УДК 004.056.53

*И. С. Шишкин, Д. А. Вислобоков, Е. С. Маленков**

ИСПОЛЬЗОВАНИЕ ИНСТРУМЕНТА Tracee ДЛЯ ОБНАРУЖЕНИЯ АТАК НА КОНТЕЙНЕРНЫЕ СРЕДЫ

Введение

Современные облачные и контейнеризированные среды требуют новых подходов к обеспечению безопасности: традиционные антивирусные и сигнатурные системы часто недостаточны для защиты от сложных атак, использующих незаметные векторы проникновения. Особенно актуальной становится защита на уровне времени выполнения (runtime) и мониторинг поведения процессов внутри контейнеров и виртуальных машин [1].

Tracee – это инструмент безопасности на основе eBPF, который отслеживает события на уровне ядра в режиме реального времени, позволяя обнаруживать широкий спектр подозрительных действий – от событий жизненного цикла контейнеров до подробного отслеживания выполнения процессов.

---

* Работа выполнена под руководством доктора технических наук, профессора кафедры ИСиЗИ ФГБОУ ВО «ТГТУ» В. Е. Дидриха.

Технологическая основа Tracee

Tracee использует технологию eBPF (extended Berkeley Packet Filter, расширенный фильтр пакетов университета Беркли). Она внедряет eBPF-программы, которые прослушивают большое число системных вызовов и другие события ядра. Эти события включают, помимо стандартных системных вызовов, «нестандартные события безопасности», которые могут быть связаны с подозрительной активностью. Так что его необходимо запускать от имени суперпользователя [1].

Механизм загрузки и выполнения программ (eBPF) в ядре Linux позволяет безопасно загружать и исполнять «программы» в пространстве ядра, ограничивая их возможности по доступу и ресурсам. Изначально eBPF задумывался для сетевых фильтраций, но позднее его возможности были расширены: его используют для трассировки, мониторинга, безопасности и других задач. eBPF имеет высокую производительность, так как работает в контексте ядра с минимальными производительными расходами. Благодаря верификации (проверки) можно убедиться, что код не повредит систему вследствие чего выполняются условия, обеспечивающие безопасное исполнение.

Программы eBPF, используемые в Tracee, могут перехватывать не только сетевые пакеты или системные вызовы, но и другие важные события ядра:

- обращения к файловой системе;
- операции с процессами (fork, exec, clone);
- изменения прав доступа, работу с памятью (mmap, mprotect);
- взаимодействие между процессами (IPC, signals);
- события загрузки модулей ядра и динамических библиотек.

Это позволяет строить целостную картину поведения процессов, охватывающую не только системные вызовы, но и низкоуровневые операции ядра. Поскольку Tracee работает через eBPF, она не требует стандартных модулей ядра, из-за чего она является более безопасной и портируемой по сравнению с решениями, которые модифицируют ядро напрямую.

Системные вызовы являются основным каналом взаимодействия приложений с ядром – это и открытие файлов, и вызов сетевых функций, и загрузка динамических библиотек. Tracee использует eBPF с момента своего создания и собирает 330 системных вызовов (и других событий, не связанных с системными вызовами) сразу после установки [2].

Пример использования

Для примера был использован контейнер nginx. Tracee позволяет наблюдать за процессами, просмотреть привилегии внутри контейнера (например, если контейнер запущен с флагом --privileged), запущенными внутри контейнера nginx.

Ниже приведен пример нескольких первых событий из трассировки контейнера с запущенным nginx.

Запуск контейнера:

```
$ docker run -it --rm nginx
```

Для трассировки событий cap_capable используем Tracee с флагом:

```
root@desk$ ./tracee.py -c -e cap_capable
TIME(s) UTS_NAME UID EVENT COMM PID PPID RET ARGS
125.000 c8520fe719e5 0 cap_capable nginx 6 1 0 CAP_SETGID
125.000 c8520fe719e5 0 cap_capable nginx 6 1 0 CAP_SETGID
124.964 c8520fe719e5 0 cap_capable nginx 1 3500 0 CAP_SYS_ADMIN
```

Зная, какие привилегии необходимы контейнеру для работы приложения, можно, следуя принципу минимума полномочий, указать в среде выполнения лишь необходимые привилегии. Либо убрать все привилегии, после чего добавить необходимые.

Для того чтобы обнаруживать запуск исполняемых файлов внутри контейнера, применяется технология eBPF. Для примера также был взят контейнер с nginx, так как в обычных условиях процессы, которые можно встретить в этом контейнере, должны принадлежать nginx. Запустив Tracee в отдельном окне терминала, был запущен контейнер nginx:

```
$ docker run --rm -d --name nginx nginx
```

После запуска контейнера Tracee показывает, как запускается исполняемый файл nginx:

```
EVENT ARGS
execve /usr/sbin/nginx
```

Теперь отправим в контейнер команду ls, чтобы увидеть ее вывод в Tracee:

```
$ docker exec -it nginx ls
```

Вывод в терминале Tracee:

```
EVENT ARGS
execve /usr/sbin/nginx
execve /bin/ls
```

Команда ls сразу выводится в терминале с Tracee. Данный пример демонстрирует, что с помощью Tracee и eBPF можно заметить запуск исполняемых файлов, которые злоумышленник мог поместить в контейнер, например программу по добычи криптовалюты.

Разработка правил для Tracee

Поскольку Tracee позволяет писать новые правила на языках Rego и Golang, это открывает множество вариантов для создания набора правил, подходящего для конкретной среды [3].

Создадим для Tracee правило, которое срабатывает при каждом доступе к файлу сокета Docker из контейнера. Это может означать, что злоумышленнику удалось смонтировать его внутри контейнера и он пытается повысить свои привилегии на хосте или атаковать другие контейнеры.

Для начала проверим, может ли Tracee обнаруживать события такого типа. Tracee может предоставить список возможных событий с помощью флага -l. Наиболее подходящим из списка является событие security_socket_connect.

После определения возможного события необходимо убедиться, что оно будет работать – использовать функцию режима необработанной трассировки Tracee (при условии, что используется обновленное ядро, поддерживающее BTF), чтобы вывести экземпляры этого события, происходящие без каких-либо привязанных правил безопасности. Эта команда будет использовать Tracee для отображения экземпляров события security_socket_connect:

```
$ docker run --rm --pid=host --privileged -v /tmp/tracee:/tmp/tracee
-it aquasec/tracee:latest trace --trace event=security_socket_connect
```

После запуска Tracee используем команду docker info для генерации некоторых событий:

```
TIME             UID  COMM  PID      TID      RET  EVENT
14:17:24:405875  0    snapd    1832024  1832050  0  security_socket_connect
14:17:24:346858 1000 docker   2608941  2608941  0  security_socket_connect
14:17:24:380853 1000 docker-app 2601199 2601199 0  security_socket_connect
ARGS
10, remote_addr: {'sa_family': 'AF_UNIX', 'sun_path': '/run/systemd/notify'}
3,  remote_addr: {'sa_family': 'AF_UNIX', 'sun_path': '/var/run/nscd/socket'}
3,  remote_addr: {'sa_family': 'AF_UNIX', 'sun_path': '/var/run/docker.sock'}
```

Из вывода видно, что есть и другие случаи доступа к сокету, которые мы, вероятно, хотим избежать в качестве предупреждений. Важно,

что предоставлены аргументы, связанные с событием. Из них видно, что аргумент remote_addr содержит имя сокета, что позволит сопоставить только те случаи, когда происходит доступ к файлу docker.sock.

Необходимо заполнить метаданные правила. Установить уникальный идентификатор, краткое описание правила и соотнести с фреймворком MITRE Container ATT&CK

```
__rego_metadoc__ := {
    "id": "RMM-1",
    "version": "0.0.1",
    "name": "Docker Socket Access Detected",
    "description": "A container accessed the Docker socket, allowing for control of the container daemon and possible privilege escalation",
    "tags": ["container"],
    "properties": {        "Severity": 0,
        "MITRE ATT&CK": "Container Administration Command" }}
```

После необходимо указать источник события. Ключевыми элементами являются названия событий, которые были определенны ранее. Установка этого параметра в значении container означает, что будут отображаться только события, которые были сгенерированы внутри контейнера.

```
eventSelectors := [    {        "source": "tracee",        "name": "security_socket_connect",        "origin": "container"    }]
```

Далее укажем, что предупреждение должно срабатывать, когда переменная remote_addr содержит строку docker.sock, чтобы не получать ложные срабатывания от других подключений к сокету внутри контейнера.

```
tracee_selected_events[eventSelector] {    eventSelector := eventSelectors[_]}tracee_match {    input.eventName == «security_socket_connect»    addr := helpers.get_tracee_argument(«remote_addr»)    contains(addr,»docker.sock»)}
```

Теперь можно протестировать правило, оно отработает и выведет в терминал предупреждение об обнаружении угрозы.

Заключение

Основным преимуществом Tracee является способность обнаруживать угрозы в режиме реального времени. Поскольку инструмент функционирует на уровне ядра, он фиксирует события с минимальной

задержкой и высокой точностью. Использование eBPF позволяет достигать высокой производительности при низких накладных расходах, а открытая архитектура обеспечивает возможность адаптации под конкретные задачи безопасности.

Список литературы

1. Liz Rice, Fundamental Technology Concepts that Protect Containerized Applications // Container Security. (2021). 203 – 206.

2. Aqua Tracee: Runtime eBPF threat detection engine [Электронный ресурс]. – URL : https://www.aquasec.com/products/tracee/#block_2 (дата обращения: 07.10.2025).

3. Tracee Runtime Security Series: Writing Custom Tracee Rules [Электронный ресурс]. – URL : https://www.aquasec.com/blog/runtime-security-tracee-rules/ (дата обращения: 08.10.2025).

Кафедра «Информационные системы и защита информации» ФГБОУ ВО «ТГТУ»