



DOI 10.28925/2663-4023.2026.35.1347

УДК 004.056

Ільїн Микола Іванович

кандидат технічних наук, доцент кафедри інформаційної безпеки

НТУУ «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна

ORCID: 0000-0002-1065-6500

m.ilin@kpi.ua

Стьопочкіна Ірина Валеріївна

кандидатка технічних наук, доцентка кафедри інформаційної безпеки

НТУУ «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна

ORCID: 0000-0002-0346-0390

i.stopochkina@kpi.ua

Новіков Олексій Миколайович

доктор технічних наук, професор, член-кореспондент НАН України

НТУУ «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна

ORCID: 0000-0001-5988-3352

o.novikov@kpi.ua

МЕТОДИ ФОРМАЛЬНОЇ ВЕРИФІКАЦІЇ ВИПРАВЛЕНЬ БІНАРНИХ ВРАЗЛИВОСТЕЙ СИСТЕМ КРИТИЧНОЇ ІНФРАСТРУКТУРИ

Анотація. Революційні зміни в індустрії кібербезпеки внаслідок розвитку генеративного штучного інтелекту суттєво вплинули на процеси пошуку, експлуатації і виправлення вразливостей програмних систем. Сучасні передові великі мовні моделі здатні автономно знаходити та виправляти вразливості програмного забезпечення на рівні експертизи, недосяжної для більшості людей. Відбуваються фундаментальні зміни в процесах кібербезпеки: агентна автономність штучного інтелекту зумовлює критичну важливість надійності цих нових процесів. В роботі досліджуються методи автоматизованого аналізу вразливостей програмного забезпечення високонадійних систем критичної інфраструктури на етапі верифікації виправлень. Проаналізовано передумови виникнення некоректних виправлень як наслідку галюцинацій – структурних особливостей сучасних великих мовних моделей в результаті донавчання з депріоритезацією відмови від відповіді. Розглянуто існуючі методи виявлення і контролю галюцинацій, для процесів синтезу виправлень програмних продуктів запропоновано підхід на основі формальної верифікації вразливостей, повноти та еквівалентності виправлень. Запропоновано практичні методи формальної верифікації із застосуванням великих мовних моделей класу LLaMA-3.1, що синтезують програмний код Python для доказу коректності як знайденої вразливості, так і відповідності виправлення досліджуваній вразливості. Показано можливість автоматичного аналізу виправлень на рівні вихідного коду (зокрема C/C++), бінарних виправлень (на рівні машинного коду, ARM) та ефективне узагальнення на рівні проміжного представлення LLVM IR (включно з мовами високого рівня, що мають фронтенди LLVM, та машинним кодом підтримуваних QEMU архітектур процесорів через можливості ліфтингу платформи rev.ng). Експериментально досліджено точність запропонованих методів на наборі даних Vul4C (144 CVE у 23 відкритих програмних продуктах, 19 різних типів вразливостей CWE).

Ключові слова: Безпека промислових систем керування, безпека контуру операційних технологій, формальна верифікація, автоматизоване виправлення вразливостей, аналіз бінарних вразливостей

ВСТУП

Системи на основі штучного інтелекту вносять революційні зміни в процеси пошуку, експлуатації і виправлення вразливостей програмного забезпечення. Сучасні передові великі мовні моделі здатні виконувати складні задачі з аудиту програмного коду на рівні людини-експерта в предметній області, автономно створюють повнофункціональні експлоїти для знайдених вразливостей, проєктують та



програмують виправлення для цільових програмних систем. Коректність результатів пошуку вразливостей при наявності експлоїту досить легко перевірити – факт контролю над потоком керування, примітивні довільного читання/запису, віддалене виконання коду відтворювані у лабораторних умовах. Разом з тим, не для всіх знайдених вразливостей конкретна велика мовна модель здатна створити подібний експлоїт, і навіть у разі успіху, коректність виправлення є відкритим питанням як і для випадку відсутності експлоїту. Цінність коректного виправлення вразливостей є фундаментальною у контексті кібербезпеки, без наявності виправлень публікація інформації про вразливості не має практичного сенсу.

Великі мовні моделі вразливі до т.зв. явища галюцинацій – створення неправдивих фактів, помилкові відповіді внаслідок прогалин даних, помилок у пам'яті. В контексті аналізу бінарних вразливостей наслідком є вигадані вразливості, некоректні виправлення, загальне зниження ефективності конвеєра аналізу. Спроба залучати людей на етапі верифікації в цілому не дала успіхів, програми винагород за знайдені вразливості (баг баунті) та автори вільного програмного забезпечення перевантажені потоком звітів про вразливості, і намагаються вводити обмеження на застосування генеративного штучного інтелекту в роботі.

Особливо чутливою тема виправлень вразливостей є для програмного забезпечення критичної інфраструктури, систем промислового контролю і керування, контуру операційних технологій (OT). Пріоритетність доступності та цілісності, ризик-орієнтовані програми оновлення висувають підвищені вимоги до коректності патчів, гарантій їх безпечності для неперервних процесів OT.

Постановка проблеми. Верифікація виправлень бінарних вразливостей є актуальною задачею, далекою від вирішення. Теоретична частина рішення має включати математичне доведення мінімальності та повноти модифікації виконуваного коду для знайденої вразливості; практична – виконання критеріїв коректності виправлення в контексті відсутності побічних ефектів порушення доступності, цілісності та конфіденційності оброблюваної в системі інформації.

Аналіз останніх досліджень та публікацій. Революційність впливу досліджень штучного інтелекту на індустрію кібербезпеки, зокрема пошуку і виправлення вразливостей програмного забезпечення, ілюструють проекти провідних дослідницьких лабораторій Anthropic [1], OpenAI [2], Zhipu AI [3]. Інтелектуальна спроможність застосовуваних великих мовних моделей в галузі аналізу бінарних вразливостей оцінена кількісно, в умовах незалежно відтворюваних тестів на основі реальних вразливостей, таких як CyberGym [4], ExploitGym [5] та ExploitBench [6]. Крім найкращих закритих мовних моделей США (Anthropic Mythos 5, OpenAI GPT-5.6 Sol), обмежено застосованих в ізолюваних OT сегментах критичної інфраструктури, високі показники в області кібербезпеки демонструють відкриті китайські моделі, як передового рівня (Qwen3.8 [7] 2.4T-A95B, Kimi K3 [8] 2.8T-A104B, GLM-5.2 [9], 5.3 [10] 743B-A40B), так і середнього (Qwen3.8 27B, Qwen3.6 27B, 35B-A3B), придатного до локального застосування на обладнанні класу NVIDIA DGX Spark. Спостерігається тенденція до публікації відкритих моделей і у США (Meta Muse Glimmer [11] 30B, Arcee Genesis-Science-1 [12]).

Галюцинація великих мовних моделей є структурною особливістю як результат навчання з винагородою за здогадки проти утримання від відповіді [13], і не може бути повністю усунута у сучасних передових моделях. Розроблено ряд методів для виявлення і корекції галюцинацій, зокрема для виявлення:

- Аналіз семантичної ентропії [14];
- Аналіз внутрішнього стану – INSIDE (EigenScore) [15], SAPLMA [16], маркери точної відповіді [17], SEP [18], Lookback Lens [19];
- Автокореляційні тести [20], [21], [22];
- RAG та зовнішні тести [23].

Для відновлення та корекції галюцинацій застосовують:

- RAG [24], [25], [26];
- Ланцюжок верифікацій (CoVe [27]);
- Донавчання на фактичних даних (різновид RLHF/DPO [28], [29], [30]);
- Обмежене декодування [31], [32];
- Обмежену автокорекцію [33], [34];
- Утримання від відповіді, залучення людини [35].

Особливості застосування виправлень в реальному часі для кіберфізичних систем з підвищеними вимогами до доступності досліджувалися у роботах [36], [37], [38].

Мета статті. В даній роботі досліджуються методи підвищення надійності результатів виправлень бінарних вразливостей систем критичної інфраструктури, на основі контролю галюцинацій великих мовних моделей в процесі автономного пошуку і виправлення вразливостей. Запропоновано методи формальної верифікації виправлень на рівні вихідного і бінарного коду, проміжного представлення LLVM IR.



РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Методи формальної верифікації програмного забезпечення на поточному етапі технологічного розвитку великих мовних моделей можуть бути застосовані до широкого класу задач аналізу бінарних виправлень. В даній роботі пропонується забезпечити перевірку коректності виправлень вразливостей за допомогою генерації формального доказу усунення вразливості та незмінності алгоритму роботи цільової програми на основі теорем Lean 4 [39], створюваних автономним агентом на базі Leanstrat-1.5 [40].

Проілюструємо ідею формальної верифікації виправлень для випадків наявного вихідного коду мовою C, бінарного виправлення для коду ARM та узагальнимо метод для випадку представлення LLVM IR [41] отриманого з мови високого рівня підтримуваної фронтом платформи LLVM та з машинного коду QEMU підтримуваного платформою rev.ng [42].

Приклад вразливого коду, що буде використовуватися для ілюстрації процесу формальної верифікації в наступних розділах, `tst1.c`:

```
#include <stdio.h>

int vuln(int idx, int val) {
    int buf[20] = {0};
    return buf[idx] = val;
}

int main() {
    int idx, val;
    scanf("%d %d", &idx, &val);
    vuln(idx, val);
}
```

Відповідна виправлена версія, `tst2.c`:

```
#include <stdio.h>

int vuln(int idx, int val) {
    int buf[20] = {0};
    if (idx >= 20 || idx < 0)
        return 0;
    return buf[idx] = val;
}

int main() {
    int idx, val;
    scanf("%d %d", &idx, &val);
    vuln(idx, val);
}
```

Верифікація виправлень на рівні вихідного коду C

У випадку наявності вихідних кодів виправлення буде мати наступний вигляд:

```
$ diff tst1.c tst2.c | tee tst.diff
4a5,6
> if (idx >= 20 || idx < 0)
>     return 0;
```

На основі `tst1.c` та виправлення `tst.diff` Leanstrat-1.5 ідентифікує CWE-121 (Stack-based Buffer Overflow, `buf[idx]` write with no bounds check) та продукує формальний доказ коректності виправлення `verify_patch.lean` на основі наступних теорем:

- 1) `guard_false_implies_valid_index`: $\neg PG \text{ idx} \rightarrow (0 \leq \text{idx} \wedge \text{idx} < 20)$;
 - 2) `patch_prevents_buffer_overflow`: $\forall \text{idx}, PG \text{ idx} \vee (0 \leq \text{idx} \wedge \text{idx} < 20)$;
 - 3) `guard_partitions_indices`: $PG \text{ idx} \rightarrow \text{True} \wedge \neg PG \text{ idx} \rightarrow (0 \leq \text{idx} \wedge \text{idx} < 20)$.
- де PG – patched guard, додана перевірка.

Верифікація виправлень на рівні машинного коду ARM

У випадку ARM бінарне виправлення буде мати наступний вигляд:

```
$ arm-none-eabi-gcc -S tst1.c tst2.c
```



```
$ diff tst1.s tst2.s | tee tst2.diff
```

```
...
34a35,44
>      cmp     r3, #19
>      bgt     .L2
>      ldr     r3, [fp, #-88]
>      cmp     r3, #0
>      bge     .L3
> .L2:
>      mov     r3, #0
>      b       .L5
> .L3:
>      ldr     r3, [fp, #-88]
...
```

На основі `tst1.s` та виправлення `tst2.diff` ідентифікується CWE-121 (Stack-based Buffer Overflow) та CWE-787 (Out-of-bounds Write):

- Вразливість `tst1.s:35-40` `str r2, [r3, #-80]` з неперевіченим `idx`;
- Виправлення `tst2.diff:35-44` перевірки `cmp r3, #19`; `bgt / cmp r3, #0`; `bge`;
- Доказ коректності з попереднього розділу.

Узагальнений випадок LLVM IR

Методи у попередніх розділи мають недоліки у вигляді залежності можливості побудови теорем формальної верифікації від мови програмування та архітектури процесора. Множина підтримуваних мов та архітектур суттєво розширюється при переході до універсального проміжного представлення, такого як LLVM IR. В цьому випадку забезпечується аналіз вихідних кодів підтримуваних LLVM фронтендів (C/C++/Objective-C, Ruby, Python, Haskell, Rust, D, PHP, Pure, Lua, Julia та ін.) та архітектур процесорів, підтримуваних існуючими ліфтерами (у випадку `rev.ng` всі підтримувані QEMU, 22+ архітектур включно з x86/x86-64, ARM/AArch64, MIPS та ін.)

У випадку LLVM IR виправлення буде мати наступний вигляд:

```
$ clang -S -emit-llvm tst1.c tst2.c
$ diff tst1.ll tst2.ll | tee tst3.diff
...
> %7 = load i32, ptr %4, align 4
> %8 = icmp sge i32 %7, 20
> br i1 %8, label %12, label %9
>
> 9:                                ; preds = %2
> %10 = load i32, ptr %4, align 4
> %11 = icmp slt i32 %10, 0
> br i1 %11, label %12, label %13
>
> 12:                                ; preds = %9, %2
> store i32 0, ptr %3, align 4
> br label %18
>
> 13:                                ; preds = %9
...
```

Вразливість ідентифікується як CWE-129 (Improper Validation of Array Index), значення `idx < 0` або `idx ≥ 20` призводить до запису за границями буфера у стеку (CWE-121/CWE-129). Формальна верифікація побудована на основі теорем `tst3_proof.lean`:

- `guard_false_implies_valid_index`: $\neg PG \text{ } idx \rightarrow 0 \leq idx < 20$;
- `patch_prevents_buffer_overflow`: $\forall idx, \text{ } PG \text{ } idx \wedge (0 \leq idx < 20)$;
- `guard_partitions_indices`: PG покриває всі індекси, розділює на ООВ/ІВ.
і успішно верифікується `lean v4.34.0-rc1`.

**Експериментальна оцінка точності автономного агента**

Запропоновані методи формальної верифікації реалізуються автономним агентом, в основі якого велика мовна модель Leanstral-1.5 119B-A6.5B також вразлива до галюцинацій. Проявляються вони у генерації вироджених випадків виду

```
theorem is_vulnerability_fixed : True := by
trivial
```

які хоча й проходять формальну верифікацію, але не доводять коректності виправлення. Такі випадки детектуються засобами аналізу програм Lean 4, проте залишають питання коректності виправлення відкритим. Оцінимо наскільки ефективним є застосування представлених методів на практиці.

В якості тестового набору даних використаємо Vul4C [43]. Він містить 144 вразливості (CVE) 19 різних типів (CWE) у 23 відкритих програмних продуктах. Для кожної вразливості надаються вихідні коди мовою C/C++, виправлення (diff), експлоїт, програмні інструкції щодо локального відтворення, ідентифікатори CVE та CWE. Ідентифікатори CVE видані у 2006-2020 рр., переважна більшість 2016-2018 рр. Набір даних зібраний для порівняння засобів автоматизованого виправлення вразливостей і має пріоритетом реалістичність представлених тестів.

В експерименті всі виправлення розбиті на окремі файли (у випадку якщо при виправленні вразливості модифіковано декілька файлів, вони розглядаються окремо). Файли збережено в окремих каталогах і перейменовано за шаблоном (scratch-XXX/ old.c, new.c, patch.diff) для попередження витoku інформації про номер CVE та CWE під час тестування (велика мовна модель не може шукати CVE в Інтернет і використовувати ім'я файлів для ідентифікації CWE). Процес верифікації протоколюється на рівні взаємодії локального агента з великою мовною моделлю, зокрема зберігаються розмірковування та виклики інструментів. Для кожного виправлення виконується

```
vibe --agent lean --auto-approve --output streaming \
--prompt "$PROMPT" | tee proof.log
```

Автономний агент має непривілейований доступ до Ubuntu Linux 24.04.4 LTS, Lean 4.33.0, Lake 5.0.0-src+d8b1897 та працює під управлінням Mistral Vibe 2.24.1.

За результатами тестування оцінюється кількість нетривіальних доказів коректності виправлення (відмінних від True by trivial в основній теоремі, що доводить функціональну еквівалентність нового коду і виправлення вразливості), кількість відновлених за вихідним кодом CWE (чи входить CWE з набору даних у список ідентифікованих агентом). Всього проаналізовано 198 виправлень, з них нетривіальних доказів 96.97% (192) та співпадінь CWE 33.84% (67).

Результати експерименту демонструють високу ефективність застосування методів на основі формальної верифікації. Їх можна покращити за допомогою додаткових моделей у змагальному циклі самокорекції (adversarial Investigator/Skeptic self-correction loop), та донавчання на даних CWE для цільового типу вхідних даних.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

В роботі досліджено методи автоматизованого аналізу вразливостей програмного забезпечення високонадійних систем критичної інфраструктури на етапі верифікації виправлень. Для процесів синтезу виправлень програмних продуктів запропоновано підхід на основі формальної верифікації вразливостей, повноти і еквівалентності виправлень. Запропоновано практичні методи формальної верифікації із застосуванням великих мовних моделей класу Leanstral-1.5, що синтезують програмний код Lean 4 для доказу коректності як знайденої вразливості, так і відповідності виправлення досліджуваній вразливості.

Показано застосовність запропонованих методів для аналізу виправлень:

- Вихідного коду (зокрема, C/C++ з виправлення у форматі diff);
- Машинного коду (зокрема, ARM за дизасембльованими лістингами бінарних застосунків);
- Проміжного представлення LLVM IR (мови високого рівня з фронтендів LLVM, машинний код підтримуваних QEMU архітектур з ліфтингом rev.ng).

Проведено оцінку точності запропонованих методів на наборі даних Vul4C, що містить 144 CVE у 23 відкритих програмних продуктах, 19 різних типів вразливостей CWE. Автономний агент з API доступом до Leanstral-1.5 синтезував 96.97% нетривіальних доказів коректності виправлення, ідентифікував визначену в наборі даних CWE у 33.84% тестів. Наявність тривіальних рішень у роботі агента є наслідком галюцинацій великої мовної моделі-верифікатора, неповний збіг класифікацій CWE заявлених та



знайдених не є критичним (класи вразливостей перетинаються, агент робить висновок за неповними даними статичного аналізу єдиного зміненого файлу).

Результати експериментального дослідження показують високу ефективність і практичну цінність методів для аналізу реальних вразливостей сучасного програмного забезпечення.

Перспективи подальших досліджень пов'язані з донавчанням відкритої мовної моделі класу Llantrale для покращення слідування інструкціям, додавання пам'яті (RAG), знань про протоколи та програмні засоби систем промислової автоматизації.

ПОДЯКА

Робота частково фінансована грантом CAREER-2048266 (IMPRESS-U) наданого Національним науковим фондом, Національною академією наук та Управлінням військово-морських досліджень США, а також за фінансової підтримки Уряду США через Проєкт США «Кібербезпека критично важливої інфраструктури України».

ДЕКЛАРАЦІЯ ПРО ШТУЧНИЙ ІНТЕЛЕКТ

У процесі підготовки рукопису в якості допоміжного інструменту використано ШІ-технології: Anthropic Claude Code, Opus 5 для технічного форматування тексту (підготовки технічних елементів рукопису, перевірка орфографії, форматування) відповідно до вимог оформлення журналу. ШІ-технології не використовувалися для генерації наукового змісту, формулювання наукових ідей, висновків та інтерпретацій результатів досліджень. Автори несуть повну відповідальність за зміст рукопису.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Anthropic (2026). Project Glasswing: Securing critical software for the AI era. <https://www.anthropic.com/glasswing>
2. OpenAI (2026). Daybreak: Frontier AI for Cyber Defenders. <https://openai.com/daybreak>
3. Zhipu AI (2026). Z.ai Security Disclosure ledger. <https://cvd.z.ai>
4. Wang, Z., Shi, T., He, J., Cai, M., Zhang, J., & Song, D. (2025). CyberGym: Evaluating AI Agents' Cybersecurity Capabilities with Real-World Vulnerabilities at Scale. *arXiv*. <https://arxiv.org/abs/2506.02548>
5. Wang, Z., Schiller, N., Li, H., Narayana, S. S., Nasr, M., Carlini, N., Qi, X., Wallace, E., Bursztein, E., Invernizzi, L., Thomas, K., Shoshitaishvili, Y., Guo, W., He, J., Holz, T., & Song, D. (2026). ExploitGym: Can AI Agents Turn Security Vulnerabilities into Real Attacks?. *arXiv*. <https://arxiv.org/abs/2605.11086>
6. Lee, S., & Brumley, D. (2026). ExploitBench: A Capability Ladder Benchmark for LLM Cybersecurity Agents. *arXiv*. <https://arxiv.org/abs/2605.14153> Website: <https://exploitbench.ai>
7. Qwen Team (2026, August). Qwen3.8-Max: A New Bar for Coding and Cowork. <https://qwen.ai/blog?id=qwen3.8>
8. NIST (2026). UK AISI/CAISI Preliminary Assessment of Kimi K3's Cyber Capabilities. <https://www.nist.gov/news-events/news/2026/07/uk-aisi-caisi-preliminary-assessment-kimi-k3s-cyber-capabilities>
9. NIST (2026). CAISI Assessment of Z.ai's GLM-5.2. <https://www.nist.gov/news-events/news/2026/07/caisi-assessment-zais-glm-52>
10. Z.ai (2026). GLM-5.3: Frontier Coding with Emergent Cyber Capabilities. <https://z.ai/blog/glm-5.3>
11. Meta Superintelligence Lab (2026). Introducing Muse Glimmer: An Open Agentic Model That Runs on Your Device. <https://research.meta.ai/blog/introducing-muse-glimmer-open-agentic-model>
12. U.S. Department of Energy (2026). Genesis Open Models. <https://genesisopenmodels.anl.gov>
13. Kalai, A. T., Nachum, O., Vempala, S. S., & Zhang, E. (2025). Why Language Models Hallucinate. *arXiv*. <https://arxiv.org/abs/2509.04664>
14. Farquhar, S., Kossen, J., Kuhn, L., & Gal, Y. (2024, June). Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017), 625-630. <https://doi.org/10.1038/s41586-024-07421-0>
15. Chen, C., Liu, K., Chen, Z., Gu, Y., Wu, Y., Tao, M., Fu, Z., & Ye, J. (2024). INSIDE: LLMs' Internal States Retain the Power of Hallucination Detection. *arXiv*. <https://arxiv.org/abs/2402.03744>
16. Azaria, A., & Mitchell, T. (2023). The Internal State of an LLM Knows When It's Lying. *arXiv*. <https://arxiv.org/abs/2304.13734>



17. Orgad, H., Toker, M., Gekhman, Z., Reichart, R., Szpektor, I., Kotek, H., & Belinkov, Y. (2025). LLMs Know More Than They Show: On the Intrinsic Representation of LLM Hallucinations. *arXiv*. <https://arxiv.org/abs/2410.02707>
18. Kossen, J., Han, J., Razzak, M., Schut, L., Malik, S., & Gal, Y. (2024). Semantic Entropy Probes: Robust and Cheap Hallucination Detection in LLMs. *arXiv*. <https://arxiv.org/abs/2406.15927>
19. Chuang, Y.-S., Qiu, L., Hsieh, C.-Y., Krishna, R., Kim, Y., & Glass, J. (2024). Lookback Lens: Detecting and Mitigating Contextual Hallucinations in Large Language Models Using Only Attention Maps. *arXiv*. <https://arxiv.org/abs/2407.07071>
20. Manakul, P., Liusie, A., & Gales, M. J. F. (2023). SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models. *arXiv*. <https://arxiv.org/abs/2303.08896>
21. Zhang, J., Li, Z., Das, K., Malin, B., & Kumar, S. (2023, December). SAC3: Reliable Hallucination Detection in Black-Box Language Models via Semantic-aware Cross-check Consistency. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 15445–15458 pp. Association for Computational Linguistics. <https://aclanthology.org/2023.findings-emnlp.1032/>
22. Yehuda, Y., Malkiel, I., Barkan, O., Weill, J., Ronen, R., & Koenigstein, N. (2024, August). InterrogateLLM: Zero-Resource Hallucination Detection in LLM-Generated Answers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 9333–9347 pp. Association for Computational Linguistics. <https://aclanthology.org/2024.acl-long.506/>
23. Tamber, M. S., Bao, F. S., Xu, C., Luo, G., Kazi, S., Bae, M., Li, M., Mendelevitch, O., Qu, R., & Lin, J. (2025). Benchmarking LLM Faithfulness in RAG with Evolving Leaderboards. *arXiv*. <https://arxiv.org/abs/2505.04847>
24. Magesh, V., Surani, F., Dahl, M., Suzgun, M., Manning, C. D., & Ho, D. E. (2025). Hallucination-Free? Assessing the Reliability of Leading AI Legal Research Tools. *Journal of Empirical Legal Studies*, 22(2), 216–242. <https://onlinelibrary.wiley.com/doi/abs/10.1111/jels.12413>
25. Alansari, A., & Luqman, H. (2026). Large Language Models Hallucination: A Comprehensive Survey. *arXiv*. <https://arxiv.org/abs/2510.06265>
26. Mishra, S., Niroula, S., Yadav, U., Thakur, D., Gyawali, S., & Gaire, S. (2026). SoK: Agentic Retrieval-Augmented Generation (RAG): Taxonomy, Architectures, Evaluation, and Research Directions. *arXiv*. <https://arxiv.org/abs/2603.07379>
27. Dhuliawala, S., Komeili, M., Xu, J., Raileanu, R., Li, X., Celikyilmaz, A., & Weston, J. (2023). Chain-of-Verification Reduces Hallucination in Large Language Models. *arXiv*. <https://arxiv.org/abs/2309.11495>
28. Tian, K., Mitchell, E., Yao, H., Manning, C. D., & Finn, C. (2023). Fine-tuning Language Models for Factuality. *arXiv*. <https://arxiv.org/abs/2311.08401>
29. Gu, Y., Zhang, W., Lyu, C., Lin, D., & Chen, K. (2025). Mask-DPO: Generalizable Fine-grained Factuality Alignment of LLMs. *arXiv*. <https://arxiv.org/abs/2503.02846>
30. Chaduvula, S., Radwan, A. Y., Farooq, A., Ioannou, Y., & Raza, S. (2026). Reducing Hallucinations in LLMs via Factuality-Aware Preference Learning. *arXiv*. <https://arxiv.org/abs/2601.03027>
31. Chuang, Y.-S., Xie, Y., Luo, H., Kim, Y., Glass, J., & He, P. (2024). DoLa: Decoding by Contrasting Layers Improves Factuality in Large Language Models. *arXiv*. <https://arxiv.org/abs/2309.03883>
32. Li, K., Patel, O., Viégas, F., Pfister, H., & Wattenberg, M. (2024). Inference-Time Intervention: Eliciting Truthful Answers from a Language Model. *arXiv*. <https://arxiv.org/abs/2306.03341>
33. Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2024). Large Language Models Cannot Self-Correct Reasoning Yet. *arXiv*. <https://arxiv.org/abs/2310.01798>
34. Tyen, G., Mansoor, H., Carbune, V., Chen, P., & Mak, T. (2024, August). LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*, 13894–13908 pp. Association for Computational Linguistics. <https://aclanthology.org/2024.findings-acl.826/>
35. Kirichenko, P., Ibrahim, M., Chaudhuri, K., & Bell, S. J. (2025). AbstentionBench: Reasoning LLMs Fail on Unanswerable Questions. *arXiv*. <https://arxiv.org/abs/2506.09038>
36. Voitsekhovskiy, A., Stopochkina, I., Sun, P., Xie, J., Ilin, M., & Novikov, O. (2026). Detection of vulnerabilities in software for unmanned aerial vehicles by using large language models. *Eastern-European Journal of Enterprise Technologies*, 1(2 (139)), 36–47. <https://doi.org/10.15587/1729-4061.2026.352029>
37. Novikov, O., Stopochkina, I., Voitsekhovskiy, A., & Ilin, M. (2024). Attack Models for Industrial Control System Elements Based on a Graph Approach and Countermeasures. In *Proceedings of the XXIV International Scientific and Practical Conference "Information Technologies and Security" (ITS 2024)*, 67–82 pp. CEUR-WS.org. <https://ceur-ws.org/Vol-4068/paper6.pdf>
38. Novikov, O., Voitsekhovskiy, A., Stopochkina, I., Ilin, M., & Ovcharuk, M. (2025, December). Petri Net-Based Analysis of UAV Networks Availability Issues in Conditions of Adversary Counteraction.



- Theoretical and Applied Cybersecurity*, 7(3), 67–73. <https://doi.org/10.20535/tacs.2664-29132025.3.346322>
39. de Moura, L., & Ullrich, S. (2021). The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction – CADE 28*, 625–635 pp. Springer. https://doi.org/10.1007/978-3-030-79876-5_37
40. Mistral AI (2026). Leanstrat 1.5: Proof Abundance for All. <https://mistral.ai/news/leanstrat-1-5/>
41. Lattner, C., & Adve, V. (2004, March). LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, 75–88 pp. <https://doi.org/10.1155/2004/977395>
42. Di Federico, A., Payer, M., & Agosta, G. (2017). rev.ng: a unified binary analysis framework to recover CFGs and function boundaries. In *Proceedings of the 26th International Conference on Compiler Construction*, 131–141 pp. Association for Computing Machinery. <https://doi.org/10.1145/3033019.3033028>
43. Hu, Y., Li, Z., Shu, K., Guan, S., Zou, D., Xu, S., Yuan, B., & Jin, H. (2025). SoK: automated vulnerability repair: methods, tools, and assessments. In *Proceedings of the 34th USENIX Security Symposium (USENIX Security 25)*.

**Mykola Ilin**

PhD, associate professor

NTUU "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

ORCID: 0000-0002-1065-6500

*m.ilin@kpi.ua***Iryna Stopochkina**

PhD, associate professor

NTUU "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

ORCID: 0000-0002-0346-0390

*i.stopochkina@kpi.ua***Oleksii Novikov**

Dr.Sc., professor, academician of National Academy of Sciences of Ukraine

NTUU "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

ORCID: 0000-0001-5988-3352

*o.novikov@kpi.ua***METHODS FOR FORMAL VERIFICATION OF BINARY VULNERABILITY PATCHES FOR CRITICAL INFRASTRUCTURE SYSTEMS**

Abstract. Revolutionary changes in the cybersecurity industry, driven by the development of generative AI, have significantly affected the processes of finding, exploiting, and fixing vulnerabilities in software systems. Modern frontier large language models can autonomously identify and fix software vulnerabilities with an expertise level unattainable for most people. Fundamental changes are taking place in cybersecurity processes: the agent autonomy of artificial intelligence determines the critical importance of the reliability of these new processes. The paper investigates methods for automated analysis of software vulnerabilities in highly reliable critical infrastructure systems at the patch verification stage. The prerequisites for the occurrence of incorrect patches as a consequence of hallucinations – structural features of modern large language models as a result of additional training with deprioritization of refusal to respond, are analyzed. Existing methods for hallucination detection and control are considered, and an approach based on formal verification of vulnerabilities, completeness, and equivalence of patches is proposed for the synthesis of software product patches. Practical methods for formal verification are proposed using large language models of the Llama-3.1 class, which synthesize Lean 4 code to prove the correctness of both the found vulnerability and the patch's correspondence to the vulnerability under study. The possibility of automatic analysis of patches at the source code level (in particular C/C++), binary patches (at the machine code level, ARM), and effective generalization at the level of intermediate representation of LLVM IR (including high-level languages with LLVM frontends, and machine code of QEMU-supported processor architectures through the lifting capabilities of the rev.ng platform) is shown. The accuracy of the proposed methods was experimentally evaluated on the Vul4C dataset (144 CVEs across 23 open-source software products and 19 different CWE types).

Keywords: Industrial control system security, operational technology security, formal verification, automated vulnerability repair, binary vulnerability analysis

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Anthropic (2026). Project Glasswing: Securing critical software for the AI era. <https://www.anthropic.com/glasswing>
2. OpenAI (2026). Daybreak: Frontier AI for Cyber Defenders. <https://openai.com/daybreak>
3. Zhipu AI (2026). Z.ai Security Disclosure ledger. <https://cvd.z.ai>
4. Wang, Z., Shi, T., He, J., Cai, M., Zhang, J., & Song, D. (2025). CyberGym: Evaluating AI Agents' Cybersecurity Capabilities with Real-World Vulnerabilities at Scale. *arXiv*. <https://arxiv.org/abs/2506.02548>
5. Wang, Z., Schiller, N., Li, H., Narayana, S. S., Nasr, M., Carlini, N., Qi, X., Wallace, E., Bursztein, E., Invernizzi, L., Thomas, K., Shoshitaishvili, Y., Guo, W., He, J., Holz, T., & Song, D. (2026). ExploitGym: Can AI Agents Turn Security Vulnerabilities into Real Attacks?. *arXiv*. <https://arxiv.org/abs/2605.11086>



6. Lee, S., & Brumley, D. (2026). ExploitBench: A Capability Ladder Benchmark for LLM Cybersecurity Agents. *arXiv*. <https://arxiv.org/abs/2605.14153> Website: <https://exploitbench.ai>
7. Qwen Team (2026, August). Qwen3.8-Max: A New Bar for Coding and Cowork. <https://qwen.ai/blog?id=qwen3.8>
8. NIST (2026). UK AISI/CAISI Preliminary Assessment of Kimi K3's Cyber Capabilities. <https://www.nist.gov/news-events/news/2026/07/uk-aisi-caisi-preliminary-assessment-kimi-k3s-cyber-capabilities>
9. NIST (2026). CAISI Assessment of Z.ai's GLM-5.2. <https://www.nist.gov/news-events/news/2026/07/caisi-assessment-zais-glm-52>
10. Z.ai (2026). GLM-5.3: Frontier Coding with Emergent Cyber Capabilities. <https://z.ai/blog/glm-5.3>
11. Meta Superintelligence Lab (2026). Introducing Muse Glimmer: An Open Agentic Model That Runs on Your Device. <https://research.meta.ai/blog/introducing-muse-glimmer-open-agentic-model>
12. U.S. Department of Energy (2026). Genesis Open Models. <https://genesisopenmodels.anl.gov>
13. Kalai, A. T., Nachum, O., Vempala, S. S., & Zhang, E. (2025). Why Language Models Hallucinate. *arXiv*. <https://arxiv.org/abs/2509.04664>
14. Farquhar, S., Kossen, J., Kuhn, L., & Gal, Y. (2024, June). Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017), 625-630. <https://doi.org/10.1038/s41586-024-07421-0>
15. Chen, C., Liu, K., Chen, Z., Gu, Y., Wu, Y., Tao, M., Fu, Z., & Ye, J. (2024). INSIDE: LLMs' Internal States Retain the Power of Hallucination Detection. *arXiv*. <https://arxiv.org/abs/2402.03744>
16. Azaria, A., & Mitchell, T. (2023). The Internal State of an LLM Knows When It's Lying. *arXiv*. <https://arxiv.org/abs/2304.13734>
17. Orgad, H., Toker, M., Gekhman, Z., Reichart, R., Szpektor, I., Kotek, H., & Belinkov, Y. (2025). LLMs Know More Than They Show: On the Intrinsic Representation of LLM Hallucinations. *arXiv*. <https://arxiv.org/abs/2410.02707>
18. Kossen, J., Han, J., Razzak, M., Schut, L., Malik, S., & Gal, Y. (2024). Semantic Entropy Probes: Robust and Cheap Hallucination Detection in LLMs. *arXiv*. <https://arxiv.org/abs/2406.15927>
19. Chuang, Y.-S., Qiu, L., Hsieh, C.-Y., Krishna, R., Kim, Y., & Glass, J. (2024). Lookback Lens: Detecting and Mitigating Contextual Hallucinations in Large Language Models Using Only Attention Maps. *arXiv*. <https://arxiv.org/abs/2407.07071>
20. Manakul, P., Liusie, A., & Gales, M. J. F. (2023). SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models. *arXiv*. <https://arxiv.org/abs/2303.08896>
21. Zhang, J., Li, Z., Das, K., Malin, B., & Kumar, S. (2023, December). SAC3: Reliable Hallucination Detection in Black-Box Language Models via Semantic-aware Cross-check Consistency. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 15445–15458 pp. Association for Computational Linguistics. <https://aclanthology.org/2023.findings-emnlp.1032/>
22. Yehuda, Y., Malkiel, I., Barkan, O., Weill, J., Ronen, R., & Koenigstein, N. (2024, August). InterrogateLLM: Zero-Resource Hallucination Detection in LLM-Generated Answers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 9333–9347 pp. Association for Computational Linguistics. <https://aclanthology.org/2024.acl-long.506/>
23. Tamber, M. S., Bao, F. S., Xu, C., Luo, G., Kazi, S., Bae, M., Li, M., Mendelevitch, O., Qu, R., & Lin, J. (2025). Benchmarking LLM Faithfulness in RAG with Evolving Leaderboards. *arXiv*. <https://arxiv.org/abs/2505.04847>
24. Magesh, V., Surani, F., Dahl, M., Suzgun, M., Manning, C. D., & Ho, D. E. (2025). Hallucination-Free? Assessing the Reliability of Leading AI Legal Research Tools. *Journal of Empirical Legal Studies*, 22(2), 216-242. <https://onlinelibrary.wiley.com/doi/abs/10.1111/jels.12413>
25. Alansari, A., & Luqman, H. (2026). Large Language Models Hallucination: A Comprehensive Survey. *arXiv*. <https://arxiv.org/abs/2510.06265>
26. Mishra, S., Niroula, S., Yadav, U., Thakur, D., Gyawali, S., & Gaire, S. (2026). SoK: Agentic Retrieval-Augmented Generation (RAG): Taxonomy, Architectures, Evaluation, and Research Directions. *arXiv*. <https://arxiv.org/abs/2603.07379>
27. Dhuliawala, S., Komeili, M., Xu, J., Raileanu, R., Li, X., Celikyilmaz, A., & Weston, J. (2023). Chain-of-Verification Reduces Hallucination in Large Language Models. *arXiv*. <https://arxiv.org/abs/2309.11495>
28. Tian, K., Mitchell, E., Yao, H., Manning, C. D., & Finn, C. (2023). Fine-tuning Language Models for Factuality. *arXiv*. <https://arxiv.org/abs/2311.08401>
29. Gu, Y., Zhang, W., Lyu, C., Lin, D., & Chen, K. (2025). Mask-DPO: Generalizable Fine-grained Factuality Alignment of LLMs. *arXiv*. <https://arxiv.org/abs/2503.02846>
30. Chaduvula, S., Radwan, A. Y., Farooq, A., Ioannou, Y., & Raza, S. (2026). Reducing Hallucinations in LLMs via Factuality-Aware Preference Learning. *arXiv*. <https://arxiv.org/abs/2601.03027>



31. Chuang, Y.-S., Xie, Y., Luo, H., Kim, Y., Glass, J., & He, P. (2024). DoLa: Decoding by Contrasting Layers Improves Factuality in Large Language Models. *arXiv*. <https://arxiv.org/abs/2309.03883>
32. Li, K., Patel, O., Viégas, F., Pfister, H., & Wattenberg, M. (2024). Inference-Time Intervention: Eliciting Truthful Answers from a Language Model. *arXiv*. <https://arxiv.org/abs/2306.03341>
33. Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2024). Large Language Models Cannot Self-Correct Reasoning Yet. *arXiv*. <https://arxiv.org/abs/2310.01798>
34. Tyen, G., Mansoor, H., Carbune, V., Chen, P., & Mak, T. (2024, August). LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*, 13894–13908 pp. Association for Computational Linguistics. <https://aclanthology.org/2024.findings-acl.826/>
35. Kirichenko, P., Ibrahim, M., Chaudhuri, K., & Bell, S. J. (2025). AbstentionBench: Reasoning LLMs Fail on Unanswerable Questions. *arXiv*. <https://arxiv.org/abs/2506.09038>
36. Voitsekhovskiy, A., Stopochkina, I., Sun, P., Xie, J., Ilin, M., & Novikov, O. (2026). Detection of vulnerabilities in software for unmanned aerial vehicles by using large language models. *Eastern-European Journal of Enterprise Technologies*, 1(2 (139)), 36–47. <https://doi.org/10.15587/1729-4061.2026.352029>
37. Novikov, O., Stopochkina, I., Voitsekhovskiy, A., & Ilin, M. (2024). Attack Models for Industrial Control System Elements Based on a Graph Approach and Countermeasures. In *Proceedings of the XXIV International Scientific and Practical Conference "Information Technologies and Security" (ITS 2024)*, 67–82 pp. CEUR-WS.org. <https://ceur-ws.org/Vol-4068/paper6.pdf>
38. Novikov, O., Voitsekhovskiy, A., Stopochkina, I., Ilin, M., & Ovcharuk, M. (2025, December). Petri Net–Based Analysis of UAV Networks Availability Issues in Conditions of Adversary Counteraction. *Theoretical and Applied Cybersecurity*, 7(3), 67–73. <https://doi.org/10.20535/tacs.2664-29132025.3.346322>
39. de Moura, L., & Ullrich, S. (2021). The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction – CADE 28*, 625–635 pp. Springer. https://doi.org/10.1007/978-3-030-79876-5_37
40. Mistral AI (2026). Leanstral 1.5: Proof Abundance for All. <https://mistral.ai/news/leanstral-1-5/>
41. Lattner, C., & Adve, V. (2004, March). LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, 75–88 pp. <https://doi.org/10.1155/2004/977395>
42. Di Federico, A., Payer, M., & Agosta, G. (2017). rev.ng: a unified binary analysis framework to recover CFGs and function boundaries. In *Proceedings of the 26th International Conference on Compiler Construction*, 131–141 pp. Association for Computing Machinery. <https://doi.org/10.1145/3033019.3033028>
43. Hu, Y., Li, Z., Shu, K., Guan, S., Zou, D., Xu, S., Yuan, B., & Jin, H. (2025). SoK: automated vulnerability repair: methods, tools, and assessments. In *Proceedings of the 34th USENIX Security Symposium (USENIX Security 25)*.

Отримано редакцією журналу / Received: 03.06.26

Прорецензовано / Revised: 16.06.26

Схвалено до друку / Accepted: 26.10.26

