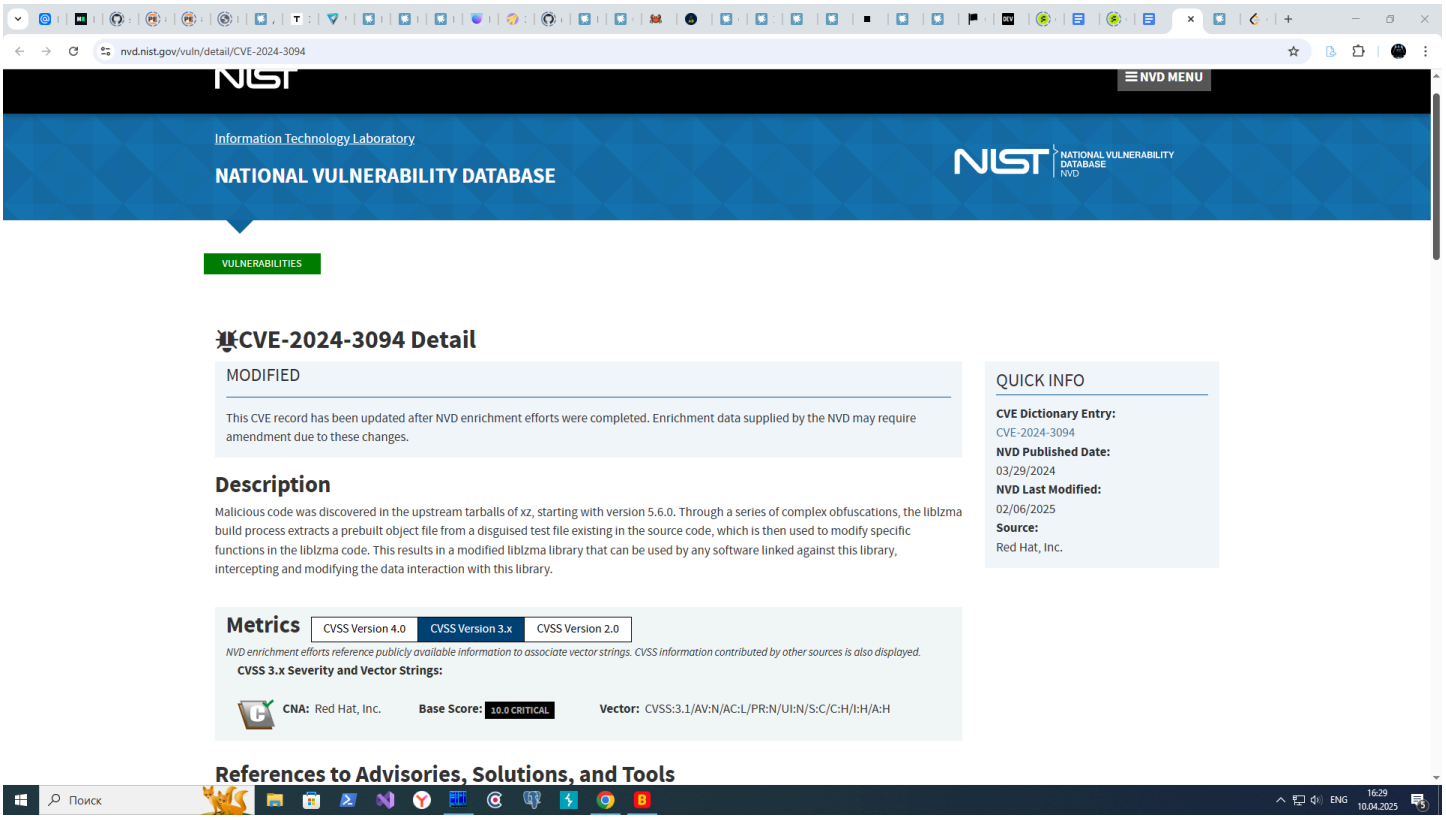
## Часть 1

### Выбор CVE

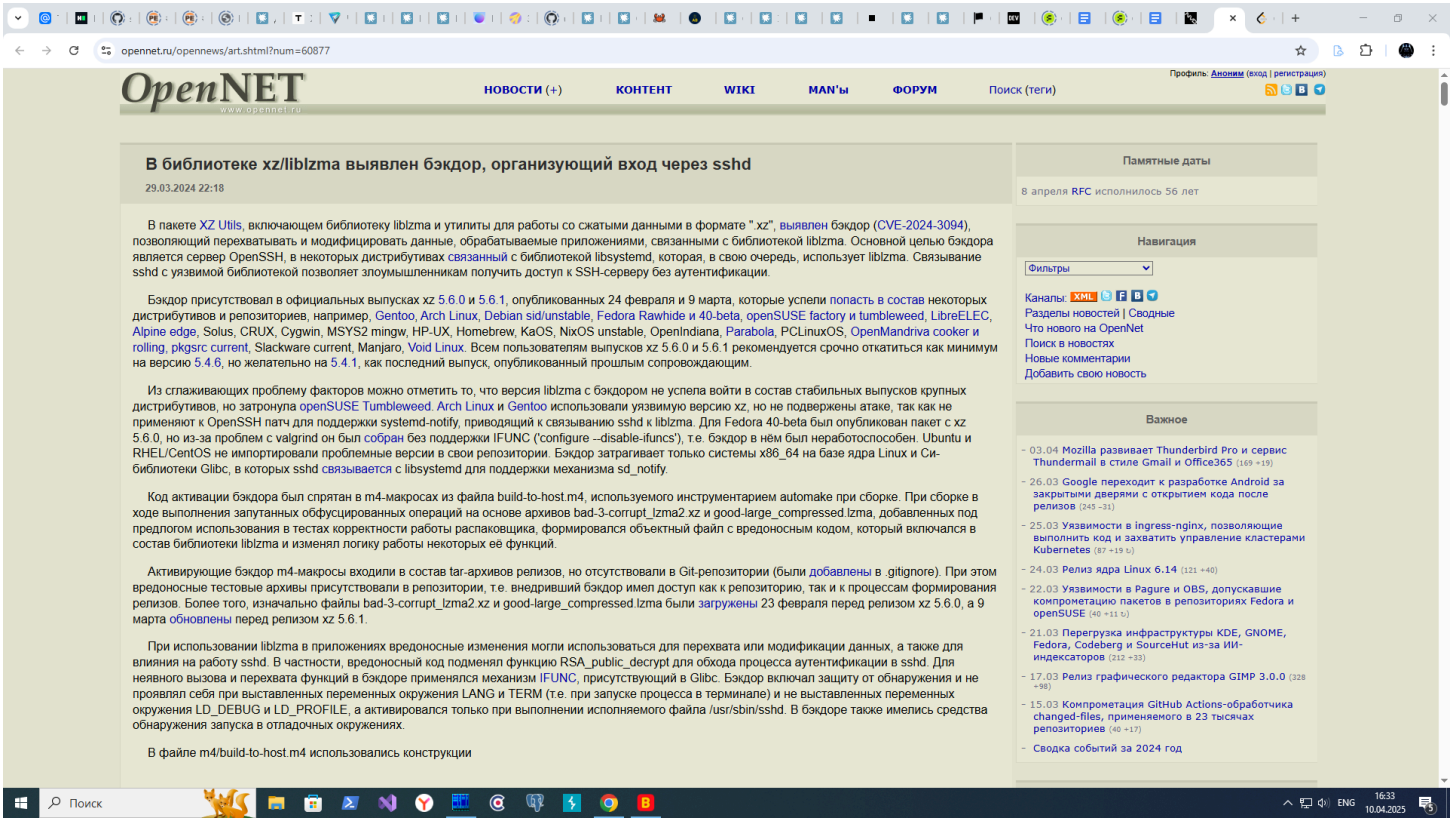
1. Для рассмотрения беру CVE-2024-3094 (CVE от 2024 года под номером 3094)

2. Описание уязвимости из базы NVD:

Вредоносный код был обнаружен в tarballs xz, начиная с версии 5.6.0. С помощью серии сложных обфускаций в процессе сборки liblzma извлекается уже собранный объектный файл из замаскированного тестового файла, существующего в исходном коде, который затем используется для изменения определенных функций в коде liblzma. Это приводит к изменению библиотеки liblzma, которые могут быть использованы любым программным обеспечением, связанным с этой библиотекой, перехватывая и изменяя взаимодействие данных с этой библиотекой.
Это позволяет выполнить произвольный код при использовании уязвимой версии в таких сервисах, как sshd (через зависимость liblzma).



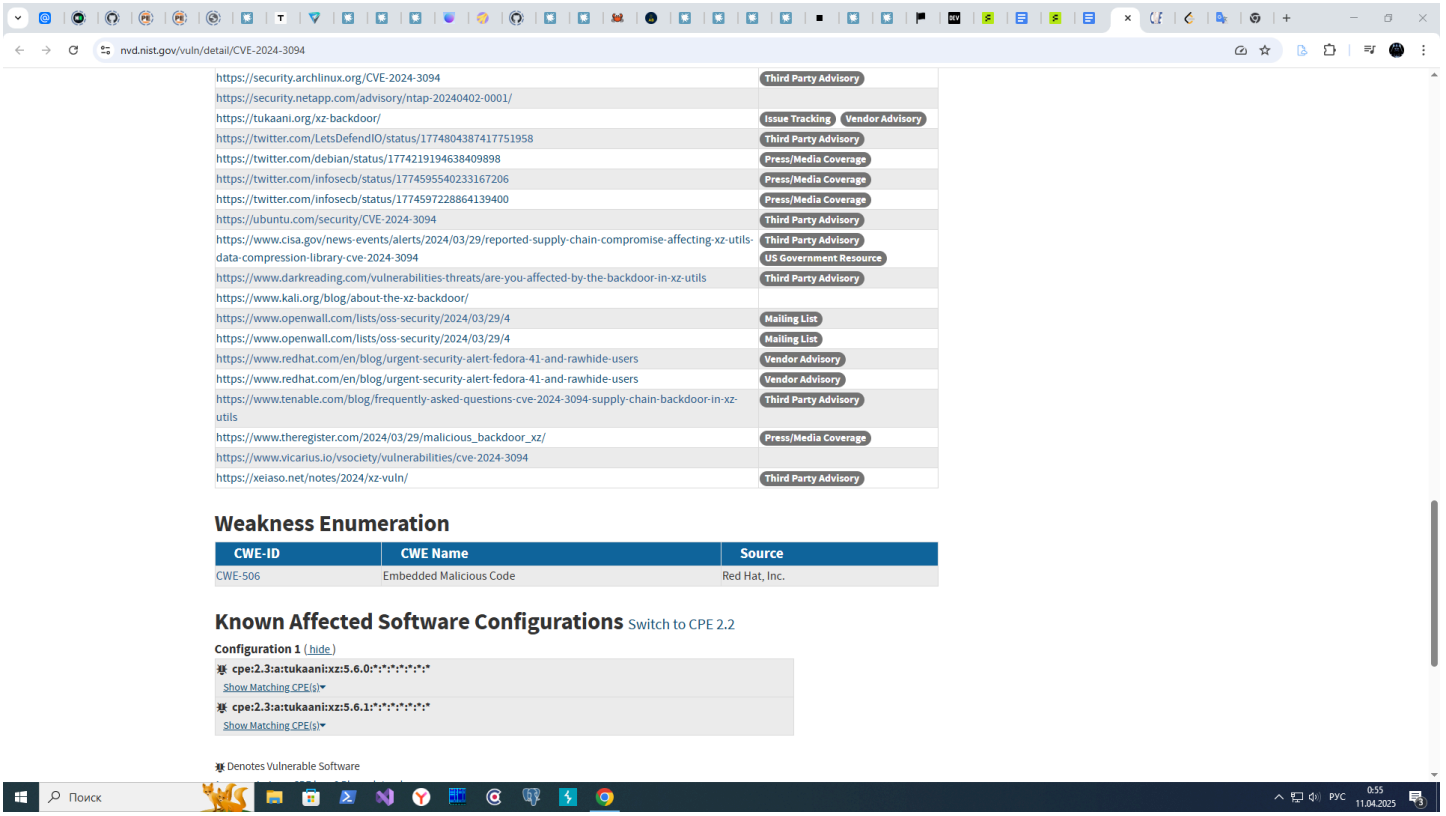

Вот, например, описание этой уязвимости на OpenNET (https://www.opennet.ru/opennews/art.shtml?num=60877):



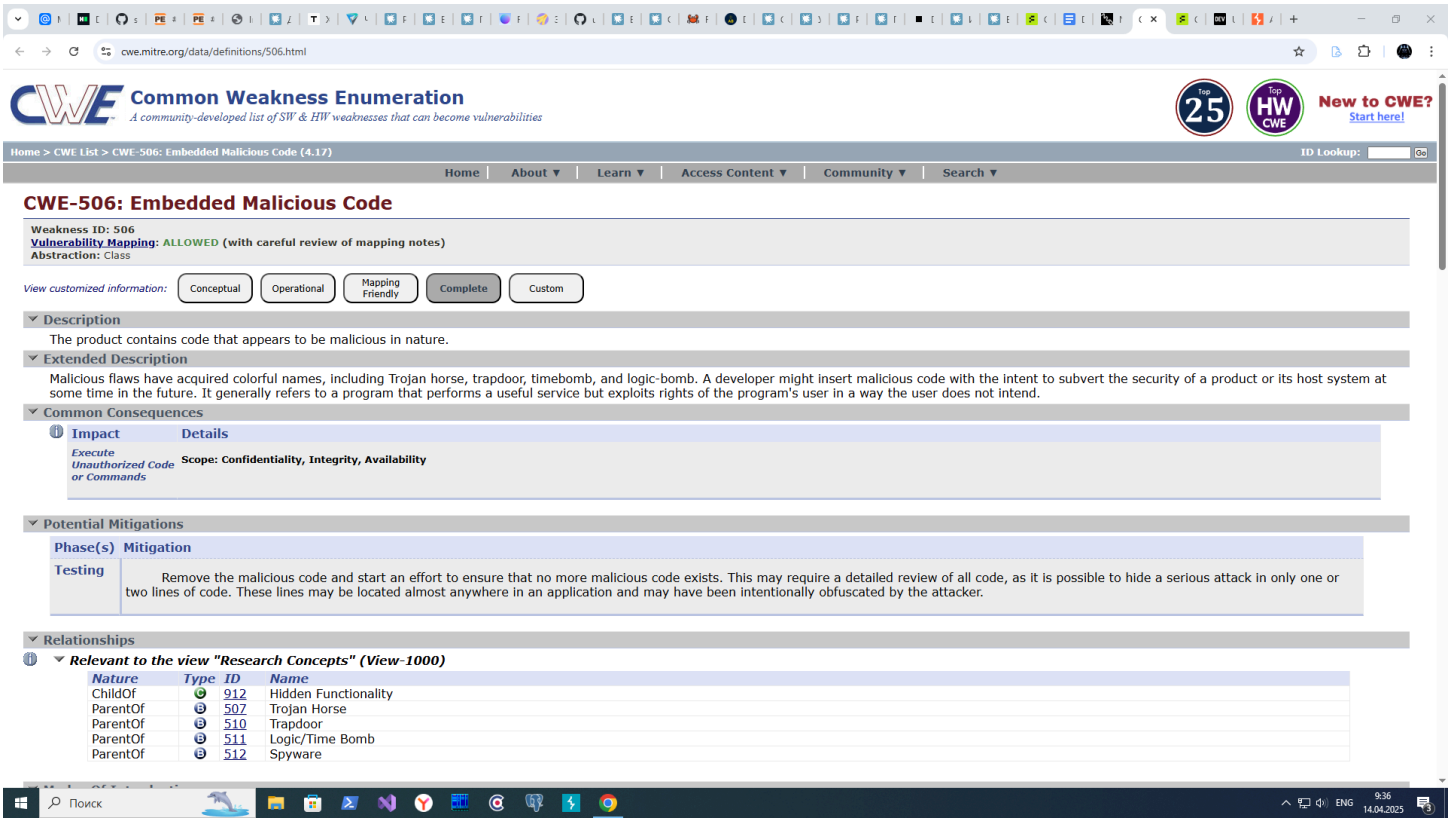

3. Определяю CWE:

CWE-506: Embedded Malicious Code (Встроенный вредоносный код)

4. Смотрю, к каким CWE относится данная CVE:
CWE-506: Embedded Malicious Code (Встроенный вредоносный код)



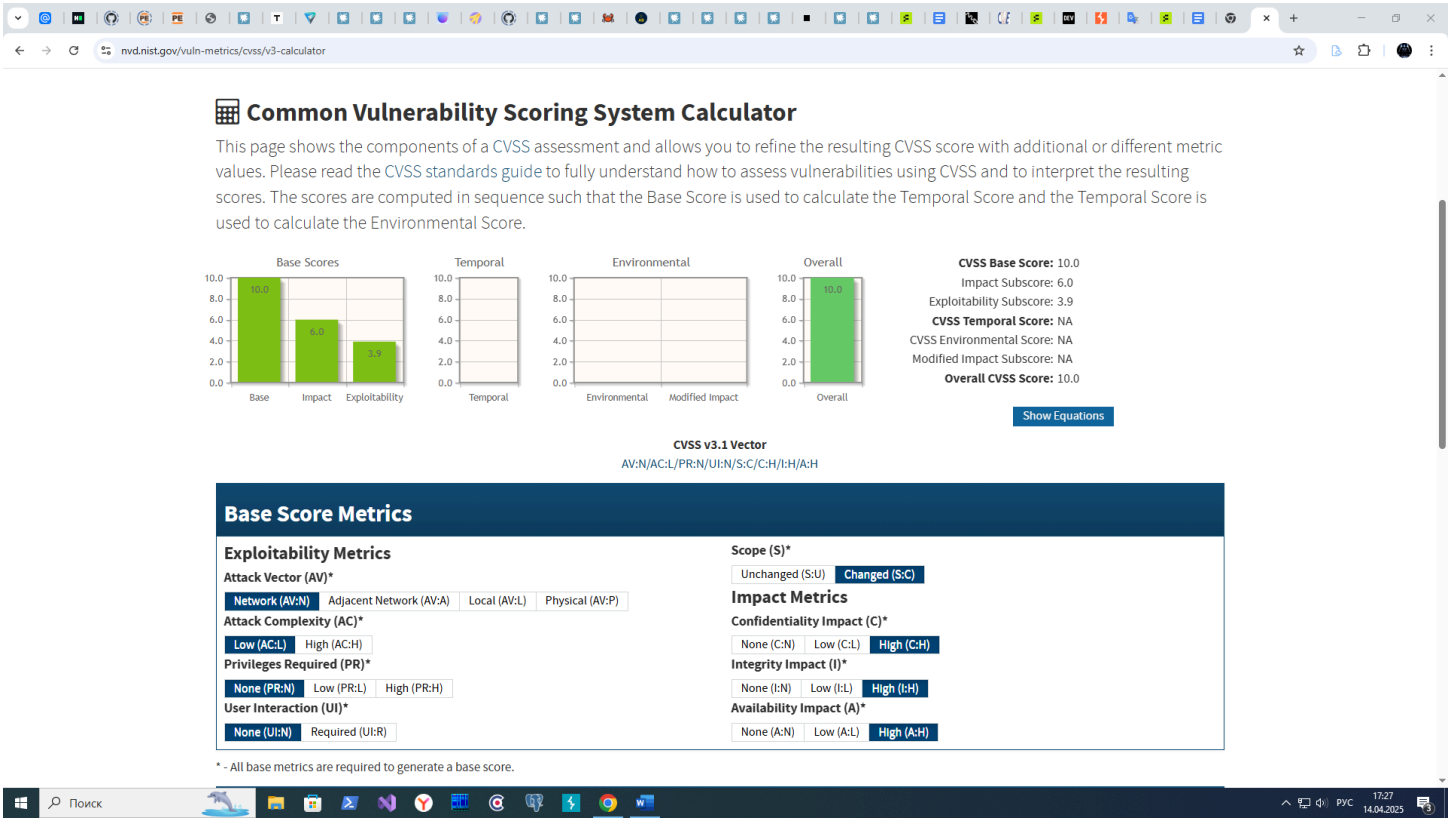

5. Краткое описание CWE, связанный с этой уязвимостью:
Продукт содержит код, который по своей природе кажется вредоносным. Намеренное внедрение вредоносного кода в легитимное ПО.




## Оценка по CVSS

1. Оценка с помощью калькулятора CVSS v3
У CVE-2024-3094 (на https://nvd.nist.gov/vuln/detail/CVE-2024-3094) следующая оценка:
Base Score: 10.0 CRITICAL
Vector:  CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
Мой расчет с помощью калькулятора Common Vulnerability Scoring System Calculator:

| Метрика | Значение | Пояснение |
|---|---|---|
| Attack Vector (AV) | Network (AV:N) | Уязвимость может эксплуатироваться удаленно - через sshd (сетевой сервис). |
| Attack Complexity (AC) | Low (AC:L) | Для эксплуатации не требуется сложных условий — достаточно использовать liblzma с внедренным вредоносным кодом. |
| Privileges Required (PR) | None (PR:N) | Эксплуатация не требует никаких привилегий. |
| User Interaction (UI) | None (UI:N) | Эксплуатация не требует никаких действий пользователя. |
| Scope (S) | Changes (S:C) | Для эксплуатации уязвимости требуется наличие liblzma с внедренным вредоносным кодом. |
| Confidentiality Impact (C) | High (C:H) | При эксплуатации уязвимости злоумышленник получает доступ к любым данным в системе |
| Integrity Impact (I) | High (I:H) | При эксплуатации уязвимости злоумышленник получает возможность модифицировать любые данные в системе |
| Availability Impact (A) | High (A:H) | При эксплуатации уязвимости злоумышленник получает возможность выполнения любых действий в системе |




2. Разбор метрик CVSS:3.1:

- Attack Vector = Network: уязвимость может эксплуатироваться удаленно, через сетевой сервис - например, через sshd;
- Attack Complexity = Low: для эксплуатации не требуется сложных условий — достаточно использовать liblzma с внедренным вредоносным кодом;
- Privileges Required = None: для эксплуатации уязвимости не требуются никакие привилегии;
- User Interaction = None: для эксплуатации уязвимости не требуется никаких действий пользователя;
- Scope = Changes: для эксплуатации уязвимости требуется наличие liblzma с внедренным вредоносным кодом;
- Confidentiality Impact = High: при эксплуатации уязвимости злоумышленник получает доступ к любым данным в системе;
- Integrity Impact = High: при эксплуатации уязвимости злоумышленник получает возможность модифицировать любые данные в системе;
- Availability Impact = High: при эксплуатации уязвимости злоумышленник получает возможность выполнения любых действий в системе (RCE).

3. Итоговый балл и уровень риска:
10.0 CRITICAL
Причины:

- Возможность атаки через сеть;
- Простота использования уязвимости;
- Отсутствие дополнительных требований (к привилегиям, к взаимодействию с пользователем);
- Максимальное воздействие на всю систему (конфиденциальность, целостность, доступность).

**Описание последствий:**

| Угроза | Описание | Последствия |
|---|---|---|
| RCE | Злоумышленник может удаленно выполнить любые команды на уязвимом сервере через liblzma с внедренным вредоносным кодом - например, через sshd сервис | Полный доступ к уязвимому серверу (root-права), возможность выполнения произвольных команд (произвольных действий) |
| Нарушение конфиденциальности | Злоумышленник может читать любые файлы на сервере | Могут быть украдены пароли, SSH-ключи, токены API, базы данных, платежные реквизиты, персональные данные, конфиги облачных сервисов |
| Нарушение целостности | Злоумышленник может изменять любые файлы на сервере | Могут быть изменены данные в базах данных, содержимое произвольных файлов, а также подписи для этих данных (при наличии соответствующих ключей) |
| Нарушение доступности | Злоумышленник может изменять доступность любых сервисов на сервере | Доступность любых сервисов на сервере может быть изменена из-за их включения/отключения, изменений правил firewall и т.д. |
| Распространение атаки на другие узлы в локальной сети | Злоумышленник может использовать скомпрометированный (захваченный) узел, как плацдарм для атаки на другие узлы в сети | Частичная или полная компрометация инфраструктуры локальной сети. Частичная или полная компрометации криптографических ключей |
| Использование устройства в качестве узла ботнета | Злоумышленник может установить дополнительно скрытое ПО для дальнейшего использования скомпрометированного (захваченного) узла в своих целях | Использование скомпрометированного (захваченного) узла, в качестве узла ботнета или для майнинга криптовалют |

**Вывод:**

1. Согласно блогу Лаборатории Касперского, проблемными являются XZ Utils версий 5.6.0 и 5.6.1 (https://www.kaspersky.ru/blog/cve-2024-3094-vulnerability-backdoor/37222/). Поэтому самым простым и логичным вариантом является использование более новых версий XZ Utils (сейчас это версия 5.8.1). В крайнем случае, если нет возможности по каким-либо причинам использовать более новую версию XZ Utils, можно использовать XZ Utils версии до 5.6.0 (хотя там тоже есть проблемы).
2. Если игнорировать наличие CVE-2024-3094 в XZ Utils, то легко можно прийти к ситуации, когда сам узел и данные на нем полностью будут под контролем злоумышленника

**Часть 2**

Безопасная разработка веб-приложений (общие принципы):

| Действие | Описание |
|---|---|
| Создание архитектуры с учетом требований DevSecOp | Необходимо изначально создавать архитектуру приложения исходя из наличия системы безопасности и требований DevSecOp (даже на стадии MVP). Иначе в противном случае можно получить ситуацию, когда система безопасности работает некорректно или неэффективно в каких-то случаях.<br>https://owasp.org/Top10/A04_2021-Insecure_Design/ |
| Использование стойких средств для аутентификации и авторизации | Необходимо использовать правильные (стойкие) средства для аутентификации и авторизации. Это относится к целому ряду подходов:<br>● управление учетными записями<br>● управление паролями (сложность, время жизни, повторное использование)<br>● хранение паролей (отказ от хранения паролей в открытом виде)<br>● 2FA<br>● использование защищенных токенов (например, JWT токенов с подписью)<br>● ограничение времени жизни пользовательской сессии<br>● использование oauth для доступа к нескольким ресурсам в рамках одной сессии<br>● и т.д.<br>Так, например, явно не стоит использовать basic access authentication в качестве подхода к аутентификации.<br>https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/ |
| Внедрение ролевой модели. Использование принципа минимальных привилегий | Необходимо использовать ролевую модель в приложении: описать роли и необходимые им права (с учетом доменной модели). При этом права для каждой роли должны быть минимально возможны. Необходимо запретить выдачу прав напрямую учетным записям - права у учетных записей должны появляться за счет членства в той или иной роли.<br>PS. Если ролевой модели будет недостаточно по каким-либо причинам, то следует использовать разграничение доступа на основе атрибутов.<br>https://owasp.org/Top10/A01_2021-Broken_Access_Control/ |
| Разделение API (на backend) с учетом ролевой модели | Необходимо методы API (на backend) сгруппировать вместе в зависимости от роли, необходимой для их выполнения (на уровне путей в url). Реализация методов API (на backend), относящихся к разным ролям может быть выполнена в разных микросервисах для физического разделения таких методов API.<br>https://owasp.org/Top10/A01_2021-Broken_Access_Control/ |
| Использование правильных средств работы с БД | Правильный подход означает, что мы должны использовать такие средства (такие библиотеки), которые исключают возможность SQL инъекций. Так, например, вместо формирования запроса вручную |

| | |
|---|---|
| | на основе пользовательского ввода, необходимо использовать параметризованные запросы<br>https://owasp.org/Top10/A03_2021-Injection/ |
| Использование правильного подхода при работе с пользовательским вводом | При обработке пользовательского ввода следует относится к нему, как если он сформирован злоумышленником: пользовательский ввод всегда должен проверяться и обрабатываться так, чтобы исключить возможность какого-либо несанкционированного воздействия - например, SQL инъекции, XSS инъекции, инъекции команд и т.д. В качестве примера такой обработки можно привести использование параметризованных запросов для работы с БД, а не ручное формирование запросов на основе пользовательского ввода.<br>https://owasp.org/Top10/A03_2021-Injection/ |
| Использование логирования | Необходимо использовать логирование для всех ошибок и внештатных ситуаций, происходящих в приложении. Также необходимо логировать все действия, которые выполняются с повышенными привилегиями.<br>https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/ |
| Создание и использование автотестов | Автотесты - это очень мощный подход для автоматической проверки работы той или иной функциональности, в том числе и для выявления регрессий. Очевидно, что автотесты должны быть созданы не только на основную функциональность, но и на аспекты, связанные с безопасностью. Очень важно, чтобы тесты создавались (когда это возможно) на все выявленные проблемы с безопасностью, чтобы в будущем не столкнуться с регрессией этих проблем. |
| Использование CI/CD | Необходимо использовать CI/CD для автоматизации сборки, выполнения автотестов, а также тестирования SAST, DAST, SCA на регулярной основе |
| Использование https протокола | На самом деле это относится больше к конфигурированию веб приложения (т.к. никак не отражается на подходах к разработке), но тем не менее - обязательно использовать протокол https (TLS 1.3) для защиты передаваемой информации. |
| Использование SAST для тестирования исходного кода | Необходимо использовать SAST для тестирования исходного кода с помощью некоторого набор заданных правил/метрик, в том числе и связанных с безопасностью. Необходимо выполнять это тестирование на регулярной основе (если добавить этот шаг, как шаг в CI/CD, то данное тестирование будет происходить автоматически). |
| Использование DAST для тестирования готового приложения | Необходимо использовать DAST для тестирования создаваемого приложения с помощью некоторого набор заданных правил/метрик, в том числе и связанных с безопасностью. Необходимо выполнять это тестирование на регулярной основе (если добавить этот шаг, как шаг в CI/CD, то данное тестирование будет происходить автоматически). |
| Использование SCA для | Необходимо использовать SCA для анализа (проверки) |

| | |
|---|---|
| анализа зависимостей | используемых зависимостей. Необходимо выполнять это тестирование на регулярной основе (если добавить этот шаг, как шаг в CI/CD, то данное тестирование будет происходить автоматически). |
| Использование открытых источников (например, NVD) для поиска информации о новых уязвимостях | Необходимо на регулярной основе делать поиск по открытым источникам (например, NVD) информации о новых обнаруженных CVE; после чего, необходимо проанализировать, возможны ли эти CVE в создаваемом веб-приложении. Если да, необходимо их исправлять согласно рекомендациям. |