1983
Размышления о том, можно ли полагаться на доверие

Кен Томпсон

AT&T Bell Laboratories

(Кен Томпсон стал солауреатом премии Тьюринга 1983 г. за участие в разработке и реализации операционной системы UNIX. См. предисловие к лекции Д. М. Ритчи «Размышления об исследованиях в области программного обеспечения» на с. 195).

До какой степени можно полагаться на утверждение, что программа не содержит «троянских коней»? Возможно, более важно — полагаться на людей, написавших эту программу.

ВВЕДЕНИЕ

Я благодарен ACM за эту награду, но не могу не чувствовать, что я обязан ею не только техническим достоинствам системы UNIX, но и тому, в какой удачный момент времени она появилась. Система UNIX завоевала популярность в связи с широким поворотом промышленности от больших центральных ЭВМ к автономным мини-машинам. Я подозреваю, что вместо меня здесь должен был бы быть Даниэл Бобров [1], если бы ему пришлось иметь дело с PDP-11, а не с PDP-10. Более того, UNIX в ее современном виде — результат труда многих людей.

Есть старая поговорка: «Танцуй с тем, кто тебя привел», согласно которой я должен говорить о UNIX. Я не участвовал в основном русле работ по UNIX в течение многих лет, однако продолжаю получать незаслуженную честь за труды других. Поэтому я не собираюсь говорить о UNIX, но хотел бы поблагодарить всех, кто внес в нее свой вклад.

Это наводит меня на мысль о Дэнисе Ритчи. Наше сотрудничество было образцом совершенства. За десять лет, которые мы проработали вместе, я могу вспомнить только один случай нескоординированной работы. Я тогда обнаружил, что мы оба написали одинаковую ассемблерную программу из 20 строк. Я сравнил наши тексты и был поражен, обнаружив, что они совпадают посимвольно. Результат нашей совместной работы был намного больше, чем вклад нас обоих по отдельности.

Я программист. Именно так я пишу в анкетах в графе «род занятий». Как программист, я пишу программы. Я хотел бы представить вам кратчайшую программу, которую я когда-либо написал. Я буду делать это в три этапа, а в конце попытаюсь собрать их в единое целое.

ЭТАП I

В колледже, когда еще не было видеоигр, мы, бывало, развлекались упражнениями в программировании. Одним из любимых упражнений было написать кратчайшую самовоспроизводящуюся программу. Так как это упражнение далеко от реальности, обычным инструментом был Фортран. На самом деле Фортран был выбран по той же причине, по которой популярны скачки на стреноженных лошадях.

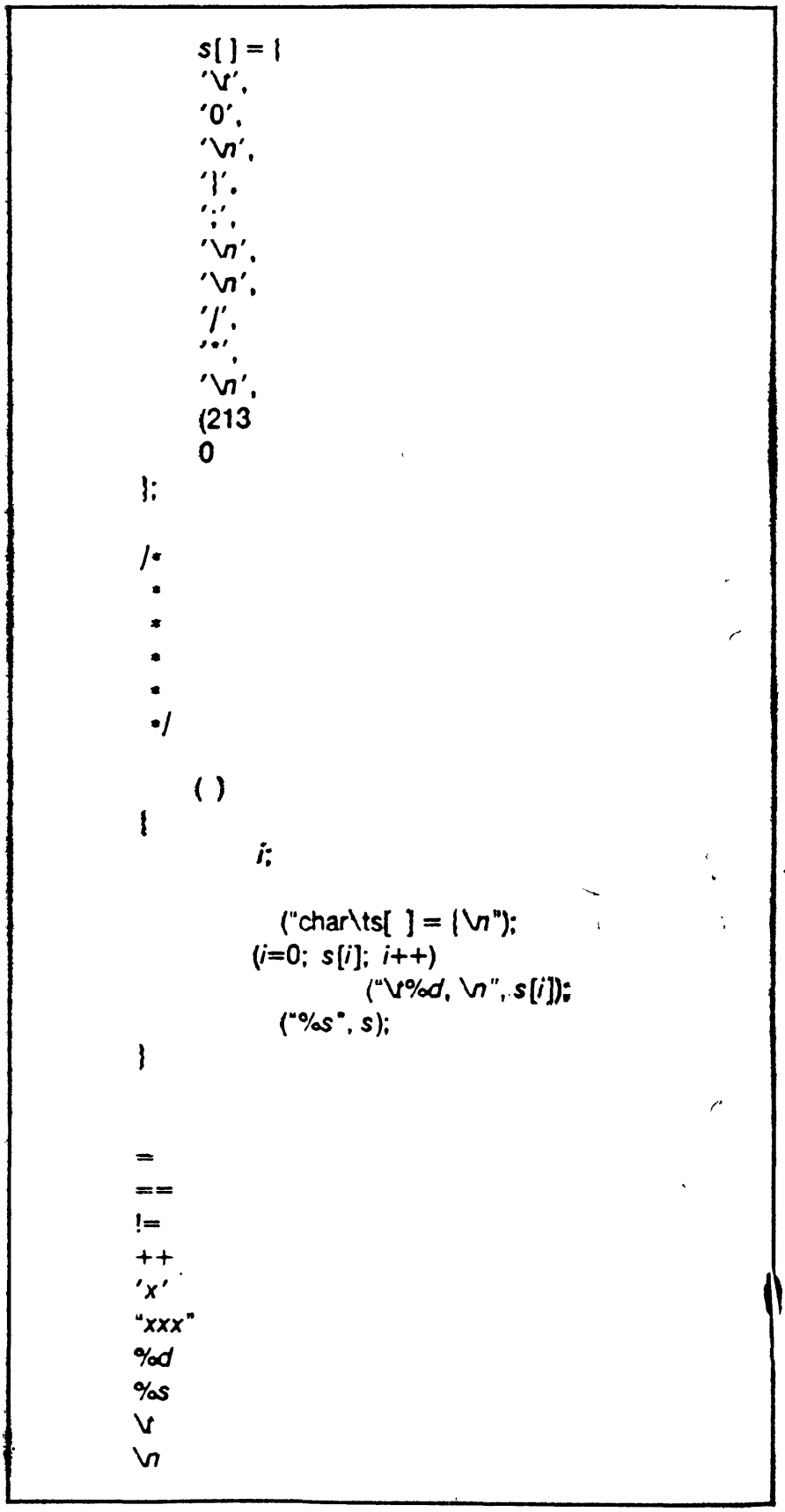
Более того, задача состоит в написании исходной программы, которая после компиляции и выполнения выдает в качестве вывода точную копию своего исходного текста. Если вам никогда не приходилось решать эту задачу, я настоятельно рекомендую попробовать сделать это самостоятельно. Удивительное открытие намного выше любой выгоды, которую вы можете извлечь, если вам скажут, как надо это делать. Требование насчет наименьшей длины было просто стимулом, чтобы проявить сноровку и определить победителя.

На рис. 1 показана самовоспроизводящаяся программа на языке программирования Си [3]. (Дотошный читатель заметит, что эта программа не является в точном смысле самовоспроизводящейся, однако выдаст в качестве результата самовоспроизводящуюся программу.) Этот текст слишком длинный, чтобы завоевать приз, но он демонстрирует прием решения и имеет два важных свойства, которые потребуются для завершения моего рассказа: 1) эта программа может быть легко порождена другой программой; 2) эта программа может содержать произвольное количество добавочного текста, который будет воспроизводиться наряду с основным алгоритмом. В приведенном примере воспроизводится даже комментарий.

ЭТАП II

Компилятор языка Си написан на Си. То, о чем я собираюсь рассказать, представляет собой одну из разновидностей проблемы курицы и яйца, которая возникает, когда компиляторы пишутся на своем собственном языке. В данном случае я буду использовать конкретный пример из компилятора Си.

Си позволяет задавать строковую константу в виде инициализированного массива символов (литер). Для обозначения



```
        s[ ] = {
        '\t',
        '0',
        '\n',
        '}',
        ';',
        '\n',
        '\n',
        '/',
        '*',
        '\n',
        (213
        0
};

/*
 *
 *
 *
 *
 */

    ( )
{
            i;

              ("char\ts[ ] = {\n");
            (i=0; s[i]; i++)
                      ("\t%d, \n", s[i]);
              ("%s", s);
}


=
==
!=
++
'x'
"xxx"
%d
%s
\t
\n
```

Рис. 1.

некоторых символов, не имеющих графического представления, в строке могут использоваться управляющие последовательности. Например,

«Здравствуй, мир \n»

является строкой символов, в которой «\n» представляет переход на новую строку.

На рис. 2.1 представлен фрагмент программы компилятора Си, в котором интерпретируются управляющие последовательности символов. Это удивительный фрагмент. Он «знает» полностью переносимым образом, какой код символа компилируется для новой строки в любой кодировке символов. Этот элемент знания позволяет ему перекомпилировать себя, увековечив таким образом знание.

Предположим, мы хотим изменить компилятор Си, чтобы включить управляющую последовательность «\v» для представления символа вертикальной табуляции. Расширение фрагмента 2.1 очевидно и представлено на рис. 2.2. Затем мы заново компилируем компилятор Си, но получаем сообщение об ошибке[1]. Ясно, что, поскольку компилятор Си в готовом виде ничего не знает о «\v», исходная программа не является правильной программой на языке Си. Заглянув в таблицу кодировки символов ASC II, мы находим, что код вертикальной табуляции — десятичное 11. Мы изменяем нашу исходную программу, как показано на рис. 2.3. Теперь старый компилятор компилирует новый исходный текст без ошибок. Мы устанав-

```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
...
```

Рис. 2.1.

```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
if(c == 'v')
        return('\v');
...
```

Рис. 2.2.

```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
if(c == 'v')
        return(11);
...
```

Рис. 2.3.

[1] На самом деле сообщения об ошибке не будет, но будет сгенерирован не тот код. Согласно правилам Си (см., например, [2]), «\v» будет интерпретироваться как символ «v», если эта управляющая последовательность неизвестна компилятору. — *Прим. перев.*

ливаем полученную готовую программу как новую официальную версию компилятора Си и теперь можем написать мобильную версию, как это сделано на рис. 2.2.

Это глубокая идея. В моем понимании она близка к понятию «обучающейся» программы. Вы просто сообщаете новое знание один раз, а затем можете использовать определение, ссылающееся само на себя.

ЭТАП III

Снова компилятор Си. На рис. 3.1 показан фрагмент компилятора Си, занимающий высокий уровень в его структуре. Подпрограмма «compile» вызывается для компиляций каждой следующей строки исходного текста. На рис. 3.2 показана модификация компилятора, которая преднамеренно неправильно компилирует исходную программу, когда в ней встречается заданный образец. Когда такое делается не преднамеренно, это называется ошибкой компилятора, или «жучком». Так как это сделано преднамеренно, то говорят, что мы имеем дело с «троянским конем».

Действительный «жучок», который я встроил бы в компилятор, распознавал бы исходный текст команды login (вход в систему UNIX). Взамен подставлялся бы такой исходный текст, чтобы команда login принимала как правильный заранее известный пароль (в зашифрованном[1]) или обычном виде). Таким образом, если бы такой компилятор был установлен в готовом (двоичном) виде и был использован для компиляции команды login, я мог бы войти в такую систему, как и любой другой пользователь.

```
compile(s)
char *s;
{
        ...
}
```

Рис. 3.1.

```
compile(s)
char *s;
{
        if(match(s, "pattern")) {
                compile("bug");
                return;
        }
        ...
}
```

Рис. 3.2.

[1] Все пароли в UNIX хранятся в зашифрованном виде в обычных текстовых файлах, в принципе доступных на чтение всем пользователям. — *Прим. перев.*

```
compile(s)
char *s;
{
        if (match(s, «образец 1»)) {
                compile («жучок 1»);
                return;
        }
        if (match(s, «образец 2»)) {
                compile («жучок 2»);
                return;
        }
        ...
}
```

Рис. 3.3.

Такая наглая вставка в программе не долго оставалась бы незамеченной. Даже при случайном внимательном просмотре исходного текста компилятора Си она вызвала бы подозрение.

Заключительный шаг представлен на рис. 3.3. Он заключается в простом добавлении еще одного «троянского коня» к уже существующему. Второй образец настроен на сам компилятор Си. Заменяющим текстом служит самовоспроизводящаяся программа (см. этап I), которая вставляет обоих «троянских коней» в компилятор. Потребуется также фаза обучения, как в примере из этапа II. Сначала мы компилируем модифицированный исходный текст нормальным компилятором Си и получаем готовую программу с «жучком». Затем устанавливаем эту готовую программу как официальный компилятор Си. Теперь можно удалить жучки из исходного текста компилятора, а новый компилятор будет воспроизводить жучки при каждой перекомпиляции. Команда login, естественно, будет оставаться с «жучком» без всякого следа в каких-либо исходных текстах.

МОРАЛЬ

Мораль ясна. Нельзя доверять программе, которую вы не написали полностью сами (особенно если эта программа пришла из компании, нанимающей таких людей, как я). Сколько бы вы не исследовали и не верифицировали исходный текст — это не защитит вас от троянской программы. Для демонстрации атаки такого рода я выбрал компилятор Си. Я мог бы выбрать любую программу, обрабатывающую другие программы — ассемблер, загрузчик или даже микропрограмму, зашитую в аппаратуру. Чем ниже уровень программы, тем труднее и труднее обнаруживать подобные «жучки». Мастерски встроенный «жучок» в микропрограмме будет почти невозможно обнаружить.

Теперь, после того как я попытался убедить вас, что мне нельзя доверять, я желаю прочесть мораль. Я хотел бы покритиковать прессу за ее отношение к «хэккерам»: банда 414, банда Далтона и т. п. Действия, совершаемые этими «детками», в лучшем случае — вандализм, а в худшем — нарушение прав владельцев и воровство. Только неадекватность уголовного кодекса спасает хэккеров от очень сурового наказания. Компании, которым угрожает такого рода деятельность (а большинство крупных компаний подвергается очень большому риску), усиленно настаивают на изменении уголовного кодекса. Несанкционированный доступ к компьютерным системам уже сейчас является серьезным преступлением в нескольких штатах, и в настоящий момент этот вопрос рассматривается законодательными органами во многих других штатах, а также в Конгрессе.

Назревает взрывоопасная ситуация. С одной стороны, пресса, кино и телевидение делают героев из хулиганов, называя их озорными детками. С другой стороны, действия, совершаемые этими детками, скоро будут наказываться годами лишения свободы.

Я видел этих деток, дававших показания в Конгрессе. Ясно, что они абсолютно не подозревают о серьезности своих действий. Очевидно, мы имеем здесь пробел в культуре. Проникновение в компьютерную систему в общественном мнении должно быть таким же неблаговидным поступком, как проникновение в дом соседа. Не должно иметь значения, что дверь у соседа открыта. Пресса должна учить, что использование компьютера не по назначению ни капли не смешнее, чем пьянство за рулем.

БЛАГОДАРНОСТИ

Я впервые прочитал о возможности такого троянского коня в документе военно-воздушных сил [4], в котором критикуются средства защиты от несанкционированного доступа ранней реализации системы Multics. Я не могу найти более точной ссылки на этот документ. Я буду благодарен тому, кто может найти эту ссылку и даст мне об этом знать.

ЛИТЕРАТУРА

1. Bobrow D. G., Burchfiel, J. D., Murphy, D. L., and Tomlinson, R. S. TENEX, a paged time-sharing system for the PDP-10. Commun. ACM 15, 3 (Mar. 1972), 135—143.
2. Kernighan B. W., and Ritchie, D. M. The C Programming Language. Prentice-Hall, Englewood Cliffs, N. J., 1978. [Имеется перевод: Керниган Б., Ритчи Д. Язык программирования Си. Москва. Финансы и статика, 1985.]
3. Ritchie D. M., and Thompson K. The UNIX time-sharing system. Commun. ACM 17, 7 (July 1974), 365—375.
4. Неизвестный документ Военно-воздушных сил.