report3

report3

Deprecated Interface Vulnerability Report for File Upload on Complaint Page

1. **Executive Summary**

- **Vulnerability Title**: Deprecated Interface and Improper File Type Validation in File Upload
- **Affected Component**: File Upload Functionality on Complaint Page (`http://localhost:3000/#/complain`)
- **Severity**: High
- **Impact**: The application allows the upload of files with deprecated and less secure file types (e.g., `.xml`), bypassing restrictions that should only permit `.pdf` and `.zip` files. This could lead to security issues such as XML External Entity (XXE) attacks, local file inclusions (LFI), or other vulnerabilities.

Summary:

The file upload functionality on the Complaint page (`http://localhost:3000/#/complain`) restricts users to uploading `.pdf` or `.zip` files. However, by examining the source code (`main.js`), it was discovered that deprecated file types, such as `.xml`, are still allowed. This discrepancy between the interface and the backend logic allows attackers to bypass the file type restrictions and upload files in insecure formats, potentially leading to further exploits.

2. **Vulnerability Details**

2.1 **Description**:

On the Complaint page, users are informed that only `.pdf` and `.zip` file types are allowed for upload. However, inspecting the `main.js` code reveals that several other file types are accepted, including `.xml`. The allowed MIME types in the source code are:

```
allowedMimeType: [
    'application/pdf',
    'application/xml',
    'text/xml',
    'application/zip',
    'application/x-zip-compressed',
    'multipart/x-zip'
]
```

This indicates that XML files, which could be used in attacks such as XML External Entity (XXE) or local file inclusion (LFI), are also permitted despite the UI restrictions. This improper validation creates a security gap where an attacker can upload malicious `.xml` files and exploit the system.

2.2 **Steps to Reproduce**:

1. Go to the Complaint page: `http://localhost:3000/#/complain`.
2. Attempt to upload a file with an extension other than `.pdf` or `.zip`, such as an `.xml` file.
3. Despite the page informing you that only `.pdf` and `.zip` files are allowed, the `.xml` file uploads successfully.
4. Review the uploaded file and verify that no restriction is applied to `.xml` files.

2.3 **Impact**:

Allowing deprecated file types such as `.xml` can result in several security issues:

- **XXE (XML External Entity) Attack**: Malicious XML files can exploit vulnerable XML parsers to access sensitive server files or exfiltrate data.
- **File Inclusion Attacks**: Improper handling of uploaded files could lead to local or remote file inclusion vulnerabilities.
- **Inconsistent Security Policy**: Discrepancies between the frontend (UI) and backend (business logic) create confusion and weaken the overall security of the application.

---

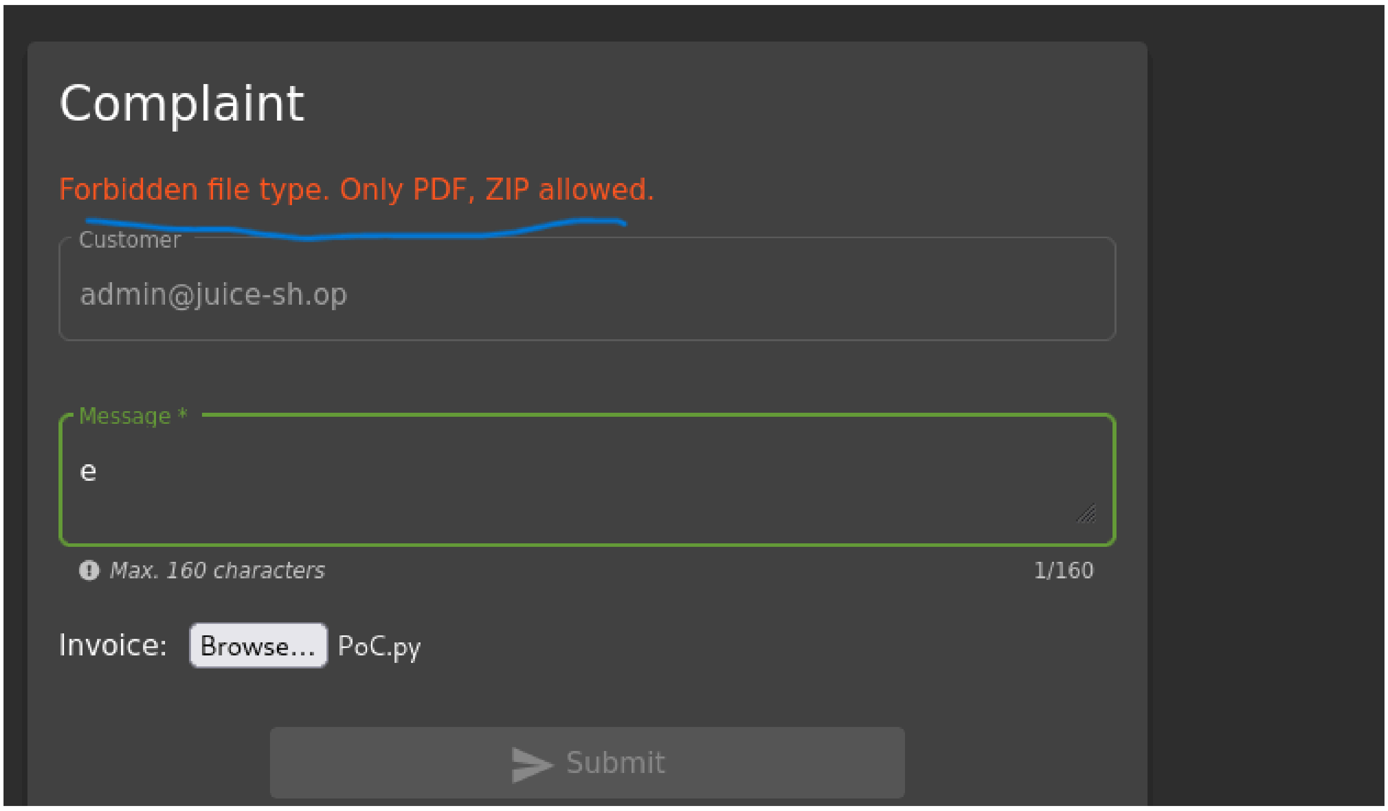
3. **Technical Details**

3.1 **Exploitability**:

The vulnerability is easily exploitable by anyone with access to the file upload feature on the Complaint page. Since the application allows `.xml` file types, an attacker can create and upload a malicious XML file and leverage vulnerabilities in the server-side handling of these files to compromise the system.

3.2 **Proof of Concept (PoC)**:

- **UI Limitation**: The UI claims that only `.pdf` and `.zip` files are allowed.
- **Source Code**: The following MIME types are allowed in `main.js`:



Complaint

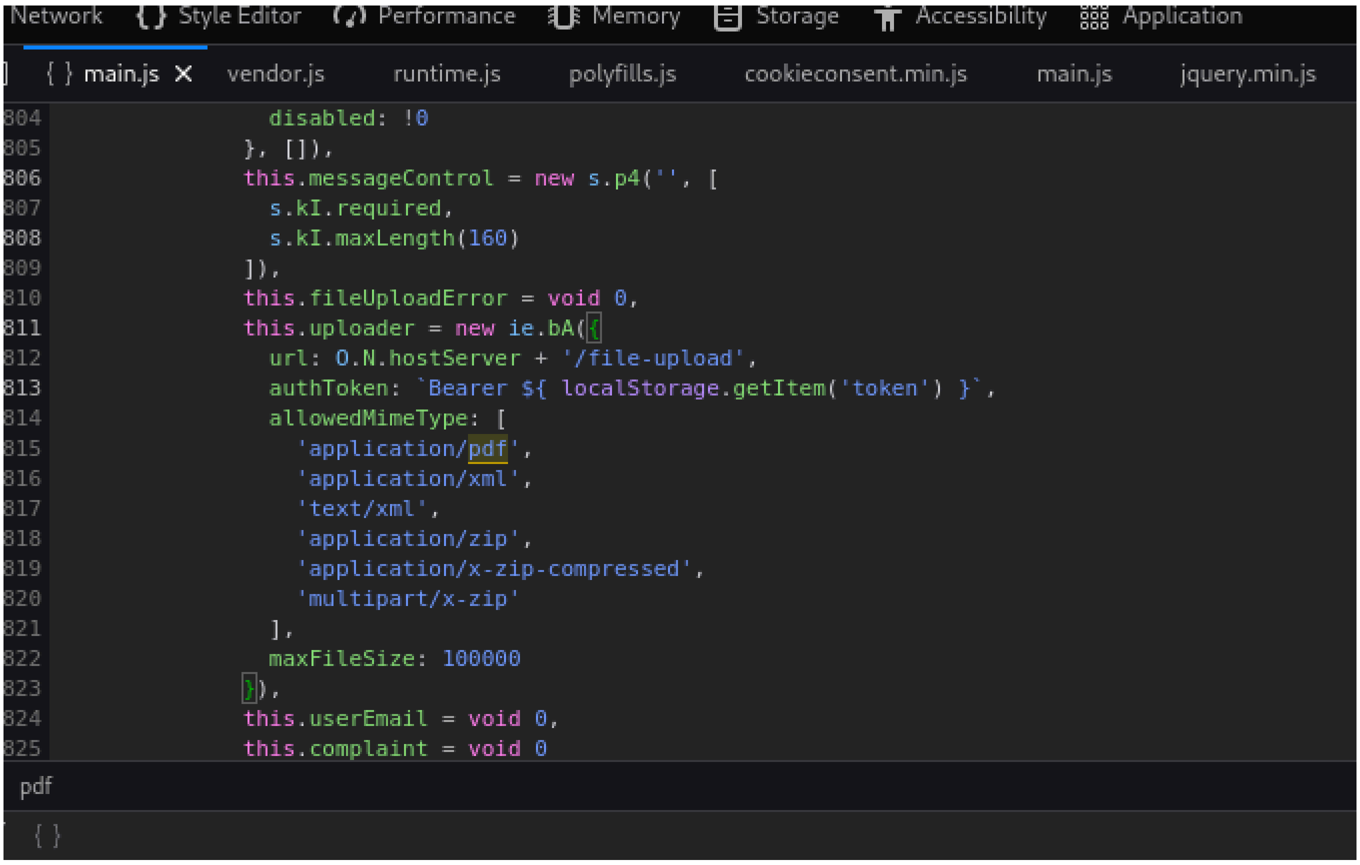
Forbidden file type. Only PDF, ZIP allowed.

Customer
admin@juice-sh.op

Message *
e

Max. 160 characters 1/160

Invoice: Browse... PoC.py



```
                disabled: !0
              }, []),
              this.messageControl = new s.p4('', [
                s.kI.required,
                s.kI.maxLength(160)
              ]),
              this.fileUploadError = void 0,
              this.uploader = new ie.bA({
                url: O.N.hostServer + '/file-upload',
                authToken: `Bearer ${ localStorage.getItem('token') }`,
                allowedMimeType: [
                  'application/pdf',
                  'application/xml',
                  'text/xml',
                  'application/zip',
                  'application/x-zip-compressed',
                  'multipart/x-zip'
                ],
                maxFileSize: 100000
              }),
              this.userEmail = void 0,
              this.complaint = void 0
```

3.3 **Root Cause Analysis**:

The vulnerability arises from the discrepancy between the allowed file types displayed to the user and the actual allowed file types in the backend code. The UI implies that only `.pdf` and `.zip` files are allowed, but the backend logic permits a broader range of file types, including insecure formats like `.xml`. The absence of proper validation and filtering of file types creates a security risk.

4. Mitigation Recommendations

4.1 **Short-Term Fix**:

- Immediately update the file upload validation to ensure that only `.pdf` and `.zip` files are allowed both on the frontend and backend. Remove any deprecated or insecure file types (e.g., `.xml`) from the `allowedMimeType` array in the `main.js` file.

  Example Code Fix:

```
allowedMimeType: [
  'application/pdf',
  'application/zip',
  'application/x-zip-compressed',
  'multipart/x-zip'
]
```

- Implement file content validation on the server side to ensure that uploaded files match their declared MIME types.

4.2 **Long-Term Fix**:

- Implement a robust content validation process that verifies the contents of uploaded files, ensuring they match their expected formats (e.g., ensuring that PDFs are actual PDFs).
- Configure the server to sanitize and validate all uploaded files properly. Ensure any parsing of XML or other complex formats is handled securely.
- Disable the processing of XML files entirely if they are not necessary for the application's functionality.
- Perform regular security audits of the application's file upload functionality to ensure no deprecated or insecure interfaces are exposed.

---

5. Risk Assessment

5.1 **Likelihood**:

- **Likelihood**: High
  The vulnerability is exploitable by anyone with access to the file upload feature, and since XML files are accepted without validation, attackers can easily upload malicious payloads.

5.2 **Impact**:

- **Impact**: High
  Allowing deprecated file types such as `.xml` opens up the possibility of critical attacks like XXE and LFI. Successful exploitation could lead to the compromise of sensitive data, access to internal files, or remote code execution.

Deprecated Interface Vulnerability Report for File Upload on Complaint Page

1. Executive Summary

- **Vulnerability Title**: Deprecated Interface and Improper File Type Validation in File Upload
- **Affected Component**: File Upload Functionality on Complaint Page (`http://localhost:3000/#/complain`)
- **Severity**: High
- **Impact**: The application allows the upload of files with deprecated and less secure file types (e.g., `.xml`), bypassing restrictions that should only permit `.pdf` and `.zip` files. This could lead to security issues such as XML External Entity (XXE) attacks, local file inclusions (LFI), or other vulnerabilities.

Summary:

The file upload functionality on the Complaint page (`http://localhost:3000/#/complain`) restricts users to uploading `.pdf` or `.zip` files. However, by examining the source code (`main.js`), it was discovered that deprecated file types, such as `.xml`, are still allowed. This discrepancy between the interface and the backend logic allows attackers to bypass the file type restrictions and upload files in insecure formats, potentially leading to further exploits.

2. Vulnerability Details

2.1 Description:

On the Complaint page, users are informed that only `.pdf` and `.zip` file types are allowed for upload. However, inspecting the `main.js` code reveals that several other file types are accepted, including `.xml`. The allowed MIME types in the source code are:

javascript

Copy code

```
allowedMimeType: [ 'application/pdf', 'application/xml', 'text/xml',
'application/zip', 'application/x-zip-compressed', 'multipart/x-zip']
```

This indicates that XML files, which could be used in attacks such as XML External Entity (XXE) or local file inclusion (LFI), are also permitted despite the UI restrictions. This improper validation creates a security gap where an attacker can upload malicious `.xml` files and exploit the system.

2.2 Steps to Reproduce:

1. Go to the Complaint page: `http://localhost:3000/#/complain`.

2. Attempt to upload a file with an extension other than `.pdf` or `.zip`, such as an `.xml` file.
3. Despite the page informing you that only `.pdf` and `.zip` files are allowed, the `.xml` file uploads successfully.
4. Review the uploaded file and verify that no restriction is applied to `.xml` files.

2.3 **Impact**:

Allowing deprecated file types such as `.xml` can result in several security issues:

- **XXE (XML External Entity) Attack**: Malicious XML files can exploit vulnerable XML parsers to access sensitive server files or exfiltrate data.
- **File Inclusion Attacks**: Improper handling of uploaded files could lead to local or remote file inclusion vulnerabilities.
- **Inconsistent Security Policy**: Discrepancies between the frontend (UI) and backend (business logic) create confusion and weaken the overall security of the application.

---

3. **Technical Details**

3.1 **Exploitability**:

The vulnerability is easily exploitable by anyone with access to the file upload feature on the Complaint page. Since the application allows `.xml` file types, an attacker can create and upload a malicious XML file and leverage vulnerabilities in the server-side handling of these files to compromise the system.

3.2 **Proof of Concept (PoC)**:

- **UI Limitation**: The UI claims that only `.pdf` and `.zip` files are allowed.
- **Source Code**: The following MIME types are allowed in `main.js`:

  javascript

  Copy code

  `allowedMimeType: [ 'application/pdf', 'application/xml', 'text/xml', 'application/zip', 'application/x-zip-compressed', 'multipart/x-zip']`
- **Exploit**: Upload a malicious `.xml` file to trigger an XXE attack.

  Example XXE Payload:

  xml

  Copy code

  `<!DOCTYPE foo [ <!ELEMENT foo ANY > <!ENTITY xxe SYSTEM "file:///etc/passwd" >]> <foo>&xxe;</foo>`
- **Result**: The file is uploaded successfully, and depending on server configuration, the system may process the XML file, triggering the XXE exploit and returning sensitive information (e.g., contents of `/etc/passwd`).

3.3 **Root Cause Analysis**:

The vulnerability arises from the discrepancy between the allowed file types displayed to the user and the actual allowed file types in the backend code. The UI implies that only `.pdf` and `.zip` files are allowed, but the backend logic permits a broader range of file types, including insecure formats like `.xml`. The absence of proper validation and filtering of file types creates a security risk.

---

4. Mitigation Recommendations

4.1 **Short-Term Fix**:

- Immediately update the file upload validation to ensure that only `.pdf` and `.zip` files are allowed both on the frontend and backend. Remove any deprecated or insecure file types (e.g., `.xml`) from the `allowedMimeType` array in the `main.js` file.

  Example Code Fix:

  javascript

  Copy code

  `allowedMimeType: [ 'application/pdf', 'application/zip', 'application/x-zip-compressed', 'multipart/x-zip']`
- Implement file content validation on the server side to ensure that uploaded files match their declared MIME types.

4.2 **Long-Term Fix**:

- Implement a robust content validation process that verifies the contents of uploaded files, ensuring they match their expected formats (e.g., ensuring that PDFs are actual PDFs).
- Configure the server to sanitize and validate all uploaded files properly. Ensure any parsing of XML or other complex formats is handled securely.
- Disable the processing of XML files entirely if they are not necessary for the application's functionality.
- Perform regular security audits of the application's file upload functionality to ensure no deprecated or insecure interfaces are exposed.

---

5. Risk Assessment

5.1 **Likelihood**:

- **Likelihood**: High
  The vulnerability is exploitable by anyone with access to the file upload feature, and since XML files are accepted without validation, attackers can easily upload malicious payloads.

5.2 **Impact**:

- **Impact**: High
  Allowing deprecated file types such as `.xml` opens up the possibility of critical attacks like XXE and LFI. Successful exploitation could lead to the compromise of sensitive data, access to internal files, or remote code execution.

---

6. Conclusion

The deprecated interface vulnerability in the file upload feature of the Complaint page allows attackers to upload insecure file types such as `.xml`, which can be exploited to launch XXE or LFI attacks. To mitigate this issue, it is critical to enforce file type restrictions both on the frontend and backend, validate the contents of uploaded files, and remove any deprecated file types from the application’s allowed MIME types. Prompt action is recommended to prevent potential exploitation.