КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (31), 2025
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2025.31.1040
УДК 004.056.5

Гавриляк Володимир Романович
Аспірант кафедри захисту інформації
Національний університет "Львівська політехніка", Львів, Україна
ORCID: 0009-0002-4114-1232
volodymyr.r.havryliak@lpnu.ua

ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ ПІДВИЩЕННЯ БЕЗПЕКИ ПРОТОКОЛУ GIT LFS

Анотація. У сучасних умовах стрімкого розвитку практик DevOps та MLOps система контролю версій Git стала стандартом індустрії, охоплюючи понад 90% ринку розробки. Зростання обсягів використання технологій машинного навчання та необхідність версіонування великих бінарних об'єктів (ML-моделей, датасетів) зумовили масове впровадження розширення Git Large File Storage (LFS). Однак, специфічна архітектура цього протоколу, що базується на відокремленні метаданих від вмісту файлів, створює нову, критичну поверхню атак, яка залишається недостатньо дослідженою в контексті безпеки ланцюгів постачання програмного забезпечення (Software Supply Chain Security). З огляду на зазначену проблематику, метою роботи обрано системний аналіз архітектурних вразливостей протоколу Git LFS та розробку комплексу практичних рекомендацій щодо підвищення рівня захищеності інфраструктури. Для досягнення цієї мети застосовано методи моделювання загроз та експериментального тестування у контрольованому середовищі, зокрема під час міграції даних між GitLab та GitHub, а також детально проаналізовано специфікацію Batch API та механізми гібридної автентифікації. За результатами дослідження експериментально підтверджено, що використання гібридної моделі транспорту (SSH для Git, HTTPS для LFS) створює проблему "розриву контексту безпеки", що ускладнює валідацію доступу. На основі аналізу ідентифіковано три групи критичних вразливостей: порушення конфіденційності через недостатню перевірку приналежності об'єктів, загрози цілісності даних через можливість підміни вмісту файлів ("отруєння" кешу) та ризики доступності через атаки на виснаження квот (Quota-based DoS). Зокрема встановлено, що відсутність перевірки хеш-сум на стороні хмарних сховищ дозволяє впровадження шкідливого коду в ML-моделі, оминаючи стандартні засоби захисту. Узагальнюючи отримані результати, у роботі систематизовано вектори атак на інфраструктуру LFS та адаптовано міжнародні практики безпеки, впровадження яких є обов'язковим для захисту LFS-серверів. Як ключові заходи запропоновано перехід до моделі "Verify-before-Write", заборону використання ключів розгортання (deploy keys) для запису даних та впровадження суворої атрибуції квот до поточного репозиторію.

Ключові слова: Git; Сховище великих файлів; DevSecOps; вразливості передачі даних.

ВСТУП

Стрімкий розвиток практик циклу розробки програмного забезпечення (DevOps) та інтеграція процесів машинного навчання (MLOps) докорінно змінили вимоги до систем контролю версій. Система Git, яка стала стандартом індустрії (згідно з результатами Stack Overflow Developer Survey, частка активних користувачів Git досягла 93,87% [1]), спочатку проєктувалася для роботи з текстовими файлами вихідного коду, що робить її не ефективною для обробки великих бінарних об'єктів (графічні ресурси, моделі штучного інтелекту, великі датасети). Впровадження розширеного інструменту Git LFS дозволдило вирішити проблему продуктивності шляхом відокремлення зберігання метаданих від вмісту файлів. Однак широке застосування цієї технології у критичну

інфраструктуру створило нову поверхню атак, котра досі залишається поза чіткою увагою кібербезпеки.

Постановка проблеми. Архітектура протоколу Git LFS передбачає складну взаємодію між клієнтом, Git-сервером та LFS-сховищем, часто використовуючи окремі механізми автентифікації та авторизації. На практиці це призводить до ситуації, коли налаштування безпеки “за замовчуванням” не відповідають сучасним вимогам захисту даних. Ключова проблема полягає полягає у відсутності сформованих стандартів безпеки для LFS-серверів.

Більшість реалізованих варіантів вразливі до маніпуляцій з токенами доступу, підробки об’єктів та атаки на відмову ресурсів. Таким чином, актуальним завданням є не лише виявлення вразливостей, а й розробка комплексних методів захисту інфраструктури, що використовує Git LFS.

Аналіз останніх досліджень і публікацій. Загальний аналіз міжнародних наукових публікацій свідчить про значний науковий інтерес до питань безпеки екосистеми Git. Спираючись на роботи останніх років, фокус змістився з безпеки окремих компонентів на комплексний захист ланцюгів постачання програмного забезпечення, зокрема у дослідженні P. Ladisa «SoK: Taxonomy of Attacks on Open-Source Software Supply Chains» [2] було систематизовано атаки на репозиторії, визначивши системи контролю версій як одну з ключових вразливих ланок. Ця тенденція підтверджується і галузевими звітами, такими як Global DevSecOps Report, які фіксують зростання інцидентів безпеки у CI/CD середовищах [3].

Проте, незважаючи на активне вивчення екосистеми DevSecOps, специфіка безпеки протоколу Git LFS залишається "сліпою плямою". Більшість сучасних робіт розглядають Git лише як транспорт для коду, ігноруючи особливості обробки великих бінарних даних. Єдиним винятком та фундаментальною працею є дослідження 2025 року авторства Y. Chen "Unveiling Security Vulnerabilities in Git Large File Storage Protocol" [4]. Автори вперше деталізували вектори атак, специфічні саме для LFS-архітектури. Водночас в українському науковому просторі ця проблематика фактично не висвітлена. Відсутні роботи, які б адаптували зазначені міжнародні висновки або пропонували б дієві механізми захисту Git LFS. Це створює прогалину в знаннях, яку необхідно заповнити для підвищення рівня кібербезпеки вітчизняних IT-продуктів.

Мета статті. Метою роботи є дослідження методів підвищення рівня захищеності протоколу Git Large File Storage та формування переліку практичних рекомендацій (best practices) щодо конфігурації LFS-серверів для мінімізації ризиків несанкціонованого доступу та атак на відмову в обслуговуванні.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати архітектурні особливості функціонування протоколу Git LFS, зокрема специфікацію Batch API та механізми роботи з файлами-вказівниками.
2. Дослідити на практиці процеси автентифікації та передачі великих бінарних об'єктів (на прикладі взаємодії платформ GitLab та GitHub), виявивши обмеження гібридних транспортних моделей (SSH/HTTPS).
3. Ідентифікувати та систематизувати вектори атак на інфраструктуру LFS, класифікувавши їх за впливом на конфіденційність, цілісність та доступність системи.
4. Запропонувати практичні рекомендації, впровадження яких мінімізує ризики витоку даних та атак на ланцюги постачання програмного забезпечення.

ТЕОРЕТИЧНІ ОСНОВНИ ДОСЛІДЖЕННЯ

Еволюція та стандартизація систем контролю версій.

Системи контролю версій (VCS - Version Control Systems) — це інструменти, які дозволяють відстежувати зміни в коді та інших файлах проєкту, повертатися до попередніх версій, працювати паралельно кільком розробникам і уникати конфліктів при злитті змін.

Еволюція VCS пройшла шлях від локальних рішень до централізованих систем (CVCS), де історія зберігалася на єдиному сервері. Проте сучасним стандартом індустрії стали розподілені системи (DVCS). Їх ключова відмінність полягає в тому, що користувачі не просто перевіряють останню версію файлів, а повністю дзеркалюють репозиторій. Це означає, що кожен розробник має повну локальну копію всієї історії проекту.

Створена у 2005 році Лінусом Торвальдсом для розробки ядра Linux, система Git стала домінуючою технологією у світі DevOps. Технічно Git відрізняється від попередників методом обробки даних. Якщо старі системи зберігали інформацію як список змін до файлів, то Git розглядає дані як серію миттєвих знімків (snapshots) мініатюрної файлової системи. Кожного разу, коли розробник фіксує зміни (commit), Git "фотографує" стан усіх файлів і зберігає посилання на цей знімок. Для забезпечення цілісності використовується механізм хешування, що робить неможливою зміну вмісту файлу без зміни його ідентифікатора[5,6].

Попри високу ефективність роботи з текстовим кодом, архітектура Git виявляє критичні недоліки при обробці великих бінарних файлів (Large Binary Blobs), таких як аудіо, відео, графічні макети або моделі машинного навчання.

Проблема полягає у розподіленій природі системи: оскільки кожен клієнт завантажує повну історію репозиторію, наявність у ньому великих файлів, що часто змінюються, призводить до експоненційного зростання обсягу даних, які необхідно передати мережею. Наприклад, зміна одного пікселя у файлі Photoshop розміром 500 МБ змушує Git зберегти нову копію всього файлу, а не лише дельту змін. Це явище призводить до "роздування" репозиторіїв, сповільнення операцій clone / fetch і, зрештою, робить стандартний Git непридатним для сучасних ML проектів, тощо.

Саме для вирішення цієї дилеми - збереження переваг Git без перевантаження клієнтів бінарними даними - було розроблено протокол Git Large File Storage (LFS), архітектура та вразливості якого є предметом подальшого аналізу.

Git LFS

Протокол Git Large File Storage (LFS) є розширенням з відкритим вихідним кодом, яке змінює парадигму обробки даних у Git, переходячи від повної децентралізації до гібридної моделі. Згідно з дослідженням Юань Чена та ін. [4], архітектура LFS (загальна схема відображена на рисунку 1) базується на трьох взаємопов'язаних компонентах: системі файлів-вказівників, механізмі фільтрації Git та спеціалізованому HTTP-протоколі Batch API.

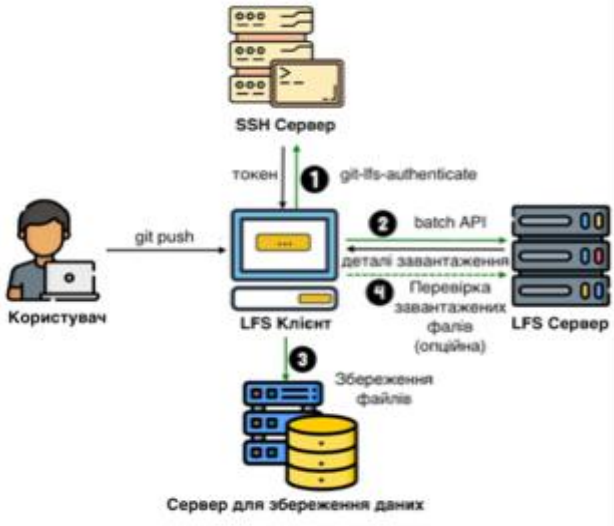

Рис. 1. Взаємодія протоколу LFS для завантаження файлу

Концепція файлів-вказівників (Pointer Files)

Основна ідея LFS полягає у відокремленні метаданих файлу від його фактичного вмісту. LFS використовує модель Content-Addressable Storage (CAS), де адресою файлу виступає хеш від його вмісту. Коли файл потрапляє під управління LFS, його бінарний вміст (BLOB) замінюється на незначний по розміру текстовий файл-вказівник. Цей підхід дозволяє Git-репозиторію залишатися компактним, оскільки він версіонує лише текстові рядки довжиною менше 1 КБ, незалежно від реального розміру вхідних файлів.

Згідно зі специфікацією, вказівник має суворий формат і містить:

Префікс версії: version https://git-lfs.github.com/spec/v1.

Ідентифікатор (OID): Хеш-сума SHA-256 від оригінального об'єкта.

Розмір (Size): Точний розмір файлу в байтах.

Приклад структури файлу-вказівника:

```
version https://git-lfs.github.com/spec/v1
oid sha256: <SHA-256 хеш оригінального файлу>
  size <розмір файлу у байтах>
```

Концепція файлів-вказівників (Pointer Files)

Робота LFS для кінцевого користувача забезпечується через механізм хуків та фільтрів Git. Як описано в технічній документації [7], LFS реєструє два основні фільтри у конфігурації .gitattributes:

- Clean Filter (при *git add*): Перехоплює файл перед його додаванням. Фільтр зчитує бінарний потік, обчислює його SHA-256 хеш, переміщує оригінальний файл у локальний кеш LFS (.git/lfs/objects), а в індекс Git записує згенерований файл-вказівник.
- Smudge Filter (при git checkout): Працює у зворотному напрямку. Коли Git намагається відновити файл у робочу директорію, фільтр зчитує вказівник, знаходить відповідний OID і підміняє текст на реальний бінарний контент з локального кешу або ініціює його завантаження з сервера. Саме на етапі роботи smudge фільтру часто виникають затримки або помилки доступу, якщо сервер не може коректно авторизувати запит на отримання контенту.

Протокол обміну даними (LFS Batch API)

• Комунікація між клієнтом і сервером відбувається через спеціалізований Batch API, який є stateless HTTP-протоколом. Процес синхронізації не є атомарним і, розділяється на дві критичні фази: Узгодження (Negotiation) та Передача (Transfer)[4].

• *Фаза Узгодження.* Клієнт відправляє HTTP POST запит. Сервер аналізує запит, перевіряє права доступу користувача (наприклад, чи має він write доступ до репозиторію) і повертає JSON-відповідь.

- Тіло запиту:

```
POST /user/repo.git/info/lfs/objects/batch
Authorization: Bearer <token>
Content-Type: application/vnd.git-lfs+json
Accept: application/vnd.git-lfs+json
{ "operation": "upload",
"objects": [{
"oid": "<object ID (file SHA-256)>",
"size": <file size in bytes>
},...] }
```

- JSON-відповідь:

```
 { "transfer": "basic",
   "objects": [{
   "oid": "<object ID>",
   "size": <file size>,
   "actions": {
          "upload": {
                 "href": "https://<upload url>",
                 "header": {
                        "Authorization": "<upload-token>",
                        ... <more headers>
          }},
          "verify": {
                 "href": "https://<verify api url>",
                 "header": {
                        "Authorization": "<verify-token>"
   }}}}, ... <more objects>]}
```

• *Фаза Передачі.* Отримавши відповідь, клієнт виконує безпосереднє завантаження даних за отриманими посиланнями (HTTP PUT/GET). Важливою архітектурною особливістю є те, що сервер зберігання даних (Storage Server) часто фізично відокремлений від LFS-сервера (API Server).

Архітектурні передумови вразливостей.

Складність протоколу Batch API створює специфічну поверхню атак, яку у дослідженні [4] класифікують як "розрив контексту безпеки" (security context gap). Оскільки LFS-сервер лише видає посилання, а фактичну перевірку даних здійснює хмарне сховище (або не здійснює зовсім), виникають ризики:

• Відсутність валідації вмісту: Хмарне сховище часто не перевіряє, чи відповідає завантажений файл заявленому OID (хешу). Це дозволяє атаку "LFS File Replacement".

• Слабкість токенів: Pre-signed URL часто мають надто довгий час життя або не прив'язані до IP-адреси клієнта, що створює ризик їх викрадення та повторного використання.

МЕТОДИКА ДОСЛІДЖЕННЯ

Підготовка експериментального середовища та аналіз транспортних протоколів

Для верифікації теоретичних векторів атак та тестування методів захисту було розгорнуто тестове середовище. Метою експерименту є відтворення сценарію міграції репозиторію з великими бінарними об'єктами між двома найпопулярнішими платформами: GitLab (як вихідна точка) та GitHub (як цільова платформа).

Архітектура

Інфраструктура експерименту побудована за схемою «Source-Client-Destination» і включає наступні компоненти:

- Source (GitLab): Репозиторій, що містить набір тестових бінарних файлів різного розміру, які відслідковуються через LFS.
- Client (Test Agent): Віртуальна машина на базі Linux (Ubuntu 22.04 LTS) зі встановленим клієнтом git-lfs. Клієнт виконує роль проміжної ланки, здійснюючи клонування (git clone --mirror) та дзеркальне відправлення (git push --mirror) даних.
- Destination (GitHub): Цільовий порожній репозиторій в організації GitHub, налаштований на прийом LFS-об'єктів.

Ініціалізація та відстеження об'єктів

Підготовка середовища розпочиналася з ініціалізації LFS-конфігурації. Для коректної обробки бінарних даних були застосовані стандартні команди відстеження:

```
git clone
https://$gitlab_token_name:$gitlab_token@${HOST}/$project_path/$project_name.g
it
/builds/$project_path/$project_name

#Додавання origin для github
        git remote add github
"https://$github_token_name:$github_token@github.com/$github_organization/$pro
ject_name.git"

  #Ініціалізація LFS
        git lfs install
  #Міграція файлів котрі більші за 100 Mb
  #Специфічний репозиторій
        git lfs migrate import --above=100Mb --include-ref=$branch -verbose
  #Весь проект
  git lfs migrate import --above=100Mb --everything -verbose
```

Проблема автентифікації на GitHub (HTTPS)

Ключовим технічним викликом, виявленим у процесі налаштування середовища, стала специфіка реалізації протоколу LFS на платформі GitHub. Як показав експеримент, при використанні SSH-протоколу виникає архітектурна колізія: Git використовує порт 22, але LFS вимагає передачі даних через HTTPS. Для вирішення цього, GitHub змушує клієнт LFS виконувати додатковий крок аутентифікації:

1. Клієнт ініціює з'єднання через SSH, використовуючи приватний ключ користувача.
2. Замість прямої передачі файлів, сервер повертає тимчасовий токен доступу (або спеціальні заголовки авторизації) та URL-адресу LFS-ендпоінту.
3. Клієнт розриває (або призупиняє) SSH-сесію і відкриває нове HTTPS-з'єднання, використовуючи отриманий токен для завантаження бінарних даних.

Такий підхід ("SSH Handover") дозволяє зберегти безпеку SSH-ключів, не передаючи їх по мережі, але ускладнює процес рукостискання (handshake) і збільшує затримки.

Альтернативна автентифікація через HTTPS та оцінка ризиків

Серед інших підходів можливий варіант повної авторизації через HTTPS (використання https:// URL та Personal Access Tokens). Хоча цей метод є архітектурно простішим (єдиний порт 443), у професійному середовищі він часто розглядається як менш безпечний у порівнянні з SSH.

Це зумовлено кількома факторами:

- Природа секретів: HTTPS часто покладається на статичні токени (PAT), які, на відміну від SSH-ключів, можуть бути легше скомпрометовані через логування, збереження в історії командного рядка або ненадійні менеджери паролів.
- Вразливість до перехоплення: У корпоративних мережах HTTPS-трафік часто підлягає інспекції (SSL Inspection/Termination), що теоретично дозволяє адміністраторам мережі або зловмисникам (у випадку MitM-атак) отримати доступ до вмісту передачі, тоді як SSH-тунель зазвичай залишається непрозорим [8].

Таким чином, експеримент підтверджує, що безпека Git LFS вимагає балансу між зручністю (HTTPS) та криптографічною стійкістю (SSH), причому механізм гібридної передачі є компромісним рішенням провайдерів.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Систематизація вразливостей та комплексні підходи до забезпечення безпеки

На основі проведеного аналізу архітектури Git LFS та результатів експериментального моделювання атак, наведено класифікацію ідентифікованих загроз та запропоновано стратегії їх нейтралізації.

Класифікація виявлених вразливостей

Узагальнюючи результати дослідження [1] та власний аналіз, вразливості екосистеми Git LFS можна розділити на три структурні групи наведені в таблиці 1.

Таблиця 1

Класифікація основних вразливостей протоколу Git LFS

Категорія безпеки	Тип вразливості	Опис та механізм виникнення
Контроль доступу *(Access Control)*	**Несанкціонований доступ до об'єктів**	Відсутність перевірки приналежності LFS-об'єкта до конкретного репозиторію. Це дозволяє зловмисникам завантажувати приватні файли, знаючи лише їх хеш (OID), навіть без прав доступу до самого проекту.
	Маніпуляції з токенами	Використання довготривалих (long-lived) або багаторазових токенів, отриманих через Batch API. Це створює високий ризик їх перехоплення (MITM) та подальшого повторного використання (Replay Attack) для доступу до даних.
Цілісність даних *(Data Integrity)*	**Отруєння LFS-кешу** *(Cache Poisoning)*	Можливість завантаження на сервер зберігання (Storage Server) файлу, зміст якого не відповідає заявленому OID. Вразливість виникає через відсутність валідації хеш-суми на стороні хмарних сховищ (S3/Azure) в момент завантаження.
	Підробка вказівників	Маніпуляція текстовими метаданими у файлах-вказівниках (Pointer Files). Дозволяє зловмиснику перенаправити

		клієнтський запит на шкідливі ресурси або підмінити посилання на об'єкт.
Доступність *(Availability)*	**Виснаження ресурсів** *(Quota Exhaustion)*	Відсутність лімітів (Rate Limiting) на кількість ініційованих сесій завантаження. Дозволяє зловмисникам масово резервувати дисковий простір та вичерпувати пропускну здатність сервера без фактичного завершення завантаження даних.

Рекомендовані підходи до побудови системи безпеки

Для нейтралізації ідентифікованих загроз та побудови надійної архітектури LFS-сервера пропонується впровадження стандартизованих властивостей безпеки. На основі аналізу роботи [1], вимоги розділено на три фундаментальні виміри: контроль доступу (конфіденційність), цілісність файлів та управління квотами (доступність) [9,10]:

- Забезпечення конфіденційності через контроль доступу.
  - Доступ до завантаження LFS-об'єктів (download) повинні мати виключно користувачі з правами “read” до конкретного репозиторію. Це унеможливлює доступ до приватних об'єктів з інших репозиторіїв.
  - Перед тим як зробити об'єкт доступним для читання, сервер зобов'язаний перевірити його вміст під час завантаження. Це критично для запобігання витоку даних, коли зловмисник може спробувати "перезаписати" файл порожнім контентом, щоб обійти перевірки.
  - Завантаження нових об'єктів (upload) дозволяється лише користувачам із явними правами “write”.
  - Ключі розгортання (Deploy Keys), налаштовані лише для читання (наприклад, для CI/CD), повинні технічно блокуватися при спробі завантаження файлів. Порушення цього правила є прямим вектором для атак на ланцюг постачання (Supply Chain Attacks).
- Гарантування цілісності даних.
  - Сервер не повинен довіряти клієнту "на слово". Кожен завантажений об'єкт повинен проходити перевірку на стороні сервера: реальний SHA-256 хеш вмісту має збігатися з заявленим OID. Відсутність такої перевірки дозволяє зловмисникам завантажувати отруєні ML-моделі з бекдорами, які клієнтські додатки можуть завантажити без додаткових перевірок.
- Захист доступності
  - Операції скачування (download) ніколи не повинні збільшувати використання квоти.
  - Списання квоти має базуватися на фактичному розмірі завантаженого файлу, а не на параметрі size, переданому клієнтом у JSON-запиті.
  - Використання ресурсів має атрибутуватися поточному репозиторію, а не кореневому репозиторію, щоб уникнути атак через масове створення форків.
  - Файли, які не враховані в квоті (наприклад, завантаження не завершено коректно), не повинні бути доступними для скачування.
  - Використання ресурсів має бути зав’язане на поточній ргілці, а не кореневій, щоб уникнути атак через масове їх створення.
  - "Завислі" (pending) об'єкти, для яких не було викликано verify API, повинні регулярно видалятися відповідними механізмами, щоб зловмисники не могли використовувати їх для тимчасового зберігання даних в обхід лімітів.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У роботі проведено комплексне дослідження архітектурної безпеки протоколу Git Large File Storage (LFS), який є критичним елементом сучасної інфраструктури розробки та MLOps. За результатами аналізу теоретичної бази та дослідження атак було встановлено, що фундаментальною проблемою безпеки LFS є "розрив контексту" між сервером управління правами (Batch API) та сервером зберігання даних (Storage Server). Експериментально підтверджено, що гібридна модель автентифікації (SSH для Git, HTTPS для LFS), яка використовується на платформах типу GitHub, створює умови для маніпуляцій з токенами доступу та атак на виснаження квот.

Основні результати дослідження:

- Систематизовано вектори атак на LFS за тріадою CIA.
- Доведено, що відсутність валідації вмісту файлів (SHA-256) на стороні хмарних сховищ створює пряму загрозу для ланцюгів постачання ПЗ, дозволяючи приховане впровадження бекдорів у ML-моделі.
- Запропоновано заходи безпеки, впровадження яких дозволяє нівелювати виявлені загрози. Ключовою рекомендацією є перехід до моделі "Verify-before-Write" та сувора прив'язка використання квот до власника репозиторію.

Враховуючи динамічний розвиток екосистеми DevSecOps, подальші наукові пошуки доцільно зосередити на таких напрямках:

- *Автоматизація аудиту.* Розробка інструментарію (open-source сканерів) для автоматизованої перевірки LFS-серверів на відповідність запропонованим властивостям безпеки.

Криптографічна стійкість: Аналіз можливостей переходу Git LFS на алгоритми хешування (SHA-3 або інші) для забезпечення довгострокової цілісності великих даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stack Overflow. (2025). *2025 Stack Overflow Developer Survey*. Stack Overflow Insights. https://survey.stackoverflow.co/2025
2. Piergiorgio, L., Plate, H., Matias, M., & Barais, O. (2023). *SoK: Taxonomy of attacks on open-source software supply chains*. In *2023 IEEE Symposium on Security and Privacy*. https://doi.org/10.1109/SP46215.2023.10179304
3. GitLab. (2024). *2024 Global DevSecOps Report: The state of AI in software development*. https://about.gitlab.com/developer-survey/
4. Chen, Y., Wang, Q., Yang, Y., Chen, Y., Li, Y., & Ji, S. (2025). *Unveiling security vulnerabilities in Git Large File Storage protocol*. In *2025 IEEE Symposium on Security and Privacy (SP)* (pp. 468–485). IEEE. https://doi.org/10.1109/sp61157.2025.00123
5. Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A quick introduction to version control with Git and GitHub. *PLOS Computational Biology, 12*(1), Article e1004668. https://doi.org/10.1371/journal.pcbi.1004668
6. Git. (n.d.). *Git*. https://git-scm.com/
7. Git Large File Storage. (n.d.). *Git Large File Storage*. https://git-lfs.com/
8. Schink, M., Wagner, A., Unterstein, F., & Heyszl, J. (2021). Security and trust in open source security tokens. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 176–201. https://doi.org/10.46586/tches.v2021.i3.176-201
9. Xiang, Z., Miller, D. J., & Kesidis, G. (2024). *BadChain: Backdoor attacks in the supply chain of large language models*. In *2024 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)* (pp. 1–18). IEEE.
10. Koohy, B., & Cito, J. (2023). *Empirical analysis of security weaknesses in CI/CD pipelines*. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)* (pp. 214–225). IEEE.

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (31), 2025
ISSN 2663 - 4023

Volodymyr Havryliak
PhD student, Information protection Department
Lviv polytechnic National University, Lviv, Ukraine
ORCID: 0009-0002-4114-1232
volodymyr.r.havryliak@lpnu.ua

RESEARCH OF METHODS AND TOOLS FOR IMPROVING GIT LFS PROTOCOL SECURITY

Abstract. In the current landscape of rapidly evolving DevOps and MLOps practices, the Git version control system has become the industry standard, capturing over 90% of the development market. The surge in machine learning adoption and the necessity of versioning large binary objects (ML models, datasets) have driven the widespread implementation of the Git Large File Storage (LFS) extension. However, the specific architecture of this protocol, based on separating metadata from file content, creates a new, critical attack surface that remains insufficiently explored within the context of Software Supply Chain Security. In light of these issues, the objective of this study is to perform a systemic analysis of architectural vulnerabilities in the Git LFS protocol and to develop a comprehensive set of practical recommendations for enhancing infrastructure security. To achieve this, threat modeling methods and experimental testing in a controlled environment were applied, specifically involving data migration between GitLab and GitHub. Additionally, the Batch API specification and hybrid authentication mechanisms were analyzed in detail. The study experimentally confirmed that using a hybrid transport model (SSH for Git, HTTPS for LFS) creates a "security context gap," complicating access validation. Based on the analysis, three groups of critical vulnerabilities were identified: confidentiality breaches due to insufficient object ownership verification, data integrity threats via file content substitution ("cache poisoning"), and availability risks due to quota exhaustion attacks (Quota-based DoS). Specifically, it was determined that the lack of hash verification on the cloud storage side allows for the injection of malicious code into ML models, bypassing standard security measures. Synthesizing the results, the paper systematizes attack vectors on LFS infrastructure and adapts international security practices mandated for LFS server protection. Key proposed measures include transitioning to a "Verify-before-Write" model, prohibiting the use of deploy keys for write operations, and implementing strict quota attribution to the current repository.

Keywords: Git; Large File Storage; DevSecOps; data transfer vulnerabilities.

REFERENCES

1. Stack Overflow. (2025). *2025 Stack Overflow Developer Survey*. Stack Overflow Insights. https://survey.stackoverflow.co/2025
2. Piergiorgio, L., Plate, H., Matias, M., & Barais, O. (2023). *SoK: Taxonomy of attacks on open-source software supply chains*. In *2023 IEEE Symposium on Security and Privacy*. https://doi.org/10.1109/SP46215.2023.10179304
3. GitLab. (2024). *2024 Global DevSecOps Report: The state of AI in software development*. https://about.gitlab.com/developer-survey/
4. Chen, Y., Wang, Q., Yang, Y., Chen, Y., Li, Y., & Ji, S. (2025). *Unveiling security vulnerabilities in Git Large File Storage protocol*. In *2025 IEEE Symposium on Security and Privacy (SP)* (pp. 468–485). IEEE. https://doi.org/10.1109/sp61157.2025.00123
5. Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A quick introduction to version control with Git and GitHub. *PLOS Computational Biology, 12*(1), Article e1004668. https://doi.org/10.1371/journal.pcbi.1004668
6. Git. (n.d.). *Git*. https://git-scm.com/
7. Git Large File Storage. (n.d.). *Git Large File Storage*. https://git-lfs.com/
8. Schink, M., Wagner, A., Unterstein, F., & Heyszl, J. (2021). Security and trust in open source security tokens. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 176–201. https://doi.org/10.46586/tches.v2021.i3.176-201

9. Xiang, Z., Miller, D. J., & Kesidis, G. (2024). *BadChain: Backdoor attacks in the supply chain of large language models*. In *2024 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)* (pp. 1–18). IEEE.
10. Koohy, B., & Cito, J. (2023). *Empirical analysis of security weaknesses in CI/CD pipelines*. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)* (pp. 214–225). IEEE.