Система мониторинга поведения транспортного протокола на уровне ядра операционной системы

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

16 января 2007 г.

Введение

Протокол Transmission Control Protocol (TCP) [1, 2] играет ключевую роль в управлении трафиком Интернет, контролируя двунаправленные соединения точка-точка. Алгоритмы протокола определяют качественные и количественные характеристики потоков данных большинства приложений Интернет.

Именно поэтому поведение реализаций протокола TCP оказывает определяющее влияние на безопасность, устойчивость, надежность и производительность современных сетей. Исследование указанных свойств TCP является весьма актуальным и требует совершенных инструментальных средств мониторинга.

Существующие общеизвестные средства мониторинга поведения TCP [3, 4] позволяют решать широкий спектр задач, но не предназначены для выявления фактов применения несанкционированных информационных воздействий, так как не предоставляют локализованные в ядре ОС данные о динамике состояний и значений переменных, отражающих поведение соединений TCP. Таким образом, задача разработки системы, предоставляющей такие данные, является актуальной.

С точки зрения безопасности такая система будет исчерпывающим решением, позволяющим интегрировать в ОС все возможные функции защиты на уровне TCP. С точки зрения моделирования поведения TCP такая система будет предоставлять исследователю всеобъемлющие данные, необходимые для построения и проверки адекватности моделей.

В статье представлен прототип системы GetTCP, представляющей собой расширение реализации TCP ядра ОС Linux. Система использует файловую систему relayfs [5] для передачи из ядра в пространство пользователя (с последующей записью в файл) полной информации об изменении во времени состояний и значений переменных ссуществующих TCP-соединений.

GetTCP работает без потерь, в реальном масштабе времени, практически не влияет как на временны́е характеристики TCP, так и на работу других сетевых компонент ядра. Предусмотрена возможность применения GetTCP в многопроцессорных системах.

Статья организована следующим образом. В разделе 1 описана архитектура системы, в разделе 2 – ее реализация. Раздел 4 содержит описание некоторых возможностей применения системы для решения задач безопасности. Эксперимент по определению реактивности GetTCP представлен в разделе 3.

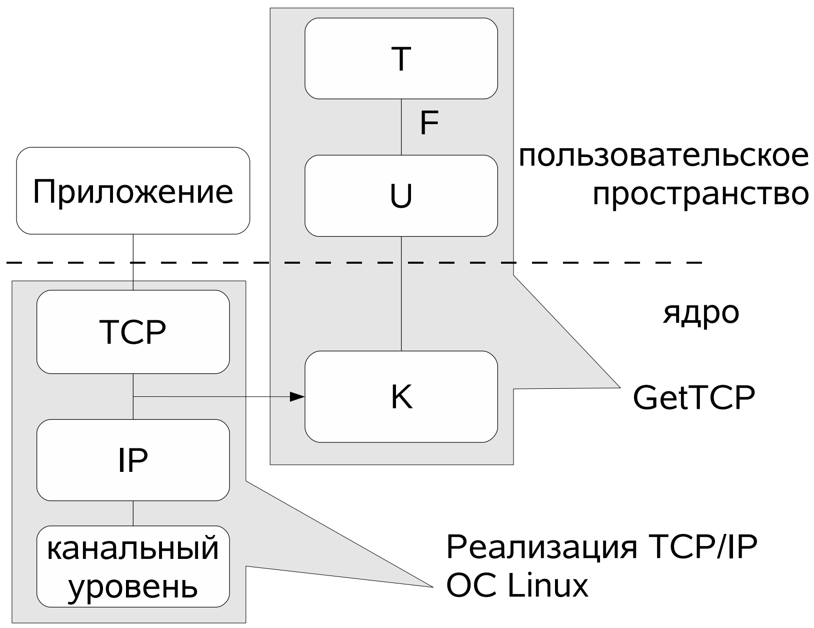

Рис. 1: Архитектура системы GetTCP

1 Основные архитектурные решения

Служебные структуры данных и переменные TCP/IP доступны только в ядре ОС Linux, где, следовательно, и должна действовать часть системы.

С другой стороны, данные нужно хранить в файлах. Надежно и с минимальным влиянием на временны́е характеристики ядра эту функцию следует реализовывать на пользовательском уровне. Причины, по которым нежелательно осуществлять запись в файл напрямую из ядра ОС (снижение надежности, потенциальные проблемы с обеспечением безопасности, отсутствие стандартного программного интерфейса и, как следствие, необходимость использования "трюков" при программировании), описаны в [6].

Архитектура системы представлена на рис. 1. Блок K работает в адресном пространстве ядра, получая данные, которые необходимо записывать, при отправке каждого сегмента TCP. Блок U работает в пользовательском адресном пространстве и записывает полученные данные в файл F. Для уменьшения объема он записывается в двоичном формате, поэтому необходим блок T для преобразования файла к виду, удобному для дальнейшего анализа.

2 Реализация

Описанная выше архитектура была реализована с использованием ядра операционной системы Linux версии 2.6.16 (дистрибутив OpenSUSE 10.1).

Очевидна необходимость передачи данных от блока K к блоку U с минимальными накладными расходами. Для этого целесообразно использовать работающую в оперативной памяти специальную файловую систему relayfs, которая предназначена для эффективной передачи больших объемов данных из адресного пространства ядра в пользовательское адресное пространство.

Блок K был реализован как загружаемый модуль ядра gettcp_kernel, непосредственно выполняющий функции мониторинга путем взаимодействия с подсистемой TCP/IP ядра. Блок U записи данных в файл F был реализован как исполняемый файл gettcp_save.

В исходный файл подсистемы TCP ядра, содержащий код отправки TCP-сегмента, были добавлены вызовы функций GetTCP, а также код, обеспечивающий запись состояния TCP, в котором произошла передача (возможно, повторная) сегмента данных и код, обеспечивающий работоспособность ядра при незагруженном модуле gettcp_kernel.

При загрузке этого модуля создается канал передачи данных relayfs, определенный для модуля gettcp_kernel как кольцевой буфер, состоящий из нескольких под-буферов, в которые записываются поступающие данные. В программе gettcp_save этот канал, согласно архитектуре relayfs, представлен как каталог файловой системы, содержащий файл управления модулем gettcp_kernel, файл характеристик реактивности системы и набор файлов данных мониторинга (по три на каждый процессор), предназначенных для дальнейшей записи в долговременную память.

Эти три файла содержат передаваемые данные и счетчики заполненных gettcp_kernel и прочитанных gettcp_save под-буферов, которые обеспечивают управление кольцевым буфером по стандартной схеме. Таким образом, осуществляется контроль потока данных от модуля gettcp_kernel к программе gettcp_save; при переполнении кольцевого буфера передача данных со стороны ядра приостанавливается и формируется информация о количестве потерянных данных о сегментах TCP. Эта ситуация может быть исключена путем подбора параметров GetTCP.

Программа gettcp_save создает для каждого процессора поток (thread), ожидающий появления новых данных, и записывающий их в в файл по мере появления. При этом поток активируется только при заполнении очередного под-буфера, а все остальное время проводит в состоянии ожидания. При активации поток читает данные, появившиеся с момента предыдущей активации, записывает их в файл и обновляет счетчик обработанных под-буферов.

Для доступа к данным, передаваемым через relayfs, программа gettcp_save использует ввод-вывод, основанный на отображения файла в память. При этом кольцевой буфер с помощью соответствующего системного вызова отображается в пользовательское адресное пространство, и дальнейший доступ к данным осуществляется как к обычному массиву, что существенно уменьшает накладные расходы за счет исключения файловых операций.

Таким образом, реактивность системы GetTCP обеспечивается за счет:

- использование расположенной в оперативной памяти файловой системы relayfs;
- реализации программы gettcp_save с помощью потоков, активирующихся только при заполнении очередного под-буфера;
- организации доступа к файлам данных как к массивам.

3 Эксперимент по определению реактивности

В ходе работы модуль gettcp_kernel системы GetTCP постоянно вычисляет время, потребовавшееся для сохранения порции данных об одном сегменте TCP в буфере для последующей передачи на пользовательский уровень. Для этого используется регистр TSC, доступный в процессоре Pentium (и в следующих за ним моделях процессоров), и содержащий счетчик, увеличивающийся на единицу с каждым тактом процессора. Применение регистра TSC позволяет достичь высокой точности измерения времени (при тактовой частоте процессора $3 \cdot 10^9$ Гц разрешение равно $\frac{1}{3} \cdot 10^{-9}$ с).

Интервал, мкс	Количество сегментов, %
$(0 - 0.32483]$	94.02
$(0.32483 - 0.64966]$	3.44
$(0.64966 - 0.97448]$	1.19
$(0.97448 - 1.2993]$	0.43
$(1.2993 - 1.6241]$	0.11
$(1.6241 - 1.9490]$	0.03
$(1.9490 - 2.2738]$	0.01
$(2.2738 - 2.5986]$	0.01
$(2.5986 - 2.9234]$	0.33
$(2.9234 - 3.2483]$	0.18
$(3.2483 - 3.5731]$	0.04
$(3.5731 - 3.8979]$	0.02
$(3.8979 - 4.2228]$	0.01
$(4.2228 - 4.5476]$	0.02
> 4.5476	< 0.1

Таблица 1: Задержки, вносимые GetTCP в передачу сегмента TCP

Измеренная величина задержки, внесенной GetTCP в передачу сегмента данных с уровня TCP на уровень IP, а также известный размер сегмента, используются для построения гистограммы распределения величины задержки и размера сегмента. Гистограмма затем может быть прочитана из пользовательской программы.

Для оценки задержки, вносимой GetTCP в передачу сегмента, был проведен эксперимент, в ходе которого с помощью утилиты для измерения производительности на транспортном уровне Iperf [7] передавался максимально возможный объем данных за фиксированный интервал времени (10 секунд). Такая передача проводилась для различных максимальных длин сегментов (от 100 до 1500 байт с шагом 50 байт).

Измерения проводились на следующей платформе: две ПЭВМ на основе процессора Celeron с тактовой частотой 3 ГГц, оперативная память объемом 512 Мб, интегрированная сетевая карта SiS900 10/100 Мбит/с, локальная сеть Ethernet 100 Мбит/с, ОС Linux, ядро 2.6.16 (дистрибутив OpenSUSE 10.1)

Полученные результаты представлены в таблице 1 и на гистограмме (рис. 2) распределения задержек, вносимых GetTCP в передачу сегмента TCP (ось ординат в логарифмическом масштабе). Для большинства сегментов (94.02%) эта задержка не превышает 0.32483 мкс.

4 Применение системы для решения задач безопасности

Особенность системы GetTCP в том, что кроме обычной информации, доступной на пользовательском уровне (заголовки сегментов TCP, заголовки IP и т.п.), становятся доступными значения внутренних переменных ядра ОС Linux, использующиеся при работе алгоритмов TCP, такие как текущий размер окна или время кругового оборота (round-trip time, RTT). Это позволяет использовать GetTCP в ситуациях, когда несанкционированное информационное воздействие сложно (или невозможно) обнаружить, имея доступ только к заголовкам TCP и IP.

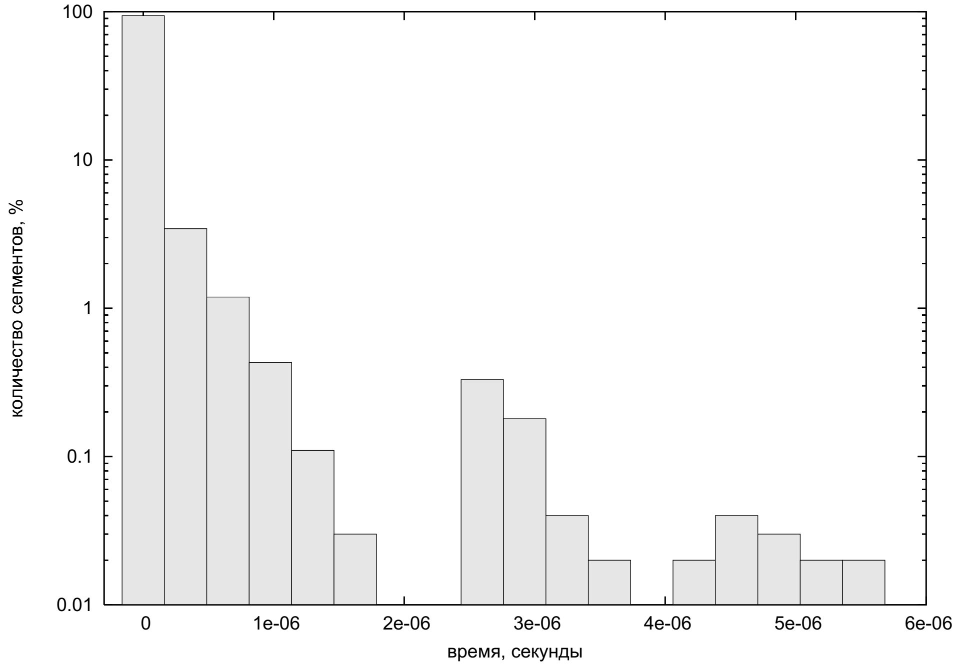

Рис. 2: Гистограмма распределения задержек, вносимых GetTCP в передачу сегмента TCP

При этом возможно построение системы, сочетающей как доступ к внутренним данным ядра ОС, так и достаточно сложные алгоритмы, выполняющиеся в пользовательском пространстве, которые нецелесообразно реализовывать в ядре ОС.

В качестве примера несанкционированного информационного воздействия можно привести описанную в [8] DoS-атаку, использующую особенности алгоритма предотвращения насыщения в TCP, и приводящую к катастрофическому падению пропускной способности TCP-соединения. Особенностью этой DoS-атаки является сравнительно небольшая интенсивность вредоносного трафика, что затрудняет ее обнаружение (в отличие от перегрузки маршрутизаторов большими объемами трафика, т.н. flood).

Суть атаки [8] в том, что с помощью трафика с определенными временны́ми характеристиками (всплески заданной длительности через заданные интервалы времени) достигается переход TCP-алгоритма жертвы в состояние ожидания повторной передачи (retransmission timeout, RTO).

Доступ к внутренним переменным алгоритма TCP позволяет определить, чем было вызвано падение пропускной способности соединения и принять необходимые меры (например, каким-либо образом изменить минимальный интервал ожидания при повторной передаче, minRTO, исключив тем самым совпадение периодов вредоносного трафика и попыток повторной передачи алгоритма TCP жертвы).

Другой класс атак на транспортном уровне — отправка сегментов с поддельным адресом отправителя и информацией, приводящей, например, к разрыву TCP-соединения (или к невозможности установить TCP-соединение). В качестве примера можно привести атаку [9]. Набор утилит, входящий в комплект dsniff, использует технологию перехвата данных, описанную, например, в [10]. Утилита tcpkill, входящая в состав dsniff, позволяет разрывать указанное со-

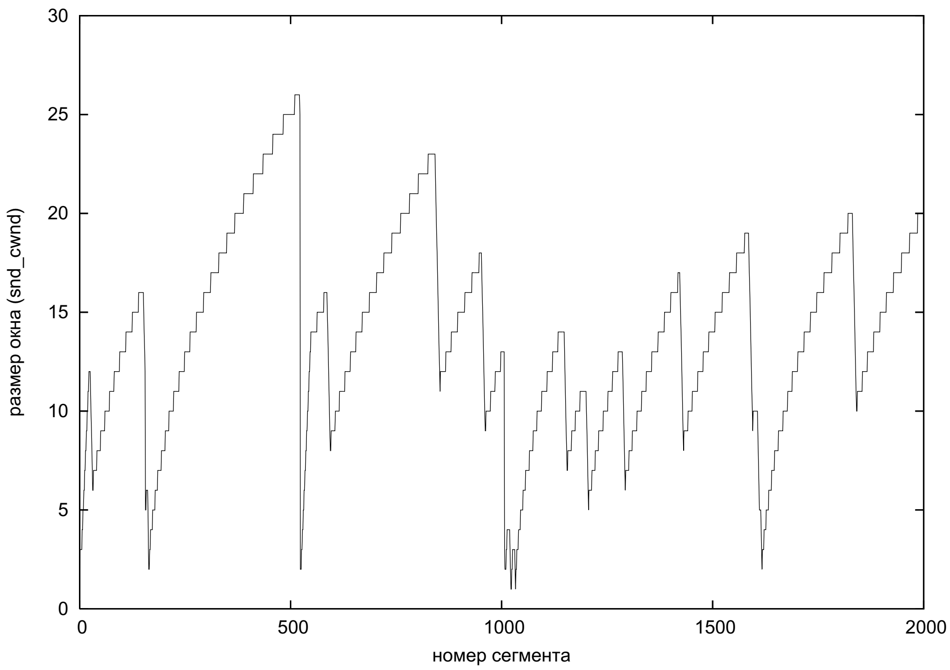

Рис. 3: График изменения размера окна TCP

единение путем отправки поддельных сегментов с установленным флагом RST.

Такие атаки можно распознавать по различным отличительным особенностям последовательности обмена сегментами. Например, активность программы tcpkill приводит к получению жертвой нескольких сегментов TCP с установленным флагом RST в ответ на один сегмент с запросом на установление соединения, тогда как в нормальных условиях этот ответ должен содержать только один сегмент с флагом RST.

5 Заключение

В статье представлен прототип системы GetTCP, позволяющий выполнять полный мониторинг поведения TCP-соединений на уровне ядра ОС Linux. При реализации прототипа использован ряд системных механизмов, обеспечивающих очень малое влияние процесса мониторинга на временны́е характеристики алгоритмов TCP ядра.

На рис. 3 показан график изменения размера окна TCP, полученный с помощью системы GetTCP.

Предлагаемый подход является перспективным для построения программных систем обнаружения несанкционированных информационных воздействий. Систему GetTCP можно рассматривать как базовое инструментальное системное средство для разработки интегрированных в операционную систему функций защиты транспортного уровня.

Список литературы

[1] Postel, J., Transmission Control Protocol, STD7, RFC793, September 1981.

[2] Allman, M., Paxon V., Stevens W., TCP Congestion Control, RFC2581.

[3] tcpdump, `http://www.tcpdump.org`

[4] tcptrace, `http://masaka.cs.ohiou.edu/software/tcptrace`

[5] High-Speed Data Relay Filesystem (relayfs),
`http://www.opersys.com/relayfs`

[6] Driving Me Nuts - Things You Never Should Do in the Kernel,
`http://www.linuxjournal.com/article/8110`

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

[8] Aleksandar Kuzmanovic, Edward W. Knightly,
Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants), ACM SIGCOMM 2003,
`http://www.sigcomm.org/sigcomm2003/papers/p75-kuzmanovic.pdf`

[9] `dsniff, http://www.monkey.org/~dugsong/dsniff`

[10] Steven Sipes, Intrusion Detection FAQ: Why your switched network isn't secure, September 10, 2000,
`http://www.sans.org/resources/idfaq/switched_network.php`