Конфигурируемая модульная система мониторинга поведения транспортного протокола на уровне ядра операционной системы

В. А. Пономарев О. Ю. Богоявленская
Ю. А. Богоявленский

27 ноября 2008 г.

1 Введение

В работе [1] описана система `GetTCP`, которая позволяет получать локализованные в ядре операционной системы (ОС) Linux данные о состоянии и динамике внутренних переменных, отражающих поведение соединений TCP. Кроме того, при разработке `GetTCP` были приняты меры по уменьшению накладных расходов, возникающих при работе системы. Указанные особенности позволяют использовать систему `GetTCP` для ряда задач, решение которых невозможно с помощью общеизвестных средств мониторинга поведения TCP. Так, система `GetTCP` применялась для сбора данных, необходимых для проверки адекватности модели TCP [2]. С точки зрения безопасности такая система предоставляет возможность интегрировать в ОС все возможные функции защиты на уровне TCP.

Вместе с тем, первоначальный вариант системы `GetTCP` обладал рядом недостатков, описанных в [3]. Наиболее существенные из этих недостатков - отсутствие библиотеки пользовательского уровня, обеспечивающей доступ к возможностям системы, а также необходимость перекомпиляции загружаемого модуля ядра при любом изменении набора перехватываемых данных.

В статье описана новая версия системы `GetTCP`, в которой были устранены указанные недостатки. Также рассматриваются примеры использования новой версии.

2 Получение управления при передаче данных

Возможность перехвата внутренних данных ядра ОС - ключевая особенность системы `GetTCP`. Очевидно, что для доступа к таким данным часть системы должна действовать в адресном пространстве ядра и получать управление процессором при отправлении каждого сегмента TCP. Рассмотрим варианты, которыми можно реализовать получение управления подсистемой перехвата при отправке сегмента.

В первоначальной реализации системы `GetTCP` применялся простой способ без использования каких-либо готовых механизмов ядра. В исходный код ядра ОС Linux, осуществляющий передачу сегмента TCP, непосредственно перед обращением к функции отправки данных сетевого уровня добавлялся код, передающий управление системе `GetTCP`.

Указанный подход требует модификации исходного кода ядра, перекомпиляции ядра и перезагрузки вычислительной системы для начала работы. В то же время в современном ядре ОС Linux существует возможность динамической установки "контрольных точек" и последующего вызова заданной при установке "контрольной точки" функции с помощью механизма `KProbes` [4]. Механизм предоставляет несколько разновидностей "контрольных точек", была исследована одна из этих разновидностей - `jprobe`.

В связи с тем, что величина задержек, вносимых системой `GetTCP` в передачу сегмента данных, является важнейшим элементом спецификации системы, были проведены эксперименты по измерению времени, необходимого для получения управления с использованием механизма `jprobe`. Измерения проводились на той же вычислительной системе, которая использовалась для экспериментов в [1] (ПЭВМ на основе процессора Celeron с тактовой частотой 3ГГц).

Выяснилось, что получение управления занимает минимум 3185 тактов процессора, что составляет около 1 мкс. Учитывая то, что в старой версии системы `GetTCP` общие накладные расходы на обработку сегмента в 94.02% случаев не превышали 0.3 мкс, было принято решение отказаться от использования `KProbes`.

Любопытно, что при проведении измерений пришлось учитывать влияние попадания или непопадания кода в кэш-память процессора (при непопадании время выполнения тестовой функции увеличивалось на несколько порядков). Поэтому проводилось большое количество (сотни тысяч) измерений с минимальными временны́ми интервалами между измерениями, чтобы код тестовой функции не успевал покинуть кэш процессора. Для минимизации этих временны́х интервалов, в свою очередь, было реализовано программное обеспечение (ПО) для тестирования на языке Си. Первоначально использовался язык Perl, однако выяснилось, что при этом код тестовой функции успевает покинуть кэш процессора и результаты измерений получаются недостоверными.

Другой механизм - kernel markers [5], стал доступен сравнительно недавно и был включен в "официальное" ядро ОС Linux в версии 2.6.24, которая вышла 24 января 2008 года (как раз в то время, когда проводились описанные выше измерения для механизма `KProbes`).

Kernel markers представляют собой статические "маркеры", заранее включенные в исходный код ядра ОС Linux (в отличие от динамических контрольных точек механизма `KProbes`, которые устанавливаются во время выполнения). Такой подход имеет свои преимущества и области применения. Статические "маркеры" сопровождаются вместе с исходным кодом ядра ОС Linux, и, следовательно, всегда согласованы с ним. Кроме того, разработчиками kernel markers были приняты специальные меры по уменьшению накладных расходов при использовании этого механизма.

В экспериментальное ПО, разработанное ранее для `KProbes`, было добавлено измерение накладных расходов механизма kernel markers. Выяснилось, что для получение управления требуется 138 тактов процессора, т.е. около 0.05 мкс (измерения проводились на той же ПЭВМ на основе процессора Celeron с тактовой частотой 3ГГц). При этом измерялось непосредственно время, затрачиваемое на вызов функции из "маркера", в отличие от подхода, примененного в [6], где измерялись косвенные "макропараметры" (время, затрачиваемое на монтирование-размонтирование тома ext3).

Учитывая указанные выше обстоятельства, было принято решение об использовании для новой версии `GetTCP` механизма kernel markers. Необходимость внесения изменений в исходный код ядра и последующей перекомпиляции ядра при использовании статических "маркеров" были признаны неизбежными, т.к. другие доступные механизмы (`jprobe`) приводят к недопустимому увеличению задержки при отправке сегмента TCP.

Для размещения “маркера” в ядре используется следующая конструкция:

```
trace_mark(gettcp_tcp_probe, "sk_buff %p integer %d", skb, id);
```

Здесь `gettcp_tcp_probe` - имя “маркера”, `skb` - указатель на внутреннюю структуру данных ядра ОС Linux (socket buffer), а `id` позволяет однозначно определить строку кода, из которой была вызвана передача сегмента TCP, и, следовательно, условия, в которых происходила передача сегмента (например, состояние алгоритма избежания перегрузок TCP).

3 Архитектура новой версии `GetTCP`

Существенным образом была переработана внутренняя архитектура части системы `GetTCP`, находящейся в ядре ОС. Эта часть была разделена на четыре подсистемы: управления, передачи данных на пользовательский уровень, получения необходимых данных, измерения накладных расходов, возникающих при работе. Для каждой из этих подсистем доступны три обязательные функции: инициализация подсистемы, сообщение о состоянии подсистемы (для просмотра пользователем), завершение работы подсистемы. Также доступны специфичные для каждой из подсистем функции (изменение конфигурации, принудительная смена под-буфера и т.п.).

Подсистема управления осуществляет получение команд от пользовательского уровня и их обработку. Команды в новой версии `GetTCP` представляют собой строки, состоящие из обязательной команды и одного или нескольких параметров, разделенных пробельными символами (см. пример ниже).

Реализация текстовых сообщений о результате выполнения команд привела бы к чрезмерному усложнению и увеличению кода, работающего в адресном пространстве ядра, что нежелательно. Поэтому было принято решение о компромиссе - реализация интерфейса посредством текстовых команд с ответами в виде кода возврата системного вызова `write`, с помощью которого подается команда. Таким образом, ошибка при выполнении команды выглядит как ошибка записи в управляющий файл:

```
# echo недопустимая команда > /sys/kernel/debug/gettcp/control
bash: echo: write error: Invalid argument
#
```

Отсутствие ошибки выглядит как успешная запись в управляющий файл:

```
# echo chan create 131072 16 > /sys/kernel/debug/gettcp/control
#
```

При обработке команд учитывается состояние, в котором находится система `GetTCP` в момент поступления команды. Например, невозможно включить перехват данных, если не задана конфигурация перехвата, или не создан канал передачи данных на пользовательский уровень.

Для реализации подобных ограничений используется конечный автомат, действующий в четырехмерном пространстве состояний, каждая точка которого может быть описана как $(chan, conf, prof, probe)$, где $chan$ - состояние подсистемы передачи данных на пользовательский уровень, $conf$ - состояние конфигурации перехватываемых данных, $prof$ - состояние подсистемы измерения накладных расходов, $probe$ - состояние подсистемы получения необходимых данных. Множество допустимых состояний, переходов между ними, а также команд, вызывающих эти переходы, описано в специальной структуре данных. Эта структура данных при необходимости легко может быть изменена и дополнена. Таким образом, обеспечивается возможность легкого расширения и модификации системы команд `GetTCP`.

Подсистема передачи данных на пользовательский уровень по сравнению с предыдущей версией `GetTCP` изменилась незначительно. Для количества заполненных и обработанных под-буферов в новой версии используются строки, а не двоичные значения. За время, прошедшее с публикации [1], механизм эффективной передачи данных из ядра на пользовательский уровень из файловой системы RelayFS трансформировался в более общий программный интерфейс relay API, для организации файлового интерфейса используется файловая система debugfs [7].

Подсистема получения данных была реализована с помощью описанного выше механизма "маркеров". Также появилась возможность динамического управления набором данных, которые необходимо перехватывать.

Подсистема измерения накладных расходов также была реализована с помощью механизма kernel markers ("маркеры" находятся в коде подсистемы получения данных). Основное отличие от предыдущей версии - возможность динамического управления конфигурацией (в новой версии можно указывать ожидаемые диапазоны измерений и количество участков гистограммы).

Значительные изменения произошли в части `GetTCP`, действующей в пользовательском адресном пространстве. Ранее эта часть фактически являлась прототипом и состояла из одной программы, сохраняющей перехваченные данные в файл для последующей обработки. В новой версии `GetTCP` реализована библиотека `libgettcp`, предоставляющая функции для создания и уничтожение канала передачи данных relay API, управления конфигурацией перехвата, запуска и остановки измерения накладных расходов, запуска и остановки перехвата, получения перехваченных данных. Указанную библиотеку можно применять как в оригинальных программах, нуждающихся в возможностях, которые предоставляет только система `GetTCP`, так и в уже существующих утилитах перехвата и обработки сетевого трафика. Для проверки возможности применения библиотеки `libgettcp` в уже существующих программах, а также для тестирования работы библиотеки была использована утилита `fprobe-ulog` [9], служащая для перехвата сетевого трафика с последующим экспортом в формате NetFlow [10]. Указанная утилита была модифицирована для использования библиотеки `libgettcp`. Объем модификаций оказался незначительным (размер полученного diff-файла составил 5.8 кБ). Проведенные эксперименты показали работоспособность полученной модификации утилиты `fprobe-ulog`, а также отсутствие потерь данных при перехвате и формировании потоков NetFlow.

В дополнение к описанным выше изменениям в код всех частей системы `GetTCP` были внесены исправления для работы на 64-битных платформах. Также была проверена компиляция и работа новой версии `GetTCP` на многоядерных платформах (AMD Athlon X2, Intel Core 2 Duo).

В настоящее время ведется работа по улучшению временны́х характеристик подсистемы перехвата и подготовке пользовательской документации. Также разрабатывается библиотека для обеспечения возможности хранения конфигурации системы `GetTCP` в отдельном файле с Си-подобным синтаксисом.

4 Эксперимент по определению реактивности

Для новой версии `GetTCP` был повторен эксперимент по определению задержек, вносимых системой в передачу сегментов TCP. Измерения проводились на следующей аппаратно-программной платформе: ПЭВМ на основе процессора Celeron с тактовой частотой 3ГГц, оперативная память объемом 1.5Гб, интегрированная сетевая карта SiS900 10/100 Мбит/с, локальная сеть Ethernet 100

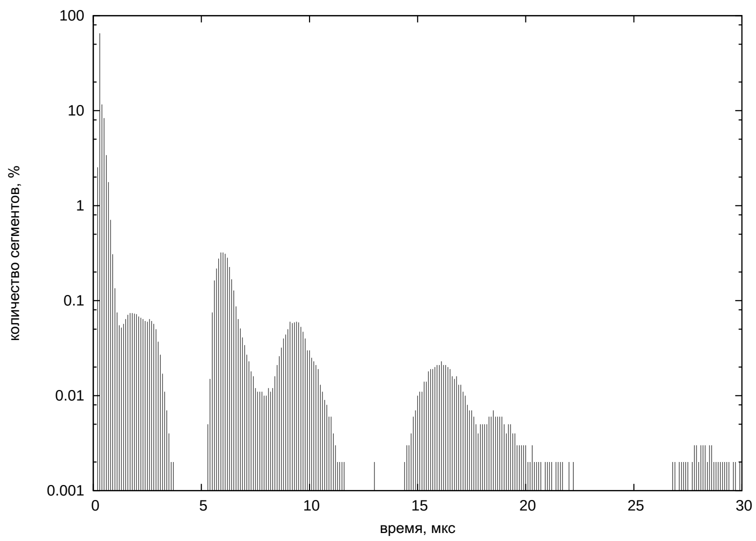

Рис. 1: Гистограмма распределения задержек, вносимых GetTCP в передачу сегмента TCP

Интервал, мкс	Количество сегментов, %
$(0, 0.2]$	2.5
$(0.2, 0.3]$	65.2
$(0.3, 0.4]$	11.6
$(0.4, 0.5]$	8.4
$(0.5, 0.6]$	3.4
$(0.6, 0.7]$	1.8
$(0.7, 0.8]$	0.7
$(0.8, 0.9]$	0.3
$(0.9, 1]$	0.1
> 1	< 6.1

Таблица 1: Задержки, вносимые GetTCP в передачу сегмента TCP

Мбит/с, ОС Linux, ядро 2.6.25.11 (дистрибутив openSUSE 11.0). Изменения аппаратной части платформы по сравнению с 2006 г. незначительны (объем оперативной памяти увеличен с 512Мб до 1.5Гб). Наиболее значительно изменилась программная часть (ядро 2.6.25 вместо 2.6.16).

Для оценки задержки, вносимой `GetTCP` в передачу сегмента, был проведен эксперимент, аналогичный описанному в [1]. С помощью утилиты Iperf [8] передавался максимально возможный объем данных за фиксированный интервал времени (120 секунд) при фиксированном размере сегмента. Для имитации наиболее неблагоприятных условий работы (минимальное время между отправками сегментов данных) был выбран размер сегмента 50 байт.

Полученные результаты представлены в таблице 1 и на гистограмме (рис. 1) распределения задержек, вносимых `GetTCP` в передачу сегмента TCP (ось ординат в логарифмическом масштабе). Для большинства сегментов (94%) задержка не превышает 1 мкс.

5 Заключение

В статье представлена новая версия системы `GetTCP`, обладающая рядом преимуществ по сравнению с предыдущей версией. При реализации использованы системные механизмы, обеспечивающие малое влияние процесса мониторинга на временны́е характеристики алгоритмов TCP ядра. Вместе с тем, при реализации новых возможностей (таких, как динамическое изменение конфигурации) не удалось избежать незначительного увеличения вносимых при перехвате задержек. В ближайшее время планируется проведение экспериментов по определению временны́х характеристик `GetTCP` на более современном оборудовании (процессор Intel Xeon E5420) и более высоких скоростях передачи данных (1 Гбит/с). Также ведется работа по трансформации системы в свободный программный продукт.

Список литературы

[1] В. А. Пономарев, О. Ю. Богоявленская, Ю. А. Богоявленский, Система мониторинга поведения транспортного протокола на уровне ядра операционной системы, материалы Второй международной научной конференции по проблемам безопасности и противодействия терроризму, стр. 349-357

[2] В. А. Пономарев, О. Ю. Богоявленская, Ю. А. Богоявленский, Проверка адекватности и модификация модели случайного потока, генерируемого транспортным протоколом TCP в сети передачи данных, труды международного семинара "Распределенные компьютерные и телекоммуникационные сети: теория и приложения (DCCN 2007)", том 2, стр. 59—64

[3] В. А. Пономарев, В. Н. Ковалев, О. Ю. Богоявленская, Ю. А. Богоявленский, Расширение функций системы мониторинга поведения транспортного протокола на уровне ядра ОС Linux, Труды XIV Всероссийской научно-методической конференции "Телематика 2007", том 1, стр. 101—102

[4] An introduction to KProbes,
`http://lwn.net/Articles/132196/`

[5] Kernel markers,
`http://lwn.net/Articles/245671/`

[6] Measuring Kernel Marker Overhead,
`http://kerneltrap.org/Linux/Measuring_Kernel_Marker_Overhead`

[7] Debugfs,
http://lwn.net/Articles/115405/

[8] The TCP/UDP Bandwidth Measurement Tool (iperf),
`http://dast.nlanr.net/Projects/Iperf`

[9] NetFlow probes: fprobe and fprobe-ulog,
`http://fprobe.sourceforge.net/`

[10] Cisco IOS NetFlow Introduction,
`http://www.cisco.com/go/netflow`