Postmortem Security Breach Analysis & Reflection:

Technical and Software Security Lessons from the 2023 MOVEit Transfer Exploit

Zachary Brown

CS-3780: Software Security

Nathan Nenninger

University of Missouri–St. Louis (UMSL)

Due Date: December 10th, 2025

I. Case Study Selection

For my postmortem analysis, I chose to examine the 2023 MOVEit Transfer zero-day vulnerability, a supply-chain style incident that rapidly became one of the most consequential software breaches in recent years. I selected MOVEit for two reasons: its clear connection to software security failures, and the unusually broad impact it had across government agencies, financial institutions, healthcare organizations, and global enterprises. Even before researching this topic in depth, I had only briefly heard about MOVEit, but once I reviewed technical analyses, including a detailed briefing from LMG Security, I realized how well this case aligned with the concepts we studied throughout CS-3780.

MOVEit Transfer is marketed as a secure managed file-transfer platform designed to handle highly sensitive data at scale. Ironically, its core functionality, processing and storing confidential files, made the consequences of a software flaw significantly more damaging. On May 27th, 2023, the threat group TA505/Clop began exploiting a previously unknown SQL injection vulnerability (CVE-2023-34362) in MOVEit Transfer's web application layer (CISA, 2023). This flaw allowed attackers to bypass authentication and deploy custom web shells, enabling unrestricted access to databases and sensitive file repositories. As LMG Security noted, this attack affected both on-premises deployments and MOVEit's SaaS cloud platform, meaning organizations relying entirely on a vendor-hosted solution were equally exposed (Davidoff & Durrin, 2023).

The timing of the attack, launched on Memorial Day weekend, was another reason I found this incident compelling. Attackers frequently target holidays, when IT staff are sparse and monitoring gaps widen, and MOVEit became another example of this trend. According to the presentation, Clop had been sitting on a beta version of the exploit since at least April 2022, waiting for an optimal moment to maximize impact (Davidoff & Durrin, 2023). This highlights the strategic dimension of real-world exploitation, where adversaries combine technical flaws with operational timing. What ultimately convinced me to choose MOVEit was the scale of the downstream breaches. For example, Zellis, a major payroll provider, was compromised through MOVEit, which then cascaded into breaches at the BBC, British Airways, and Boots, exposing payroll data for tens of thousands of employees. This demonstrated how a single software vulnerability can ripple across entire industries when trusted third-party tools are involved.

Because MOVEit illustrates failures in input validation, secure coding, web-application hardening, and secure SDLC practices, it aligns naturally with the content of this course. It also gives me an opportunity to analyze how web security principles, dependency management, and incident response all converge when a real vulnerability is exploited at scale. For these reasons, MOVEit is an ideal case study for this assignment.

II. Technical Analysis (MOVEit Transfer / CVE-2023-34362)

In this section, I unpack how the MOVEit Transfer zero-day was actually exploited, which software vulnerabilities were involved, and which secure-coding “deadly sins” it represents.

1. Vulnerabilities Exploited (CVE IDs)

The main flaw exploited in the 2023 attacks was CVE-2023-34362, a critical unauthenticated SQL injection (SQLi) vulnerability in the MOVEit Transfer web application. Progress Software confirmed that the issue affected all supported versions of MOVEit Transfer and allowed attackers to gain direct access to the underlying database over HTTP/HTTPS.

During incident response and follow-on research, additional related SQLi vulnerabilities were disclosed:

- **CVE-2023-35036** and **CVE-2023-35708** – further SQL injection issues in the web UI.
- **CVE-2023-36934** and **CVE-2023-36932** – additional SQLi paths (unauthenticated and authenticated) that could be used to access the MOVEit database.

However, the initial mass exploitation campaign by TA505/Clop in late May 2023 primarily abused **CVE-2023-34362** to deploy a custom web shell and steal data at scale.

2. How the SQL Injection Worked (Input Handling & Query Construction)

At a high level, MOVEit Transfer exposes a .NET web front end (including components like *moveitisapi.dll* and *human.aspx*) that handles user logins, file browsing, and automated transfer workflows. Researchers from Huntress, Akamai, and Imperva all observed that the vulnerable endpoint accepted **user-controlled parameters** which were concatenated into SQL statements without proper sanitization or parameterization.

Conceptually, the vulnerable code looked something like this:

```
// Pseudocode – not the real source

string username = Request["user"];

string query = "SELECT * FROM users WHERE username = '" +
username + "'";

DataTable result = ExecuteQuery(query);
```

Because the application failed to neutralize special characters (quotes, semicolons) and did not use parameterized queries, an attacker could send a crafted HTTP request such as:

```
user='; INSERT INTO ...;--
```

Within MOVEit's real code, the injected payloads were more complex and targeted internal tables that store configuration and credentials. According to Imperva and Unit 42, the attack chain began with SQLi used to retrieve or modify administrative credentials and internal configuration, which then allowed unrestricted file upload or direct placement of malicious files in the webroot.

This is a textbook instance of the secure-coding failure sometimes summarized as *"failing to protect SQL queries from user input"*—one of the classic **"24 Deadly Sins of Software Security"** and closely aligned with OWASP's "Injection" category.

3. From SQLi to Web Shell: LEMURLOOT

Once the database was exposed via SQLi, TA505/Clop used it as a stepping stone to deploy a custom ASP.NET web shell dubbed **LEMURLOOT**.

Key technical details:

- The web shell was typically written as an `.aspx` page and dropped into the MOVEit webroot under names like `human2.aspx`, mimicking the legitimate `human.aspx` component.
- Using the SQLi, attackers could:
  - Write the file directly via database-backed file storage routines, or
  - Abuse admin-level functionality (gained by modifying DB records) to upload the file as if it were a legitimate resource.
- **LEMURLOOT** then provided a remote command-and-control interface that allowed attackers to:
  - Enumerate files and folders in the MOVEit repository
  - Read configuration data, including Azure Blob Storage settings and keys
  - Create or delete MOVEit user accounts
  - Download files in bulk for exfiltration

At this point, the SQLi had effectively been escalated into full remote code execution (RCE) and data-layer compromise via the web shell.

4. Attack Chain: Step-by-Step

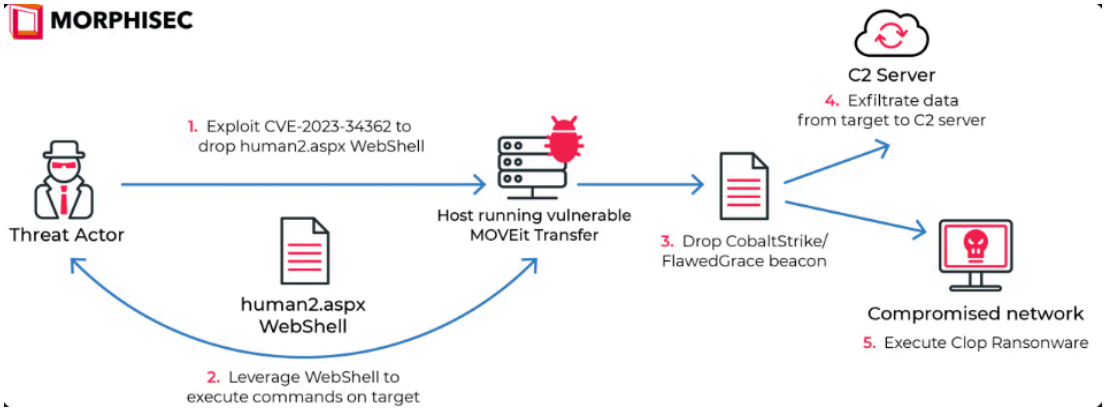
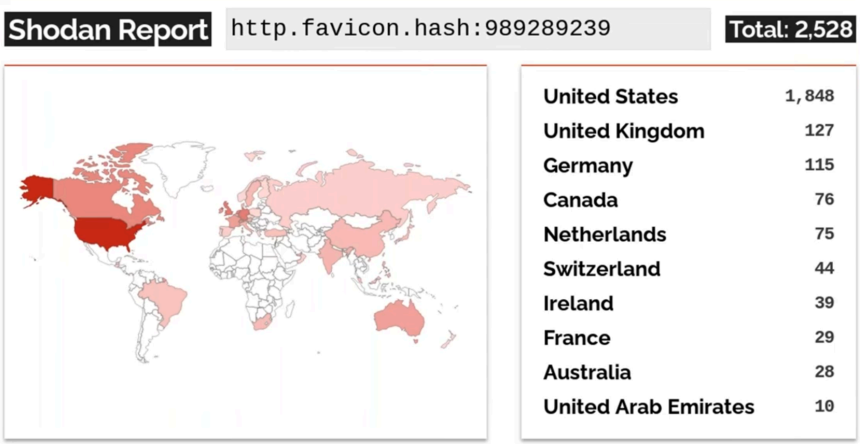


Figure 1: *High-level attack chain for the MOVEit Transfer exploit, showing the sequence from external reconnaissance through SQL injection, backdoor installation, and data exfiltration (Morphisec, 2023).*

Putting the above pieces together, the attack chain looked roughly like this (based on Mandiant, CISA, Huntress, Akamai, and the LMG Security briefing):

- **Reconnaissance & Targeting**
  - Clop scanned the internet (e.g., via Shodan) for publicly exposed MOVEit Transfer servers. LMG Security's scan alone identified ~1,800 exposed servers in the U.S. (Davidoff & Durrin, 2023). This aligns with Akamai's estimate of ~2,500 internet-facing MOVEit instances globally.



- **Initial Exploitation – SQL Injection**
  - Attackers sent specially crafted HTTP(S) requests to vulnerable MOVEit endpoints.

- The malicious parameters were concatenated into SQL queries inside `moveitisapi.dll` or related handlers, resulting in unauthenticated SQL commands being executed against the MOVEit database.

❖ **Privilege Escalation & Configuration Discovery**

- Via SQLi, attackers enumerated the database, extracted MOVEit configuration, and in some cases reset or inserted administrative credentials.

❖ **Web Shell Deployment (LEMURLOOT)**

- Using the obtained privileges or file-handling routines, Clop uploaded the ASP.NET web shell (`human2.aspx`) into the webroot.

- The web shell was reachable via HTTPS, blending in with legitimate traffic.

❖ **Data Enumeration & Exfiltration**

- Through **LEMURLOOT**, attackers listed folders, queued large downloads, and exfiltrated data from MOVEit's local storage and, in some cases, connected Azure Blob Storage containers.

❖ **Extortion & Ransom**

- Instead of immediately deploying ransomware, Clop focused on data theft and extortion. Victims were instructed to contact the group via Tor to negotiate, as described both on Clop's leak site and in the LMG Security video (Davidoff & Durrin, 2023).

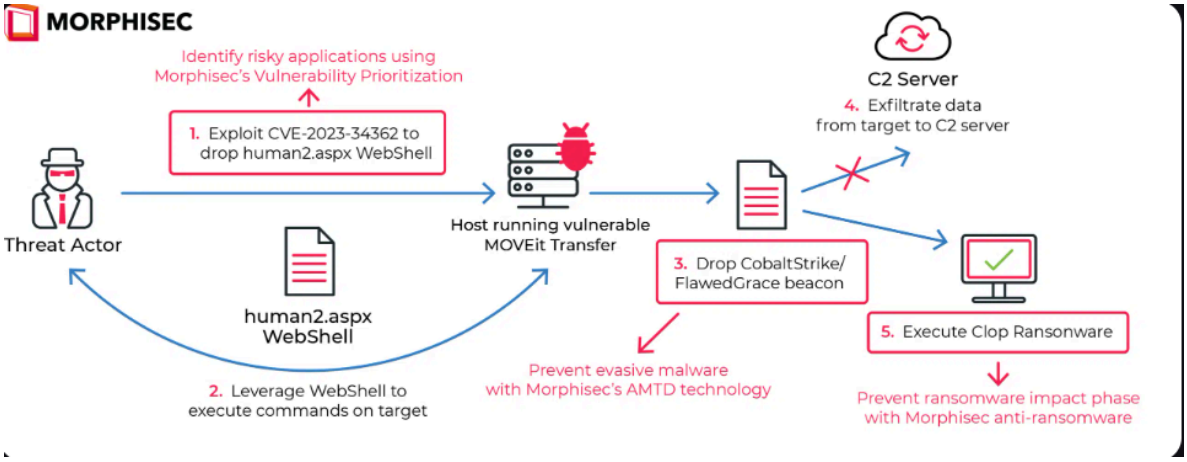


Figure 2: *Stages of the MOVEit Transfer breach, illustrating how threat actors progressed from initial SQL injection to system compromise, privilege escalation, and large-scale data theft (Morphisec, 2023).*

From a software-security standpoint, the most important observation is that every major step depended on the original input-validation failure in the web application. There was no need for buffer overflows, format-string bugs, or low-level memory corruption; the attack lived entirely at the application and database layers.

5. Poor Software Engineering / DevSecOps Practices

Several coding and process failures contributed to the vulnerability and its impact:

- **Lack of Proper Input Validation and Parameterization**
  - The core bug is a failure to treat user input as untrusted and to use parameterized SQL queries or ORM safeguards. This maps directly to OWASP's "Injection" category and one of the *"Deadly Sins"* of insecure database access.
- **Insufficient Threat Modeling & Secure SDLC**
  - MOVEit is explicitly marketed as "secure managed file transfer," yet a critical SQLi in its main web front end suggests that input handling for high-risk endpoints was not systematically threat-modeled or fuzz-tested, especially considering Clop allegedly had a working exploit as early as April 2022 (Davidoff & Durrin, 2023).
- **Exposure of a High-Value Service Directly to the Internet**
  - Many organizations ran MOVEit as a public-facing service, sometimes without compensating controls (WAF rules, strict IP allow-lists). Once SQLi existed, the blast radius was huge because the service sat directly on the perimeter handling sensitive data (Morphisec, 2023).
- **Dependency & Patch Management Gaps (on the customer side)**
  - Although Progress Software released patches quickly once the issue became public, many organizations needed days or weeks to identify where MOVEit was running (often at third-party suppliers) and to patch or isolate it, which allowed Clop to expand their victim list (Progress Software, 2023)
- **Logging & Forensics Limitations**
  - Several analyses note that, without advanced observability or WAF logs, it was difficult for victims to reconstruct exactly what SQL payloads were sent or which data was taken. This highlights another *"Deadly Sin"*: *insufficient logging and monitoring* for critical security events.

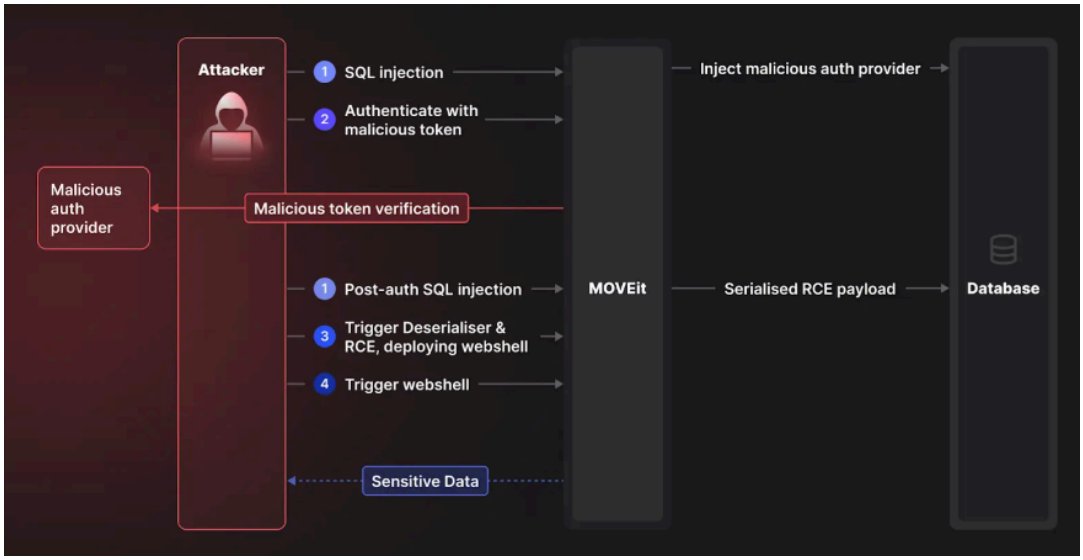


Figure 3: *Visual depiction of the four MOVEit Transfer vulnerabilities forming the chained exploit path for CVE-2023-34362, including the SQL injection entry point and subsequent stages of compromise (Miggo, 2023).*

III. Course Connection & Prevention (MOVEit Breach Through the Lens of CS-3780)

The MOVEit Transfer exploitation campaign provides a near-perfect case study for many of the core principles taught in CS-3780. Although the breach did not depend on low-level memory corruption or cryptographic failures, it demonstrates how weaknesses in input validation, database interaction, deserialization, authentication logic, and secure SDLC practices can be just as devastating as classic buffer overruns. By mapping MOVEit's failure points to course modules and tools, it becomes clear that many elements of this attack were preventable with standard secure coding practices and the systematic application of developer-side security testing.

1. Web Vulnerabilities & Input Validation (Weeks 2–3)

The primary entry point for CVE-2023-34362 was a textbook SQL injection flaw, one of the earliest and most persistent topics in secure coding. SQLi occurs when untrusted user input is concatenated directly into a query, violating fundamental principles of input validation and output encoding. MOVEit's vulnerable endpoint accepted parameters through the web interface and passed them directly to the underlying database layer without using parameterized queries, prepared statements, or stored procedures, all defenses emphasized in Week 2 and Week 3 of the course.

This flaw directly maps to the *"Deadly Sin"* of *Failing to Protect SQL Queries from User Input,* and also aligns with the OWASP Top 10 category of Injection attacks. Modern development stacks (such as .NET, Java, PHP, Python, or Node.js) all provide safe abstractions that prevent

SQLi, meaning the MOVEit vulnerability reflects a lapse not only in coding discipline but also in code review, threat modeling, and API design. SQLi is one of the most preventable vulnerabilities in software engineering, which underscores the severity of this oversight (Progress Software, 2023; Mandiant, 2023).

2. Deserialization, Token Handling & Application Logic Flaws (Weeks 2–5)

The Miggo diagram *(Figure 3)* highlights a deeper issue: after exploiting SQLi, attackers injected parameters that interfered with MOVEit's authentication token verification. This suggests weaknesses in server-side deserialization logic, allowing a malicious token to bypass the normal authentication workflow. Week 4 and 5 discussions on unsafe deserialization and insecure data structures directly apply here.

Once a malicious authentication provider was introduced (again via SQLi), the server trusted attacker-supplied identity material. The vulnerability was not merely that SQL could be injected, but that the application logic itself allowed trust relationships to be modified at runtime, something secure SDLC methodologies strongly warn against.

This connects to several secure-coding sins, including:

- ➢ Improper Access Control
- ➢ Failing to Validate All Inputs
- ➢ Trusting Data from the Client Without Verification

3. Memory Tools, Debuggers & Fuzzing (Weeks 6–7)

Even though the MOVEit flaw was not a buffer overflow or heap corruption issue, many of the tools used in Weeks 6–7, like fuzzers, dynamic analysis tools, taint tracking, and sanitizers, could have discovered the SQL injection path before release.

Fuzzing techniques are especially relevant. A structured web fuzzer sending malformed parameters, unexpected special characters, or SQL meta-syntax to MOVEit endpoints would likely have triggered anomalous SQL behavior. Tools like:

- ❖ Burp Suite Intruder
- ❖ OWASP ZAP Fuzzer
- ❖ AFLNet (protocol fuzzing)
- ❖ .NET Security CodeScan

- Microsoft's SDL Input Fuzzers

could have produced evidence of exploitable behavior even without full source-code access.

Had MOVEit undergone the secure SDL stages emphasized in class, particularly threat modeling, code review, and fuzz testing, the SQL injection would likely have been discovered during development rather than by a criminal group.

4. Cryptography & Key Management (Week 9)

While cryptographic primitives were not broken during the breach, the attack bypassed cryptography entirely by gaining administrative-level access to MOVEit's data stores. This is a key lesson from Week 9: cryptography cannot protect data if the application's access control and input validation layers are compromised.

Furthermore, MOVEit's integration with Azure Blob Storage meant the SQLi allowed attackers to retrieve API keys and blob connection strings stored in its configuration tables. This directly ties to secure key-management principles taught in Week 9, reinforcing that secrets must:

- be hardened through least privilege
- use isolated secret vaulting
- never be retrievable in plaintext even by the main application

5. Network Concurrency, Sessions & Thread Safety (Weeks 10–12)

The breach also touches on topics from Weeks 10–12 regarding:

- session handling
- thread safety
- race conditions in distributed systems

Once the web shell (*human2.aspx*) was deployed, attackers interacted with MOVEit as a privileged "user," leveraging APIs that were never designed to be invoked outside the normal session lifecycle. This exposed weaknesses in session authorization checks. For example; MOVEit did not require re-authentication for certain administrative file-access operations once the shell was running.

Additionally, because MOVEit is a multi-threaded, asynchronous web application, insecure internal assumptions about the order of operations (e.g., trusting a database write that inserted a malicious identity provider) enabled attackers to chain SQLi into full RCE. This aligns with the Week 12 concept that concurrency bugs are not limited to low-level languages, they can also occur at the application logic level when developers assume serialized execution where none exists.

6. Secure SDLC, Logging & Detection (Weeks 13–14)

Perhaps the most important connection to the course is the failure of **secure SDLC practices**, including:

- Threat modeling
- Automated secure code scanning
- Strict code reviews
- Logging & observability
- Patch readiness and emergency response workflows

The LMG Security briefing stresses that many organizations were unable to determine whether data was taken because logging and forensics data were missing or incomplete (Davidoff & Durrin, 2023). Week 14's emphasis on log integrity, combined with SIEM, WAF logs, and application telemetry, would have significantly reduced investigation time.

Furthermore, the attack illustrates the "usable security vs. secure security" tradeoff. MOVEit's design focused on creating an easy-to-use managed file transfer solution, but usability decisions such as:

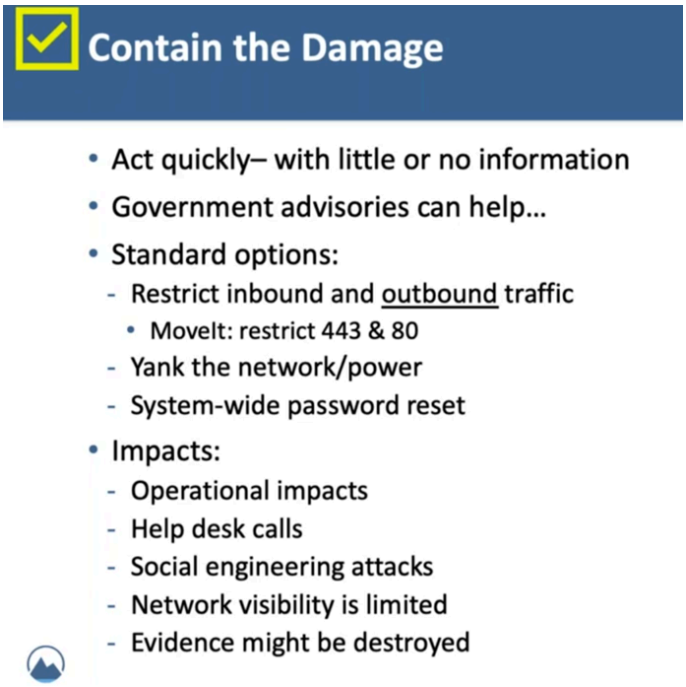
- broad administrative privileges
- exposed public-facing endpoints
- insufficient segregation of configuration data

all created opportunities for attackers once a single input-handling flaw was found.

7. How Course Tools Could Have Prevented the MOVEit Breach

Based on CS-3780 techniques, several prevention strategies map directly to the vulnerability:

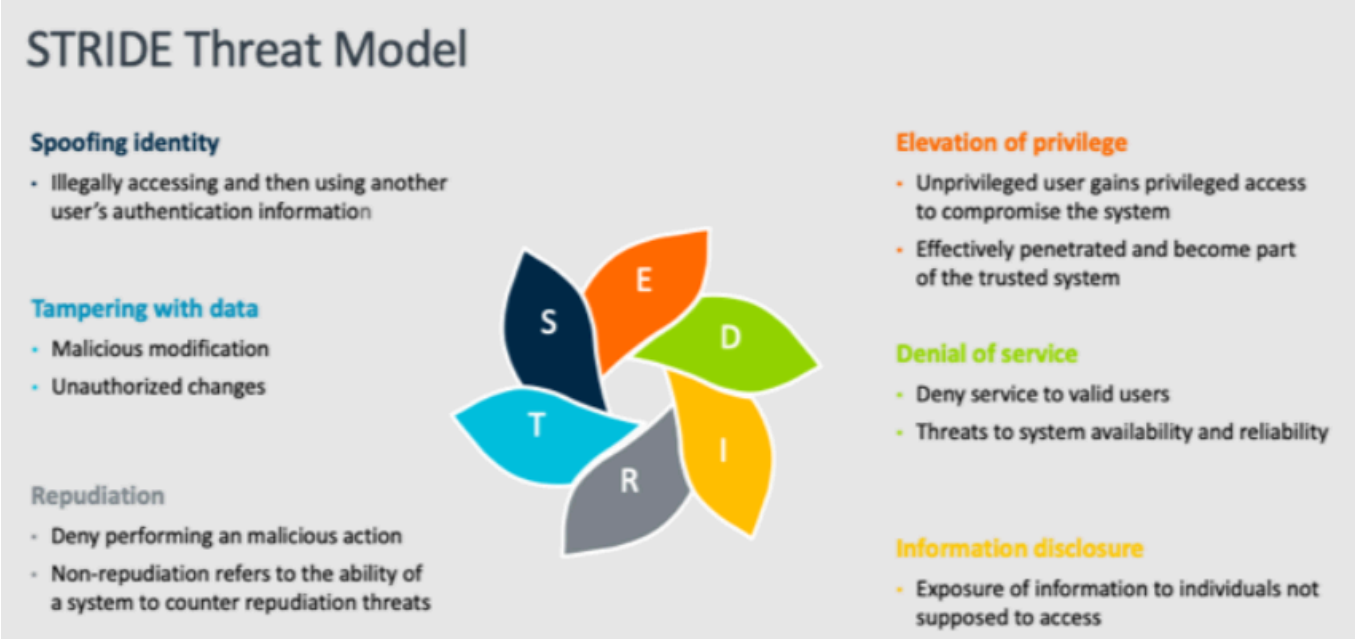
[Course Topic/Tool]	[How It Would Have Helped]
Static Code Analysis	Would flag dynamic SQL string concatenation.
Dynamic Application Security Testing (DAST)	Automated SQL injection detection at runtime.
Fuzzing (Burp/ZAP)	Would discover SQLi immediately.
Threat Modeling (STRIDE)	Identifies SQLi under “Tampering” and “Elevation of Privilege.”
Secure Logging & Monitoring	Enables rapid detection of abnormal SQL queries or shell access.
Principle of Least Privilege	Prevents SQLi from escalating into full administrative access.
Secret Vaulting	Protects database/API keys even if SQLi occurs.



8. Lightweight Threat Model (STRIDE)

To connect directly to Week 13's SDLC content, here is a simplified STRIDE mapping:

[STRIDE Category]	[MOVEit Example]
S – Spoofing	Attacker injects malicious identity provider and logs in as trusted user.
T – Tampering	SQLi modifies database tables and authentication workflows.
R – Repudiation	Lack of adequate logs hindered attribution and impact assessment.
I – Information Disclosure	Sensitive data in MOVEit storage and Azure Blob containers exfiltrated.
D – Denial of Service	Not the main goal, but orgs had to shut servers down to contain damage.
E – Elevation of Privilege	SQLi → RCE → administrative control → data theft.



IV. Broader Lessons & Impact

The MOVEit Transfer breach has had far-reaching consequences for the software industry, government agencies, and the broader cybersecurity community. Because the vulnerability affected organizations across finance, healthcare, aviation, state governments, and third-party service providers, the scale and operational disruption forced the industry to confront long-standing weaknesses in secure software development, supply chain dependencies, and incident response readiness. One of the most significant lessons is that secure coding failures in widely used commercial products can become global cybersecurity events, with ripple effects that extend far beyond the vendor responsible for the software.

First, the MOVEit incident reinforced the ongoing reality that supply chain vulnerabilities are among the highest-impact threats in modern computing. Many victims were not attacked directly but were compromised because their payroll providers, file-transfer vendors, or downstream partners used MOVEit Transfer. This mirrors the systemic risk observed in incidents like SolarWinds and Log4Shell, demonstrating that organizations must evaluate the security posture of all third-party software and services, not just their own systems. Both the U.S. Cybersecurity and Infrastructure Security Agency (CISA) and multiple national CERT teams issued emergency advisories urging organizations to strengthen third-party risk management and to adopt continuous monitoring approaches rather than relying on annual vendor questionnaires (CISA, 2023).

Another major industry impact was the renewed pressure to adopt Secure Software Development Frameworks (SSDF), such as NIST's SP 800-218. These frameworks emphasize secure design, threat modeling, code review requirements, and automated testing pipelines, especially for high-risk systems that process sensitive data. The widespread exploitation of CVE-2023-34362 showed that even mature enterprise software can still harbor SQL injection vulnerabilities, one of the oldest and most thoroughly documented categories of software flaws. This failure accelerated conversations across the industry about mandating secure coding standards and requiring software vendors to demonstrate compliance with secure SDLC practices. Progress Software released multiple post-incident statements outlining plans to expand internal security testing, code audits, and independent assessments, indicating that vendor accountability is becoming a market expectation rather than an optional practice.

The breach also had a profound impact on organizational incident response strategy. MOVEit customers were forced to immediately shut down critical file-transfer systems, pivot to manual processes, and navigate a chaotic environment where Clop publicly threatened victims, demanded self-reporting, and leaked data online. These events reaffirmed the importance of maintaining robust logging, audit trails, and offline forensic copies, practices frequently emphasized in software security courses but often underprioritized in real-world deployments. The difficulty many victims experienced while determining what data was stolen revealed the operational cost of insufficient observability and incomplete logging. As a result, more organizations have begun investing in application-level telemetry, web-application firewalls (WAF), and continuous monitoring tools to detect anomalous behavior early.

Furthermore, the MOVEit breach highlighted the evolving business model of cybercriminal groups, who increasingly prefer data-theft-and-extortion attacks over traditional ransomware encryption. Clop's strategy, exfiltrate first, extort second, shifted the industry's understanding of modern threat actor incentives. This has influenced both cybersecurity insurance policies and regulatory discussions, as governments worldwide explore requirements for timely breach disclosure, vendor transparency, and stronger reporting obligations during active exploitation events.

Lastly, the incident shifted expectations for public-private threat intelligence collaboration. Researchers at Mandiant, Huntress, and LMG Security provided rapid analysis, detection signatures, and defensive guidance, enabling thousands of organizations to respond faster than they otherwise could have. The MOVEit exploitation campaign demonstrated that no single organization can defend itself in isolation; security now depends on open sharing of indicators, telemetry, and remediation steps across the entire ecosystem.

V. Personal Reflection

As I worked through this case study and connected the MOVEit breach to the topics covered in CS-3780, I realized how much this course has reshaped the way I think about software and the responsibility that comes with building and securing it. The most eye-opening part of the class for me was understanding how seemingly small coding decisions, like how a query is constructed or how input is validated, can escalate into global security failures. Before this course, I viewed cybersecurity issues as mostly "network problems," but now I see how deeply they originate within the software itself. SQL injection, insecure deserialization, improper access control, and weak session handling are no longer abstract terms to me; they are real entry points that attackers exploit, and MOVEit was a perfect example of how devastating the consequences can be when they are overlooked.

This course also changed how I approach software design, testing, and debugging. I used to focus primarily on whether something "worked," meaning whether it produced the correct output. Now I understand that functionality alone is not enough, software must be safe under hostile conditions, resilient to malformed input, and defensively built so that a single oversight cannot cascade into a complete compromise. I now appreciate the purpose of fuzzing, secure SDLC processes, static analysis tools, and threat modeling sessions. What used to feel like extra steps now seem essential safeguards that every real project should include.

In terms of my future, this course reinforced my desire to move deeper into cybersecurity, but with a focus on *cloud security.* I'm fascinated by how cloud platforms introduce both new opportunities and new risks, especially when vulnerabilities like MOVEit show how third-party services and misconfigurations can compromise entire organizations. At the same time, I won't deny that this field can feel overwhelming. The mistakes we studied, such as the SQL injection flaw in MOVEit, were not small oversights. They were failures that affected

millions of people, disrupted critical services, and caused financial and reputational damage across multiple industries. It's intimidating to think about the responsibility that comes with writing secure code, and the fear of being the person who accidentally creates a vulnerability is something I'm taking seriously.

However, I am also learning that fear can be productive. It keeps me aware, motivates me to follow secure coding practices carefully, and pushes me to ask the right questions instead of assuming things are safe. This course showed me that developing secure software at scale is incredibly challenging, not because developers lack skill, but because modern systems are complex, interconnected, and constantly evolving. But with structured processes, continuous learning, and a mindset that prioritizes security from the start, those challenges become manageable. This is the mindset I plan to carry with me as I move further into cybersecurity and cloud security as a professional career path.

References/Sources:

Davidoff, S., & Durrin, M. (2023). *MOVEit Vulnerability Deep Dive*. LMG Security. YouTube. https://www.youtube.com/watch?v=XJ_RDexC6zg

CISA. (2023). *Progress MOVEit Transfer vulnerability (CVE-2023-34362).* Cybersecurity and Infrastructure Security Agency. https://www.cisa.gov/news-events/alerts/2023/06/01/progress-moveit-transfer-vulnerability

Huntress. (2023). *MOVEit Transfer zero-day exploited in the wild (CVE-2023-34362) – Technical analysis.* Huntress Labs. https://www.huntress.com/blog/moveit-transfer-critical-zero-day-exploited-in-the-wild

Mandiant. (2023). *CL0P exploitation of MOVEit Transfer vulnerability: SQLi to mass data theft.* Mandiant Threat Intelligence. https://www.mandiant.com/resources/blog/clop-ransomware-moveit-transfer-exploited

Microsoft Security Threat Intelligence. (2023). *Threat actor Lace Tempest exploits MOVEit Transfer vulnerability.* Microsoft. https://www.microsoft.com/security/blog

Miggo. (2023). *How to detect the MOVEit breach with OpenTelemetry.* https://miggo.io/post/how-to-detect-the-moveit-breach-with-opentelemetry

Morphisec. (2023). *How to protect against the MOVEit Transfer exploit.* Morphisec Cybersecurity Blog. https://www.morphisec.com/blog/how-to-protect-against-the-moveit-transfer-exploit

Progress Software. (2023). *MOVEit Transfer security advisory: CVE-2023-34362, CVE-2023-35036, CVE-2023-35708.* Progress Software Corporation. https://www.progress.com/security/moveit-transfer

U.S. Cybersecurity & Infrastructure Security Agency. (2022). *Secure Software Development Framework (SSDF), Version 1.1.* NIST SP 800-218. https://csrc.nist.gov/publications/detail/sp/800-218/final