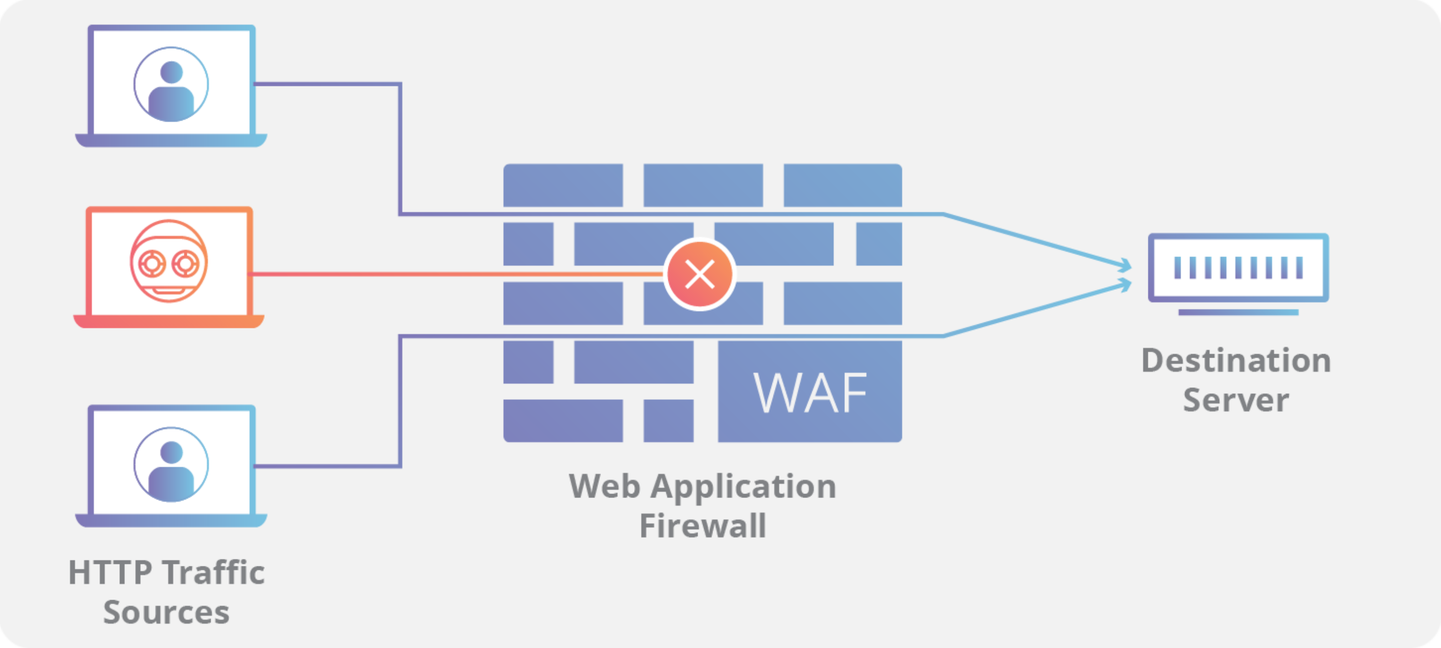
Developer Knowledge

- OWASP Guidance
- Developers sharing strategies
- XKCD comic



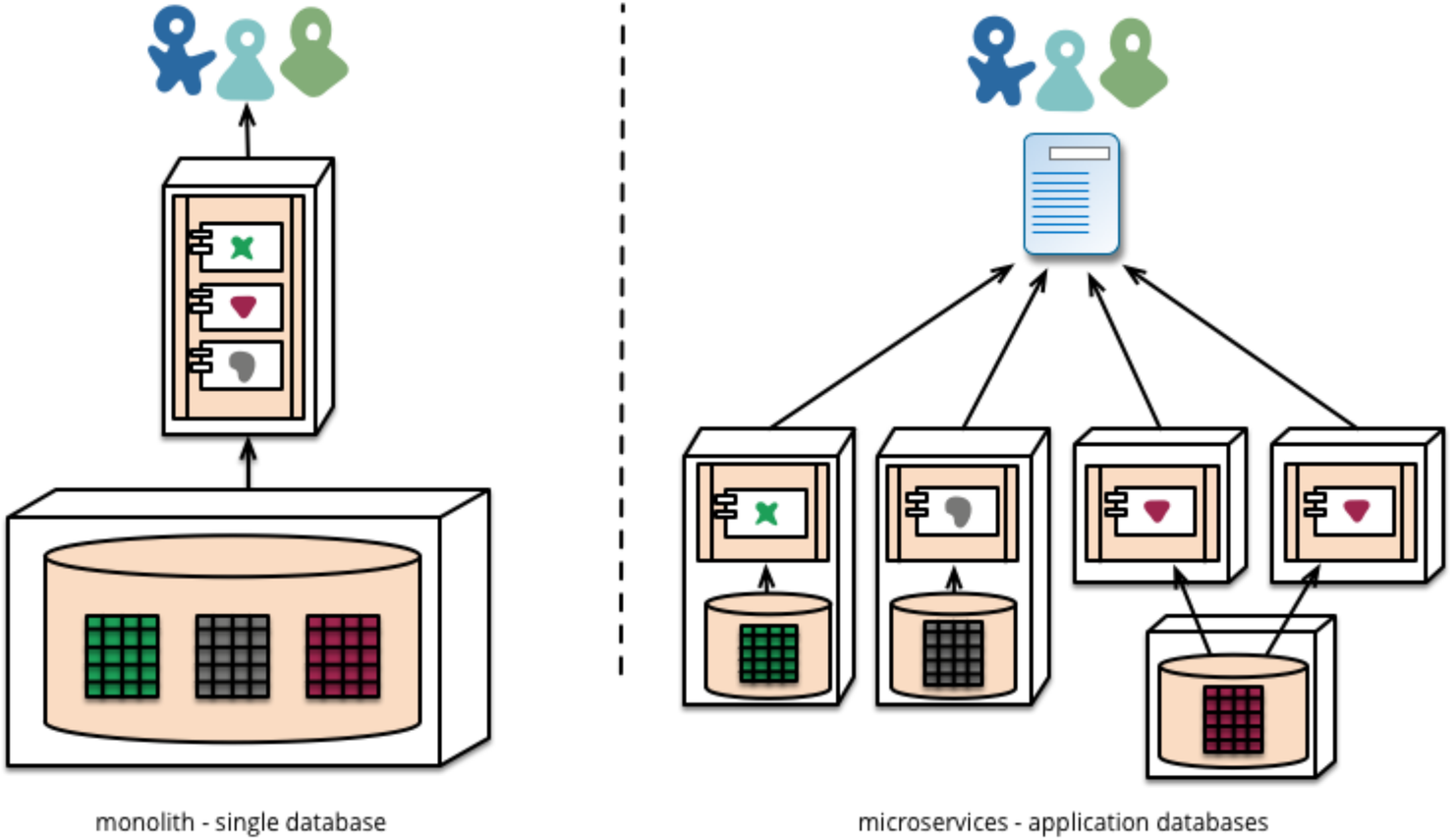
Web Application Firewalls

- Process requests from users and reject requests that look like attacks
- Mentioned in first PCI-DSS in 2004
- Bolt on security
- Growth exploded in the Cloud



Microservices

- New architectural pattern in 2000s and 2010s
- Enforced architecturally the principle of least privilege
- Limited blast radius of SQL Injection



monolith - single database

microservices - application databases

Martin Fowler - "Microservices"

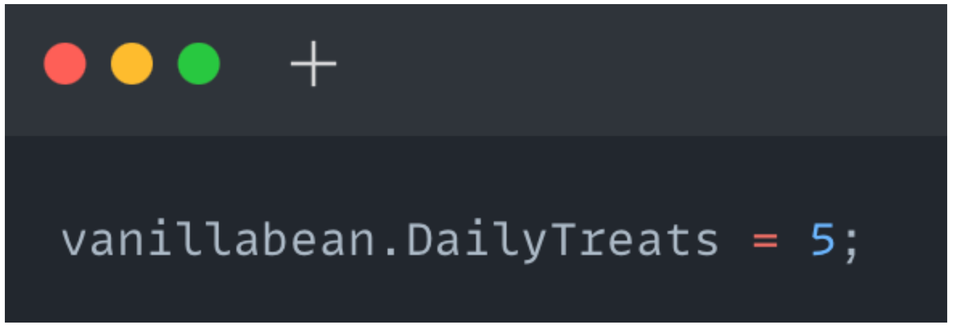
NoSQL

- New type of databases that were non-relational
- Gained popularity in the 2010s
- NoSQL Injections possible

Object-Relational Mappers (ORM)

- Extremely easy for developers to use
- Ruby on Rails was the catalyst for other languages

Using SQL



Using ORM

The Disappearance of SQL Injections

- Developer Knowledge
- Web Application Firewalls (WAF)
- Microservices
- NoSQL
- Object-Relational Mapping (ORM)

Paved Paths and Golden Roads

- Almost accidentally solved
- People take the path of least resistance
- Ensure it is easier to be secure than not

...did SQL Injections disappear?

3rd Place Most Dangerous Weakness

- MITRE's 2023 CWE Top 25 Most Dangerous Software Weaknesses
- SQL Injections #3
- Up from #6 in 2022

Why are SQL Injections #3 if Pentesters aren't seeing them?

'; DROP TABLE Presentation; --