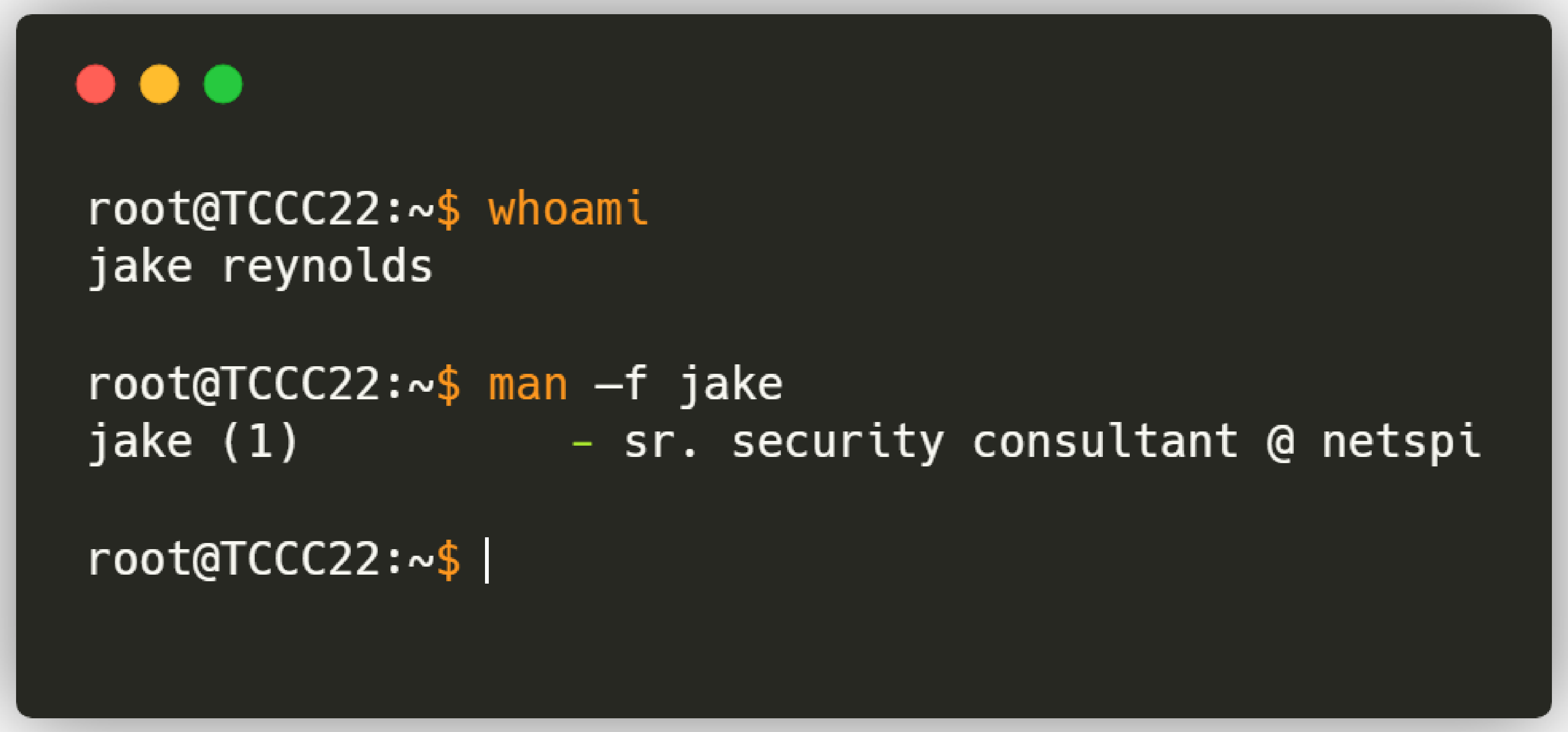
NETSPI

PAYING THE EQUITAX

WEB SECURITY IN 2018

OUTLINE

- What are we all doing here
- OWASP
- Hack the planet
  - Code examples in
    - Java
    - C#
    - NodeJS
    - Javascript
- ✌



```
root@TCCC22:~$ whoami
jake reynolds

root@TCCC22:~$ man –f jake
jake (1)          - sr. security consultant @ netspi

root@TCCC22:~$
```

WHAT ARE WE ALL DOING HERE

codeSecurity === jobSecurity

WHAT ARE WE ALL DOING HERE

- Web vulnerabilities
- OWASP Top 10
  - Open Web Application Security Project (OWASP)
  - 501(c)(3) worldwide not-for-profit charitable organization focused on improving the security of software.

#	OWASP Top 10 2017
1	(SQL) Injection
2	Broken Authentication
3	Sensitive Data Exposure
4	XML External Entities (XXE)
5	Broken Access Control
6	Security Misconfiguration
7	Cross-Site Scripting (XSS)
8	Insecure Deserialization
9	Components with known vulnerabilities
10	Insufficient logging and monitoring

WEB SECURITY IN ~~2018~~ 2004

2004	2007	2010	2013	2017
Unvalidated Input	XSS	Injection	Injection	Injection
Broken Access Control	Injection Flaws	XSS	Broken Authentication and Session Management	Broken Authentication
Broken Authentication and Session Management	Malicious File Execution	Broken Authentication and Session Management	XSS	Sensitive Data Exposure
XSS	IDOR	IDOR	IDOR	XXE
Buffer Overflow	CSRF	CSRF	Security Misconfiguration	Broken Access Control
Injection Flaws	Information leakage and improper error handling	Security Misconfiguration	Sensitive Data Exposure	Security Misconfiguration
Improper Error Handling	Broken Authentication and Session Management	Insecure Cryptographic storage	Missing function level access control	XSS
Insecure Storage	Insecure Cryptographic storage	Failure to restrict URL access	CSRF	Insecure Deserialization
Application DoS	Insecure communications	Insufficient transport layer encryption	Components with known vulnerabilities	Components with known vulnerabilities
Insecure Configuration Management	Failure to restrict URL access	Unvalidated redirects and forwards	Unvalidated redirects and forwards	Insufficient logging and monitoring

WEB SECURITY IN ~~2018~~ 2004

2004	2007	2010	2013	2017
Unvalidated Input	XSS	Injection	Injection	Injection
Broken Access Control	Injection Flaws	XSS	Broken Authentication and Session Management	Broken Authentication
Broken Authentication and Session Management	Malicious File Execution	Broken Authentication and Session Management	XSS	Sensitive Data Exposure
XSS	IDOR	IDOR	IDOR	XXE
Buffer Overflow	CSRF	CSRF	Security Misconfiguration	Broken Access Control
Injection Flaws	Information leakage and improper error handling	Security Misconfiguration	Sensitive Data Exposure	Security Misconfiguration
Improper Error Handling	Broken Authentication and Session Management	Insecure Cryptographic storage	Missing function level access control	XSS
Insecure Storage	Insecure Cryptographic storage	Failure to restrict URL access	CSRF	Insecure Deserialization
Application DoS	Insecure communications	Insufficient transport layer encryption	Components with known vulnerabilities	Components with known vulnerabilities
Insecure Configuration Management	Failure to restrict URL access	Unvalidated redirects and forwards	Unvalidated redirects and forwards	Insufficient logging and monitoring

WEB SECURITY IN ~~2018~~ 2004

2004	2007	2010	2013	2017
Unvalidated Input	XSS	Injection	Injection	Injection
Broken Access Control	Injection Flaws	XSS	Broken Authentication and Session Management	Broken Authentication
Broken Authentication and Session Management	Malicious File Execution	Broken Authentication and Session Management	XSS	Sensitive Data Exposure
XSS	IDOR	IDOR	IDOR	XXE
Buffer Overflow	CSRF	CSRF	Security Misconfiguration	Broken Access Control
Injection Flaws	Information leakage and improper error handling	Security Misconfiguration	Sensitive Data Exposure	Security Misconfiguration
Improper Error Handling	Broken Authentication and Session Management	Insecure Cryptographic storage	Missing function level access control	XSS
Insecure Storage	Insecure Cryptographic storage	Failure to restrict URL access	CSRF	Insecure Deserialization
Application DoS	Insecure communications	Insufficient transport layer encryption	Components with known vulnerabilities	Components with known vulnerabilities
Insecure Configuration Management	Failure to restrict URL access	Unvalidated redirects and forwards	Unvalidated redirects and forwards	Insufficient logging and monitoring

WEB SECURITY IN ~~2018~~ 2004

2004	2007	2010	2013	2017
Unvalidated Input	XSS	Injection	Injection	Injection
Broken Access Control	Injection Flaws	XSS	Broken Authentication and Session Management	Broken Authentication
Broken Authentication and Session Management	Malicious File Execution	Broken Authentication and Session Management	XSS	Sensitive Data Exposure
XSS	IDOR	IDOR	IDOR	XXE
Buffer Overflow	CSRF	CSRF	Security Misconfiguration	Broken Access Control
Injection Flaws	Information leakage and improper error handling	Security Misconfiguration	Sensitive Data Exposure	Security Misconfiguration
Improper Error Handling	Broken Authentication and Session Management	Insecure Cryptographic storage	Missing function level access control	XSS
Insecure Storage	Insecure Cryptographic storage	Failure to restrict URL access	CSRF	Insecure Deserialization
Application DoS	Insecure communications	Insufficient transport layer encryption	Components with known vulnerabilities	Components with known vulnerabilities
Insecure Configuration Management	Failure to restrict URL access	Unvalidated redirects and forwards	Unvalidated redirects and forwards	Insufficient logging and monitoring

WEB SECURITY IN 2018

2004	2007	2010	2013	2017
Unvalidated Input	XSS	Injection	Injection	Injection
Broken Access Control	Injection Flaws	XSS	Broken Authentication and Session Management	Broken Authentication
Broken Authentication and Session Management	Malicious File Execution	Broken Authentication and Session Management	XSS	Sensitive Data Exposure
XSS	IDOR	IDOR	IDOR	XXE
Buffer Overflow	CSRF	CSRF	Security Misconfiguration	Broken Access Control
Injection Flaws	Information leakage and improper error handling	Security Misconfiguration	Sensitive Data Exposure	Security Misconfiguration
Improper Error Handling	Broken Authentication and Session Management	Insecure Cryptographic storage	Missing function level access control	XSS
Insecure Storage	Insecure Cryptographic storage	Failure to restrict URL access	CSRF	Insecure Deserialization
Application DoS	Insecure communications	Insufficient transport layer encryption	Components with known vulnerabilities	Components with known vulnerabilities
Insecure Configuration Management	Failure to restrict URL access	Unvalidated redirects and forwards	Unvalidated redirects and forwards	Insufficient logging and monitoring

CROSS-SITE SCRIPTING (XSS)

CROSS-SITE SCRIPTING (XSS)

- Attack the user’s browser context (via javascript)
- Attacks can steal
  - Passwords
  - Credit card numbers
  - Session tokens
  - CPU time for bitcoin mining
- Exploitable due to poor input encoding and validation

CROSS-SITE SCRIPTING (XSS)

- Demo: https://codepen.io/JacobReynolds/pen/ZxapqW?editors=1010

CROSS-SITE SCRIPTING (XSS) | CAUSES

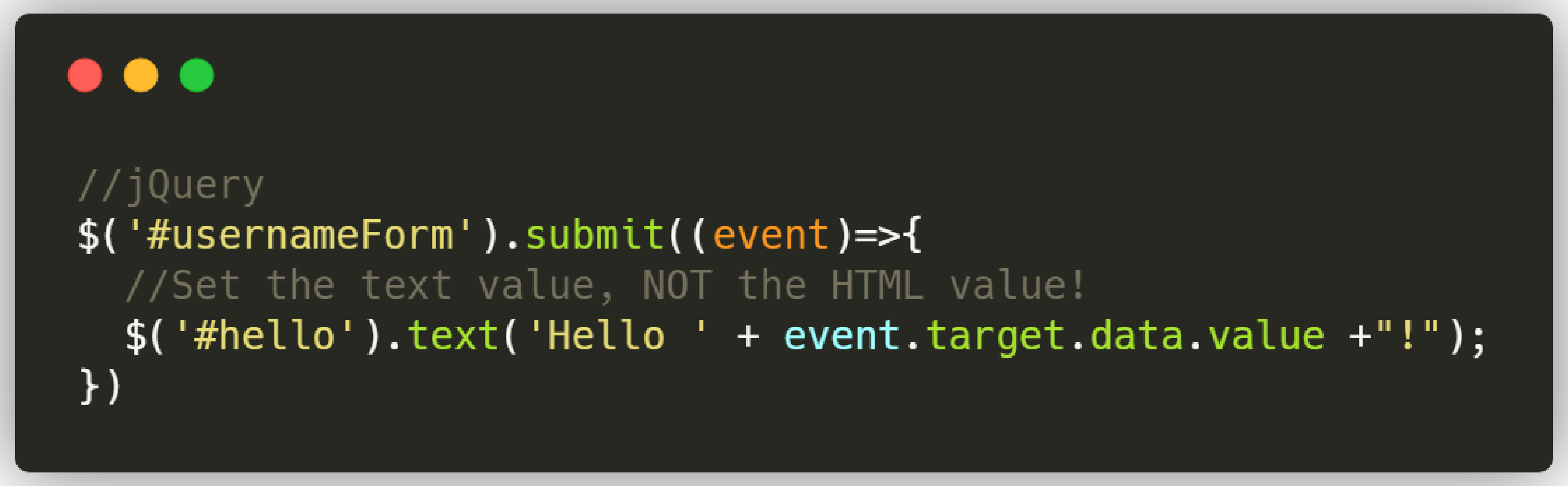
- Direct input reflection
- Blacklisting
- Lack of encoding

CROSS-SITE SCRIPTING (XSS) | FIXED IT BOSS!

CROSS-SITE SCRIPTING (XSS) | REMEDIATION

- Users suck and will try to break your stuff
- Whitelist every parameter you can
- HTML encode anything else

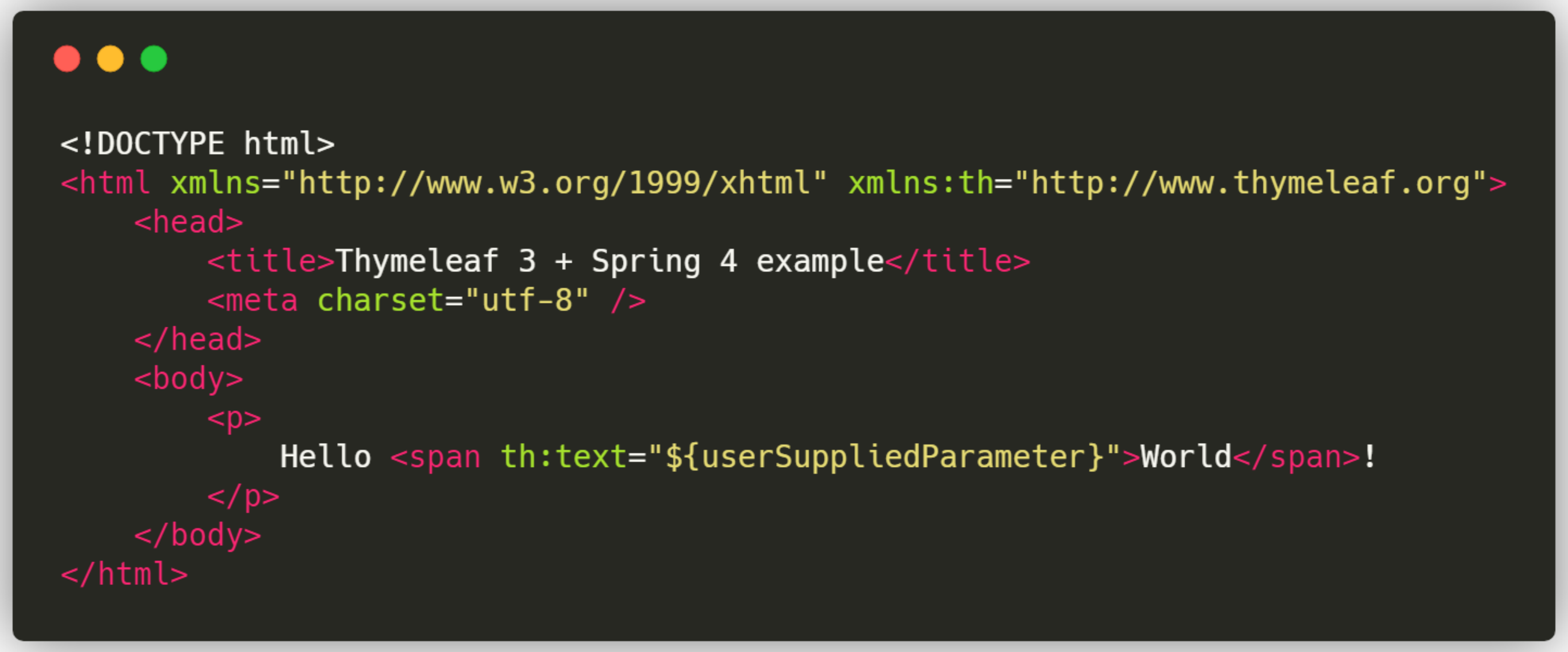
CROSS-SITE SCRIPTING (XSS) FRONTEND REMEDIATION | JAVASCRIPT



```
//jQuery
$('#usernameForm').submit((event)=>{
  //Set the text value, NOT the HTML value!
  $('#hello').text('Hello ' + event.target.data.value +"!");
})
```

View online

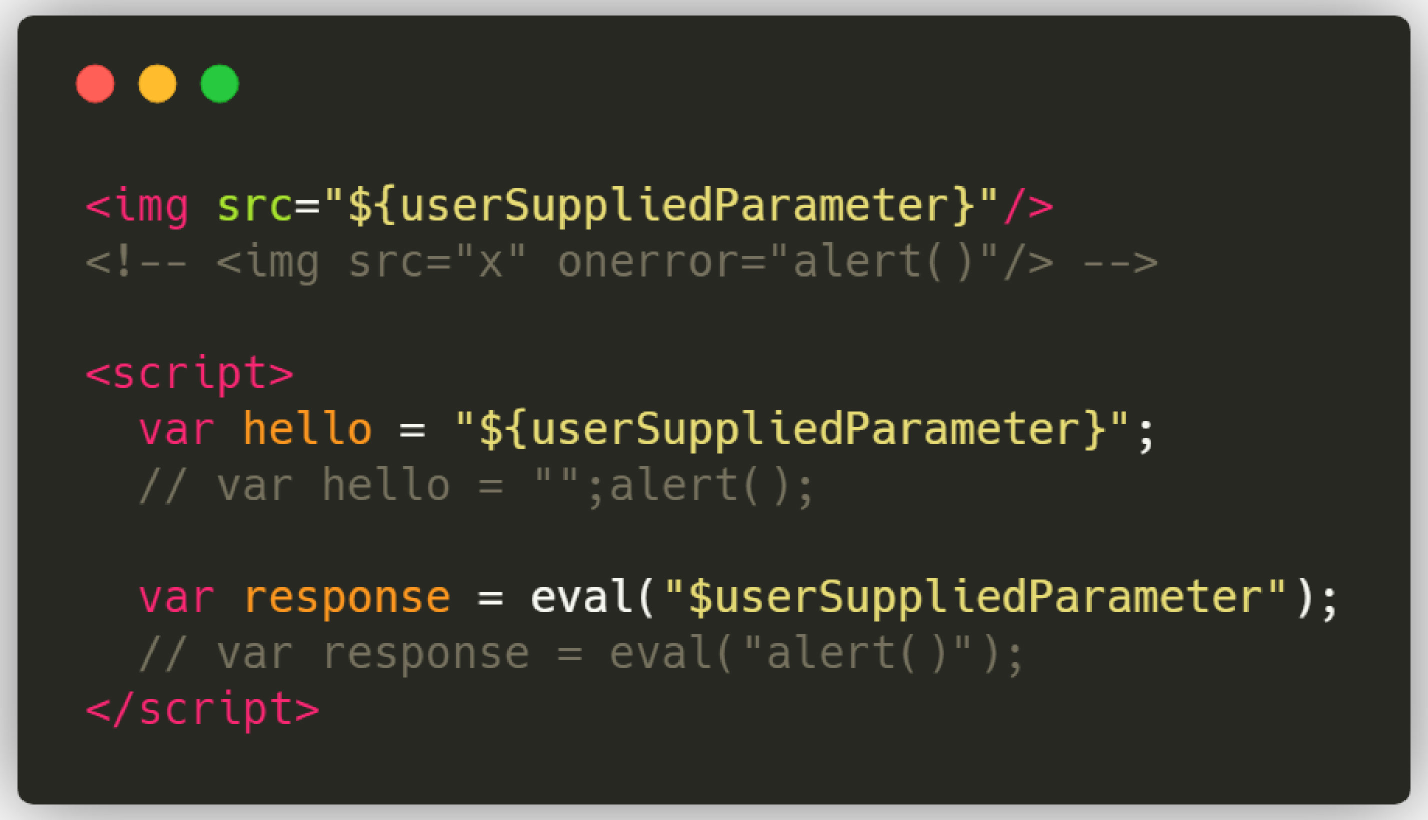
CROSS-SITE SCRIPTING (XSS) TEMPLATE REMEDIATION | SPRING



```html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Thymeleaf 3 + Spring 4 example</title>
        <meta charset="utf-8" />
    </head>
    <body>
        <p>
            Hello <span th:text="${userSuppliedParameter}">World</span>!
        </p>
    </body>
</html>
```

View online

CROSS-SITE SCRIPTING (XSS) TEMPLATE REMEDIATION | NO-NOS



```
<img src="${userSuppliedParameter}"/>
<!-- <img src="x" onerror="alert()"/> -->

<script>
  var hello = "${userSuppliedParameter}";
  // var hello = "";alert();

  var response = eval("$userSuppliedParameter");
  // var response = eval("alert()");
</script>
```

View online

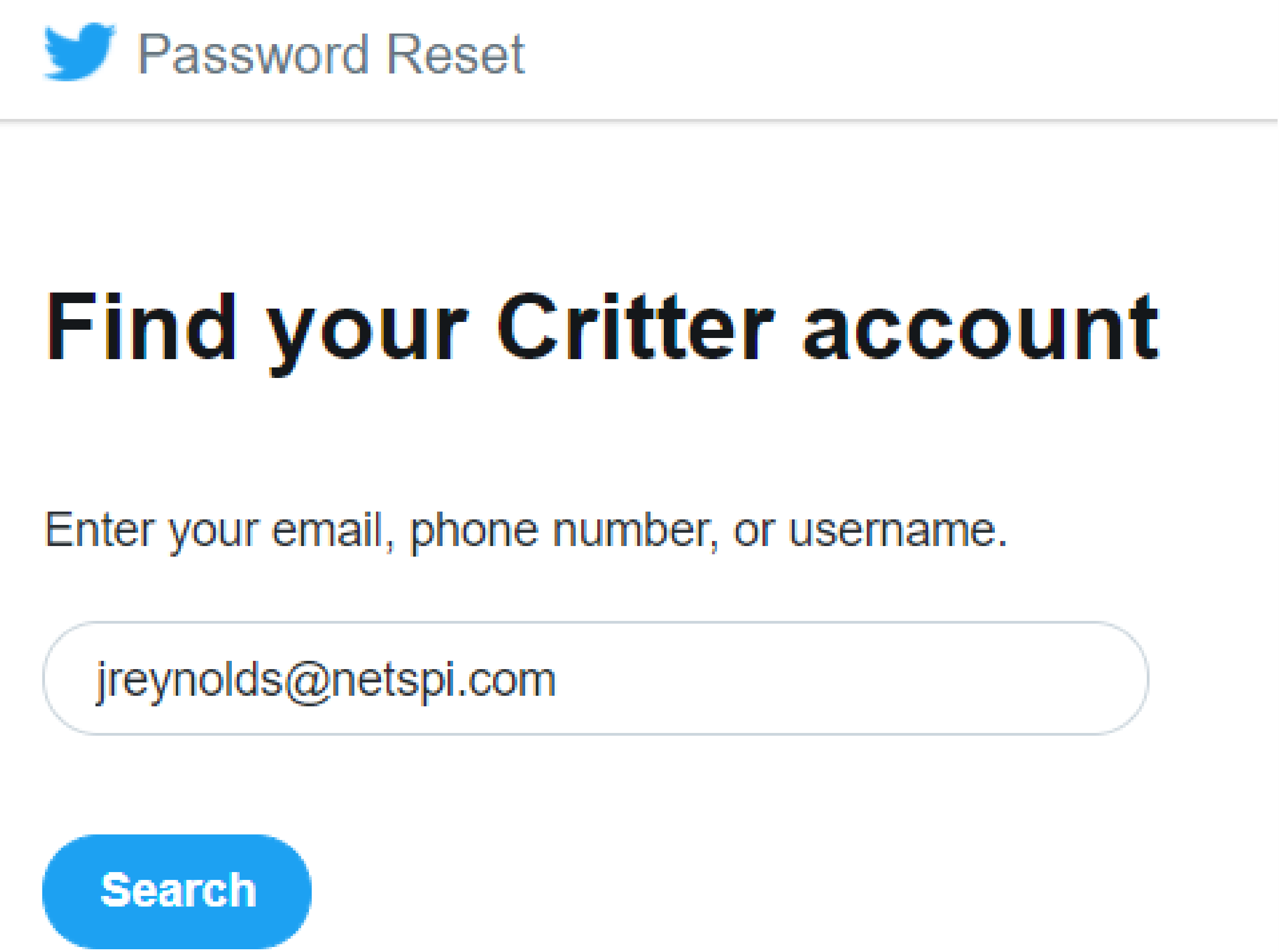
BROKEN AUTHENTICATION

BROKEN AUTHENTICATION

- Information disclosure
- Session handling
- Account exploitation

BROKEN AUTHENTICATION | CAUSES

- Information disclosure
  - Verbose error messages

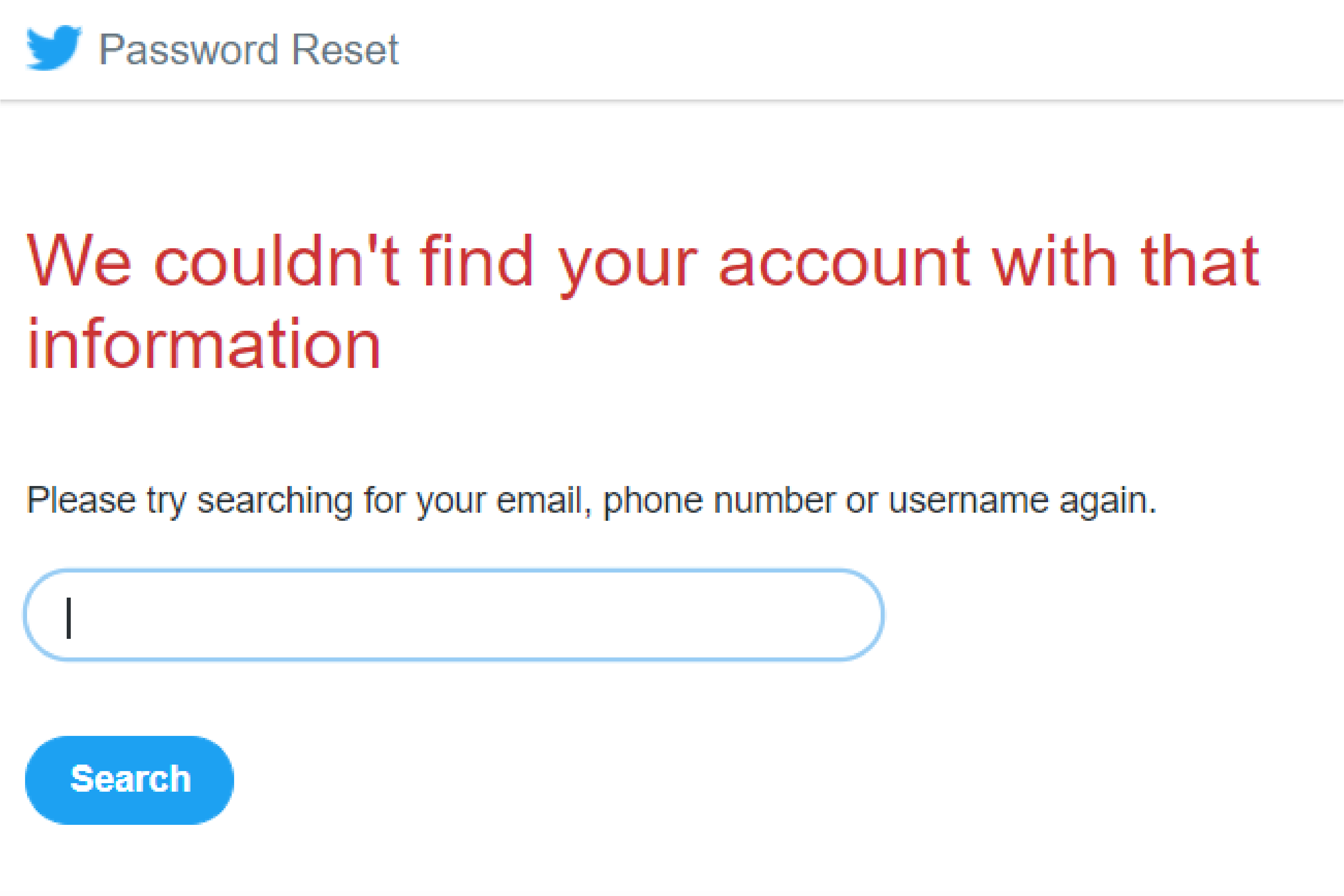


Password Reset

Find your Critter account

Enter your email, phone number, or username.

jreynolds@netspi.com



Password Reset

We couldn't find your account with that information

Please try searching for your email, phone number or username again.

BROKEN AUTHENTICATION | CAUSES

- Information disclosure
  - Verbose error messages
- Session handling
  - Session expiration
  - Session destruction
  - Session rotation
- Account exploitation
  - Weak password policies and storage
  - Weak lockout policies
  - Lack of multifactor

BROKEN AUTHENTICATION | REMEDIATION

- Rotate, destruct, and renew session tokens as often as possible
- Enforce strict lockout policies
- Give generic error messages
- Follow NIST Special Publication 800-63B for password policies
  - 5cr3w p@$$w0rd c0mpl3x!ty, passwordLengthAndUniquenessIsWhatMatters
  - Don’t require password rotation unless an action occurs
  - Securely store passwords
  - Multifactor
  - Don’t allow common passwords

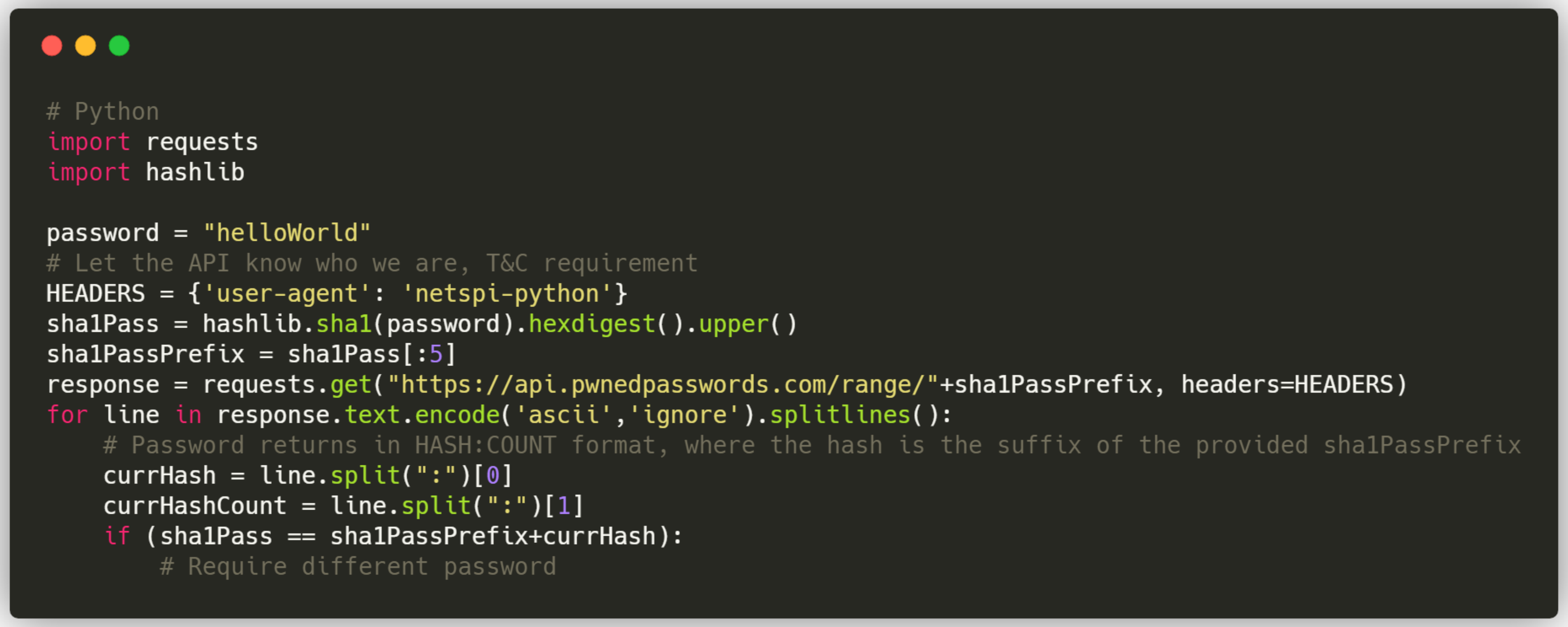
BROKEN AUTHENTICATION | K-ANONYMITY

- Don’t allow common passwords
- *k*-anonymity
  - Each person in a *k*-anonymous size-*n* data set cannot be distinguished from *k*-1 others in the data set
  - Securely and anonymously verify passwords without disclosing their passwords
- Cloudflare and Troy Hunt
  - haveibeenpwned.com
  - Send first 5 characters of your SHA-1 password hash, get any matches starting with those 5
  - Compare on your servers, not theirs

BROKEN AUTHENTICATION | K-ANONYMITY

- P@ssw0rd -> SHA1 -> **21BD1**2DC183F740EE76F27B78EB39C8AD972A757
- GET https://api.pwnedpasswords.com/range/**21BD1**
  - …
  - 2DC183F740EE76F27B78EB39C8AD972A757:47205
  - …

BROKEN AUTHENTICATION | PYTHON



```python
# Python
import requests
import hashlib

password = "helloWorld"
# Let the API know who we are, T&C requirement
HEADERS = {'user-agent': 'netspi-python'}
sha1Pass = hashlib.sha1(password).hexdigest().upper()
sha1PassPrefix = sha1Pass[:5]
response = requests.get("https://api.pwnedpasswords.com/range/"+sha1PassPrefix, headers=HEADERS)
for line in response.text.encode('ascii','ignore').splitlines():
    # Password returns in HASH:COUNT format, where the hash is the suffix of the provided sha1PassPrefix
    currHash = line.split(":")[0]
    currHashCount = line.split(":")[1]
    if (sha1Pass == sha1PassPrefix+currHash):
        # Require different password
```

View online

SQL INJECTION

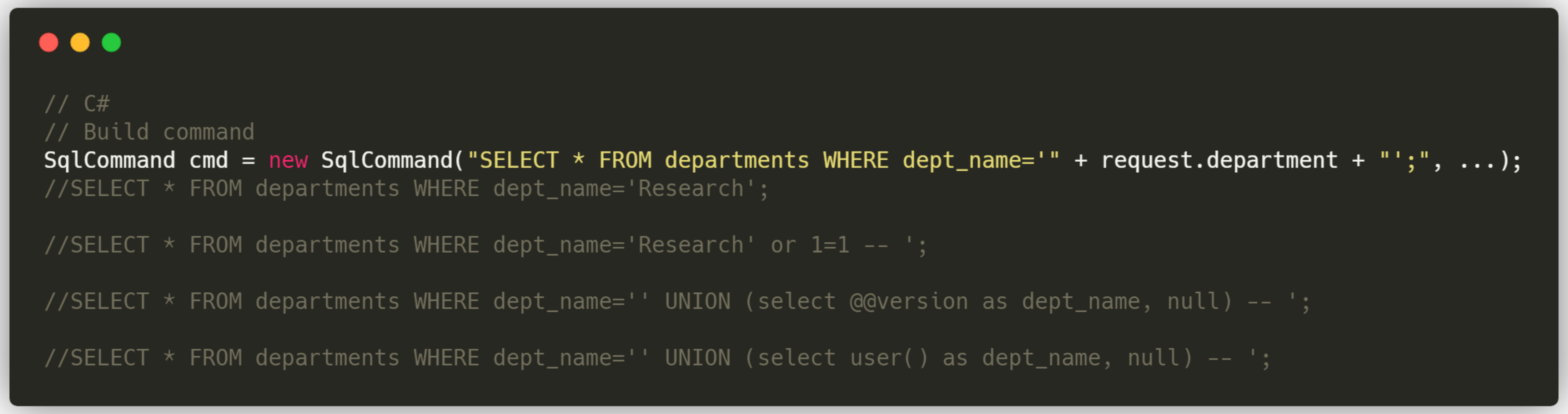
SQL INJECTION

- OWASP TOP 10, #1 for almost a decade
- Appending user-supplied input to server-side SQL queries
- Up to and including arbitrary command execution on the database and related servers
- https://codecurmudgeon.com/wp/sql-injection-hall-of-shame/

SQL INJECTION | DEMO

- http://localhost:3000/0/2

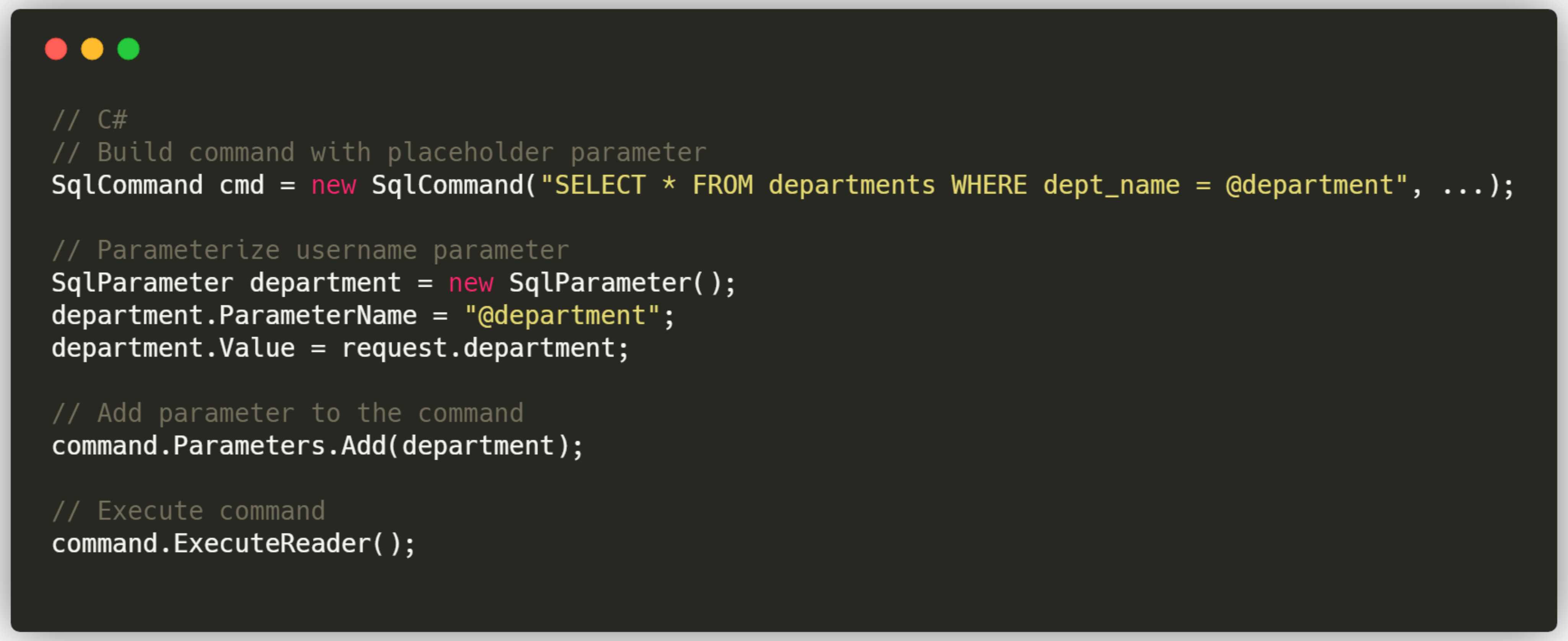
SQL INJECTION | DEMO



```
// C#
// Build command
SqlCommand cmd = new SqlCommand("SELECT * FROM departments WHERE dept_name='" + request.department + "';", ...);
//SELECT * FROM departments WHERE dept_name='Research';

//SELECT * FROM departments WHERE dept_name='Research' or 1=1 -- ';

//SELECT * FROM departments WHERE dept_name='' UNION (select @@version as dept_name, null) -- ';

//SELECT * FROM departments WHERE dept_name='' UNION (select user() as dept_name, null) -- ';
```

View online

SQL INJECTION | REMEDIATION



```
// C#
// Build command with placeholder parameter
SqlCommand cmd = new SqlCommand("SELECT * FROM departments WHERE dept_name = @department", ...);

// Parameterize username parameter
SqlParameter department = new SqlParameter();
department.ParameterName = "@department";
department.Value = request.department;

// Add parameter to the command
command.Parameters.Add(department);

// Execute command
command.ExecuteReader();
```

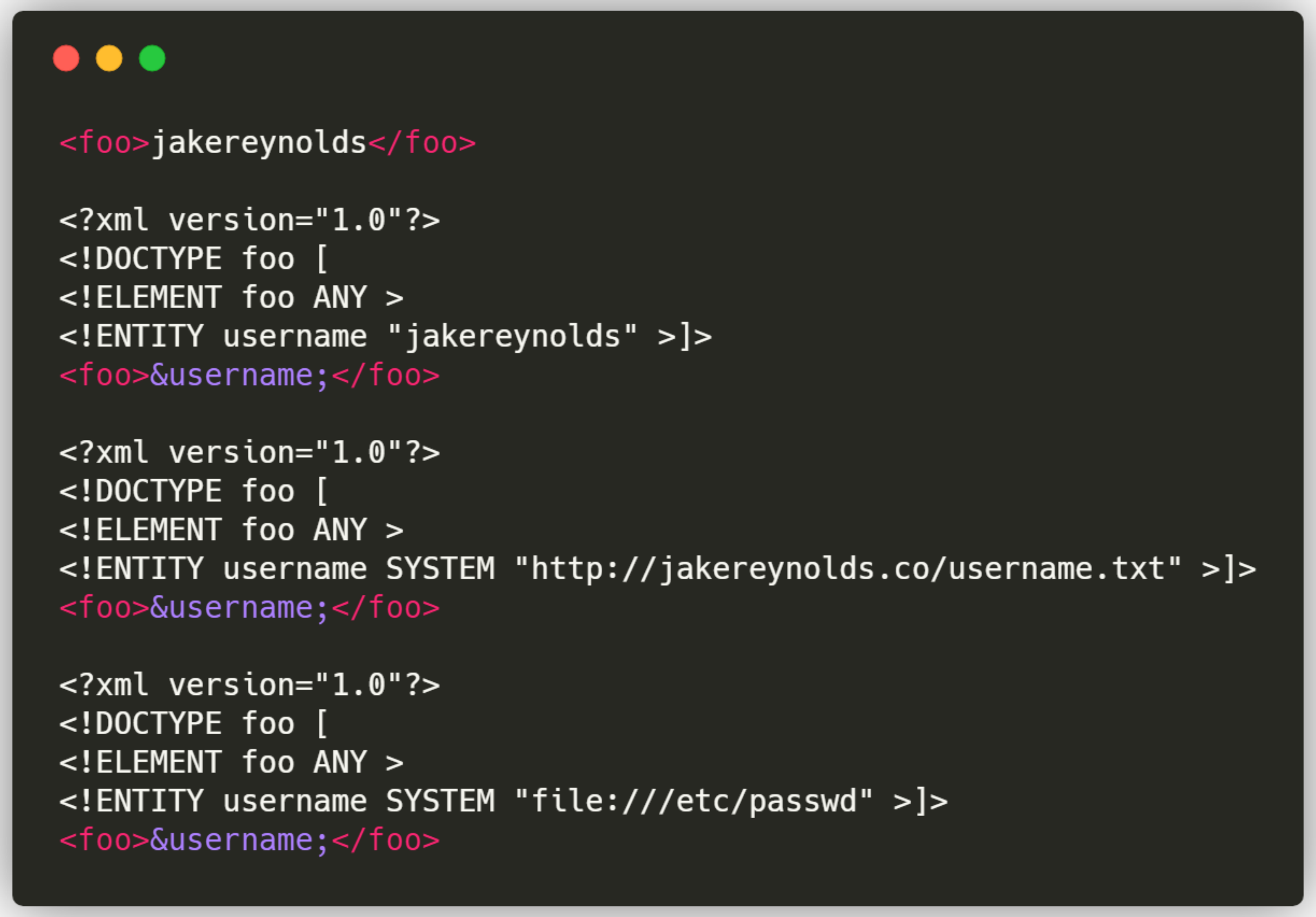
View online

XML EXTERNAL ENTITY INJECTION (XXE)

XML EXTERNAL ENTITY INJECTION (XXE)

- Built-in functionality of XML that allows inclusion of external resources via external entities
- External entities can be http://, and more importantly file://
- Also allows DoS, and SSRF

XML EXTERNAL ENTITY INJECTION (XXE)



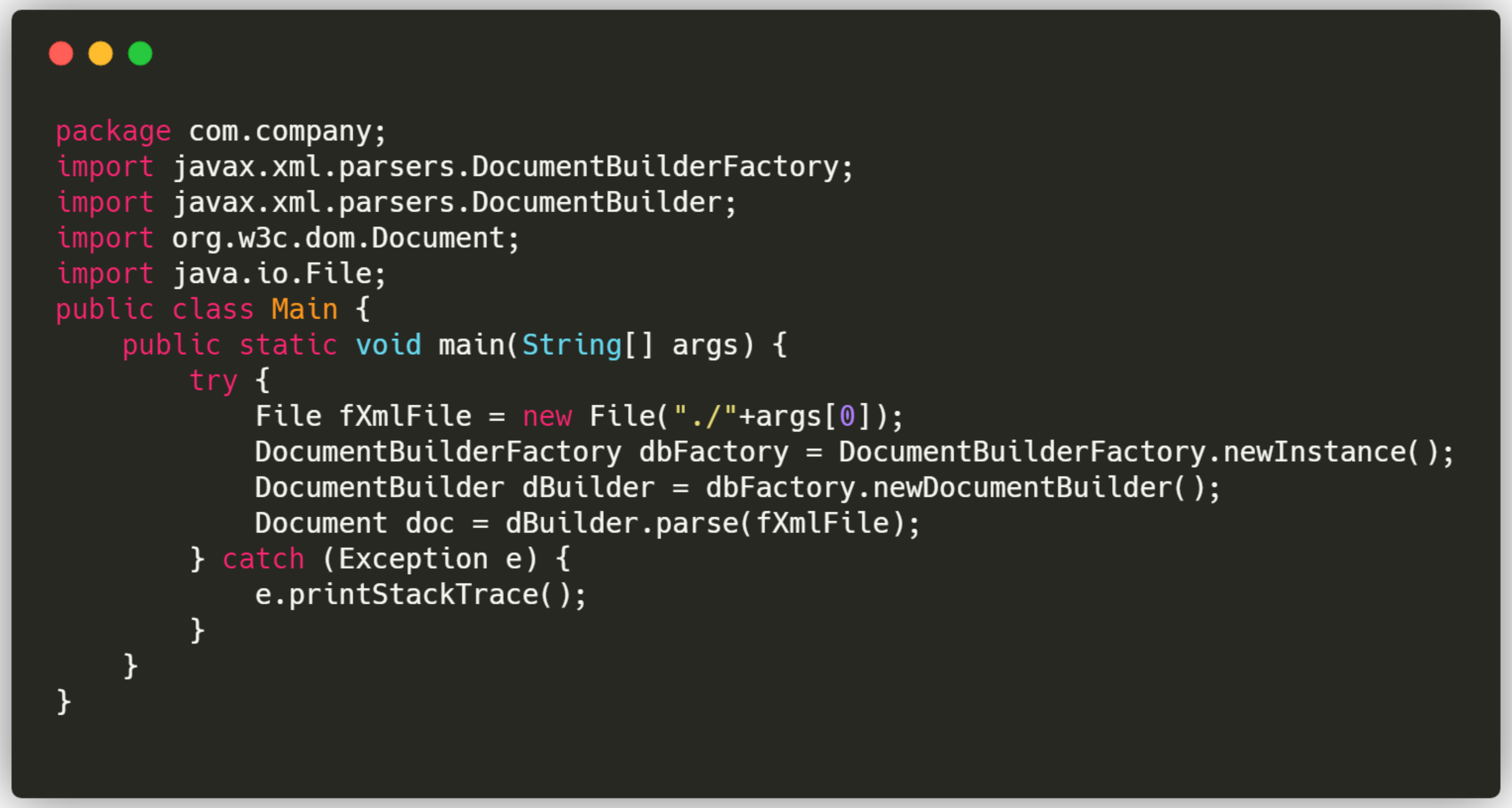
```
<foo>jakereynolds</foo>

<?xml version="1.0"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY username "jakereynolds" >]>
<foo>&username;</foo>

<?xml version="1.0"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY username SYSTEM "http://jakereynolds.co/username.txt" >]>
<foo>&username;</foo>

<?xml version="1.0"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY username SYSTEM "file:///etc/passwd" >]>
<foo>&username;</foo>
```

View online

XML EXTERNAL ENTITY INJECTION (XXE) | DEMO

- http://localhost:3000/0/1

XML EXTERNAL ENTITY INJECTION (XXE) | CODE



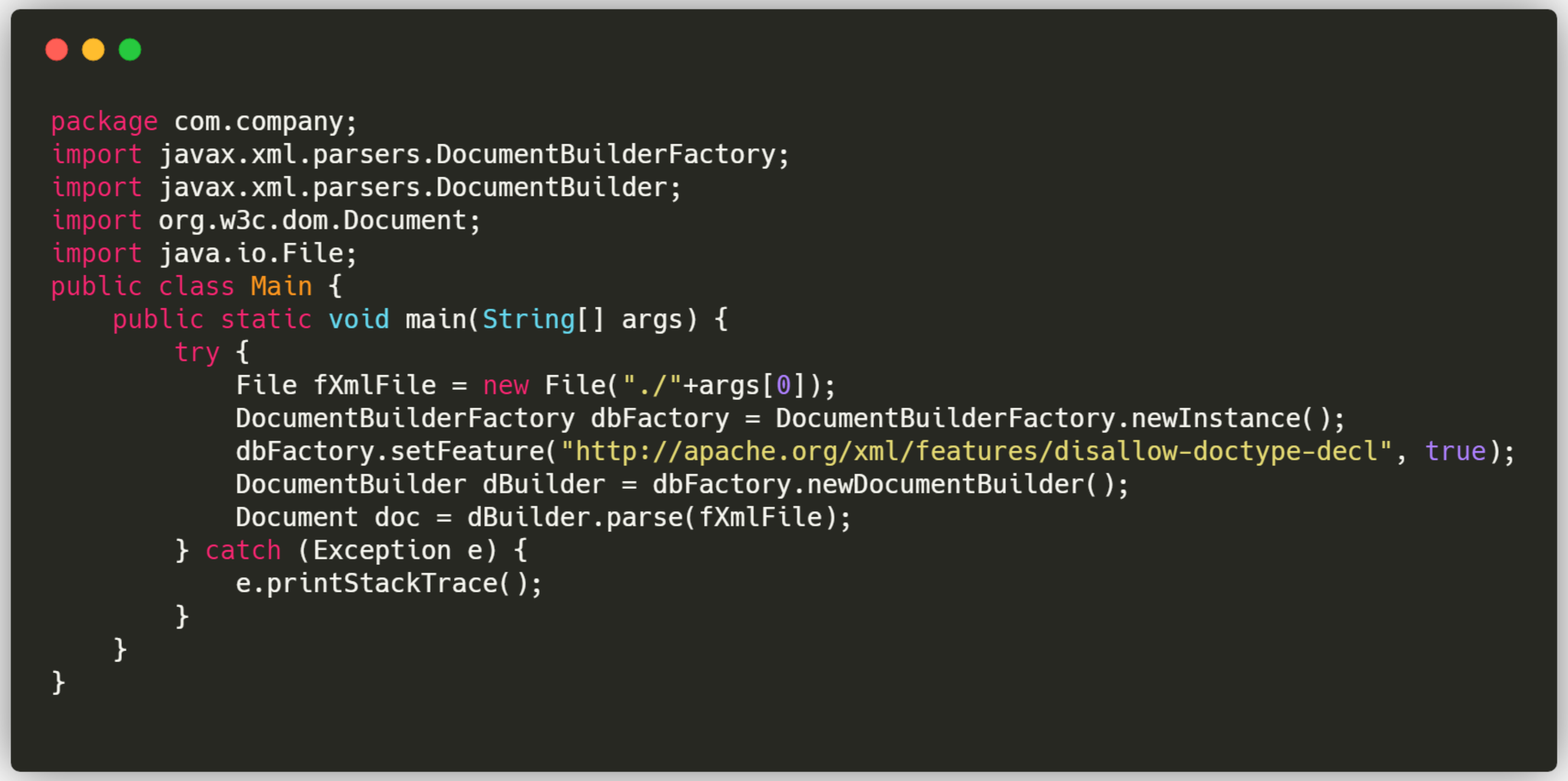
```java
package com.company;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import java.io.File;
public class Main {
    public static void main(String[] args) {
        try {
            File fXmlFile = new File("./"+args[0]);
            DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(fXmlFile);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

View online

XML EXTERNAL ENTITY INJECTION (XXE) | REMEDIATION

- Please don’t use XML
- If you have to, disable doctypes/entities
- Java
  - https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet#Java
- .NET
  - https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet#.NET
- NodeJS
  - Why are you using XML?

XML EXTERNAL ENTITY INJECTION (XXE) | REMEDIATION



```
package com.company;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import java.io.File;
public class Main {
    public static void main(String[] args) {
        try {
            File fXmlFile = new File("./"+args[0]);
            DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
            dbFactory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(fXmlFile);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

View online

INSECURE DESERIALIZATION

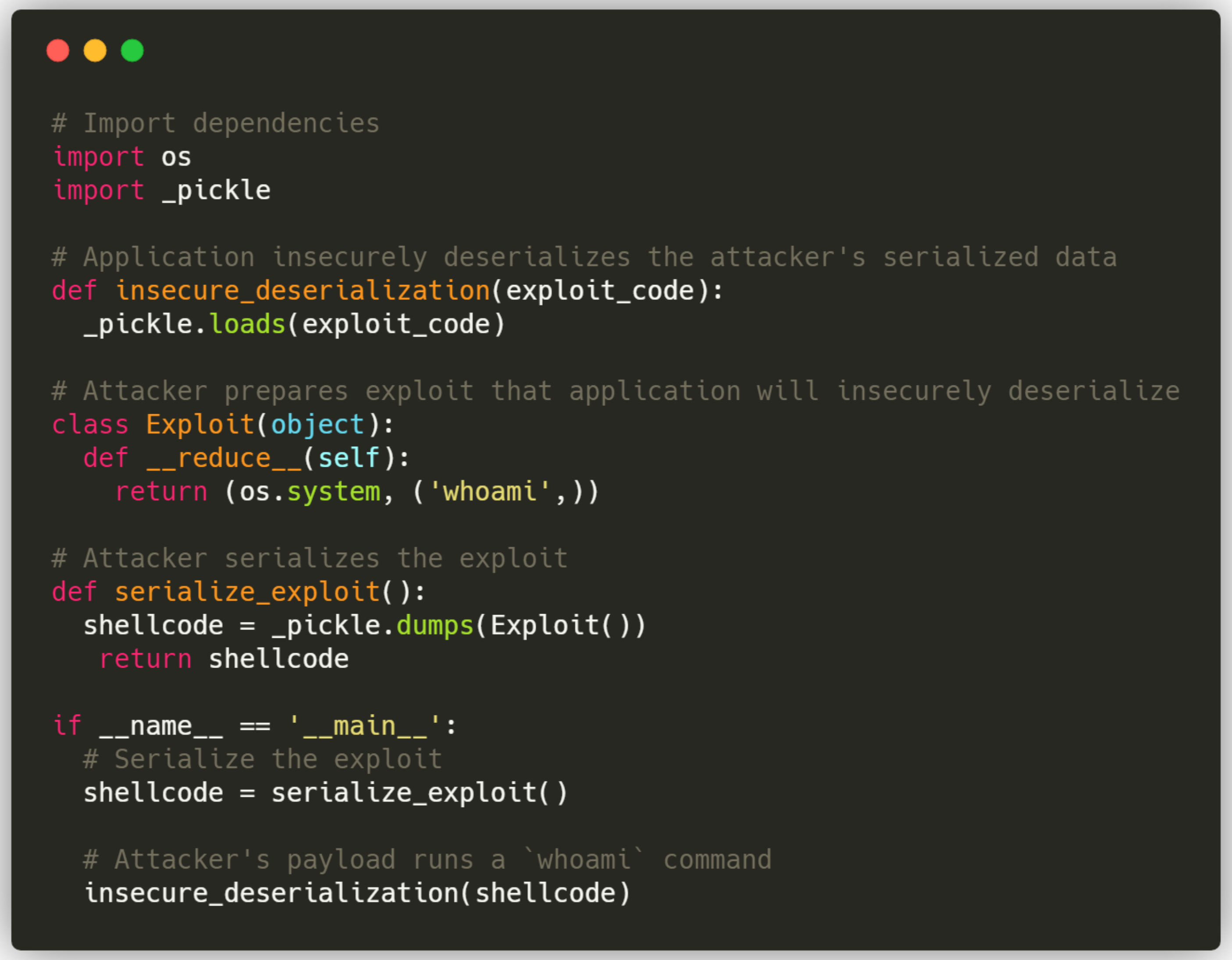
INSECURE DESERIALIZATION

- Serialize data structures and classes for storage or communication
- Commonly allows
  - AuthN/AuthZ bypasses
  - Command execution

INSECURE DESERIALIZATION | AUTH BYPASS

- a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"**user**"; i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
- a:4:{i:0;i:1;i:1;s:5:"Mallory";i:2;s:5:"**admin**";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

INSECURE DESERIALIZATION



```
# Import dependencies
import os
import _pickle

# Application insecurely deserializes the attacker's serialized data
def insecure_deserialization(exploit_code):
  _pickle.loads(exploit_code)

# Attacker prepares exploit that application will insecurely deserialize
class Exploit(object):
  def __reduce__(self):
    return (os.system, ('whoami',))

# Attacker serializes the exploit
def serialize_exploit():
  shellcode = _pickle.dumps(Exploit())
   return shellcode

if __name__ == '__main__':
  # Serialize the exploit
  shellcode = serialize_exploit()

  # Attacker's payload runs a `whoami` command
  insecure_deserialization(shellcode)
```

View online

INSECURE DESERIALIZATION | REMEDIATION

- Don’t use serialized objects
- If you have to, only allow primitive types
- If it’s for immutability, add a hash of the data

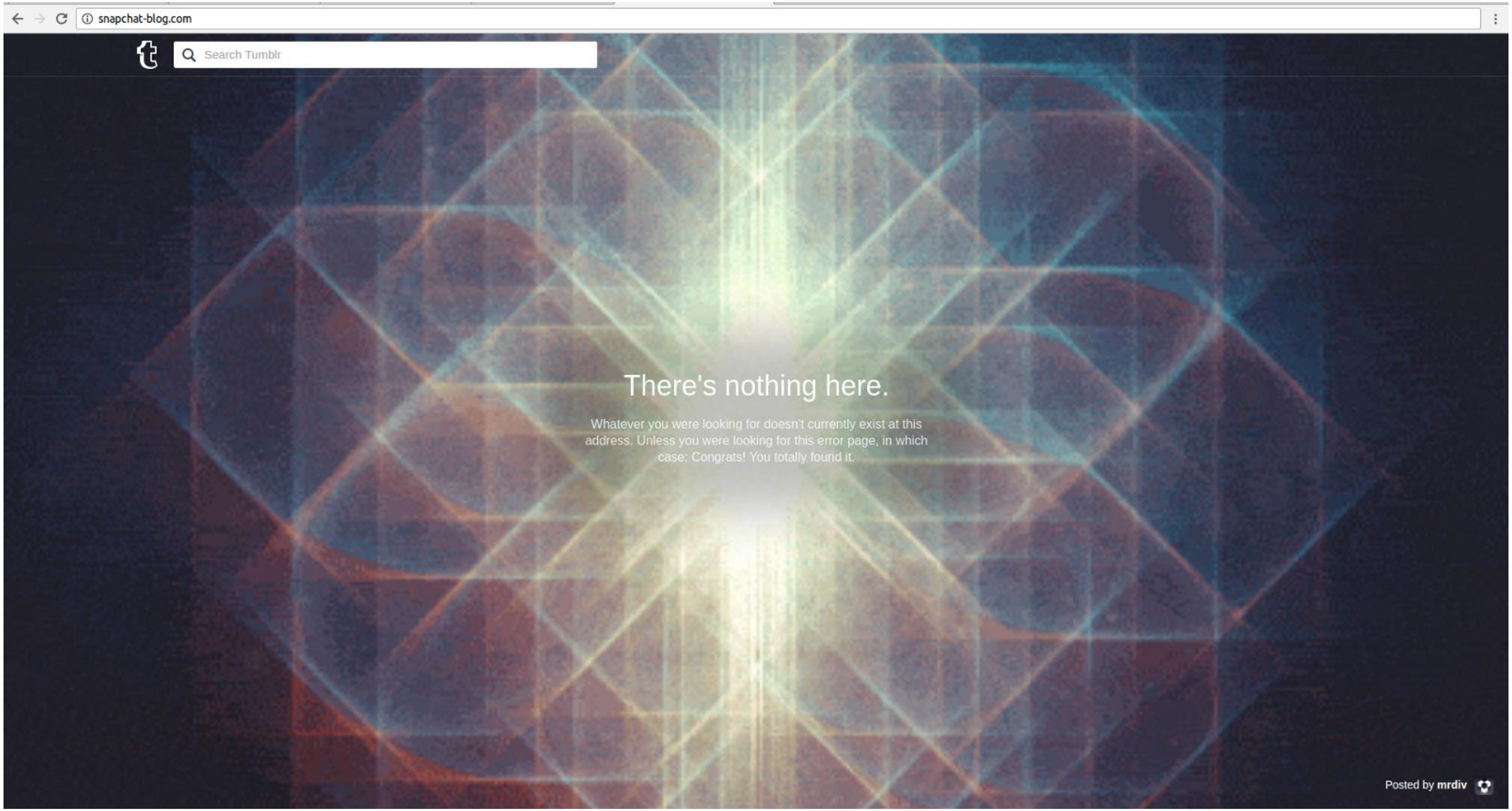
IS THAT IT?

- There will always be vulnerabilities and you’ll never know them all
- Defend yourself against the known
- Be aware enough to detect and defend against the unknown

WHAT ELSE IS THERE?

- Anything that is or could be public-facing is part of the website's attack surface
- Servers
  - DNS
  - Mail
  - Web
  - SSH
- Domain names
- Social media accounts
- Developers

SUBDOMAIN TAKEOVER



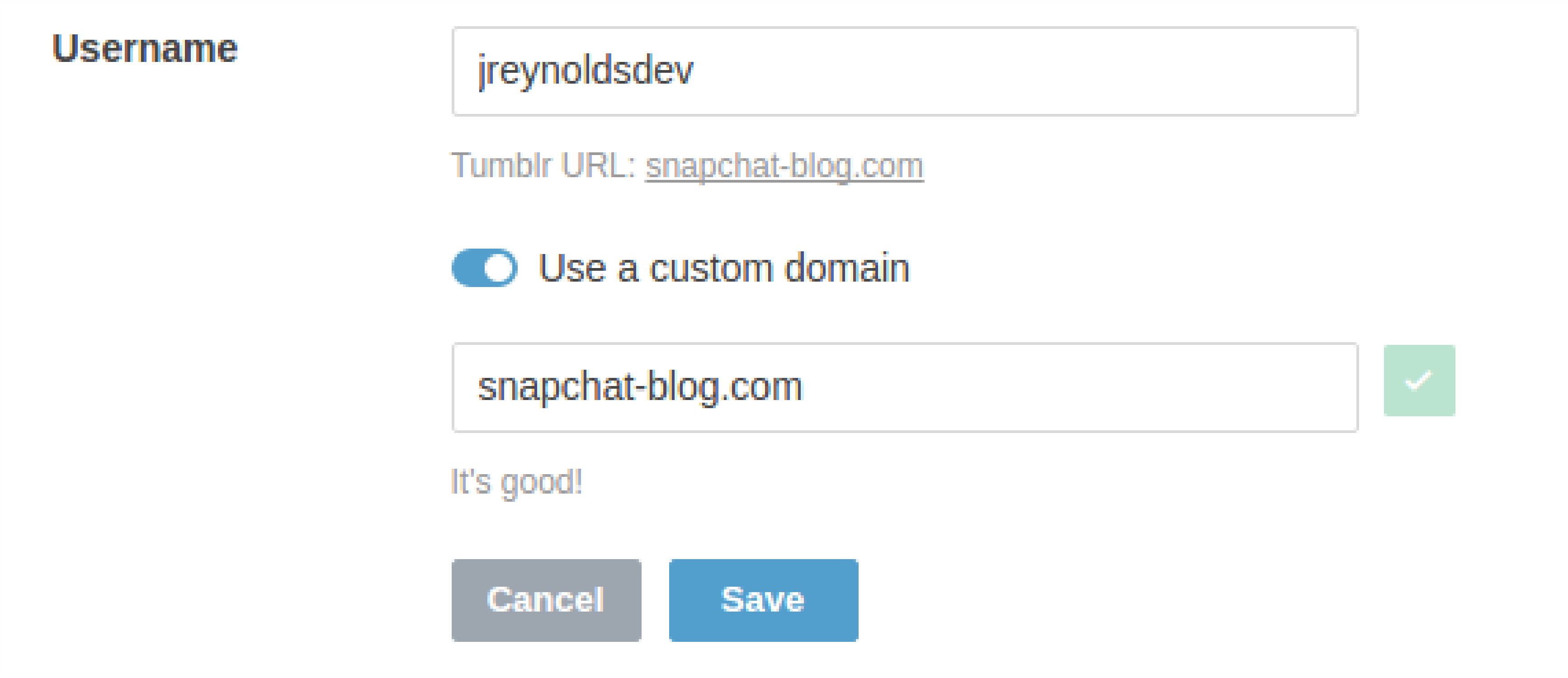
There's nothing here.

Whatever you were looking for doesn't currently exist at this address. Unless you were looking for this error page, in which case: Congrats! You totally found it.

SUBDOMAIN TAKEOVER

```
~$ nslookup snapchat-blog.com
Non-authoritative answer:
Name: snapchat-blog.com
Address: 66.6.32.21
```

SUBDOMAIN TAKEOVER



Username: jreynoldsdev

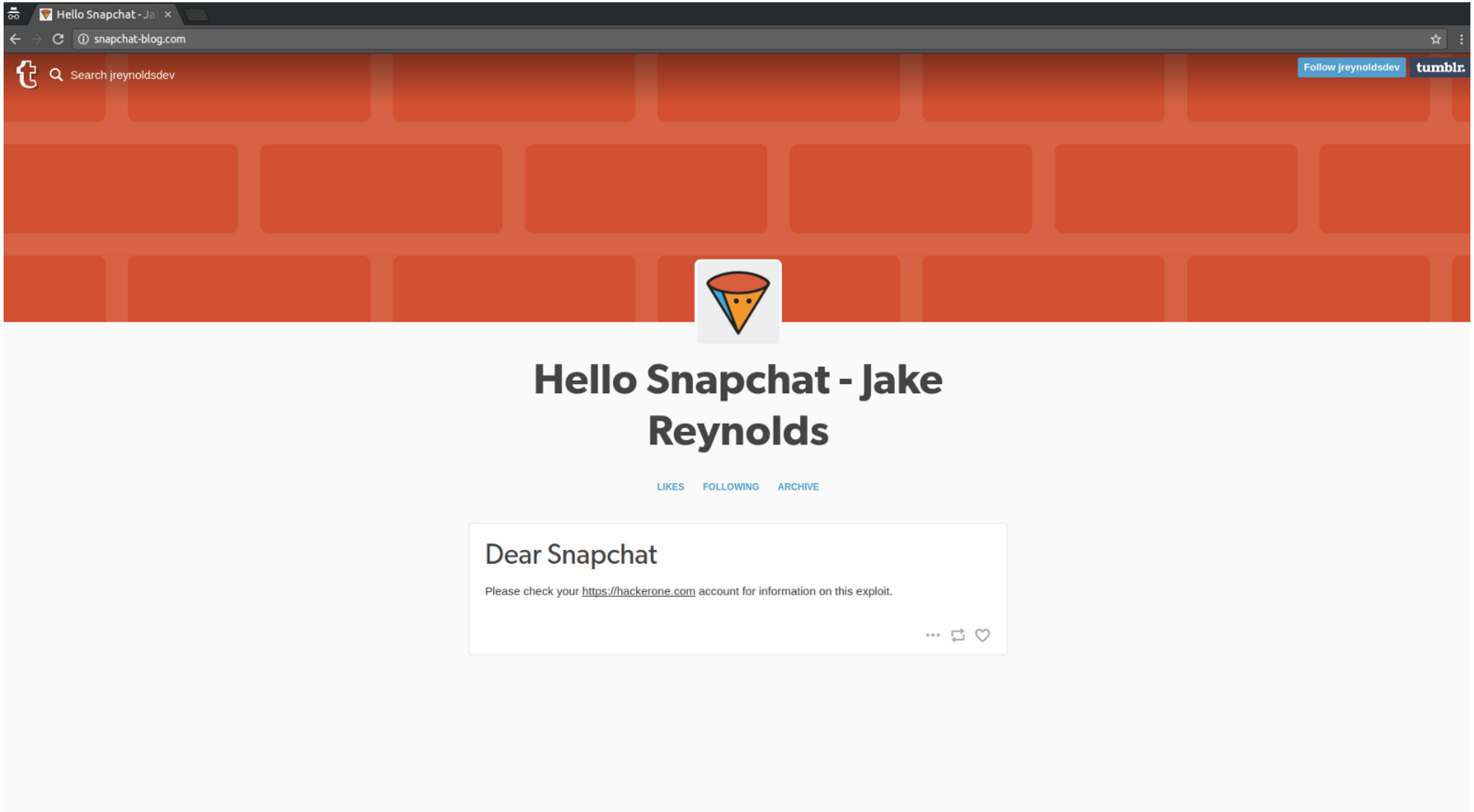
Tumblr URL: snapchat-blog.com

Use a custom domain

snapchat-blog.com ✓

It's good!

SUBDOMAIN TAKEOVER



Hello Snapchat - Jake Reynolds

Dear Snapchat

Please check your https://hackerone.com account for information on this exploit.

IS THAT IT?

- There will always be vulnerabilities and you'll never know them all
- Defend yourself against the known
- Be aware enough to detect and defend against the unknown

REFERENCES

- https://stackoverflow.com/questions/1265282/recommended-method-for-escaping-html-in-java
- https://www.dotnetperls.com/htmlencode-htmldecode
- https://github.com/jmiguelsamper/thymeleaf3-spring-helloworld/blob/master/src/main/webapp/WEB-INF/templates/index.html
- https://www.mkyong.com/spring-boot/spring-boot-hello-world-example-thymeleaf/
- https://www.acunetix.com/blog/articles/what-is-insecure-deserialization/
- https://www.troyhunt.com/ive-just-launched-pwned-passwords-version-2/
- https://www.owasp.org/index.php/Top_10-2017_A8-Insecure_Deserialization

NETSPI

MINNEAPOLIS | NEW YORK | PORTLAND | DENVER | DALLAS

Empowering enterprises to scale & operationalize their security programs, globally.