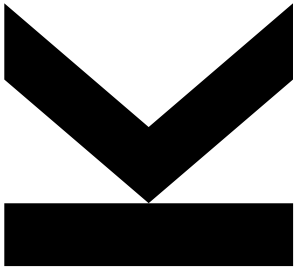
Programming problems



Server-side issues caused by developers

JOHANNES KEPLER
UNIVERSITÄT LINZ

Michael Sonntag
Institute of Networks and Security

INSTITUTE
OF NETWORKS
AND SECURITY

XML External Entities (XXE)

XXE

- The applications accepts some form of XML
  - Direct upload of XML documents (→ modern office documents!), XML metadata in other files, imports XML form another website, SAML (security/authentication language; based onXML), SOAP, inserting unescaped client input into a local XML document etc
    - This may be way down in some library or sub-protocol!
  - Someone might implement their own XML parser
    - “This is simple – just a few string operations; no need for a full parser!”
- Some parser somewhere accepts DTDs
  - DTDs can come from anywhere, e.g. some external server, or the local network (=scanning the LAN)
- The parser accepts external elements for these DTDs
  - The parser might be able to restrict this, but this would have to be explicitly configured

XXE: Examples

- **<?xml version="1.0" encoding="ISO-8859-1”?>**
  <!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM “file:///etc/password”>] >
  <foo>&xxe;</foo>
  Result: Password file included & displayed (or used as input & then…)
- **<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>**
  - Scanning the local network

XXE: Detection

- Try to include an XML bomb (see later) and look for parsing time/memory issues
- Include some local external entity & look whether a request appears
  - E.g. on the local host firewall
- Scan all input for XML
  - Decompress first, scan also binary files (metadata!)
    - XMP metadata in pictures, documents properties etc
- Check all places where XML is created or used
  - Are (external) DTDs deactivated?
  - How is the data validated: DTD or XSD or not at all?
  - Where is the XML coming from? Any external input?
- Escaping might not help: these are not “problematic” characters but normal text interpreted wrongly
  - Escaping would have to be extensive (at least all “<“!)

XXE: Prevention

- Ideally: disable all DTDs
  - This means, everything must already be directly inside the XML
  - Caveat: make sure what to put in it (escaping!)
- Second line of defense: disable external entities
  - Everything must be local
  - This can still cause problems, like “bombs” (=DoS)
- Use both (best case) or at least the second (if DTD **must** be used and this **cannot** be changed)
- Make sure to do this for all places XML is parsed
  - Typically this cannot be set generally, but only when creating an XML parser. These need to be set as “parameters” in a parser-dependent way.
- Validation needed: use XSD (=XML schemas)
- Fallback: use less complex formats, like JSON

Insecure deserialization

Insecure deserialization

- Objects are stored or transmitted in binary/encoded form and later deserialized to re-obtain programming objects
  - While they are “objects”, they are protected by the OS and the application; but in between they are “just data”
  - This is not simple to exploit: you have to know exactly how the object is serialized, what/where the elements are, and what they mean to be able to **usefully** change them!
    - Merely damaging them is easy…
- Three problems:
  - Data content may change (easy to do)
  - Code may change: difficult
    - Only if code is serialized too; typically code must already exist locally for objects to be deserialized (but see e.g. JavaScript or reflection!)
  - (Additional) Objects may be created artificially from scratch

Insecure deserialization

- Applies to:
  - Remote Procedure Calls (RPC), web services
    - Can be remote or (unlikely) locally (=between processes)
  - Persistence (databases, files)
  - HTML content sent between server and browser
    - As cookies, JSON etc

Insecure deserialization: Examples

- Storing the server state on the client
  - As data only, or as actual serialized objects
  - Serialized objects may be modified to create completely different objects, of which constructors might be executed immediately!
- Creating a subclass rendering a “private” data element “public”, and therefore modifiable by the attacker
- Storing user attributes in a cookie
  - User name, id, permissions etc
  - If “admin” values are know, they can be changed trivially
- Replacing an object by org.apache.commons.collections.functors.InstantiateTransformer
  - This class will create a new object based on reflection, i.e. you can specify whatever object you want to have created as text (=Text2Code → insecure and cannot be secured!)
    - Since version 3.2.2 disabled (because of security reasons ☺)

Insecure deserialization: Prevention

- Do not deserialize any objects containing code
  - Everything should only be a “data wrapper”
- Sign/MAC all objects at a trusted source, and deserialize them at a trusted destination **after** signature verification
- Verify the data type and the value of each data member **before** deserialization/object construction
  - Example: Java “resolveClass” is called before any deserialization is done → throw an exception if not as expected
- Verify data type & value of each member **after** deserialization
  - I.e., even if it comes from a trusted source!
  - Not foolproof, but better than nothing
- Performing deserialization in low-privilege processes, and verify the results before using these objects
- Monitoring: any errors, or their frequency (tries by attacker)

Race conditions

Web application synchronisation

- Web applications are multithreaded, as many users may use them simultaneously
  - This does not mean we can ignore it for a single user!
- Example: discount code/voucher
  - Procedure:
    - Check voucher is valid
    - Credit amount to user
    - Invalidate voucher
  - Exploit:
    - Send multiple “cash in voucher” requests simultaneously
    - Multiple credits before being invalidated
- Detection: same as between users / general race conditions
- Prevention:
  - Remember that users can use multiple tabs
  - Every “transaction” must be performed atomic

Web application synchronisation

- A variant of this attack was used to siphon off cryptocoins:
- An application allows you to invest money, and later get it back (e.g. if you are not satisfied)
  - Implemented by Ethereum smart contracts
- Exploit:
  - Send 1 coin
  - Ask for one coin back
    - Subtract the donation from your own account
    - Call “credit” function of donator to give the donation back
    - Unfortunately, the “credit” function did not merely add the coin to its own account, it immediately asked again for a refund…
  - Terminates only after all funds have been exhausted!
    - Validates the no-reentrancy requirement
  - Solution: None. The buggy contract is already on the blockchain and cannot be changed anymore. → Hard fork

Missing Function Level Access Control

Failure to restrict URL access

- Some access protection (e.g. username+password) exists, but „protected“ pages can be accessed by knowing their URL
  - „Secret“ URLs (security by obscurity) are not a protection: the login status must actually be verified!
  - Same applies to different authentication levels: if you are a “normal” user, can you access “administrative” pages when knowing their URL?
- Detection:
  - Spider the complete application with the highest possible permissions and store each URL
  - Try accessing these URLs with all lesser permissions and check that access is denied properly
    - Check for each user/group/role! Authentication alone is insufficient, authorization for this “set of users” must be checked too!
- Similar to insecure direct object reference…

Failure to restrict URL access

- Examples:
  - http://www.vulnerab.le/admin_page
    - Administrative rights should be required for accessing this page
  - Typical: if permissions are lacking, buttons or links to pages are just not shown, but actual access is not checked
- How to prevent this:
  - Use a framework for authentication and authorization
    - Preferably role-based (or: groups…) to reduce administration
      - Design a matrix: Who + What → Allowed/Prohibited
    - Should be in the business logic layer; not presentation alone!
    - Or place check on every single page at the very start
  - Deny all access by default to all pages (except login)
    - Require an explicit configuration to grant access to a page
  - Workflows, form submission…: check at every stage, not only at the first page/form or at rendering the form
    - Form submission: verify that the user is allowed to submit it

Using Known Vulnerable Components

Old components

- Process for updating all software: OS, web server, application server, libraries, framework, DB, application
  - Similarly: process for installing/duplication
- Disable/Remove/Uninstall everything
  - Re-enable only those elements which are actually needed
  - Make sure to understand all security settings
- Check for unused elements:
  - Ports: only open those really needed
  - Pages: only “used” pages should be on the webserver
  - Defaults: passwords, accounts…
- Procedures for closing accounts
  - And plans for what to do with their data
- Try to have development, QA and production environments configured exactly the same

Third-party elements

- Your application (or the framework/CMS you use) is secure, but what if you include “external elements”, e.g. plugins?
- Example: 600 plugins and 722 themes – the most popular third party plugins for Wordpress
  - Result: 25 plugins/themes had at least one vulnerability
    - Mostly automated testing, so not an in-depth assessment!
    - In total 49 exploits were found
  - Most surprising: large number of E-Mail header injections
  - Common problems in these plugins:
    - XSS; mostly search text and contact form fields
    - Referrer headers (and others) unsecured
      - An attacker has full control over all HTTP headers he sends, not only over the HTML content!
  - Others: PHP wrapper allows to use scripts as open proxies
    - Request not “x.zip” (=send local file), but “http://www.other.com/x?y” (retrieves file and sends it → DDoS!)

Akamai: State of the Internet report Q2/2015, https://www.stateoftheinternet.com/

Third-party elements

- Do you really know the source of the plugins?
  - In the study (see reference below) they found a vulnerable plugin, where the company named in the documentation repeatedly stated “It’s not ours”
  - Reason: some inspection (=code) before inclusion, perhaps E-Mail contact, but no further identity verification!
- Effect: this is “anonymous code” created by “someone you don’t know”, who “might be unreachable”, who has written “something” according to “an arbitrary standard of quality”
  - Be careful!

JOHANNES KEPLER
UNIVERSITÄT LINZ

INSTITUTE
OF NETWORKS
AND SECURITY

https://www.ins.jku.at

THANK YOU FOR YOUR ATTENTION!

Michael Sonntag
michael.sonntag@ins.jku.at
+43 (732) 2468 - 4137
S3 235 (Science park 3, 2nd floor)

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at