Data-Driven Software Security Assessment Report

Stall Statements

File **ZOIALibrarian_local.py** contains 2 stall statements:

1. Line number: 888, statement: print(conn, e)
2. Line number: 890, statement: print(conn, e)

Dependency Vulnerabilities

The following dependency has vulnerabilities:

1. urllib3 == 1.0

    - CVE-2020-26137 **CVSS:** 6.4
        - **Name:** Blind SQL Injection
        - **Summary:** Unavailable
        - **Solution:** If possible, upgrade to the latest package release.
          Otherwise:
          Security by Obscurity is not a solution to preventing SQL Injection. Rather than suppress error messages and exceptions, the application must handle them gracefully, returning either a custom error page or redirecting the user to a default page, without revealing any information about the database or the application internals. Strong input validation - All user-controllable input must be validated and filtered for illegal characters as well as SQL content. Keywords such as UNION, SELECT or INSERT must be filtered in addition to characters such as a single-quote(') or SQL-comments (--) based on the context in which they appear.

    - CVE-2021-33503 **CVSS:** 5.0
        - **Name:** XML Entity Expansion
        - **Summary:** Web Logs Tampering attacks involve an attacker injecting, deleting or otherwise tampering with the contents of web logs typically for the purposes of masking other malicious behavior. Additionally, writing malicious data to log files may target jobs, filters, reports, and other agents that process the logs in an asynchronous attack pattern. This pattern of attack is similar to "Log Injection-Tampering-Forging" except that in this case, the attack is targeting the logs of the web server and not the application.
        - **Solution:** If possible, upgrade to the latest package release.
          Otherwise:
          Design: Use libraries and templates that minimize unfiltered input. Use methods that limit entity expansion and throw exceptions on attempted entity expansion. Implementation: Disable altogether the use of inline DTD schemas in your XML parsing objects. If must use DTD, normalize, filter and white list and parse with methods and

routines that will detect entity expansion from untrusted sources.

- CVE-2019-11324 **CVSS:** 5.0
  - **Name:** Creating a Rogue Certification Authority Certificate
  - **Summary:** An adversary exploits a weakness in the MD5 hash algorithm (weak collision resistance) to generate a certificate signing request (CSR) that contains collision blocks in the "to be signed" part. The adversary specially crafts two different, but valid X.509 certificates that when hashed with the MD5 algorithm would yield the same value. The adversary then sends the CSR for one of the certificates to the Certification Authority which uses the MD5 hashing algorithm. That request is completely valid and the Certificate Authority issues an X.509 certificate to the adversary which is signed with its private key. An adversary then takes that signed blob and inserts it into another X.509 certificate that the attacker generated. Due to the MD5 collision, both certificates, though different, hash to the same value and so the signed blob works just as well in the second certificate. The net effect is that the adversary's second X.509 certificate, which the Certification Authority has never seen, is now signed and validated by that Certification Authority. To make the attack more interesting, the second certificate could be not just a regular certificate, but rather itself a signing certificate. Thus the adversary is able to start their own Certification Authority that is anchored in its root of trust in the legitimate Certification Authority that has signed the attackers' first X.509 certificate. If the original Certificate Authority was accepted by default by browsers, so will now the Certificate Authority set up by the adversary and of course any certificates that it signs. So the adversary is now able to generate any SSL certificates to impersonate any web server, and the user's browser will not issue any warning to the victim. This can be used to compromise HTTPS communications and other types of systems where PKI and X.509 certificates may be used (e.g., VPN, IPSec).
  - **Solution:** If possible, upgrade to the latest package release.
    Otherwise:
    Certification Authorities need to stop using the weak collision prone MD5 hashing algorithm to hash the certificates that they are about to sign. Instead they should be using stronger hashing functions such as SHA-256 or SHA-512.

- CVE-2018-20060 **CVSS:** 5.0
  - **Name:** Unavailable
  - **Summary:** Blind SQL Injection results from an insufficient mitigation for SQL Injection. Although suppressing database error messages are considered best practice, the suppression alone is not sufficient to prevent SQL Injection. Blind SQL Injection is a form of SQL Injection that overcomes the lack of error messages. Without the error messages that facilitate SQL Injection, the adversary constructs input strings that probe the target through simple Boolean SQL expressions. The adversary can determine if the syntax and structure

of the injection was successful based on whether the query was executed or not. Applied iteratively, the adversary determines how and where the target is vulnerable to SQL Injection.
    - **Solution:** If possible, upgrade to the latest package release.

- CVE-2019-11236 **CVSS:** 4.3
    - **Name:** Web Logs Tampering
    - **Summary:** An attacker submits an XML document to a target application where the XML document uses nested entity expansion to produce an excessively large output XML. XML allows the definition of macro-like structures that can be used to simplify the creation of complex structures. However, this capability can be abused to create excessive demands on a processor's CPU and memory. A small number of nested expansions can result in an exponential growth in demands on memory.
    - **Solution:** If possible, upgrade to the latest package release.
      Otherwise:
      Design: Use input validation before writing to web log Design: Validate all log data before it is output