УДК 004.056

ЗАХИСТ ТРАНЗАКЦІЙ MODBUS ЗА ДОПОМОГОЮ HMAC ТА SCTP

М. В. Правдюк[1,а], О. М. Барановський[1,б]

[1] Навчально-науковий Фізико-технічний інститут

Анотація

У роботі розглянуто актуальну проблему забезпечення кібербезпеки промислових протоколів передачі даних. Зокрема, проаналізовано вразливості протоколу Modbus, що широко використовується в системах промислової автоматизації. З метою підвищення стійкості транзакцій Modbus до потенційних загроз запропоновано нову архітектуру, яка передбачає використання протоколу транспортного рівня SCTP та застосування криптографічного механізму автентифікації HMAC. Для верифікації ефективності запропонованого підходу було проведено серію експериментальних досліджень в умовах змодельованих кібератак.

Ключові слова: Modbus, кібербезпека, SCTP, HMAC, автентифікація, промислові протоколи, криптографія, кібератаки

Вступ

Modbus — це відкритий протокол прикладного рівня, який набув широкого поширення у сфері SCADA-систем, промислової автоматизації та індустріального Інтернету речей (IoT). Його популярність обумовлена простотою реалізації, гнучкістю адаптації до різних типів інтерфейсів (RS-232, RS-485, TCP/IP) та відкритістю специфікації, що забезпечує сумісність між різними виробниками обладнання.

Однак, попри ці переваги, класичні реалізації протоколу, зокрема Modbus TCP, не мають вбудованих механізмів кіберзахисту. Усі передані дані не шифруються, не автентифікуються і не перевіряються на цілісність. Це створює значні ризики при використанні Modbus у критичних інфраструктурах — зокрема в енергетиці, виробництві, транспорті та водопостачанні.

Переваги	**Недоліки**
• Простота реалізації • Адаптація до RS-232, RS-485, TCP/IP • Відкритість специфікації • Сумісність між виробниками	• Відсутність шифрування даних • Немає автентифікації повідомлень • Відсутня перевірка цілісності • Ризик атак типу Man-in-the-Middle

Таблиця 1. Переваги та недоліки протоколу Modbus

Зі зростанням кількості інцидентів у сфері промислової кібербезпеки, проблематика захисту протоколу Modbus набуває особливої актуальності. У 2024 році було зафіксовано серію атак типу «людина посередині» (Man-in-the-Middle, MiTM) на енергетичні об'єкти, які використовували незахищені канали з Modbus TCP [1]. Ці атаки дозволяли зловмисникам змінювати критично важливі параметри роботи обладнання без виявлення втручання з боку систем безпеки.

У цьому контексті виникає необхідність розробки рішень, що підвищують захищеність протоколу Modbus без радикальної зміни його структури або втрати сумісності. Метою цієї роботи є створення архітектури захищеної транзакції Modbus, яка базується на поєднанні двох підходів:

- використанні SCTP (Stream Control Transmission Protocol) як транспортного протоколу, що надає розширені можливості з контролю зв'язку, захисту від спуфінгу, підтримки мультистримінгу та захисту від DoS-атак;
- інтеграції HMAC (Hash-Based Message Authentication Code) як криптографічного механізму для перевірки цілісності та автентичності повідомлень, що передаються в рамках транзакцій Modbus.

1. Запропоноване рішення

Для усунення критичних вразливостей Modbus TCP (відсутність автентифікації, контроль цілісності, захист від атак) запропоновано гібридне рішення:

- заміна TCP на SCTP як транспортного протоколу,
- додавання HMAC-SHA256 до кожного Modbus-повідомлення.

[а] email1@mail.com
[б] email2@mail.com

1.1. Використання SCTP як транспортного рівня

SCTP (Stream Control Transmission Protocol) — транспортний протокол, розроблений IETF [2], який має ряд технічних переваг над TCP, критично важливих для захисту.

Переваги SCTP:

- 4-way handshake: механізм встановлення з'єднання із cookie-обміном — захист від SYN Flood.
- Association verification: перевірка IP та портів обох кінців — зменшує ризик IP-spoofing.
- Multi-streaming: до 65 535 незалежних потоків в одному з'єднанні — запобігає Head-of-Line blocking.
- Multi-homing: можливість використовувати кілька IP-адрес на обох сторонах — підвищення стійкості до DoS.
- Partial reliability: контроль над тим, які дані мають бути доставлені строго, а які — опційно (для real-time Modbus RTU over IP).

Практична реалізація:

- Сервер і клієнт повинні мати встановлену SCTP-підтримку (на рівні ОС). Наприклад, в Linux — модуль `lksctp`.
- У Python — через бібліотеку `pysctp` або сокет-обгортку з підтримкою `AF_INET/SOCK_SEQPACKET/SOL_SCTP`.
- Режим роботи: `one-to-one`, з активною перевіркою IP/портів під час ініціалізації асоціації.

Використання:

У Modbus TCP структура повідомлення зберігається, але транспортується через SCTP-пакети. Після встановлення асоціації дані передаються як message-oriented пакети без необхідності розмітки довжини.

1.2. HMAC для перевірки автентичності та цілісності

HMAC (Hash-based Message Authentication Code) — криптографічний механізм, який забезпечує автентичність та цілісність повідомлення шляхом використання секретного ключа [3].

$$\mathrm{HMAC}(K, M) = H\left((K \oplus \mathrm{opad}) \parallel H\left((K \oplus \mathrm{ipad}) \parallel M\right)\right) \tag{1}$$

де:

- K — секретний симетричний ключ;
- M — повідомлення (Modbus TCP payload);
- H — хеш-функція SHA-256;
- `opad`, `ipad` — зовнішнє та внутрішнє заповнення згідно зі специфікацією HMAC.

Технічні параметри

- Довжина HMAC: 256 біт (32 байти)
- Секретний ключ: $\geq$ 128 біт (рекомендовано 256 біт)
- Розташування: HMAC додається в кінець кожного Application Data Unit (ADU)

Алгоритм перевірки

1. Отримати повідомлення з доданим HMAC
2. Відокремити HMAC від payload
3. Обчислити $\mathrm{HMAC}(K, M)$ згідно (1)
4. Порівняти з отриманим значенням
5. У разі невідповідності — відкинути повідомлення

Захисні властивості

- Захист від MITM-атак: неможливість генерації HMAC без ключа
- Захист від replay-атак: використання таймштампів або nonce у M
- Захист від модифікації: зміна M призводить до невідповідності HMAC

Практична реалізація

Мова програмування	**Python**
Бібліотека	`hmac.new(key, msg, hashlib.sha256).digest()`
Керування ключами	Попередній обмін або захищений канал
Інтеграція	Модифікація драйвера або проксі-рішення

2. Експериментальна перевірка

2.1. Інфраструктура експерименту

Master-пристрій: ноутбук з ОС Ubuntu 22.04, Python 3.10, бібліотека `pymodbus` версії 3.0.0.

Slave-пристрій: віртуальна машина (VirtualBox), на якій запущено сервер на Python з прийомом SCTP-пакетів через модуль `socket`.

Комунікаційний протокол: SCTP в режимі `one-to-one` (`SOCK_SEQPACKET`), використано бібліотеку `pysctp`.

HMAC: реалізація з використанням Python-бібліотеки `hmac`, хеш-функція — SHA-256 з модуля `hashlib`.

Інструменти моніторингу: Wireshark 4.0.0 для перехоплення SCTP-трафіку, `time` та `perf` для вимірювання продуктивності.

2.2. Конфігурація HMAC

- **Довжина секретного ключа K:** 256 біт (32 байти), збережений як байтовий рядок у HEX-форматі
- **Повідомлення M:** тіло Modbus ADU, яке включає заголовок та PDU
- **opad та ipad:** фіксовані байтові послідовності ($0x5c$ і $0x36$ відповідно, повторені до довжини блоку SHA-256 — 64 байти)
- **Позиція HMAC у повідомленні:** додається як останні 32 байти ADU

Перевірка HMAC виконується шляхом повторного обчислення на стороні Slave та байтового порівняння з отриманим значенням.

2.3. Конфігурація SCTP

- **Використано:** `AF_INET/SOCK_SEQPACKET/IPPROTO_SCTP`
- **Режим:** `one-to-one` асоціація (аналог TCP)
- **Кількість потоків (streams):** встановлено `in_streams=2`, `out_streams=2`
- **Контроль доставки:** порядок зберігається в межах потоку, між потоками — незалежно
- **Багатогомінг:** не активований (але підтримується)
- **Додатково:** для перевірки цілісності асоціацій використано `SCTP_AUTHENTICATION_EVENT`

2.4. Хід експерименту

1. Master створює Modbus ADU (Function Code 0*x*03, Read Holding Registers).
2. До ADU додається HMAC-SHA256 на основі спільного ключа.
3. Повідомлення надсилається через SCTP-сокет.
4. Slave отримує пакет, витягує HMAC та payload.
5. Slave обчислює локальний HMAC і порівнює з отриманим.
6. Якщо значення збігається — запит обробляється; інакше — ігнорується.

2.5. Результати

- **Середній час обчислення HMAC (SHA-256):** 46–62 μs.
- **Розмір накладних витрат:** +32 байти до кожного повідомлення (256 біт).
- **Стійкість до MITM:** перехоплене повідомлення з підробленим HMAC було успішно відкинуто.
- **Стійкість до Replay-атак:** у конфігурації з timestamp (типу Unix epoch) в payload було реалізовано тайм-ліміт у 2 секунди.
- **Робота SCTP:** після штучного перезапуску інтерфейсу мережі, SCTP автоматично перепідключився через альтернативну IP-адресу (у режимі багатогомінгу).
- **Порівняння з TCP:** TCP в аналогічному сценарії втратив з'єднання; SCTP продовжив передачу без втрати асоціації.

Висновки

Огляд актуального стану безпеки протоколу Modbus TCP засвідчив відсутність вбудованих механізмів автентифікації та цілісності даних, що створює передумови для широкого спектра атак. У роботі запропоновано інтегроване рішення, яке поєднує:

- Використання HMAC для перевірки достовірності Modbus-повідомлень
- Протокол SCTP як захищенішу альтернативу TCP

Експериментальна реалізація демонструє такі результати:

- Успішне виявлення підроблених/модифікованих повідомлень
- Стійкість до атак типу MiTM (*man-in-the-middle*)
- Захист від replay-атак за рахунок використання timestamp
- Середній час обробки HMAC: 46–62 μs
- Накладні витрати: лише +32 байти на повідомлення

Запропонований підхід пропонує оптимальний баланс між:

- Рівнем безпеки
- Продуктивністю системи
- Простотою інтеграції в існуючу інфраструктуру

Перспективи подальших досліджень включають:

- Адаптацію рішення для інших промислових протоколів
- Інтеграцію з системами керування ключами
- Оптимізацію продуктивності для високонавантажених систем

Перелік використаних джерел

1. *Constantin L.* ICS malware FrostyGoop disrupted heating in Ukraine, remains threat to OT worldwide. — 06/23/2024. — URL: https://www.csoonline.com/article/3476858/ics-malware-frostygoop-disrupted-heating-in-ukraine-remains-threat-to-ot-worldwide.html.
2. *Stewart R.* Stream Control Transmission Protocol. — 12/28/2007. — URL: https://tools.ietf.org/html/rfc4960.
3. *Okta.* HMAC (Hash-Based Message Authentication Codes) Definition. — 08/28/2024. — URL: https://www.okta.com/identity-101/hmac/.