OWASP Analysis

# Chosen Topics

## XML External Entity

### Explanation

Some XML parsers have a known “vulnerability”, where external XML entities can be loaded into an XML File. These external entities have the capability to load files (either local or on a shared server), retrieve websites via a HTTP GET request and, in some cases, might lead to RCE (Remote Code Execution).

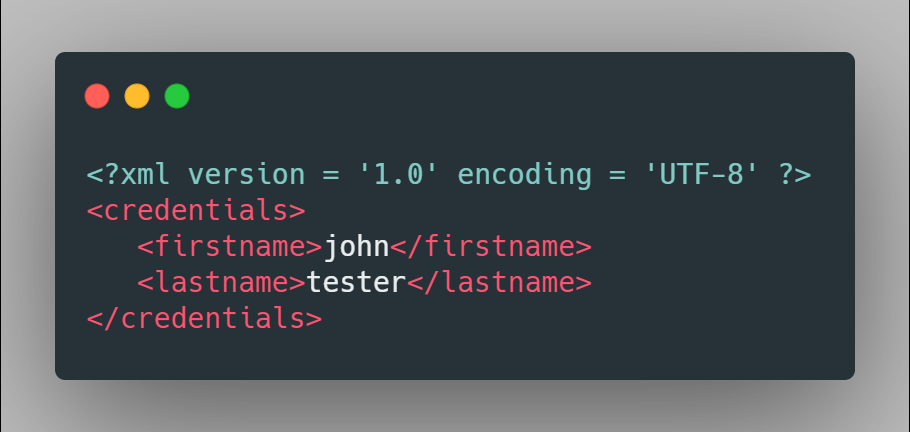
### Possible Prevention Methods

Nowadays most XML parsers have the option to load external entities turned off by default, but in some cases it might be necessary for a developer to use this feature set, and they would have to activate DTDs (Document Type Definitions). For these types of situations, OWASP maintains a cheatsheet on Github for XXE Prevention, which can be found here:

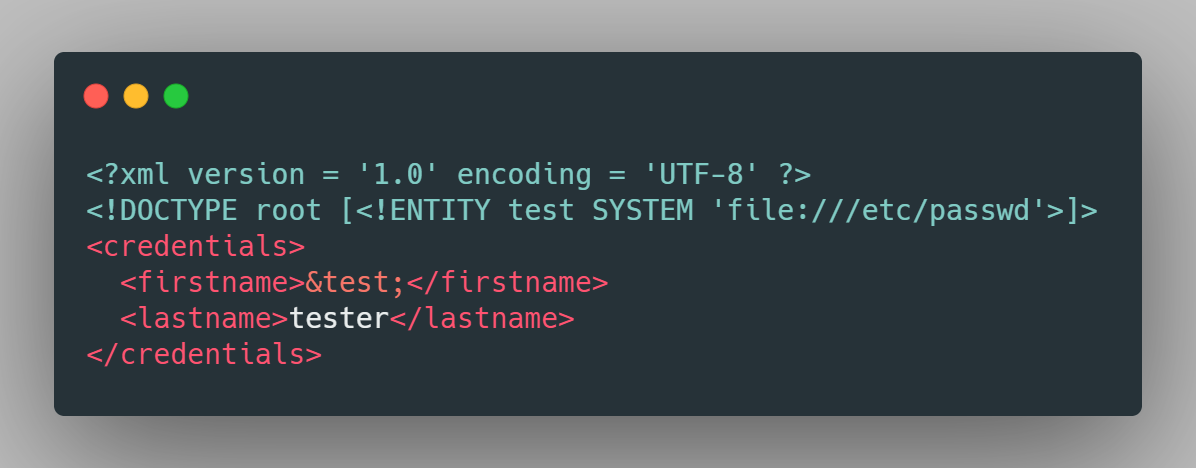
<https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.md>

### Example

Here we can see a typical XML configuration, which in our example is sent via HTTP to our Server.



On a System with a vulnerable XML parser, a working payload could look something like this:



We inject a DTD with an Entity that uses the “SYSTEM” keyword to load a file from the system. We then reference the entity in our request with “&”. If the data we sent to the server gets displayed back to us, like how it is likely the case for a username, we can retrieve the file in our user interface:



## Insecure Deserialisation

### Explanation

Serialisation is when data is converted into an easily portable format. Deserialisation is the opposite. This makes it easier to transport and store things like objects or collections by only sending/storing a single stream of binary data or a single string. The problem arises when the serialised data can be intercepted and even modified. For example, if you store some information about a user in a file which is only serialised, it’s possible for a malicious actor to deserialise the data and read the user’s information or change it however they please.

### Possible Prevention Methods

The only way to guarantee safe deserialisation is to never deserialise objects from insecure sources; however, there are some other methods to ensure the damage is limited:

* Providing integrity checks in the serialised objects such as a checksum.
* Enforcing strict type constraints.
* Encrypting serialised data.
* Extensive logging of deserialisation anomalies.
* Limiting deserialisation to low-privilege environments.

### Example

In this example a user’s login data has been serialised to send to the server for authentication:

C:4:"User":52:{a:3:{i:0;s:8:"username";i:1;s:8:"**password**";i:2;b:0;}}

Here you can see the user’s username and password is clearly visible, which is already quite insecure, but there is also a Boolean on the end. If an attacker were to modify the string to change its state like so:

C:4:"User":52:{a:3:{i:0;s:8:"username";i:1;s:8:"password";i:2;**b:1**;}}

It might give them privileges they weren’t meant to have.

# OWASP Top-10 Rankings since 2013

|  |  |  |
| --- | --- | --- |
| **Rank** | **Security Risks 2013** | **Security Risks 2017** |
| **A1** | Injection | Injection |
| **A2** | Broken Auth. / Session Mgmt. | Broken Auth. / Session Mgmt. |
| **A3** | Cross-Site Scripting | Cross-Site Scripting |
| **A4** | Insecure Direct Object References | Broken Access Control |
| **A5** | Security Misconfiguration | Security Misconfiguration |
| **A6** | Sensitive Data Exposure | Sensitive Data Exposure |
| **A7** | Missing Function Level Access Control | Insufficient Attack Protection |
| **A8** | Cross-Site Request Forgery | Cross-Site Request Forgery |
| **A9** | Using Components with known Vulns. | Using Components with known Vulns. |
| **A10** | Unvalidated Redirects / Forwards | Under protected APIs |

<https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2017_Release_Candidate1_English.pdf>

# OWASP Top 10 vs Proactive Controls

The OWASP proactive controls are measures which should be implemented in every project to ensure its security. They are as follows:

1. Define Security Requirements
2. Leverage Security Frameworks and Libraries
3. Secure Database Access
4. Encode and Escape Data
5. Validate All inputs
6. Implement Digital Identity
7. Enforce Access Controls
8. Protect Data Everywhere
9. Implement Security Logging and Monitoring
10. Handle All Errors and Exceptions

<https://owasp.org/www-project-proactive-controls/>

The main difference between these two lists is that the Top 10 Vulnerabilities lays out what issues are likely to appear and how to protect against them, while the Proactive Controls are — as the name implies — proactive measures to protect against all manner of threats.