Введение

Taint – анализ является одним из методов поиска бинарных уязвимостей в программном обеспечении (ПО) и облегчает исследование вредоносного программного обеспечения (ВПО). Кроме архитектуры x86, для которых данный метод уже реализован, существует архитектура ARM, для которой публично доступных готовых средств пока еще не существует. ARM – процессоры используются мобильными телефонами, карманными компьютерами, а также устройствами IoT. Для таких устройств уже существует ВПО [1], имеются уязвимости, эксплутация которых может привести к выполнению произвольного кода здоумышленника. Поэтому для этой архитектуры необходимо реализовать taint – анализ. Целью данной работы является сравнительный анализ публично доступных средств, на основе которых можно реализовать taint – анализ.

В процессе работы потребуется решить следующие задачи:

- Подготовить рабочую среду – реальное или виртуальное устройство с архитектурой ARM;
- Рассмотреть публично доступные средства для архитектуры ARM, на основе которых можно реализовать taint – анализ. Выявить их достоинства и недостатки;
- Выбрать наиболее подходящее средство и детально изучить принципы его работы.

Подготовка рабочей среды: плата beaglebone black

Самым простым способом является использование одноплатных компьютеров, таких как BeagleBone black или Paspberry Pi. Плата BeagleBone black является более новой и производительной. Поэтому все действия по настройке и использованию будут относиться именно к ней.

Одноплатный компьютер BeagleBone black (см. рис. 1) имеет интерфейсы USB Host, USB Device, Ethernet, HDMI, поддерживает карты памяти формата SD или microSD. В качестве операционной системы может использоваться Linux или Android. Наличие портов USB обеспечивает подключения клавиатуры и мыши, HDMI выход позволяет подключать к одноплатному компьютеру телевизор или монитор через переходник HDMI-DVI. Процессор от компании Texas Instruments серии Sitara AM3359AZCZ100 с тактовой частотой в 1ГГц содержит в своем составе вычислительное ядро Cortex-A8 и графический ускоритель SGX530 . На плате установлено 512 Мб оперативной памяти DDR3L, Flash память eMMC объемом в 2 ГБ , на которой установлена операционная система Angstrom Linux. Для загрузки другого дистрибутива Linux или ОС Android можно использовать карту памяти формата microSD, подключаемую к соответствующему слоту на плате [2].

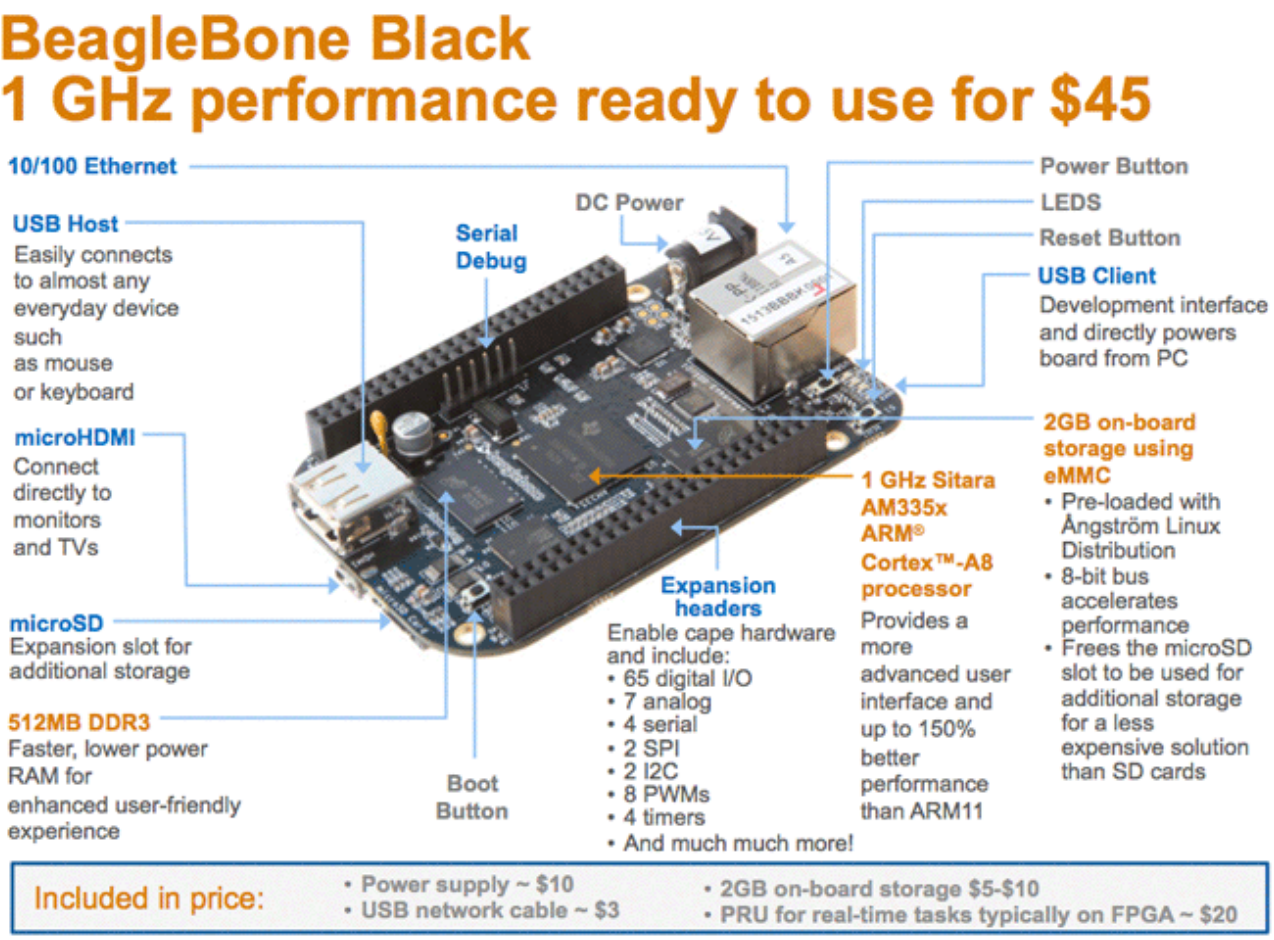

Рисунок 1 – Одноплатный компьютер BeagleBone Black

Сначала необходимо установить операционную систему (ОС) на плату. Для этого нужна отформатированная SD – карта и образ ОС Linux debian 9 [3], который потребуется записать на карту. Сделать это можно с помощью программы Etcher [4]. После этого SD – карту с записанным образом нужно вставить в слот и подключить питание к плате. Через пару минут запустится ОС, подключиться к которой можно через USB по протоколу ssh. Более подробно с настройками можно ознакомиться на сайте [5].

Подготовка рабочей среды: эмулятор qemu

Если под рукой не имеется одноплатного ARM – компьютера, то можно использовать эмулятор QEMU.

Для установки Linux в QEMU необходимо выполнить следующее:

1. Загрузить с ресурса [6] ядро ОС Linux debian 9 Stretch (vmlinuz), а также временную файловую систему (initrd), которая будет использоваться операционной системой при загрузке.
2. Создать виртуальный жесткий диск для ОС

   ```
   qemu-img create -f qcow2 hda.qcow2 5G
   ```

3. Запустить QEMU и осуществить установку ОС (~ 1, 5 часа)

   ```
   qemu-system-arm -M virt -m 1024 \
   -kernel installer-vmlinuz \
   -initrd installer-initrd.gz \
   -drive if=none,file=hda.qcow2,format=qcow2,id=hd \
   -device virtio-blk-device,drive=hd \
   -netdev user,id=mynet \
   -device virtio-net-device,netdev=mynet \
   -nographic -no-reboot
   ```

4. Так как при таком режиме установки не получится установить загрузчик, то придется вручную найти файлы vmlinux и initrd, которые использовались во время установки и заменить ими исходные файлы. Перед этим необходимо примонтировать виртуальный жесткий жиск QEMU, на который была установлена ОС.

```
modprobe nbd max_part=16
qemu-nbd -c /dev/nbd0 hda.qcow2
cp /dev/nbd0/boot/new_vmlinuz old_vmlinuz_path
cp /dev/nbd0/boot/new_initrd old_initrd_path
```

5. После замены файлов ОС можно запускать и использовать

```
qemu-system-arm -M virt -m 1024 \
 -kernel new_vmlinuz \
 -initrd new_initrd.img \
 -append 'root=/dev/vda2' \
 -drive if=none,file=hda.qcow2,format=qcow2,id=hd \
 -device virtio-blk-device,drive=hd \
 -netdev user,id=mynet \
 -device virtio-net-device,netdev=mynet \
 -nographic
```

Подробнее об установке Linux для ARM на QEMU можно узнать из ресурсов [7] и [8]. Стоит также отметить, что QEMU тратит много вычислительных ресурсов и времени на эмуляцию архитектуры ARM. Поэтому лучше применить ускорители, использующие средства аппаратной виртуализации, такие как qemu-kvm (Linux) и intel HAXM driver (Windows).

Критерии выбора средства для реализации taint – анализа

Обычно taint – анализ применяется для исследования программы во время исполнения. Поэтому он реализуется с помощью средств динамической бинарной инструментации (DBI).

Средства DBI должны обладать следующими характеристиками:

- Малые накладные расходы (overhead) на инструментацию кода. Средства DBI часто сильно замедляют работу программ, вследствие чего анализ занимает продолжительное время.
- Низкая гранулярность. То есть существет возможность производить инструментацию кода на уровне инструкций и базовых блоков. Средства, работающие на более высоких уровнях, не подходят для реализации taint – анализа.
- Применимость в сфере ИБ. Средство может быть создано для различных целей, например, профилирования, анализа покрытия кода и т.д. Но не каждое из таких средств сможет поддерживать произвольную модификацию кода программы во время её исполнения.

Кроме перечисленных выше критериев стоит учитывать наличие уже существующих наработок, расширений, дополнительного API, которые помогли бы быстрее приступить к реализации taint – анализа.

Исследование средства DynamoRIO

DynamoRIO - это средство динамической бинарной инструментации кода, работающее во время выполнения программы [9]. DynamoRIO предоставляет интерфейс для создания динамических инструментов применяемых в сферах исследования и анализа, профилирования, инструментирования, оптимизации программ и т.д. Более того, он может быть применен в сфере ИБ, так как способен обрабатывать самомодифицирующийся код, использовать динамический патчинг. DynamoRIO не ограничено во вставке инструкций в ассемблерный код программы, и позволяет произвольно изменять его. DynamoRIO поддерживает архитектуры IA-32 / AMD64 / ARM / AArch64 и обеспечивает эффективное управление приложениями, работающими на операционных системах Windows, Linux и Android.

На рисунке 2 изображена схема работы DynamoRIO. Оперирует оно на уровне базовых боков программы. Для каждого базового блока выполняются следующие действия:

1. Копирование кода базового блока приложения в кэш кода DynamoRIO
2. Анализ базового блока, вставка новых инструкций, callback - функций
3. JIT – рекомпиляция измененного базового блока
4. Переключение констекста: DynamoRIO -> приложение
5. Выполнение инструкций процессором
6. Переключение констекста: приложение -> DynamoRIO
7. Возвращение контроля к DynamoRIO и загрузка следующего блока

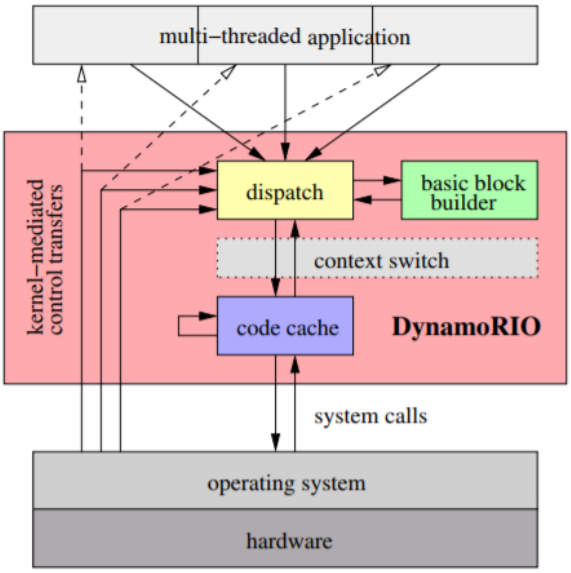

Рисунок 2 – Схема работы DynamoRIO

Поскольку действия по инструментации происходят непосредственно в кэше, DynamoRIO существенно снижает свои накладные расходы. Несмотря на это, они весьма значительные (см. рис. 3).

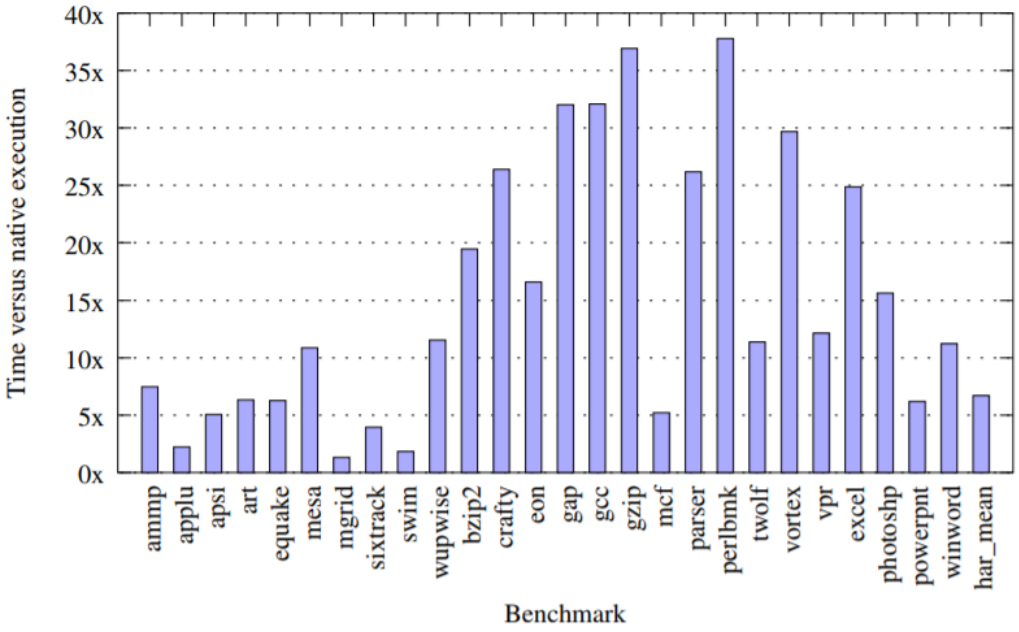

Рисунок 3 – Результаты тестов производительности DynamoRIO

Исследование средства Mambo

Mambo – это средство динамической бинарной инструментации для архитектур ARM/ARM64 [10]. Оно является относительно новым и прогрессивным средством, протестированным бенчмарками SPEC CPU2000, SPEC CPU2006, многопоточным бенчмарком PARSEC, а также успешно работающим с такими программами, как LibreOffice, GIMP, GCC и др. По функционалу и предоставляемым API оно похоже на DynamoRIO. Однако обладает некоторыми преимуществами:

- Небольшие накладные расходы. В связи с тем, что Mambo спроектировано именно под ARM, оно использует особенности этой архитектуры для оптимизации скорости своей работы. Результаты сравнения производительности Mambo и DynamoRIO показаны на рис. 4.
- Простой в использовании интерфейс. По сравнению с DynamoRIO, интерфейс Mambo выглядит проще и удобнее.
- Небольшой размер ядра. Одним из приоритетов при разработке Mambo было сохранение небольшой кодовой базы, чтобы исследователи могли легко понимать и изменять это средство.

К сожалению, Mambo создавалось не для решения задач ИБ, а для анализа покрытия кода и профилирования. Оно не способно обрабатывать упакованные программы (см. рис. 5)

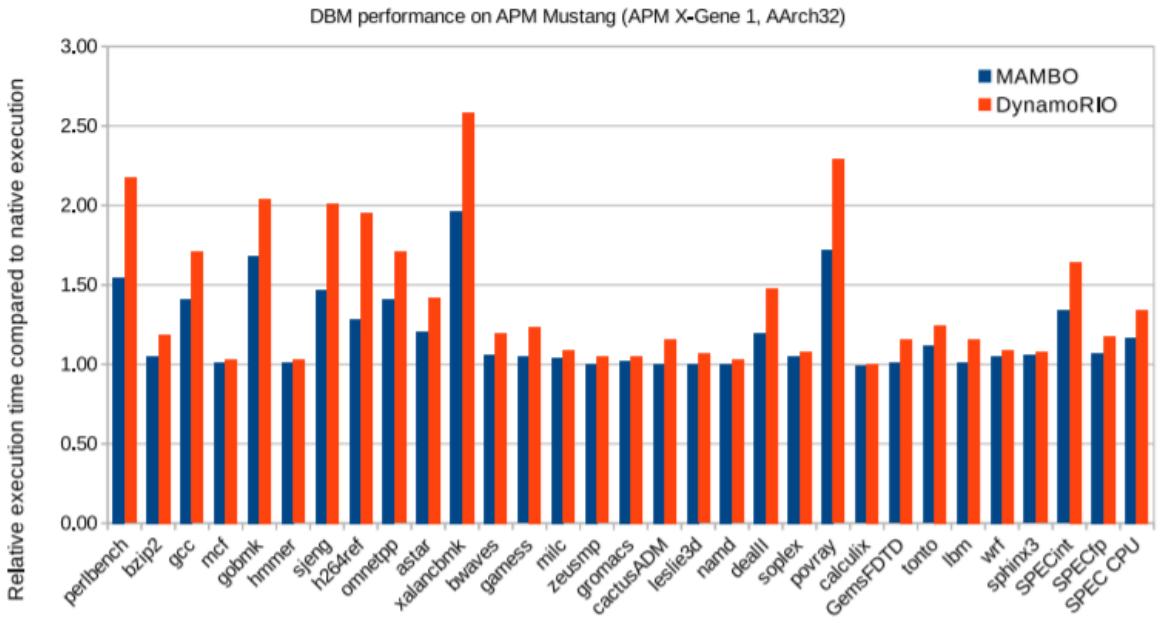

Рисунок 4 – Результаты тестов производительности Mambo по сравнению с DynamoRIO

```
debian@debian:~/Mambo/mambo$ ./dbm /bin/ls
api          dispatcher.c        pie             scanner_common.h  test
common.c     dispatcher.S        plugins         scanner_public.h  traces.c
common.h     elf_loader          plugins.h       scanner_thumb.c   util.h
dbm          kernel_sigaction.h  README.md       signals.c         util.S
dbm.c        LICENSE             scanner_a64.c   syscalls.c
dbm.h        makefile            scanner_arm.c   syscalls.h
We're done; exiting with status: 0
debian@debian:~/Mambo/mambo$
debian@debian:~/Mambo/mambo$
debian@debian:~/Mambo/mambo$ ./dbm /home/debian/DbiTest/ls_packed
Segmentation fault
debian@debian:~/Mambo/mambo$
```

Рисунок 5 – Аварийное завершение Mambo при запуске упакованной программы

Исследование средства Scorpio

Scorpio – средство динамической бинарной инструментации, сделанное для архитектур: x86, MIPS, ARM, SPARC, PowerPC [11]. Работает на ОС Windows, Mac, Linux, BSD. В отличие от средств DynamoRIO и Mambo, Scorpio использует механизм хуков (hooking). Механизм hooking радикально отличается от JIT – трансляции. Вместо того, чтобы перекомпилировать код приложения на ходу, в базовый блок вместо одной из инструкций вставляется инструкция безусловного перехода - hook (jmp, b, bl), которая передает управление особому блоку кода, называемого трамплином (trampoline). Этот блок кода отвечает за смену контекстов, вызов callback – функции и возвращение программы в исходное состояние, которое было до прыжка в трамплин (см. рис. 6).

Общий алгоритм работы метода hooking:

1. Выбрать одну из инструкций в базовом блоке, сохранить её и записать на её место hook (например, bl hook_func; ldr pc, [hook_func]). Когда приложение выполнит hook – инструкцию, то начнет выполняться код трамплина;
2. Сохранить контекст программы;
3. Осуществить вызов callback – функции;
4. Восстановить контекст программы после вызова callback – функции;
5. Выполнить инструкцию, которая была заменена на hook;
6. Сделать безусловный переход на инструкцию, которая следует за хуком, и продолжить обычное выполнение программы.

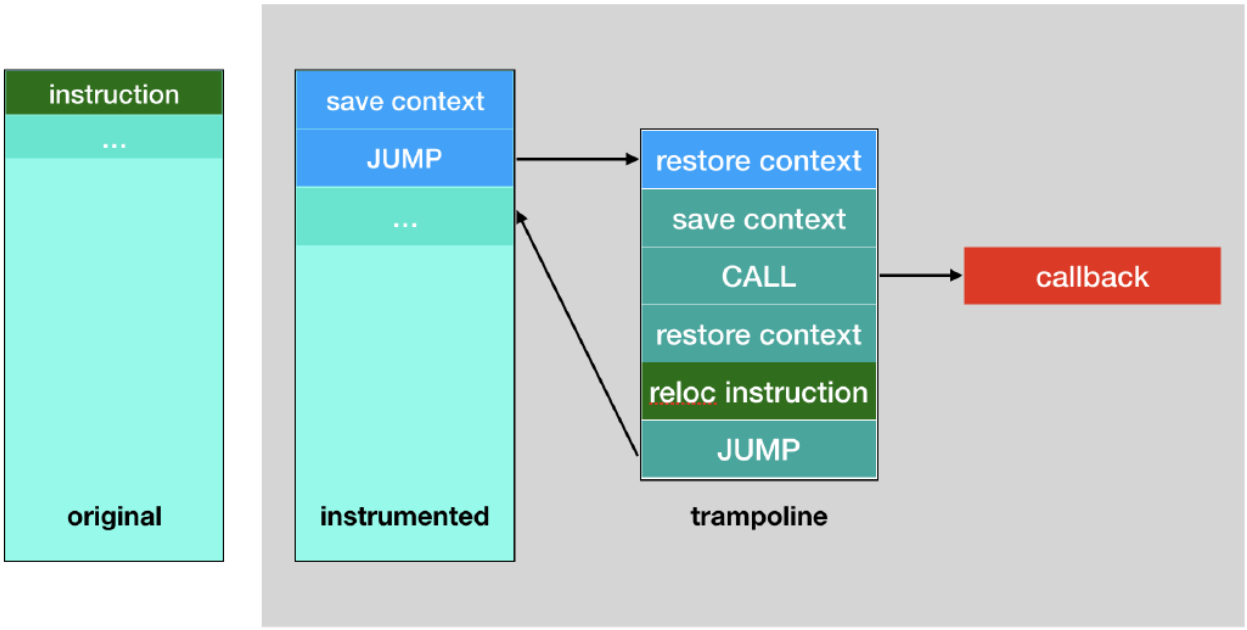

Рисунок 6 – Схема работы метода jump – trampoline

Существует четыре различных метода инструментаци при использовании хуков:

1. Jump – trampoline: происходит прыжок на трамплин, затем следует вызов callback – функции и возобновление работы программы (см. рис. 6)
2. Jump – callback: происходит прыжок не на трамплин, а на саму callback – функцию. При этом восстанавливать контекст и замененную хуком инструкцию приходится делать в этой же функции (см. рис. 7).
3. Call – trampoline: то же самое, что и в п.1, но используется вызов функции (call, bl), а не безусловный переход (jmp, b)
4. Call – trampoline: то же самое, что и в п.2, но используется вызов функции (call, bl), а не безусловный переход (jmp, b)

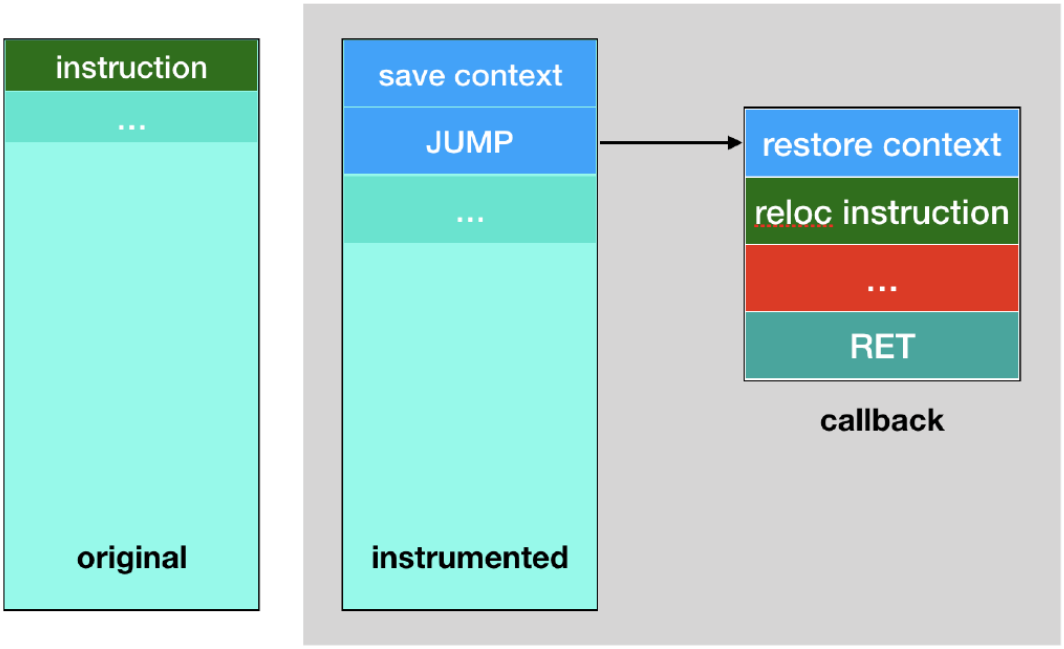

Рисунок 6 – Схема работы метода jump – callback

Основным преимуществом средства Scorpio является скорость работы. Разработчики заявляют, что их средство значительно превосходит по скорости остальные фреймворки (до 100x). В то же время метод hooking не обладает достаточной гибкостью по сравнению с методом JIT – translation, используемым в DynamoRIO и Mambo. Само средство в данный момент находится на стадии тестирования.

Результаты сравнительного анализа

Результаты исследований средств DBI показаны в таблице №1. Символ «+» означает, что средство подходит для реализации taint – анализа по данному критерию, символ «++» - очень подходит, символ «–» - не подходит, символ «?» - не удалось выяснить.

Таблица №1 – Результаты исследований средств DBI

Название средства	Используемый метод	Скорость работы	Применимость в ИБ	Расширения, наработки
DynamoRIO	JIT - translation	+	+	++
Mambo	JIT – translation	++	-	-
Scorpio	hooking	++	?	-

Из таблицы видно, что DynamoRIO лучше подходит для реализации taint – анализа. Благодаря тому, что его основе были созданы расширения Umbra, DrMemory и публично доступный проект DrTaint, реализовать taint – анализ будет проще.

Анализ библиотеки Umbra

Библиотека Umbra – это расширение DBI – фреймворка DynamoRIO, реализующее механизмы управления теневой памятью (shadow memory) процесса [10]. Теневая память - это абстрактная структура данных, сопоставляющая свое виртуальное адресное пространство с адресным пространством процесса по определенной схеме. Например, 1 байт памяти процесса может отображаться на 1 виртуальный байт теневой памяти или 4 байта памяти процесса могут отображаться на 1 виртуальный байт и т.д. С помощью теневой памяти становится возможным контролировать каждый байт памяти процесса на каждом шаге его работы.

Библиотека Umbra предоставляет следующие возможности:

- Создание и инициализация теневой памяти, выбор схемы отображения
- Операции чтения и записи в теневую память
- Обход всех регионов памяти процесса или теневой памяти
- Инструментация кода для доступа к теневой памяти

Клиент может использовать предоставленное библиотекой Umbra API для доступа к теневой памяти, не зная фактического адреса в теневой памяти. Поданный на вход адрес автоматически преобразуется в адрес теневой памяти. Более того, опытные пользователи могут создать специальную теневую память (только для чтения) или напрямую обращаться к ней для повышения производительности. Так как основой для реализации taint – анализа служит именно теневая память, то библиотека Umbra подходит для этой цели как нельзя кстати.

Анализ библиотеки DrTaint

Библиотека DrTaint - это средство, задачей которого является проведение taint – анализа на архитектуре ARM [11]. Основано оно на DBI – фреймворке DynamoRIO и его расширении Umbra. Проект DrTaint на данный момент находится на стадии разработки. Он реализует базовые возможности taint – анализа и способен обрабатывать лишь небольшое количество ассемблерных инструкций.

Для сборки средства DrTaint потребуются:

- Исходные коды проекта DynamoRIO
- Исходные коды проекта DrMemory
- Исходные коды проекта DrTaint

Все действия по сборке будут проходить на целевой машине, так как справиться с ошибками, возникшими во время кросс – компиляции под ARM, разобраться не удалось.

Сначала необходимо собрать проект DynamoRIO:

```
git clone https://github.com/DynamoRIO/dynamorio.git
mkdir build_arm
cd build_arm
cmake ../
make –j
```

После сборки DynamoRIO нужно собрать расширение DrMemory:

```
git clone https://github.com/DynamoRIO/drmemory.git
cd drmemory
make/git/devsetup.sh
cd ..
mkdir build_arm
cd build_arm
cmake ../
make –j
```

Наконец, можно собрать сам DrTaint. Так как он находится в разработке, то нужно подправить пути к cmake – файлам DynamoRIO и DrMemory в Cmakelists.txt. После этого выполнить сборку:

```
git clone https://github.com/toshipiazza/drtaint.git
*** fix paths in cmakelists.txt **
mkdir build_arm
cd build_arm
cmake ../
make
```

Более подробно о сборке проектов DynamoRIO и DrMemory можно узнать из источников [13] и [14].

Проект DrTaint состоит из двух основных частей:

1. Drtaint_shadow. Эта часть отвечает за инициализацию библиотеки, обработку исключительных ситуаций во время её работы (handling faults), реализует базовый интерфейс для взаимодействия с теневой пямятью и регистрами.
2. DrTaint. Эта часть берет на себя основную логику работы. В ней находятся обработчики различных инструкций ассемблера ARM (см. рис. 7). Каждый из обработчиков реализует одну из политик taint – анализа, речь о которых пойдет ниже.

Библиотека DrTaint работает следующим образом. Сначала происходит инициализация теневой памяти и регистров, заполнение всей теневой памяти нулевыми значениями (clear). Интерфейс библиотеки позволяет во время исполнения программы пометить определенный регион памяти или регистр тегом (к примеру, буфер с input - данными), при этом регион теневой памяти, который отображается на реальный, помечается ненулевым значением (taint). Если участок памяти помечен, то необходимо отслеживать распросранение tainted – данных по карте памяти программы. В связи с этим DrTaint в фоновом режиме перехватывает каждую инструкцию ассемблера и перед её исполнением вызывает обработчик для неё. Таким образом реализуется taint – анализ: отслеживание распространения помеченных (tainted, untrusted) данных в программе. Более подробно показать принцип работы библиотеки можно на псевдокоде:

```
DRTAINT: MAKE_TAINTED(user_input)

APP: user_input = input(stdin);                  // user_input is tainted

APP: some_result = process_data(user_input)  // some_result is tainted

APP: call_vuln_func(some_result)                 // alert! function call with untrusted data
```

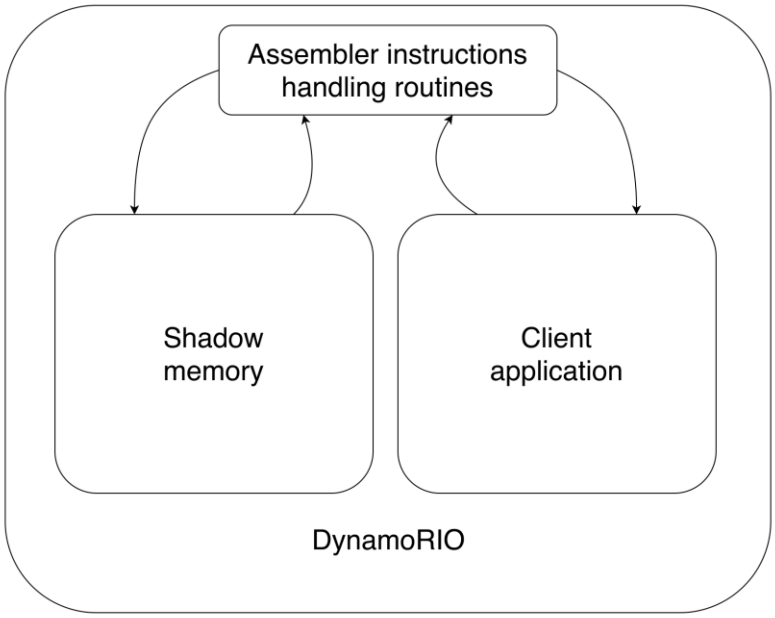

Рисунок 7 – Схема работы DrTaint

Существует четыре политики taint – анализа, согласно которым происходит распространение tainted – данных:

1. Copy Propagation: tainted - данные копируются из одного места в другое командами: mov, mvn, rev, rrx, ldr, ldrd, str, strd, ldm, stm и др.
2. Arithmetic Propagation: tainted - данные являются операндами математического или логического выражения (add, sub, mull, sdiv, and, orr, ror, lsl, lsr, asr и др.).
3. Address Propagation: tainted - данные могут быть использованы для расчета адреса в памяти (ldr - offset addressing, preindex addressing).
4. Control Propagation: tainted - данные могут распространяться посредством влияния на значение другой переменной (все инструкции зависящие от флагов состояния или воздействующие на них)

В библиотеке DrTaint частично реализованы политики 1 и 2, возможно, с ошибками в реализации. Поэтому еще предстоит сделать много работы по доработке, тестированию и оптимизации библиотеки.

Заключение

В процессе работы был проведен сравнительный анализ средств на пригодность к реализации taint – анализа с помощью них. Выяснилось, что DBI – фреймворк DynamoRIO наиболее подходит, потому что у него приемлемый overhead, его можно применять с сфере ИБ, и существуют основанные на нем проекты, облегчающие реализацию taint – анализа на архитектуре ARM. Фреймворки Mambo и Scorpio выигрывают DynamoRIO по скорости работы, однако проигрывают ему по останым критериям.

В качестве средства для доработки была выбрана библиотека DrTaint в связи с тем, что она уже реализует часть требуемого функционала. После детального исследования принципов работы DrTaint было решено добавить новые возможности в библиотеку, протестировать её и оптимизировать по скорости работы.

Предполагается, что доработанное средство поможет реверс - инженерам в исследовании ВПО, а также в поиске бинарных уязвимостей в ПО, связанных с ошибками работы с памятью.

Список источников

1. Malware source code samples [Электронный ресурс]: https://github.com/ifding/iot-malware
2. What is BeagleBone Black [Электронный ресурс]: http://beagleboard.org/BLACK
3. BeagleBone Black Images [Электронный ресурс]: https://beagleboard.org/latest-images
4. Flash OS images [Электронный ресурс]: https://etcher.io/
5. BeagleBone getting started [Электронный ресурс]: http://beagleboard.org/getting-started
6. Linux debian images [Электронный ресурс]:
7. http://http.us.debian.org/debian/ dists/stretch/main/installer-armhf/current /images/netboot/
8. Installing linux on qemu [Электронный ресурс]: https://translatedcode.wordpress. com/2016/11/03/installing-debian-on-qemus-32-bit-arm-virt-board/
9. Emulating linux on arm [Электронный ресурс]: https://gist.github.com/bruce30262 /e0f12eddea638efe7332
10. Efficient, Transparent, and Comprehensive Runtime Code Manipulation \ Derek L. Bruening, 2004
11. low-overhead dynamic binary instrumentation and modification tool for ARM [Электронный ресурс]: https://github.com/beehive-lab/mambo
12. SKORPIO: Advanced Binary Instrumentation Framework [Электронный ресурс]: http://groundx.io/docs/Opcde2018-skorpio.pdf
13. Umbra: Shadow Memory Extension [Электронный ресурс]: ttp://drmemory.org/docs/page_umbra.html
14. Very WIP taint analysis for DynamoRIO (ARM) [Электронный ресурс]: ttps://github.com/toshipiazza/drtaint/
15. Instructions for building DynamoRIO [Электронный ресурс]: https://github.com/DynamoRIO/dynamorio/wiki/How-To-Build
16. Instructions for building DrMemory [Электронный ресурс]: https://github.com/DynamoRIO/drmemory/wiki/How-To-Build