OVERVIEW OF CYBERSECURITY

DR. SAMUEL SAMBASIVAM
LEAD DOCTORAL PROFESSOR
DOCTOR OF COMPUTER SCIENCE (DCS) PROGRAM
“It is the mark of an educated mind to be able to entertain a thought without accepting it.”

— Aristotle, Metaphysics
“I’m applying for the Information Security position. Here is a copy of my resumé, encoded, encrypted and shredded.”
INCREASING TRENDS CONTINUED FROM 2017

DDOS attacks to increase

One concern is use of IoT to pre-stage attacks

Consolidation of security companies

Increase in Manages Security Service (MSS)

Insider Threat

More concerns as employees become less loyal to companies

Talent Brain-Drain

Focus on Mobile Security
TRENDS TO WATCH FOR IN 2018

Increase in Cloud Migration

Companies Putting Trust in Cloud – Some say without due diligence

Ransomware Attacking the Cloud

Undefined Enterprise Network (Lines blurred by the cloud)

Third Party Risk

Big Data Solution (Ferrets out threats, finds vulnerabilities)

Machine Learning and Artificial Intelligence

Data Theft moving to Data Manipulation

Increase in Cybersecurity Regulation
TRENDS TO WATCH FOR IN 2018

Increase in Cloud Migration

Companies Putting Trust in Cloud – Some say without due diligence

Ransomware Attacking the Cloud

Undefined Enterprise Network (Lines blurred by the cloud)

Third Party Risk

Big Data Solution (Ferrets out threats, finds vulnerabilities)

Machine Learning and Artificial Intelligence

Data Theft moving to Data Manipulation

Increase in Cybersecurity Regulation
TRENDS TO WATCH FOR IN 2018

Hacking Becomes More Organized, Smarter:

- Evolution of the Blockchain in Tracking Bitcoin Transactions
- SSL Abuse/Exploitation of Protocols
- DNSMessenger Attacks – Queries Trigger PowerShell Commands without Direct Infection by Malware
- Machine learning as a means of Triggering Focused Social Engineering
- Growth of Online Purchasing Driving Theft
- More Wide Spread Outages of Internet (Are orgs ready – Business Continuity)

(Cyber Pearl Harbor)
TRENDS TO WATCH FOR IN 2018

US Govt Cybersecurity Increasing:
$14B in 2016
$19B in 2017
$ ?? in 2018
ADDITIONAL CONCERNS FOR 2018

Drones - UAV Activity (easily attack critical infrastructure, target leaders, etc.)
  (Drone-jacking)
Artificial Intelligence – Development of AI manipulated attacks across the Internet
  As an attack vector
  As a solution to cybersecurity solutions
Social Media websites
  Robots
Autonomous Cars (Concern for coordinated attacks)
Smart City Infrastructure
Internet of Things (moving to the enterprise, too)
  Home Security Vulnerabilities
COMPUTER SECURITY
OVERVIEW

- System/Software Security
  - Set UID Programs
  - Buffer Overflow Attack
- Cyber/Web Security
  - SQL Injection Attack
  - XSS Attack
- Network Security
  - Packet Sniffing and Spoofing
  - Attacks on the TCP Protocol
SQL INJECTION ATTACK
SQL INJECTION ATTACK
Brief Tutorial of SQL

• **Log in to MySQL:** We will use MySQL database, which is an open-source relational database management system. We can log in using the following command:

```bash
$ mysql -u root -p seedubuntu
Welcome to the MySQL monitor.
...
mysql>
```

• **Create a Database:** Inside MySQL, we can create multiple databases. “SHOW DATABASES” command can be used to list existing databases. We will create a new database called dbtest:

```sql
mysql> SHOW DATABASES;
.......;
mysql> CREATE DATABASE dbtest;
```
SQL Tutorial: Create a Table

• A relational database organizes its data using tables. Let us create a table called employee with seven attributes (i.e. columns) for the database “dbtest”

• We need to let the system know which database to use as there may be multiple databases

• After a table is created, we can use describe to display the structure of the table

```sql
mysql> USE dbtest
mysql> CREATE TABLE employee (  
  ID INT (6) NOT NULL AUTO_INCREMENT,  
  Name VARCHAR (30) NOT NULL,  
  EID VARCHAR (7) NOT NULL,  
  Password VARCHAR (60),  
  Salary INT (10),  
  SSN VARCHAR (11),  
  PRIMARY KEY (ID)  
);
mysql> DESCRIBE employee;

+-------+--------------+-----+----------------+-----------+-----------+----------+
| Field | Type         | Null| Key | Default | Extra     |
+-------+--------------+-----+-----+---------+-----------+----------+
| ID    | int(6)       | NO  | PRI | NULL    | auto_increment |
| Name  | varchar(30)  | NO  |    | NULL    |           |          |
| EID   | varchar(30)  | NO  |    | NULL    |           |          |
| Password | varchar(60) | YES |   | NULL    |           |          |
| Salary | int(10)      | YES |   | NULL    |           |          |
| SSN   | varchar(11)  | YES |   | NULL    |           |          |
+-------+--------------+-----+-----+---------+-----------+----------+
SQL Tutorial: Insert a Row

• We can use the INSERT INTO statement to insert a new record into a table:

```sql
mysql> INSERT INTO employee (Name, EID, Password, Salary, SSN) VALUES ('Ryan Smith', 'EID5000', 'paswd123', 80000, '555-55-5555');
```

• Here, we insert a record into the “employee” table.

• We do not specify a value of the ID column, as it will be automatically set by the database.
SQL Tutorial: SELECT Statement

- The SELECT statement is the most common operation on databases
- It retrieves information from a database

Asks the database for all its records, including all the columns

Asks the database only for Name, EID and Salary columns
SQL Tutorial: WHERE Clause

• It is uncommon for a SQL query to retrieve all records in a database.

• WHERE clause is used to set conditions for several types of SQL statements including SELECT, UPDATE, DELETE etc.

• The above SQL statement only reflects the rows for which the predicate in the WHERE clause is TRUE.

• The predicate is a logical expression; multiple predicates can be combined using keywords AND and OR.

• Lets look at an example in the next slide.
SQL Tutorial: WHERE Clause

• The first query returns a record that has EID5001 in EID field
• The second query returns the records that satisfy either EID=‘EID5001’ or Name=‘David’
SQL Tutorial: WHERE Clause

• If the condition is always True, then all the rows are affected by the SQL statement

```
mysql> SELECT * FROM employee WHERE 1=1;
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>EID</th>
<th>Password</th>
<th>Salary</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>EID5000</td>
<td>passwd123</td>
<td>80000</td>
<td>555-55-5555</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>EID5001</td>
<td>passwd123</td>
<td>80000</td>
<td>555-66-5555</td>
</tr>
<tr>
<td>3</td>
<td>Charlie</td>
<td>EID5002</td>
<td>passwd123</td>
<td>80000</td>
<td>555-77-5555</td>
</tr>
<tr>
<td>4</td>
<td>David</td>
<td>EID5003</td>
<td>passwd123</td>
<td>80000</td>
<td>555-88-5555</td>
</tr>
</tbody>
</table>

• This 1=1 predicate looks quite useless in real queries, but it will become useful in SQL Injection attacks
SQL Tutorial: UPDATE Statement

- We can use the UPDATE Statement to modify an existing record

```
mysql> UPDATE employee SET Salary=82000 WHERE Name='Bob';
mysql> SELECT * FROM employee WHERE Name='Bob';
```

```
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>EID</th>
<th>Password</th>
<th>Salary</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Bob</td>
<td>EID5001</td>
<td>passwd123</td>
<td>82000</td>
<td>555-66-5555</td>
</tr>
</tbody>
</table>
```
SQL Tutorial: Comments

MySQL supports three comment styles

- Text from the # character to the end of line is treated as a comment
- Text from the “--” to the end of line is treated as a comment.
- Similar to C language, text between /* and */ is treated as a comment

```sql
mysql> SELECT * FROM employee;  # Comment to the end of line
mysql> SELECT * FROM employee;  -- Comment to the end of line
mysql> SELECT * FROM /* In-line comment */ employee;
```
A typical web application consists of three major components:

- SQL Injection attacks can cause damage to the database. As we notice in the figure, the users do not directly interact with the database but through a web server. If this channel is not implemented properly, malicious users can attack the database.
Getting Data from User

• This example shows a form where users can type their data. Once the submit button is clicked, an HTTP request will be sent out with the data attached.

![Form Example]

• The HTML source of the above form is given below:

```html
<form action="getdata.php" method="get">
  EID:  <input type="text" name="EID"><br>
  Password:  <input type="text" name="Password"><br>
  <input type="submit" value="Submit">
</form>
```

• Request generated is:

```
http://www.example.com/getdata.php?EID=EID5000&Password=passwd123
```
Getting Data from User

• The request shown is an HTTP GET request, because the method field in the HTML code specified the get type

• In GET requests, parameters are attached after the question mark in the URL

• Each parameter has a name=value pair and are separated by “&”

• In the case of HTTPS, the format would be similar but the data will be encrypted

• Once this request reached the target PHP script the parameters inside the HTTP request will be saved to an array $_GET or $_POST. The following example shows a PHP script getting data from a GET request

```php
<?php
    $eid = $_GET['EID'];
    $pwd = $_GET['Password'];
    echo "EID: $eid --- Password: $pwd\n";
?>
```
Connecting to MySQL Database

- PHP program connects to the database server before conducting query on database using.
- The code shown below uses new mysqli(...) along with its 4 arguments to create the database connection.

```php
function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="dbtest";

    // Create a DB connection
    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error . "\n");
    }
    return $conn;
}
```
How Web Applications Interact with Database

• Construct the query string and then send it to the database for execution.
• The channel between user and database creates a new attack surface for the database.
Launching SQL Injection Attacks

• Everything provided by user will become part of the SQL statement. Is it possible for a user to change the meaning of the SQL statement?

• The intention of the web app developer by the following is for the user to provide some data for the blank areas.

```sql
SELECT Name, Salary, SSN
FROM employee
WHERE eid=' ' and password=' ';
```

• Assume that a user inputs a random string in the password entry and types “EID5002’#” in the eid entry. The SQL statement will become the following

```sql
SELECT Name, Salary, SSN
FROM employee
WHERE eid='EID5002' '#' and password='xyz';
```
Launching SQL Injection Attacks

- Everything from the # sign to the end of line is considered as comment. The SQL statement will be equivalent to the following:

```sql
SELECT Name, Salary, SSN
FROM employee
WHERE eid='EID5002'
```

- The above statement will return the name, salary and SSN of the employee whose EID is EID5002 even though the user doesn’t know the employee’s password. This is security breach.

- Let’s see if a user can get all the records from the database assuming that we don’t know all the EID’s in the database.

- We need to create a predicate for WHERE clause so that it is true for all records.

```sql
SELECT Name, Salary, SSN
FROM employee
WHERE eid='a' OR 1=1
```
Launching SQL Injection Attacks using cURL

• More convenient to use a command-line tool to launch attacks.
• Easier to automate attacks without a graphic user interface.
• Using cURL, we can send out a form from a command-line, instead of from a web page.

```
curl 'www.example.com/getdata.php?EID=a’ OR 1=1 &Password=
```

• The above command will not work. In an HTTP request, special characters are in the attached data needs to be encoded or they maybe mis-interpreted.
• In the above URL we need to encode the apostrophe, whitespace and the # sign and the resulting cURL command is as shown below:

```
curl 'www.example.com/getdata.php?EID=a%27%20
       OR%201=1%20%23&Password=
```

Name: Alice -- Salary: 80000 -- SSN: 555-55-5555<br/>
Name: Bob -- Salary: 82000 -- SSN: 555-66-5555<br/>
Name: Charlie -- Salary: 80000 -- SSN: 555-77-5555<br/>
Name: David -- Salary: 80000 -- SSN: 555-88-5555<br/>
Modify Database

• If the statement is UPDATE or INSERT INTO, we will have chance to change the database.

• Consider the form created for changing passwords. It asks users to fill in three pieces of information, EID, old password and new password.

• When Submit button is clicked, an HTTP POST request will be sent to the server-side script `changepassword.php`, which uses an UPDATE statement to change the user’s password.
Modify Database

• Let us assume that Alice (EID5000) is not satisfied with the salary she gets. She would like to increase her own salary using the SQL injection vulnerability. She would type her own EID and old password. The following will be typed into the “New Password” box:

```plaintext
New Password paswd456’, salary=100000 #
```

• By typing the above string in "New Password" box, we get the UPDATE statement to set one more attribute for us, the salary attribute. The SQL statement will now look as follows.

```sql
UPDATE employee
SET password='paswd456’, salary=100000 #'
WHERE eid=’EID5000’ and password='paswd123’";
```

• What if Alice doesn’t like Bob and would like to reduce Bob’s salary to 0, but she only knows Bob’s EID (eid5001), not his password. How can she execute the attack?

<table>
<thead>
<tr>
<th>EID</th>
<th>EID5001’ #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Password</td>
<td>anything</td>
</tr>
<tr>
<td>New Password</td>
<td>paswd456’, salary=0 #</td>
</tr>
</tbody>
</table>
Multiple SQL Statements

• Damages that can be caused are bounded because we cannot change everything in the existing SQL statement.

• It will be more dangerous if we can cause the database to execute an arbitrary SQL statement.

• To append a new SQL statement "DROP DATABASE dbtest" to the existing SQL statement to delete the entire dbtest database, we can type the following in the EID box:

   ```
   EID a'; DROP DATABASE dbtest; #
   ```

• The resulting SQL statement is equivalent to the following, where we have successfully appended a new SQL statement to the existing SQL statement string:

   ```
   SELECT Name, Salary, SSN
   FROM employee
   WHERE eid='a'; DROP DATABASE dbtest;
   ```

• The above attack doesn’t work against MySQL, because in PHP’s mysqli extension, the mysqli::query() API doesn’t allow multiple queries to run in the database server.
Multiple SQL Statements

• The code below tries to execute two SQL statements using the $mysqli->query() API

```php
/* testmulti_sql.php */
<?php
$mysqli = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$res = $mysqli->query("SELECT 1; DROP DATABASE dbtest");
if (!$res) {
    echo "Error executing query: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

• When we run the code, we get the following error message:

```
$ php testmulti_sql.php
Error executing query: (1064) You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DROP DATABASE dbtest' at line 1
```

• If we do want to run multiple SQL statements, we can use $mysqli->multi_query(). [not recommended]
The Fundamental Cause

Mixing data and code together is the cause of several types of vulnerabilities and attacks including SQL Injection attack, XSS attack, attacks on the system() function and format string attacks.
Countermeasures: Filtering and Encoding Data

• Before mixing user-provided data with code, inspect the data. Filter out any character that may be interpreted as code.
• Special characters are commonly used in SQL Injection attacks. To get rid of them, encode them.
• Encoding a special character tells parser to treat the encoded character as data and not as code. This can be seen in the following example

Before encoding:   aaa' OR 1=1 
After encoding:   aaa\' OR 1=1 

• PHP’s mysqli extension has a built-in method called mysqli::real_escape_string(). It can be used to encode the characters that have special meanings in SQL. The following code snippet shows how to use this API.

```php
/* getdata_encoding.php */

<?php
    $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
    $eid = $mysqli->real_escape_string($_GET["EID"]);  // 1
    $pwd = $mysqli->real_escape_string($_GET["Password"]);  // 2
    $sql = "SELECT Name, Salary, SSN
            FROM employee
            WHERE eid='\$eid' and password='\$pwd'";
?>
```
Countermeasures: Prepared Statement

• Fundament cause of SQL injection: mixing data and code
• Fundament solution: separate data and code.
• Main Idea: Sending code and data in separate channels to the database server. This way the database server knows not to retrieve any code from the data channel.
• How: using prepared statement
• Prepared Statement: It is an optimized feature that provides improved performance if the same or similar SQL statement needs to be executed repeatedly. Using prepared statements, we send an SQL statement template to the database, with certain values called parameters left unspecified. The database parses, compiles and performs query optimization on the SQL statement template and stores the result without executing it. We later bind data to the prepared statement
Countermesures: Prepared Statement

The vulnerable version: code and data are mixed together.

```php
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
       FROM employee
       WHERE eid = '$_eid' and password='$_pwd'";
$result = $conn->query($sql);
```

Using prepared statements, we separate code and data.

```php
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
       FROM employee
       WHERE eid = ? and password=?";

if ($stmt = $conn->prepare($sql)) {  
    $stmt->bind_param("ss", $eid, $pwd);  
    $stmt->execute();  
    $stmt->bind_result($name, $salary, $ssn);  
    while ($stmt->fetch()) {  
        printf ("%s %s %s\n", $name, $salary, $ssn);
    }
}
```

Send code

Send data

Start execution
Why Are Prepared Statements Secure?

• Trusted code is sent via a code channel.
• Untrusted user-provided data is sent via data channel.
• Database clearly knows the boundary between code and data.
• Data received from the data channel is not parsed.
• Attacker can hide code in data, but the code will never be treated as code, so it will never be attacked.
Summary

• Brief tutorial of SQL
• SQL Injection attack and how to launch this type of attacks
• The fundamental cause of the vulnerability?
• How to defend against SQL Injection attacks?
• Prepared Statement
THANK YOU FOR JOINING THIS SESSION!

Dr. Samuel Sambasivam
Lead Doctoral Professor - DCS
Phone: 626-945-9809
Email: SSambasivam@coloradotech.edu
QUESTIONS?