Глава 4: Результат и анализ безопасности

4.1. Используемый стек программного обеспечения

4.1.1. Инструментарий

4.1.1.1. Среда разработки

Для кросс-компиляции компонентов Безопасного мира используется инструментарий RISC-V GNU, в частности, GCC версии 11.1.0 и Binutils 2.38.

Как сама Безопасная ОС, так и Доверенные приложения компилируются с флагами для архитектуры RISC-V.

В качестве основной системы для разработки рекомендуется использовать ОС на базе Linux (например, Ubuntu 20.04). Система сборки требует CMake версии 3.14 или выше. Отладка осуществляется с помощью GDB, который подключается к GDB серверу QEMU, что позволяет инспектировать состояние ядра Безопасной ОС на самом низком уровне.

4.1.2. Среда эмуляции

4.1.2.1. QEMU с поддержкой WorldGuard

Для работы используется надежная среда эмуляции на базе модифицированной версии QEMU. Такая среда предоставляет виртуальную платформу, симулирующую архитектуру RISC-V с расширением WorldGuard, что обеспечивает воспроизводимое тестирование без необходимости в физическом оборудовании.

Эмуляция основана на форке QEMU версии v9.1.50 riscv64-softmmu, в который были внесены патчи для реализации ISA-расширения WorldGuard. Эти модификации включают реализацию специфичных для WorldGuard регистров управления и состояния (CSR), а также логики контроля доступа к памяти на основе идентификатора мира (world ID).

4.1.2.2. Конфигурация QEMU

Система эмулируется с использованием машинной модели virt с 4 ядрами: харядро 0 эксклюзивно выделено для Безопасной ОС, а остальные - для Linux. Файл дерева устройств (DTB) определяет физическую карту памяти, используемых устройств.

QEMU запускается со следующими флагами:

```
qemu-system-riscv64 -M virt,wg=on -m 4G -nographic -smp 4 -bios sbi.bin
```

4.1.3. Linux

4.1.3.1. Linux с поддержкой WorldGuard

Стандартное ядро Linux выступает в роли операционной системы Обычного мира. В этой среде работают пользовательские приложения и драйвер ядра, отвечающий за инициацию взаимодействия с Безопасной ОС и её Доверенными приложениями.

В проекте используется ядро Linux, основанное на LTS-версии 6.12.0. Оно включает набор внешних патчей, которые в содержат драйвер символьного устройства, реализующий клиентскую часть протокола TEE. Этот драйвер предоставляет стандартный интерфейс на основе ioctl для взаимодействия пользовательских приложений с Безопасным миром.

4.1.3.2. Конфигурация Linux

Ядро собирается с минимальной конфигурацией на основе defconfig для архитектуры RISC-V. Ключевая включенная опция - это CONFIG_WG_TEE. Конфигурация нацелена на создание компактного образа ядра с минимальным набором драйверов, необходимых для запуска в среде QEMU.

4.1.3.3. Образ Linux

Загрузочный компонент Linux состоит из образа на базе дистрибутива debian 10. Файловая система создается примонтированном в QEMU диске.

4.1.4. Система сборки

4.1.4.1. Конфигурация CMake

Для компиляции всех программных компонентов проекта используется централизованная и модульная система сборки на основе CMake версии не менее 3.14.

4.1.4.2. Дизайн системы сборки

Система сборки организована с корневым файлом CMakeLists.txt, который подключает подкаталоги с ядром Безопасной ОС, стандартными библиотеками и Доверенными приложениями.

4.1.4.3. Процесс сборки Доверенного приложения (ДП)

Каждое Доверенное приложение собирается как отдельный позиционно-независимый ELF-файл. Процесс сборки ДП включает компиляцию его исходного кода с использованием стандартной библиотеки Безопасной ОС и обработку файла-манифеста. Манифест, определяющий UUID приложения и его начальные права доступа, встраивается в специальную секцию итогового ELF-файла, откуда он считывается Безопасной ОС при инициализации.

4.1.5. Интеграция с CI

4.1.5.1. Настройка непрерывной интеграции

Для обеспечения качества и стабильности кода используется конвейер непрерывной интеграции (CI), который автоматизирует процессы сборки и тестирования.

Автоматизация подтверждает, что вносимые изменения не приводят к регрессиям, и

гарантирует работоспособность всего программного стека.

Проект использует GitHub Actions в качестве платформы для CI. Рабочий процесс настроен так, что он запускается при каждой отправке изменений (push) в основные ветки. Он последовательно выполняет серию задач: сначала собирается кросс-компилятор, затем модифицированный QEMU, OpenSBI, Безопасная ОС со своими ДП и, наконец, ядро Linux.

4.1.5.2. Скрипты автоматизированного тестирования

На заключительном этапе конвейера CI выполняется автоматизированный тест, реализованный на Python. Этот скрипт запускает полностью собранную систему в QEMU с поддержкой WorldGuard, в Безопасной ОС при этом запускаются тестовые сценарии. Таким образом проверяется, что и Безопасная ОС, и Linux успешно загрузились, а также подтверждается корректная инициализация драйвера связи и выполнение тестовой команды в образцовом Доверенном приложении.

4.2 Демонстрация работы Безопасной ОС

В данном разделе демонстрируется практическое применение Безопасной ОС на примере полного цикла: от сборки всего программного стека и разработки минимального Доверенного приложения (ДП) до его запуска в эмулируемой среде RISC-V. Цель - проверить сквозную интеграцию всех компонентов системы, от клиентского приложения в Обычном мире (Normal World) до самого ДП, работающего в Безопасном мире (Secure World).

4.2.1. Сборка программного стека

В этом подразделе приводится краткое руководство по сборке всех необходимых программных компонентов из исходного кода для получения загрузочного образа системы, готового к запуску в среде эмуляции QEMU.

Клонирование репозиториев проекта

Первым шагом является клонирование исходного кода всех ключевых компонентов: модифицированного QEMU с поддержкой WorldGuard, адаптированной прошивки OpenSBI, ядра Linux с драйвером поддержки TEE, а также Безопасной ОС вместе с её библиотеками и примерами ДП.

клонирование производится командой

```
git clone git@github.com:HALIP192/TEEgRISCv.git
cd TEEgRISCv
git submodule update --init --recursive
```

Сборка инструментария для кросс-компиляции

Для компиляции всех артефактов требуется инструментарий RISC-V GNU, который обеспечивает двоичную совместимость (ABI) между компонентами. Процесс включает настройку кросс-компилятора GCC и утилит для целевой архитектуры riscv64.

Для начала необходимо установить следующие пакеты:

```
sudo apt install -y \
 git                    \
 make                   \
 cmake                  \
 bear                   \
 device-tree-compiler   \
 cpio                   \
 libncurses-dev         \
 gawk                   \
 flex                   \
 bison                  \
 openssl                \
 libssl-dev             \
 dkms                   \
 libelf-dev             \
 libudev-dev            \
 libpci-dev             \
 libiberty-dev          \
 autoconf               \
 bc                     \
 python3                \
 python3-venv           \
 ninja-build            \
 libglib2.0-dev         \
 libgcrypt20-dev        \
 zlib1g-dev             \
 autoconf               \
 automake               \
 libtool                \
 libpixman-1-dev        \
 libslirp-dev
```

Загрузка тулчейна:

```
mkdir -p toolchain;
cd toolchain;
wget $TOOLCHAIN_URL --output-document "sc-dt.tar.gz";
tar xf "sc-dt.tar.gz";
```

Сборка QEMU с поддержкой WorldGuard

Среда эмуляции создаётся путём сборки реализации QEMU с расширением WorldGuard. Эмулятор конфигурируется для машины virt с riscv64-softmmu, что активирует симуляцию аппаратных механизмов, необходимых для разделения на два мира.

сборка QEMU производится командой

```
mkdir -p build;
cd build;

../configure \
 --target-list=riscv64-softmmu \
 --disable-capstone \
```

```
 --disable-curl \
 --disable-guest-agent \
 --disable-libusb \
 --disable-vnc \
 --enable-slirp \
 --disable-docs;

make -j $(nproc)
```

Сборка модифицированного OpenSBI

Прошивка OpenSBI компилируется из репозитория, содержащего изменения стандартного процесса загрузки. Эти изменения позволяют OpenSBI инициализировать Безопасную ОС на первом ядре процессора перед передачей управления ядру Linux на остальных ядрах, тем самым устанавливая аппаратное разделение миров.

сборка OpenSBI производится командой

```
mkdir -p build;
cd build;

make \
  -C ../ \
  PLATFORM=generic \
  FW_PAYLOAD_PATH=kernel.bin \
  NWD_FW_PAYLOAD_PATH=LinuxImage \
  -j $(nproc)
```

Сборка образа Linux с поддержкой WorldGuard

Ядро Linux конфигурируется с опцией CONFIG_WG_TEE, которая включает драйвер для взаимодействия с Безопасной ОС. Этот драйвер предоставляет стандартный интерфейс на основе системного вызова ioctl, позволяя приложениям Обычного мира обращаться к TEE.

сборка Linux производится командой:

```
mkdir -p build;
cd build;

make \
 -C ../ \
 O=build \
 ARCH=riscv \
 LLVM=1 \
 teegriscv_defconfig \

make \
 -C ../ \
 O=build \
 ARCH=riscv \
 LLVM=1 \
 -j 16
```

Сборка Безопасной ОС

Сборка Безопасной ОС выполняется с помощью системы CMake. В ходе этого процесса компилируется микроядро, его стандартные библиотеки, а манифесты и ELF-файлы предопределённых Доверенных приложений упаковываются в конечный образ Безопасной ОС в соответствии с картой памяти платформы.

```
build.py \
 -manifest manifest.toml \
 -build build \
 -root blowfish

cmake -B build blowfish
cd build
make
```

Сборка загрузочного образа Linux

Загрузочный образ Linux создаётся с помощью скрипта create_image.sh поверх дистрибутива debian 10:

```
./create_image.sh examples/hello_world
```

Пример доверенного приложения

Реализация простого доверенного приложения

Для иллюстрации процесса разработки ДП создаётся базовое приложение, которое предоставляет Обычному миру простую функцию генерации случайных чисел.

Определение манифеста для модели полномочий

Для ДП создаётся файл манифеста, в котором указываются его уникальный идентификатор (UUID) и минимально необходимый набор полномочий. Затем этот манифест регистрируется в манифесте корневой задачи Безопасной ОС, что даёт ей право на запуск данного ДП.

пример манифеста:

```
uuid = "8aaaf200-2450-11e4-abe2-0002a5d5c51b"
name = "hello_world"

[kernel]
factory = ["create_vmo"]
vmspace = ["allocate"]

crypto = []
```

Сборка доверенного приложения

ДП компилируется в позиционно-независимый ELF-файл. Его манифест встраивается в специальную секцию этого файла, что позволяет Безопасной ОС при загрузке извлечь информацию о его идентификаторе и правах доступа:

```
build.py \
 -manifest hello.toml \
 -build build \
 -root blowfish
```

Демонстрация выполнения

В этом подразделе описаны шаги по запуску полностью собранной системы и показан полный цикл взаимодействия между клиентским приложением в Linux и примером ДП.

Загрузка системы

Система запускается с помощью QEMU с собранным загрузочным образом. Консольный вывод подтверждает успешную передачу управления от OpenSBI к Безопасной ОС и последующей загрузки ядра Linux:

```
OpenSBI v1.6
   _____                    ______ _____ ______
  / ___ \                  / _____|  _ \_   _|
 | |   | |_ __   ___ _ __ | (____ | |_) || |
 | |   | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |___| | |_) |  __/ | | |____) | |_) || |_
  \_____/| .__/ \___|_| |_|_____/|____/_____|
         | |
         |_|

Platform Name               : riscv-virtio,qemu

Switching... 80200000, wid 0
[BOOT] Blowfish start..
[SBI TIMER] Timer frequency is 0000000000989680
[PAGE_ARRAY] Page array base 0xffffffe0000ce000 size 00000000000004ce
[WGC] DRAM: Vendor ID 00000000
[WGC] DRAM: Num slots 00000010
[WGC] UART: Vendor ID 00000000
[WGC] UART: Num slots 00000001
[CONSOLE] Unregistering console Sbi console
[CONSOLE] Registering console Uart console
[TASK] System task initialized
[NWD] Registered NWD channel at pa 00000000802cb000
[ROOTTASK] Spawned root task
[BOOT] Blowfish boot end...
Booting NWD payload on CPU 1... 80400000
Switching... 80400000, wid 1
```

Инициализация драйвера Linux

После загрузки Linux его модуль ядра для взаимодействия с TEE загружается командой insmod. Наличие устройства tee0 подтверждает успешную инициализацию драйвера TEE:

```
# ls /dev/tee*
/dev/tee0
```

Открытие сессии с доверенным приложением

Клиентское приложение в Linux использует TEE Client API для вызова `TEEC_OpenSession`, передавая UUID арифметического ДП. Это действие инициирует запрос к Безопасной ОС, которая проверяет его и создаёт новый экземпляр сессии для указанного ДП.

Пример кода, открывающий сессию с ДП:

```
TEEC_Result res;
TEEC_Session session;
TEEC_UUID uuid = TA_HELLO_WORLD_UUID;
uint32_t ret;

res = TEEC_InitializeContext("test", &ctx);
if (res != 0) {
        printf("Failed to open context %d\n", res);
        return 0;
}

res = TEEC_OpenSession(&ctx, &session, &uuid, 0, NULL, NULL, &ret);
if (res != 0) {
        printf("Failed to open seesion %d\n", res);
        return 0;
}

...
```

Вызов функции ДП и получение ответа

Далее клиентское приложение вызывает `TEEC_InvokeCommand`, передавая в качестве параметров целое число - seed генератора случайных чисел.

Запрос передаётся в ДП через очередь запросов, ДП выполняет вычисление случайного числа, и результат возвращается клиенту через очередь ответов.

Пример кода, вызывающего функцию ДП:

```
TEEC_Result res;
uint32_t ret;
TEEC_Operation op = {};

...

printf("Before call value %x\n", 0xdeadbeef);
op.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INOUT, TEEC_NONE, TEEC_NONE
                                 TEEC_NONE);
op.params[0].value.a = 0xdeadbeef;

res = TEEC_InvokeCommand(&session, TA_INC_ARG, &op, &ret);
if (res != 0) {
        printf("Failed to invoke command %d\n", res);
        return 0;
```

```
}

printf("After call value %x\n", op.params[0].value.a);
```

Демонстрация работы с разделяемой памятью

ДП так же имеет возможность работать с разделяемой памятью, которая доступна для чтения и записи из ДП. Для создания участка разделяемой памяти вызывается TEEC_AllocateSharedMemory, который возвращает указатель на участок памяти.

Пример кода, создающего участок разделяемой памяти:

```
TEEC_Result res;
uint32_t ret;
TEEC_Operation op = {};
TEEC_SharedMemory shmem = {};

res = TEEC_AllocateSharedMemory(&ctx, &shmem);
if (res != 0) {
        printf("Failed to allocate shmem %d\n", res);
        return 0;
}

void *data = mmap(NULL, 1 << 12, PROT_READ | PROT_WRITE,
                  MAP_SHARED, shmem.fd, 0);

assert(data != MAP_FAILED);

*(unsigned *)data = 0x12345;
printf("Shmem before call %x\n", *(unsigned *) data);

op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INOUT, TEEC_NONE,
                                 TEEC_NONE, TEEC_NONE);
op.params[0].memref.parent = &shmem;
op.params[0].memref.size = 1 << 12;
op.params[0].memref.offset = 0;

res = TEEC_InvokeCommand(&session, TA_SHMEM, &op, &ret);
if (res != 0) {
        printf("Failed to invoke command %d\n", res);
        return 0;
}
```

4.3 Анализ безопасности

В данном разделе оценивается уровень защищённости Безопасной ОС посредством анализа устойчивости её архитектуры к различным типам угроз, определённым в модели угроз из Главы 2. Для каждого сценария описывается тестовая среда, имитируемая атака и фактическая реакция системы.

4.3.1 Устойчивость к атакам из Обычного мира

В этом подразделе рассматривается защищённость Безопасной ОС от атак со стороны скомпрометированной ОС Linux, работающей в Обычном мире.

4.3.1.1 Несанкционированный доступ к коду Безопасной ОС и ДП

Был проведён тест, в ходе которого процесс Обычного мира с повышенными привилегиями пытался инспектировать память, содержащую бинарный код ядра Безопасной ОС и ДП, используя механизм /dev/mem.

Эти попытки провалились, поскольку защищённые области памяти не отображены в адресное пространство Обычного мира.

4.3.1.2 Несанкционированный доступ к защищённой памяти

Была смоделирована атака, в ходе которой специально разработанный драйвер ядра Linux пытался выполнить чтение и запись физических страниц памяти, принадлежащих ядру Безопасной ОС и её Доверенным приложениям.

Аппаратные контроллеры WorldGuard корректно распознали эти операции как недопустимую транзакции. В результате попытки доступа были заблокированы на аппаратном уровне, что привело к генерации исключения типа ошибка шины в Обычном мире.

Исключение было сгенерировано на том ядре, которое инициировало недопустимую операцию, т.е. на ядре с ОС Linux. ОС Linux в этом случае проигнорировал исключение данного типа, так как обработчик данного типа исключений не реализован за ненадобностью.

4.3.1.3 Попытки повредить очереди в разделяемой памяти

Канал связи через разделяемую память был протестирован путём намеренной отправки некорректно сформированных запросов из драйвера Linux. Эти запросы содержали неверные идентификаторы команд, ошибочные размеры сообщений и параметры, нарушающие ожидаемый формат.

Обработчик сообщений в Безопасной ОС успешно проверял каждый входящий запрос, отбрасывая недействительные и регистрируя ошибки, что предотвратило отказ в обслуживании на стороне ядра.

4.3.1.4 Эксплуатация протокола межмирового взаимодействия (CWC)

Для проверки целостности протокола CWC на устойчивость к состоянию гонки был разработан драйвер в Linux, который пытался перезаписать сообщения в разделяемой очереди во время их обработки Безопасной ОС. Lock-free очередь, полагающаяся на атомарные операции с относительными индексами заголовка, доказала свою устойчивость.

Любое повреждение индексов, приводившее к доступу за пределы буфера, игнорировалось, что предотвращало повреждение данных и рассинхронизацию состояния очереди.

4.3.2 Устойчивость к ошибочным Доверенным приложениям

Критически важным аспектом архитектуры Безопасной ОС является её способность изолировать и сдерживать ошибочные Доверенные приложения, гарантируя, что они не смогут скомпрометировать ядро или другие ДП.

4.3.2.1 Изоляция между ДП

Была инсценирована атака, в ходе которой одно ДП пыталось получить прямой доступ к памяти или использовать дескриптор канала связи, принадлежащий другому ДП.

Ядро Безопасной ОС отклонило эти попытки на этапе проверки системного вызова. Поскольку у вызывающего ДП отсутствовало необходимое полномочие (capability) на целевой объект в его манифесте, операция была запрещена, что подтвердило как логическую, так и физическую изоляцию между ДП.

4.3.2.2 Механизм контроля на основе полномочий

Интерфейс системных вызовов Безопасной ОС был подвергнут фаззингу с использованием некорректно сформированных запросов, включая использование недействительных, закрытых или поддельных дескрипторов объектов.

Каждый нелегитимный вызов был последовательно отклонён ядром. Механизм контроля полномочий проверяет, что предоставленный дескриптор существует и что вызывающая задача обладает необходимыми правами для выполнения операции, эффективно предотвращая эскалацию привилегий или несанкционированные действия.

4.3.2.3 Защита от некорректного использования ресурсов ДП

Было разработано вредоносное ДП, пытавшееся исчерпать системные ресурсы путём непрерывного выделения памяти и создания неограниченного числа дескрипторов объектов.

Безопасная ОС корректно обработала эти попытки, применяя квоты на ресурсы для каждой задачи. Как только квота была достигнута, последующие запросы на выделение ресурсов завершались неудачей без сбоя самого ДП или нарушения стабильности ядра.

4.3.2.4 Атаки по побочным каналам

Основной защитой от атак по побочным каналам является архитектурная изоляция - выделение отдельного физического ядра для Безопасной ОС. Это смягчает многие атаки, основанные на времени выполнения и доступе к кэш-памяти, которые эксплуатируют общие ресурсы ядра.

Однако атаки, использующие общий кэш последнего уровня или шину памяти, не могут быть полностью устранены таким подходом и требуют аппаратного обеспечения с функциями партиционирования кэша, что выходит за рамки текущего прототипа.

4.3.3 Дополнительные сценарии атак и ограничения

В этом разделе рассматриваются угрозы, которые либо выходят за рамки текущего прототипа, ориентированного на программное обеспечение, либо представляют собой неотъемлемые сложности проектирования TEE.

4.3.3.1 Физические атаки

Рассматриваются сценарии, такие как прямое зондирование DRAM, прослушивание шины или атаки с внедрением сбоев (гличинг).

Текущая система не предлагает специальных программных средств защиты от этих угроз. Предполагается, что устойчивость к физическим атакам обеспечивается базовой аппаратной платформой и не входит в область оценки данного проекта.

4.3.3.2 Сложность доверенной вычислительной базы (ДВБ)

Доверенная вычислительная база включает в себя ядро Безопасной ОС, его конфигурацию и весь доверенный код, исполняемый в Безопасном мире. Хотя само микроядро спроектировано минималистичным, каждое загруженное Доверенное приложение по своей сути расширяет ДВБ. Поэтому общая безопасность системы зависит от тщательной проверки и верификации каждого ДП перед его развёртыванием.

4.3.3.3 Атаки на цепь доверия

Безопасность всей TEE зависит от процесса безопасной загрузки, который устанавливает непрерывную цепь доверия от аппаратного корня.

Данная оценка предполагает целостность загрузчика (OpenSBI) и исходного образа Безопасной ОС.

Скомпрометированный компонент загрузки может отключить WorldGuard или загрузить вредоносную Безопасную ОС, полностью подрывая безопасность системы. Полноценная реализация безопасной загрузки не была интегрирована в прототип.

4.4 Перспективы развития

В данном разделе изложены перспективные направления для дальнейшей работы, нацеленные на расширение функциональности платформы, повышение её производительности и усиление защищённости.

4.4.1 Расширение функциональности Доверенных приложений

В этом разделе рассматриваются усовершенствования фреймворка Доверенных приложений, которые позволят им предоставлять более сложные сервисы, сохраняя при этом минимальный размер доверенной вычислительной базы и надёжную изоляцию.

4.4.1.1 Защищённое хранилище

Дальнейшая работа предполагает реализацию сервиса защищённого хранилища, который предоставит ДП доступ к персистентной памяти, защищённой от несанкционированного вмешательства.

Этот сервис должен включать API для щифрования данных, при котором информация шифруется ключом, уникальным для конкретного ДП и аппаратной платформы.

Такой подход гарантирует конфиденциальность и целостность данных даже при их хранении в небезопасной памяти.

4.4.1.2 Аттестация

Реализация механизма аттестации является следующим критически важным шагом. Он позволит Доверенному приложению криптографически доказывать свою подлинность и целостность кода удалённой стороне (удалённая аттестация) или другому ДП (локальная аттестация).

Такой механизм будет построен на основе цепи доверия и предоставит верифицируемое подтверждение того, что в подлинной TEE исполняется корректное программное обеспечение.

4.4.1.3 Корень доверия

Для построения завершённой модели безопасности необходима интеграция формального аппаратного корня доверия (Root of Trust). Это подразумевает использование неизменяемого загрузочного кода и однократно программируемой (OTP) памяти для создания верифицируемой цепи доверия. Такой подход гарантирует, что OpenSBI и Безопасная ОС являются подлинными и не были скомпрометированы перед загрузкой.

4.4.1.4 Криптографические сервисы

Чтобы уменьшить ДВБ и исключить ошибки в криптографических реализациях внутри каждого ДП, необходимо предоставить централизованный криптографический сервис. Он будет предлагать стандартизированный API для распространённых операций: симметричного и асимметричного шифрования, хеширования и генерации случайных чисел. Использование аппаратных ускорителей, обеспечит как производительность, так и надёжность.

4.4.1.5 Портирование на реальное оборудование RISC-V с WorldGuard

Ключевым этапом валидации проекта является портирование системы из среды QEMU на физическое оборудование с реализацией расширения WorldGuard. Эта работа потребует выбора подходящих кремниевых платформ, адаптации прошивки OpenSBI под конкретные пакеты поддержки платформ (BSP) и разработки инфраструктуры для аппаратной отладки.

Это необходимо для анализа производительности в реальных условиях и оценки устойчивости к физическим атакам.

4.4.2 Производительность и управление памятью

4.4.2.1 Поддержка многоядерности в Безопасном мире

Текущая архитектура выделяет для Безопасного мира одно процессорное ядро.

Расширение Безопасной ОС для поддержки многоядерного исполнения значительно повысит производительность при параллельных вычислениях.

Эта задача сопряжена с необходимостью разработать планировщик для нескольких ядер в Безопасном мире, обеспечить когерентность разделяемого состояния между ядрами и адаптировать модель полномочий для многопоточных экземпляров ДП.

4.4.2.2 Динамическая загрузка Доверенных приложений

Для повышения эффективности использования памяти и гибкости системы ключевым направлением является добавление поддержки динамической загрузки и выгрузки ДП во время исполнения.

Это потребует создания безопасного загрузчика, который выполняет криптографическую проверку подписи ДП. Загрузчик также должен анализировать манифест приложения для корректной интеграции его полномочий и гарантировать полную изоляцию и последующее освобождение его памяти.

4.4.3 Усиление безопасности

4.4.3.1 Формальная верификация компонентов

Для достижения наивысшего уровня гарантий безопасности следует применить методы формальной верификации к наиболее критичным компонентам ДВБ.

Это включает использование специализированных инструментов, таких как средства проверки моделей (model checkers), для построения математических доказательств корректности логики микроядра, механизма контроля полномочий и примитивов IPC.

Такой подход позволит формально доказать такие свойства системы, как безопасность работы с памятью и корректность распространения прав доступа.

4.4.3.2 Усиление защиты от атак

Постоянное усиление защиты системы от современных угроз является необходимым направлением работы.

Дальнейшие усилия должны быть сосредоточены на реализации конкретных мер противодействия атакам по побочным каналам, например, через использование криптографических алгоритмов с постоянным временем выполнения и техник разделения кэша.

Кроме того, надёжность системы можно повысить за счёт систематического фаззинга API ядра и разработки защитных механизмов от атак с внедрением сбоев и спекулятивного выполнения.

4.4.3.3 Реализация механизма безопасного обновления

Для обеспечения долгосрочной безопасности и поддержки платформы необходимо разработать механизм её безопасного обновления.

Такой механизм должен гарантировать целостность новых образов Безопасной ОС и защищать систему от попыток отката на более старые, уязвимые версии (anti-rollback). Этого можно достичь с помощью криптографической подписи пакетов обновлений и использования монотонного счётчика версий, хранящегося в защищённой энергонезависимой памяти.

Механизм должен поддерживать как удалённые обновления (через сеть), так и локальные (с внешних носителей).

Заключение

References

Appendices

Sample TA Code

TA Manifests

Secure OS Code

Secure Standard Library Code