УДК 004.45
DOI https://doi.org/10.24144/2616-7700.2025.46(1).138-148

М. О. Богдюк

Національний університет біоресурсів і природокористування України,
аспірант кафедри комп'ютерних систем, мереж та кібербезпеки
m.bogdiuk@nubip.edu.ua
ORCID: https://orcid.org/0009-0000-4743-4023

АНАЛІЗ ПРОЦЕСУ ЗАВАНТАЖЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ НА БАЗІ АРХІТЕКТУРИ X86-64 НА ПРЕДМЕТ ВРАЗЛИВОСТЕЙ ТА ВИЗНАЧЕННЯ МЕХАНІЗМІВ ЙОГО ЗАХИСТУ

У статті проведено систематичний аналіз механізмів завантаження сучасної комп'ютерної системи архітектури x86-64 та потенційних загроз, що виникають у процесі її ініціалізації. Розглянуто методи компрометації прошивки, низькорівневих режимів виконання та засобів перевірки цілісності завантаження. Особливу увагу приділено атакам, що спрямовані на обхід захисних технологій зазначених систем. На основі аналізу наукових публікацій визначено потенційні вразливості компонентів комп'ютерної системи, які використовуються при її ініціалізації, та які необхідно ізолювати від потенційних загроз (зокрема небезпечних програм), а також визначені напрями подальших наукових досліджень.

Ключові слова: завантаження, компрометація прошивки, вразливості, обхід захисних технологій, ізолювання.

1. Вступ. В умовах цифрової трансформації всіх сфер діяльності суспільства, дедалі більше компаній та державних установ залежить від ефективного та безпечного функціонування інформаційних, електронних комунікаційних, та відповідно комп'ютерних систем. Ці системи використовуються зокрема для керування банківськими транзакціями, медичною інформацією, енергетичними мережами, військовою технікою та озброєнням. Вразливості в їхньому операційному середовищі можуть призвести не лише до значних фінансових втрат, але й становити загрозу безпеці держави, життю та здоров'ю людей.

Паралельно з розвитком засобів захисту інформації, методи кібератак також еволюціонують. Зловмисники застосовують все більш складні техніки, зокрема вбудовують шкідливе програмне забезпечення (ПЗ) в прошивку чипсета чи вбудованих пристроїв (Advanced Persistent Threats, APT). З огляду на це, дослідження вразливостей завантажувального ланцюга комп'ютерних систем та ефективних методів їх нейтралізації є вкрай важливим для забезпечення безпеки сучасних інформаційних систем.

Програмне забезпечення, що виконується на системному рівні, характеризується високою складністю, інтеграцією апаратних засобів та численними взаємозв'язками між компонентами, що створює ідеальне середовище для появи нових вразливостей. Завантаження операційної системи є критично важливим етапом, оскільки саме в цей момент відбувається ініціалізація апаратних компонентів, виконання прошивки та перевірка цілісності завантажуваних компонентів.

У цій статті розглядаються ключові етапи завантаження систем на основі x86-64, їх потенційні вразливості та сучасні методи захисту.

Основні вектори атак під час завантаження включають компрометацію прошивки (UEFI, SMM), обходи механізмів Secure Boot, підміну завантажувачів та атаки на рівні гіпервізора та ядра операційної системи.

Метою статті є системний аналіз процесу ініціалізації комп'ютерної системи та потенційних вразливостей для подальшої розробки засобів попередження та захисту від атак.

Об'єктом дослідження є процес ініціалізації комп'ютерної системи.

Предметом дослідження є методи атак на компоненти комп'ютерної системи, які використовуються при її ініціалізації, та методи захисту від них.

Методи дослідження: аналіз та синтез. Для кожного етапу ініціалізації комп'ютерної системи розглядаються наступні критерії:

- Поверхня атаки.
- Відомі вразливості.
- Засоби протидії.

На основі них шляхом порівняльного аналізу ці критерії узагальнюються на весь процес ініціалізації.

2. Зміст дослідження. Архітектура x86-64 обрана через високий рівень її використання в державних та комерційних інформаційних та електронних комунікаційних системах. Вона досі лишається домінантною та актуальною серед десктопних та ноутбучних систем. Це відрізняє її від ARM64, яка має фрагментовану екосистему, що включає різні вендорські реалізації без єдиного стандарту прошивки та механізмів безпеки. Точних даних про поширеність архітектур немає через комерційну таємницю, однак згідно [1] поширеність Windows становить 70%, а це в основному x86 + x86-64 платформа, системи Windows-on-ARM з достатньою продуктивністю почали випускатись лише у 2024 році.

На рис. 1 зображений процес запуску x86-64 системи. Зліва блоки TPM (Trusted Platform Module) та TEE (Trusted Execution Environment) є компонентами, що можуть бути використані будь-яким іншим компонентом, що розміщений нижче на діаграмі, стрілки до них не зображені задля кращого сприйняття діаграми. Зеленим кружечком позначено умовне «кільце привілеїв процесора» від -3 до 3. Це розширення принципу кілець привілеїв, сформульованого в [2], де рівень привілеїв міг бути від 0 до 3, однак практично архітектура x86-64 передбачає рівні привілеїв, нижчі за рівень ядра ОС, для прикладу на -1 рівні виконується гіпервізор, який має повний контроль над ядром (рівень 0).

Завантаження системи на базі x86-64 починається з запуску *Intel ME / AMD Secure Technology* (колись PSP). Це міні операційна система, яка керує процесором. Зокрема, вона перевіряє цілісність UEFI. Водночас, вона має небезпечні аспекти функціонування, а саме: має необмежений доступ до пам'яті та периферійних пристроїв, і працює, коли комп'ютер вимкнений, але має живлення. З однієї сторони це дозволяє адміністратору керувати всіма комп'ютерами мережі (ME), з іншої є потенційною точкою атаки. Зокрема, згідно дослідження [3], у процесорів Intel 2017 року ME базувалась на пропрієтарній версії Minix із численними вразливостями та відкритими мережевими портами. Особливо небезпечним є те, що експлойти можуть записатись у Flash-пам'ять, таким чином уникаючи стирання при перезапуску. Єдиним дієвим розв'язанням цієї проблеми автори вважають застосування апаратних компонентів, які мають повністю

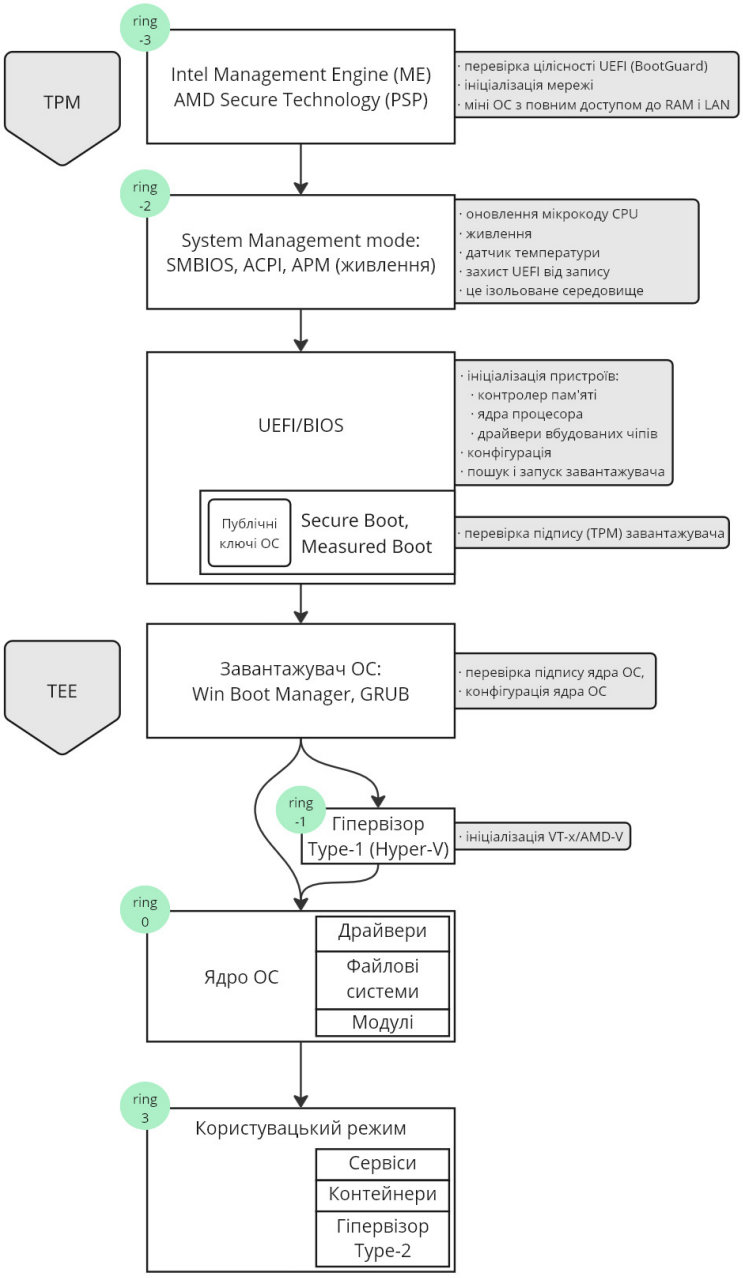

Рис. 1. Процес запуску операційної системи.

відкритий вихідний код прошивок для всієї системи, включно з процесором і чипсетом.

У статті [4], Intel стверджує, що використовує лише порти 623, 664, 5900 та 16992-16995 — їх можна заблокувати зовнішнім файрволом. В системах виробництва AMD мережеві порти не відкриті, тому вразливості обмежуються передачею спеціальних команд через SPI [5], [6], зокрема через драйвер чипсета amdpsp.sys [7], що реєструє пристрій \Device\amdpsp.

TPM — це апаратний криптографічний модуль, інтегрований у платформу (материнську плату чи процесорний модуль), який забезпечує основні довірені функції, пов'язані з безпекою обчислювальної системи. Серед ключових завдань

TPM виокремлюють зберігання та захист криптографічних ключів, виконання операцій шифрування/розшифрування та верифікацію цілісності апаратних і програмних компонентів.

Згідно зі специфікацією Trusted Computing Group (2003), TPM має незалежну апаратну реалізацію, що включає криптографічні співпроцесори, механізм генерації випадкових чисел (RNG) та захищену пам'ять для зберігання ключів. Завдяки цьому TPM може: генерувати криптографічні ключі у захищеному середовищі, недоступному для решти процесів системи; зберігати їх та використовувати для шифрування і підпису. Також TPM дозволяє перевіряти цілісність операційної системи шляхом перевірки контрольних сум.

Хоча є кілька досліджень, де ключ з TPM можна дістати апаратно, прослуховуючи шини SPI, I2C, LPC [8] [9], та Moghimi et al. [10] знайшли вразливості в fTPM та STMicroelectronics TPM, що дозволяють добути приватні ключі еліптичних алгоритмів ECDSA, та ECSchnorr шляхом аналізу побічного каналу (а саме, часу обробки), причому віддалено, без безпосереднього доступу до пристрою.

Після ME/PSP починає ініціалізацію *System Management Mode* (SMM) — спеціальний режим роботи процесора, який використовується для управління енергоспоживанням, апаратного моніторингу, оновлення мікрокоду процесора, емуляції апаратури та початкової ініціалізації UEFI, зокрема захищає секцію коду UEFI від запису. Код SMM виконується в ізольованій області пам'яті (SMRAM), що робить його недоступним для операційної системи та інших привілейованих компонентів. Після запуску системи, SMM зупиняється і продовжується лише у випадку генерації System Management Interrupt (SMI) — високопріоритетного немаскованого апаратного переривання. Згенерувати його можна:

- Апаратно: подати сигнал на пін SMI [11].
- Читання або запис в спеціальний I/O порт (залежить від материнської плати, деколи 0xB2) [12].
- Через APIC (Advanced Programmable Interrupt Controller) [13].
- Спеціальна команда, доступна з UEFI/BIOS.

Коли відбувається подія, що генерує SMI, процесор зберігає свій поточний стан і переходить у SMM, виконуючи код, розташований у SMRAM. Після завершення обробки процесор відновлює свій попередній стан і повертається до нормальної роботи.

Після перевірки цілісності і захисту від запису, починає виконуватись *Unified Extensible Firmware Interface*. UEFI виконує початкову ініціалізацію апаратного забезпечення та драйверів критично важливих пристроїв (мережа, NVME, AHCI), перевіряє його справність та знаходить завантажувальні пристрої. Стандарт не прив'язаний до архітектури, тому може бути реалізований на ARM пристроях, однак для економії виробники обходяться мінімальною нестандартизованою реалізацією.

Згідно дослідження [14], кількість атак на UEFI різко зросла у 2016-2017 і продовжує зростати. Детальніше вразливості UEFI узагальнені в [15], де описані механізми атак і засоби протидії, а також додатково підкреслюється важливість регулярного оновлення прошивки та контролю її цілісності.

Оскільки виробники випускають оновлення лише кілька років після випуску материнської плати, вразливості у UEFI часто залишаються назавжди. Однією з таких вразливостей є група LogoFAIL, знайдена Binarly Research Team [16]. Суть в тому, що користувач може завантажити будь-який рисунок екрану завантаження (наприклад, в EFI:\EFI\OEM\Logo.jpg). Для простоти користування, його цифровий підпис не перевіряється, адже він може бути згенерований користувачем. Це створює лазівку для завантаження непідписаного коду в чип SPI. Таким чином, достатньо лише знайти вразливість модуля, що зчитує зображення, і не думати як завантажити непідписаний код.

UEFI має механізм *Secure Boot*, який забезпечує перевірку цифрового підпису всіх завантажувальних компонентів ядра ОС (включаючи драйвери та модулів), тим самим обмежує виконання несанкціонованого коду на ранніх етапах завантаження.

Secure Boot також може створювати труднощі для користувачів операційних систем, відмінних від Windows. В своїй презентації Intel [17] розглядає виклики, з якими стикаються користувачі Linux при використанні Secure Boot. Зокрема, обговорюється проблема сумісності, коли Secure Boot може блокувати завантаження непідписаних або самостійно підписаних завантажувачів, що ускладнює встановлення та використання Linux-дистрибутивів. Для вирішення цієї проблеми було запропоновано використання завантажувача "shim який підписується довіреним сертифікатом і дозволяє завантажувати підписані ядра Linux. Крім того, у звіті обговорюється концепція Machine Owner Key (MOK), яка надає користувачам можливість додавати власні ключі для підпису завантажувачів та модулів ядра, забезпечуючи більшу гнучкість та контроль над процесом завантаження. Хоча деякі дистрибутиви Linux отримали підписані завантажувачі для сумісності з Secure Boot, процес налаштування все ще може бути складним для пересічних користувачів. А деякі виробники апаратного забезпечення взагалі не надають зручних інтерфейсів для керування налаштуваннями Secure Boot, що ускладнює процес його вимкнення або налаштування для підтримки альтернативних операційних систем.

UEFI вважається довіреним середовищем, оскільки код є підписаним. Компоненти, що завантажуються після нього, вважаються не довіреними, однак є випадки, коли користувацький код мусить обробити конфіденційні дані. Без спеціальної апаратної підтримки, будь-які дані, доступні користувацькому режиму, доступні і ядру системи. Для ізоляції таких даних і обчислень існує середовище захищеного виконання (*Trusted Execution Environment*, TEE), або захищений анклав – це апаратно-програмна концепція, яка створює «острівок довіри» всередині загальної обчислювальної платформи. Завдяки ізоляції, шифруванню та механізмам віддаленої атестації, анклав гарантує безпечне виконання коду та збереження конфіденційності критичних даних навіть тоді, коли сама операційна система частково скомпрометована. Це спеціалізоване обчислювальне середовище, у якому код і дані можуть бути ізольовані від решти системи й оброблятися у безпечній зоні. Основна ідея полягає в тому, щоб важлива логіка та конфіденційна інформація (наприклад, криптографічні ключі, паролі тощо) були недоступні для решти операційної системи або потенційно скомпрометованих процесів. Анклави мають власну адресу пам'яті, захищену шифруванням на апаратному рівні. Це унеможливлює прочитання або втру-

чання у вміст анклаву навіть з боку операційної системи чи гіпервізора. Код і дані, які необхідно захистити, завантажуються у пам'ять анклаву у зашифрованому вигляді, і апаратні засоби процесора розшифровують дані лише під час виконання, зберігаючи їх ізольованими від решти пам'яті. У випадку витоку пам'яті або фізичного доступу до неї, зловмисник не зможе прочитати інформацію анклаву, оскільки ключі шифрування зберігаються на рівні процесора, недоступному назовні. Під час виконання інструкції анклаву, доступ до пам'яті за межами анклаву обмежений, а самій операційній системі заборонено втручатися у процеси всередині анклаву. Результати обчислень або службова інформація, які виходять за межі анклаву, можуть бути додатково перевірені або зашифровані. Доступна також можливість верифікації цілісності коду, для унеможливлення модифікації коду зловмисником [18].

TEE досі обмежені в використанні через складність. Paju et al. [19] дослідили анклави і дійшли висновку, що усі доступні фреймворки для реалізації довірених контейнерів ("tcon") потребують суттєвого переписування наявного коду, а практична їхня імплементація є складнішою, ніж задекларовано виробниками SDK. Крім того, у більшості публікацій основна увага зосереджена на Intel SGX та Arm TrustZone, унаслідок чого програмного забезпечення для менш поширених платформ AMD SEV, RISC-V та графічних TEE значно менше. Zhao et al. [20] запропонували мінімалістичну архітектуру ядра ОС, орієнтовану на середовище довіреного виконання. Їх реалізація повністю вміщається в TEE, займаючи всього 100кБ. Ця архітектура спрямована на протидію атакам на рівні фізичного доступу до плати (board-level physical attacks). Зокрема, вони демонструють, як можна істотно скоротити поверхню атаки за рахунок зменшення коду та кількості компонентів усередині довіреного середовища. Такий підхід дає змогу ускладнити несанкціоноване втручання в роботу TEE та підвищити загальний рівень безпеки системи, водночас зберігаючи основні функціональні можливості для захищеного виконання.

Зворотною стороною TEE є те, що анклав є чудовим середовищем для виконання самих шкідливих програм: він невидимий для антивірусів і не дозволяє приєднання відлагоджувальника, таким чином унеможливлюючи аналіз [21]. З огляду на це, провайдери хмарних сервісів можуть цілком вимикати SGX на рівні UEFI для уникнення позовів щодо розкриття приватних даних.

Самі анклави теж можуть бути скомпрометовані, як показано в [22]: Schwarz et al. розробили програмне забезпечення, здатне непомітно проникнути у захищений SGX анклав іншої програми. Поки що не існує засобів протидіяти таким атакам, оскільки це доволі нова технологія, плюс антивіруси не мають можливості аналізувати вміст анклавів.

Після успішної ініціалізації апаратного забезпечення керування передається завантажувачу (*bootloader*): UEFI шукає на жорсткому диску спеціальний розділ і починає виконання завантажувача. Якщо увімкнений CSM, UEFI емулює BIOS і пробує завантажитись зі спеціального Boot розділу жорсткого диску. Якщо увімкнений Secure Boot (чи аналог), спершу перевіряється підпис завантажувача, щоб впевнитись в цілісності. Проте, якщо Secure Boot вимкнено або неправильно налаштовано, зловмисник може замінити завантажувач на шкідливий, отримавши повний контроль над системою. А оскільки розділом UEFI на диску є незашифрований FAT32, записати туди може будь-хто з фізичним

доступом.

Найпоширенішими завантажувачами є Bootmgr (Windows), GRUB (Linux), systemd-boot (Linux) та U-Boot (ARM).

Реальним прикладом застосування вразливості Bootmgr є BlackLotus, досліджений ESET Research [23]. Це UEFI-bootkit, який експлуатує вразливість CVE-2022-21894 [24], що дає змогу підписаному, але вразливому завантажувачу Windows обходити Secure Boot, що в свою чергу робить можливим виконання довільного коду на ранніх етапах завантаження системи. Це досягається завдяки параметру "truncatememory" у конфігурації завантаження (BCD): він дозволяє видалити з карти пам'яті області, що містять політику Secure Boot. Це призводить до того, дозволяється активація небезпечних параметрів, таких як "bootdebug", "testsigning" та "nointegritychecks", які порушують цілісність процесу завантаження, але корисні для відлагодження ядра. Після успішного впровадження BlackLotus модифікує завантажувальний процес, уникаючи засобів виявлення, а також розгортає постійний бекдор, що забезпечує подальше виконання шкідливого коду навіть після перезавантаження.

Завантажувач завантажує ядро операційної системи в оперативну пам'ять та передає йому керування. Ядро ініціалізує необхідні драйвери, налаштовує апаратні ресурси та запускає системні процеси, завершуючи процес завантаження. Використання підписаних модулів ядра та механізмів контролю цілісності допомагає запобігти виконанню несанкціонованого коду. Основною небезпекою монолітних ядер є повний доступ всіх до всіх, будь-який драйвер може прочитати будь-яку область пам'яті. На помилках коду модулів ядра якраз і ґрунтуються вразливості.

Щоб дотримуватись принципу мінімального доступу, McKee et al. [25] розробили засіб Hardware-Assisted Kernel Compartmentalization (HAKC): код та дані модулів у ядрі операційної системи розділяються в окремі секції, кожна з яких має власну політику доступу, що унеможливлює несанкціоноване зчитування або модифікацію з боку інших складових системи. У разі, коли потрібно передати керування за межі секції, право «власності» на ці дані тимчасово переходить до цільової секції, а після повернення виклику відновлюється початковий власник. Така схема дає змогу ізолювати код і дані всередині ядра без залучення додаткових елементів довіри (Trusted Computing Base) під час виконання ізольованого коду. Для мінімізації надлишкових обчислень, система HAKC покладається на апаратні механізми безпеки для дотримання визначених політик доступу. Це дозволяє посилити загальну безпеку системи, дотримуючись принципів найменших привілеїв та сегментування, за якого кожна партиція має лише ті права, які необхідні їй для реалізації покладених на неї функцій. Автори виміряли, що програми з активним HAKC сповільнюються на 1.6%–24%, при цьому захищаючи від багатьох CVE, пов'язаних з модулями ядра, або роблячи завдання зловмисника тяжче, оскільки зона доступної пам'яті значно менша.

3. Підсумок. У табл. 1 наведено визначені поверхні атак для кожного етапу ініціалізації комп'ютерної системи на основі архітектури x86-64.

Таблиця 1.

Поверхні атаки компонентів комп'ютерної системи на основі x86-64

Компонент	**Поверхня атаки**
Intel ME	Відкриті Ethernet порти: 623, 664, 5900 та 16992-16995 [4]. Шина SPI, MMIO (memory-mapped I/O).
AMD PSP	Шина SPI, MMIO [5], зокрема через драйвер \Device\amdpsp [7].
TPM	Фізичне прослуховування шин SPI, I2C, LPC [8, 9], Атаки шляхом аналізу часу обробки [10] (timing attack).
SMM	Апаратний SMI пін [11]. Спеціальний порт I/O (часто 0xB2) [12]. APIC [13]. Команда UEFI/BIOS.
UEFI	Запис модифікованої прошивки в SPI чип [15], часто з драйвером. Підміна завантажувача [15]. Підміна ресурсів завантажувача [16].
TEE	Може бути середовищем неконтрольованого виконання [21]. Можливе проникнення в чужі анклави.
Завантажувач	Підміна завантажувача [23, 24].
Ядро ОС	Системні виклики (syscall). Підміна ядра. Підміна модулів чи драйверів [25].

4. Висновки та перспективи подальших досліджень. У процесі аналізу поверхонь атаки на різні компоненти, визначено наступні методи захисту:

- налаштування зовнішнього файрволу для заборони портів 623, 664, 5900 та 16992-16995;
- фільтрація запитів до шин SPI, I2C, LPC;
- фільтрація запитів до I/O портів та APIC;
- визначення критичних областей віддзеркаленої пам'яті MMIO та фільтрація доступу до неї;
- рандомізація часу доступу до функцій TPM шляхом додавання випадкових затримок;
- заборона запису в розділ EFI;
- заборона або вимкнення TEE, або вимога явного дозволу на використання кожного конкретного випадку.

Такі методи захисту, як фільтрація, є досить складними для реалізації, на сьогодні єдиних підходів не сформовано, тому їх ефективна реалізація є предметом подальших наукових досліджень.

Список використаної літератури

1. Desktop Operating System Market Share Worldwide (2025). URL: `https://gs.statcounter.com/os-market-share/desktop/worldwide` (date of access: 28.03.2025)
2. Tanenbaum A. S., and Bos H. Modern operating systems (4th ed.). Pearson Education, 2014. 1136 p.
3. Minnich R., Shun Lim G., O'Leary R., Koch C., and Chen X. Replace your exploit-ridden firmware with a Linux kernel. 2017. URL: `https://www.mikrocontroller.net/attachment/353216/Linuxcon\_2017\_NERF.pdf`
`https://www.youtube.com/watch?v=iffTJ1vPCSo` (date of access: 20.03.2025).

4. Intel® Active Management Technology: Privacy Statement. 2023. URL: https://www.intel.com/content/www/us/en/privacy/intel-active-technology-vpro.html (date of access: 20.03.2025).
5. AMD Processor Vulnerabilities. AMD Security Bulletin AMD-SB-7009, 2023. URL: https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7009.html (date of access: 20.03.2025).
6. AMD Embedded Processors Vulnerabilities — February 2024. AMD Security Bulletin AMD-SB-5001, 2024. URL: https://www.amd.com/en/resources/product-security/bulletin/amd-sb-5001.html (date of access: 20.03.2025).
7. Economou K. Vulnerability Report. AMD, 08/04/2021. URL: https://dl.packetstormsecurity.net/2109-Advisories/AMD_PSP_Vulnerability_Report.pdf (date of access: 20.03.2025).
8. TPM Sniffing, 2021. URL: https://blog.scrt.ch/2021/11/15/tpm-sniffing (date of access: 21.02.2025).
9. Stealing the BitLocker Key from a TPM. URL: https://astralvx.com/stealing-the-bitlocker-key-from-a-tpm (date of access: 21.02.2025).
10. Moghimi, D., Sunar, B., Eisenbarth, T., & Heninger, N. TPM-FAIL: TPM meets Timing and Lattice Attacks. 2019. ArXiv. abs/1911.05673.
11. Robert R. Collins, Intel's System Management Mode. Dr. Dobb's Journal. 1997. URL: https://www.rcollins.org/ddj/Jan97/Jan97.html (date of access: 21.02.2025).
12. Masayo Y., Hiroyuki T., Computer system for reading/writing system configuration using I/O instruction. 1999. URL: https://worldwide.espacenet.com/publicationDetails/biblio?CC=US\&NR=5963738 (date of access: 01.03.2025).
13. 82093AA I/O Advanced Programmable Interrupt Controller (IOAPIC). URL: https://web.archive.org/web/20161130153145/http://download.intel.com/design/chipsets/datashts/29056601.pdf (date of access: 01.03.2025).
14. Warfield N. Know Your Enemy and Yourself: A Deep Dive on CISA KEV. 2022. URL: https://eclypsium.com/blog/know-your-enemy-and-yourself-a-deep-dive-on-cisa-kev (date of access: 01.03.2025).
15. Surve, P.P., Brodt, O., Yampolskiy, M., Elovici, Y., and Shabtai, A. SoK: Security Below the OS — A Security Analysis of UEFI. 2023. ArXiv. abs/2311.03809.
16. Finding LogoFAIL: The Dangers of Image Parsing During System Boot. 2023. URL: https://www.binarly.io/blog/finding-logofail-the-dangers-of-image-parsing-during-system-boot (date of access: 05.03.2025).
17. UEFI Secure Boot in Linux. Intel Developer Zone. 2013. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/sf13-stts002-100p-820238.pdf (date of access: 05.03.2025).
18. Costan V., and Devadas S. Intel SGX explained. IACR Cryptology ePrint Archive. 2016. 086.
19. Paj A., Javed M. O., Nurmi J., Savimäki J., McGillion B., and Brumley B. B. SoK: A Systematic Review of TEE Usage for Developing Trusted Applications. *Proceedings of the 18th International Conference on Availability, Reliability and Security.* 2023. pp. 1–15.
20. Zhao S., Zhang Q., Qin Y., Feng W., and Feng D. Minimal kernel: an operating system architecture for TEE to resist board level physical attacks. *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019).* 2019. pp. 105–120.
21. Zhang Z., Zhang X., Li Q., Sun K., Zhang Y., Liu S., Liu Y., and Li X. See through Walls: Detecting Malware in SGX Enclaves with SGX-Bouncer. *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security.* 2021.
22. Schwarz M., Weiser S., and Gruss D. Practical Enclave Malware with Intel SGX. *International Conference on Detection of intrusions and malware, and vulnerability assessment.* 2019.
23. BlackLotus UEFI bootkit: Myth confirmed. ESET. 2023. URL: https://www.welivesecurity.com/2023/03/01/blacklotus-uefi-bootkit-myth-confirmed (date of access: 07.03.2025).
24. CVE-2022-21894: Bypassing Windows Secure Boot with BCD settings. 2022. URL: https://github.com/Wack0/CVE-2022-21894 (date of access: 07.03.2025).
25. McKee D., Giannaris Y., Ortega C., Shrobe H., Payer M., Okhravi H., and Burow N. Preventing Kernel Hacks with HAKC. *NDSS Symposium.* 2022.

Bogdiuk M. O. Analysis of the Boot Process of Computer Systems Based on the x86-64 Architecture for Vulnerabilities and Identification of Measures to Protect Against Them.

The article presents a systematic analysis of the boot process of modern x86-64 computer systems, with a focus on identifying potential vulnerabilities that may emerge during its initialization. It explores methods of firmware modification, low-level execution modes, and weaknesses in integrity verification procedures. Particularly, most attention is devoted to attack vectors designed to circumvent built-in security. Based on a comprehensive review of recent scientific literature, the study identifies potential vulnerabilities of the components of the boot sequence and suggests ways to isolate them from potential threats (e.g. dangerous applications). The study outlines directions for future research in the domain of secure system boot processes.

Keywords: boot process, firmware modification, vulnerability, bypass built-in security, isolation.

References

1. Desktop Operating System Market Share Worldwide (2025). Retrieved from https://gs.statcounter.com/os-market-share/desktop/worldwide
2. Tanenbaum, A. S., & Bos, H. (2014). *Modern operating systems (4th ed.).* Pearson Education.
3. Minnich, R., Shun Lim, G., O'Leary, R., Koch, C., & Chen, X. (2017). Replace your exploit-ridden firmware with a Linux kernel. Retrieved from https://www.mikrocontroller.net/attachment/353216/Linuxcon_2017_NERF.pdf
https://www.youtube.com/watch?v=iffTJ1vPCSo
4. Intel® Active Management Technology: Privacy Statement. (2023). Retrieved from https://www.intel.com/content/www/us/en/privacy/intel-active-technology-vpro.html
5. AMD Processor Vulnerabilities. (2023). AMD Security Bulletin AMD-SB-7009. Retrieved from https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7009.html
6. AMD Embedded Processors Vulnerabilities — February 2024. AMD Security Bulletin AMD-SB-5001. Retrieved from https://www.amd.com/en/resources/product-security/bulletin/amd-sb-5001.html
7. Economou, K. (2021). Vulnerability Report. AMD, 08/04/2021. Retrieved from https://dl.packetstormsecurity.net/2109-Advisories/AMD_PSP_Vulnerability_Report.pdf
8. TPM Sniffing. (2021). Retrieved from https://blog.scrt.ch/2021/11/15/tpm-sniffing
9. Stealing the BitLocker Key from a TPM. Retrieved from https://astralvx.com/stealing-the-bitlocker-key-from-a-tpm
10. Moghimi, D., Sunar, B., Eisenbarth, T., & Heninger, N. (2019). TPM-FAIL: TPM meets Timing and Lattice Attacks. ArXiv, abs/1911.05673.
11. Robert R. Collins. (1997). Intel's System Management Mode. *Dr. Dobb's Journal.* Retrieved from https://www.rcollins.org/ddj/Jan97/Jan97.html
12. Masayo, Y., Hiroyuki, T. (1999). Computer system for reading/writing system configuration using I/O instruction. Retrieved from https://worldwide.espacenet.com/publicationDetails/biblio?CC=US\&NR=5963738
13. Intel Corporation. 82093AA I/O Advanced Programmable Interrupt Controller (IOAPIC). Retrieved from https://web.archive.org/web/20161130153145/http://download.intel.com/design/chipsets/datashts/29056601.pdf
14. Warfield, N. (2022). Know Your Enemy and Yourself: A Deep Dive on CISA KEV. Retrieved from https://eclypsium.com/blog/know-your-enemy-and-yourself-a-deep-dive-on-cisa-kev
15. Surve, P. P., Brodt, O., Yampolskiy, M., Elovici, Y., & Shabtai, A. (2023). SoK: Security Below the OS — A Security Analysis of UEFI. ArXiv, abs/2311.03809.
16. Binarly Research Team. (2023). Finding LogoFAIL: The Dangers of Image Parsing During System Boot. Retrieved from https://www.binarly.io/blog/finding-logofail-the-dangers-of-image-parsing-during-system-boot
17. Intel Corporation. UEFI Secure Boot in Linux. Intel Developer Zone (2013). Retrieved from

https://www.intel.com/content/dam/develop/external/us/en/documents/sf13-stts002-100p-820238.pdf
18. Costan, V., & Devadas, S. (2016). Intel SGX explained. IACR Cryptology ePrint Archive, 2016, 086.
19. Paju, A., Javed, M. O., Nurmi, J., Savimäki, J., McGillion, B., & Brumley, B. B. (2023). SoK: A Systematic Review of TEE Usage for Developing Trusted Applications. In *Proceedings of the 18th International Conference on Availability, Reliability and Security.*
20. Zhao, S., Zhang, Q., Qin, Y., Feng, W., & Feng, D. (2019). Minimal kernel: an operating system architecture for TEE to resist board level physical attacks. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019).*
21. Zhang, Z., Zhang, X., Li, Q., Sun, K., Zhang, Y., Liu, S., Liu, Y., & Li, X. (2021). See through Walls: Detecting Malware in SGX Enclaves with SGX-Bouncer. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security.*
22. Schwarz, M., Weiser, S., & Gruss, D. (2019) Practical Enclave Malware with Intel SGX. In *International Conference on Detection of intrusions and malware, and vulnerability assessment.*
23. ESET Research. (2023). BlackLotus UEFI bootkit: Myth confirmed. ESET. Retrieved from https://www.welivesecurity.com/2023/03/01/blacklotus-uefi-bootkit-myth-confirmed
24. CVE-2022-21894: Bypassing Windows Secure Boot with BCD settings. (2022). Retrieved from https://github.com/Wack0/CVE-2022-21894
25. McKee, D., Giannaris, Y., Ortega, C., Shrobe, H., Payer, M., Okhravi, H., & Burow, N. (2022). Preventing Kernel Hacks with HAKC. *NDSS Symposium.*

Одержано 10.04.2025