Костюкова Нина Ивановна
**ВОЗНИКНОВЕНИЕ УГРОЗ**
Адрес статьи: www.gramota.net/materials/1/2011/5/14.html


Источник
**Альманах современной науки и образования**
Тамбов: Грамота, 2011. № 5 (48). С. 39-42. ISSN 1993-5552.
Адрес журнала: www.gramota.net/editions/1.html
Содержание данного номера журнала: www.gramota.net/materials/1/2011/5/

сы. Срок их жизни заметно короче прочих, поскольку взаимодействие их между собой ускоряет взрыв. Наверняка что-то подобное работает и с антивеществом.

Как версия: тот же электрон - стабилен в виде облака с зарядом. Именно как облако. Протон, получив отрицательный заряд и став антипротоном, стремится принять форму облака и взрывается. Хотя эта теория требует, конечно, экспериментальной проверки.

Еще пара теорий-следствий, которые не проверить экспериментально:

1. Вселенная (во всеобъемлющем для нас смысле) - результат взаимодействия всего 2х частиц (назовем их Богами). Момент начала взаимодействия Богов - Большой Взрыв.

При этом взаимодействие постоянно ослабевает - Вселенная расширяется, энергия более высоких состояний уходит все ниже и ниже по оси Состояний (энтропия). Подобная теория объясняет непрерывное расширение Вселенной без введения загадочной «темной энергии». Она и дальше будет только расширяться, поскольку взаимодействие со временем только ослабевает. При этом ничего не мешает Богам (а я считаю и их не последней стадией, а ось Состояний бесконечной прямой, а не лучом), то легко допустить бесконечное множество Вселенных на разных стадиях развития, каждая из которых *замкнута и конечна в пространстве и времени, для тех, кто пытается осознать ее изнутри*. То есть, мысленно двигаясь по оси Состояний, на определенном этапе мы увидим Вселенные и их взаимодействия между собой, их скопления и т.д.

2. Фон из эфирных частиц должен быть чрезвычайно сильным - ведь, на мой взгляд, именно освобождение «эфирных» частиц из атомов в термоядерной реакции с превышением их «стягивающего» фона (или поля) взрывает звезды.

Хотя, возможно, и это будет возможно проверить со временем.

3. Наконец осознав существование Оси Состояний - мы когда-нибудь сможем двигаться по ней. Наш разум уже это делает. Физическое перемещение между уровнями Оси Состояний кажется еще невероятным, но когда-то и летающие аппараты тяжелее воздуха казались также невероятными.

Осознав структуру окружающего мира, мы сможем лучше приспособить его к нашим потребностям.

---

УДК 519.6

*Нина Ивановна Костюкова*
*Институт вычислительной математики и математической геофизики СО РАН*

## ВОЗНИКНОВЕНИЕ УГРОЗ©

**Безопасность программ**

Условимся, что под термином «***программа***» мы в равной степени будем понимать обычные программы пользователей; специальные программы, рассчитанные на нарушение ***безопасности системы***; а также разнообразные системные утилиты и коммерческие прикладные программы, которые отличаются более высоким профессиональным уровнем разработки и, тем не менее, могут содержать отдельные недоработки, позволяющие захватчикам атаковать системы. Программы могут порождать проблемы двух типов: они, во-первых, могут перехватывать и модифицировать данные в результате действий пользователя, который к этим данным не имеет доступа; и во-вторых, используя упущения в защите компьютерных систем, программы могут или обеспечивать доступ к системе пользователям на это права, или блокировать доступ к системе законных пользователей. К сожалению, количество возможных слабых точек, которые могут содержаться в той или иной программе, значительно превышает число известных технологий устранения экспозиций. Это обусловлено двумя причинами:

- во-первых, качество программы всегда не превышает квалификации ее разработчика: очевидные экспозиции и нарушения могут быть выявлены и устранены достаточно легко, однако, чем выше уровень подготовки программиста, тем более тщательно он способен скрыть умышленные механизмы, разработанные для нарушения безопасности системы;
- во-вторых, имеет место трудно разрешимый компромисс между эффективностью и удобством компьютерной системы в работе и степенью обеспечения в ней требований безопасности.

Чем более высокие требования предъявляются к безопасности системы, тем большее количество ресурсов системы затрачивается на обеспечение этих требований, тем сильнее снижается производительность системы, и увеличиваются сроки решения задач, наконец, тем неудобнее работать в данной системе пользователям. С другой стороны, чем больше ресурсов выделяется для решения текущих задач, тем меньше возможностей по обеспечению должного уровня безопасности. В основном небезопасность программ состоит в том, что они могут быть использованы как средства получения критичной информации (данных), циркули-

рующих в системе, тем более, что данные в компьютерной системе обычно хранятся в виде, непонятном для большинства людей. Впрочем, целью атаки могут быть и сами программы. Причин тому несколько:

1. В современном мире программы могут быть товаром, приносящим немалую прибыль, особенно тому, кто первый начнет тиражировать программу в коммерческих целях и оформит авторские права на нее. Поэтому опасность похищения почти готовой программы конкурентом - не такая уж надуманная возможность.

2. Программы могут становиться также объектом атаки, имеющей целью модифицировать эти программы некоторым образом, что позволило бы в будущем провести атаку на другие объекты системы. Особенно часто объектом атак такого рода становятся программы, реализующие функции защиты системы.

Рассмотрим несколько типов программ и приемы, которые наиболее часто используются для атак программ и данных.

**Как залезть в программу**

*Как залезть в программу? Очень просто - через люк!*

Люком называется не описанная в документации на программный продукт возможность работы с этим программным продуктом. Сущность использования люков состоит в том, что при выполнении пользователем некоторых не описанных в документации действий он получает доступ к возможностям и данным, которые в обычных условиях для него закрыты (в частности - выход в привилегированный режим). Люки чаще всего являются результатом забывчивости разработчиков. В процессе разработки программы разработчики часто создают временные механизмы, облегчающие ведение отладки за счет прямого доступа к отлаживаемым частям продукта. Пусть например, для начала работы с продуктом требуется выполнить некоторую последовательность действий, предусмотренных алгоритмом, - ввести пароль, установить значения некоторых переменных и так далее. При нормальной работе продукта эти действия имеют определенный смысл. Но во время отладки, когда разработчику необходимо тестировать некоторые внутренние части программы и волей-неволей приходится выполнять ту же операцию входа добрый десяток - а то и более - раз на дню, безобидные в общем-то правила, затрудняющие тем не менее доступ к отлаживаемым частям, начинают не на шутку раздражать. Что же делает программист? Правильно: в течение получаса он программирует некоторый дополнительный механизм, не предусмотренный изначальным алгоритмом программы, но позволяющий не выполнять надоедливых действий или выполнять их автоматически - например, при нажатии определенной клавиши (комбинации клавиш) или при вводе определенной последовательности символов. Все - люк готов! По окончании отладки большинство люков убирается из программы; но люди есть люди - зачастую они забывают о существовании каких-то мелких «лючков». Иногда, правда, разработчики сознательно оставляют люки в своих продуктах, особенно в ранних версиях, когда весьма вероятна возможность доработки продукта. Одним из наиболее показательных примеров использования «забытых» люков является, пожалуй, широко известный в компьютерном мире инцидент с вирусом Морриса. Одной из причин, обусловивших возможность распространения этого вируса, была ошибка разработчика программы электронной почты, входящей в состав одной из версий операционной системы UNIX, приведшая к появлению малозаметного лючка. Для вас, наверное, будет небесполезно знать, что американские специалисты оценивают ущерб, нанесенный в результате этого инцидента, более чем в сто миллионов долларов. Люки могут образовываться также в результате часто практикуемой технологии разработки программных продуктов «***сверху вниз***». При этом программист приступает к написанию сразу управляющей программы, заменяя предполагаемые в будущем программы так называемыми «***заглушками***» - группами команд, имитирующими или просто обозначающими место подсоединения будущих подпрограмм. В процессе разработки по мере готовности реальных подпрограмм эти подпрограммы заменяют соответствующие заглушки. В теории моментом завершения разработки конечной программы по такой технологии можно считать момент замены последней заглушки реальной подпрограммой (при условии, что все подпрограммы отлажены и синхронизированы). В действительности дело обстоит несколько сложнее. Вся беда в том, что авторы часто оставляют заглушки в конечном программном продукте, передаваемом в эксплуатацию. Делают это порой неумышленно: например, на ранних стадиях разработки предполагалось наличие в конечном продукте некоторой подпрограммы, однако в процессе разработки выяснилось, что это подпрограмма в силу каких-либо причин не нужна. Но заглушка-то на предполагавшемся месте подключения подпрограммы осталась! Более того, удалить заглушку, не заменяя ее подпрограммой, бывает весьма сложно. Все это может спровоцировать программиста на то, чтобы оставить такую заглушку на месте «до лучших времен». Возможен вариант, когда заглушки оставляются в конечной программе сознательно, в расчете на подключение в дальнейшем к работающей программе новых подпрограмм, реализующих некоторые новые возможности, либо предполагая возможное подключение к программе тестирующих средств для более точной настройки программы. Еще одним распространенным источником люков является так называемый «неопределенный ввод». Это случай, когда в программе нет синтаксического контроля вводимых данных. В этом случае существует потенциальная возможность нарушения безопасности системы. Неопределенный ввод - частная реакция прерывания. То есть в общем случае захватчик может умышленно пойти на создание в системе некоторой нестандартной ситуации, позволившей бы ему выполнить необходимые действия. Например, он может искусственно вызвать аварийное завершение программы, работающей в привилегированном режиме, с тем, чтобы перехватить управление, оставшись в этом привилегированном режиме.

Борьба с возможностью прерывания (в терминологии безопасности) в конечном итоге выливается в предусмотрение при разработке программ комплекса механизмов, образующих так называемую «защиту от дурака». Смысл этой защиты состоит в том, чтобы гарантированно отсекать всякую вероятность обработки неопределенного ввода и разного рода нестандартных ситуаций (в частности, ошибок). Короче, качественная программа должна работать корректно и не приводить к появлению возможности нарушения безопасности компьютерной системы, даже если с самой программой работают некорректно - например, если за клавиатурой терминала случайно оказалась обезьяна. Таким образом, люк (или люки) может присутствовать в программе ввиду того, что программист:

1. забыл удалить его;
2. умышленно оставил его в программе для обеспечения тестирования или выполнения оставшейся части отладки;
3. умышленно оставил его в программе в интересах облегчения окончательной сборки конечного программного продукта;
4. умышленно оставил его в программе с тем, чтобы иметь скрытое средство доступа к программе уже после того, как она вошла в состав конечного продукта.

В первом случае люк - неумышленная, но серьезная брешь в безопасности системы. Во втором и третьем случаях люк - серьезная экспозиция безопасности системы. Наконец, в последнем случае люк - первый шаг к атаке системы. В любом случае люки - это возможность получить управление вашей системой в обход защиты.

**Программы без документации**

Существуют программы, реализующие, помимо функций, описанных в документации, и некоторые другие функции, в документации не описанные. Такие программы называются ***троянскими конями***. Например, вы запускаете какую-то нужную вам программу - графический пакет, расчетную задачу или игру, наконец, - а через некоторое время обнаруживаете, что в вашей директории исчезли все файлы. Поскольку кроме запуска известной вам программы вы ничего другого не делали, вполне вероятно, что вы заподозрите что-то неладное и решите запустить ту же программу в другой (лучше специально для этого созданной) директории. Если эффект повторится - вы имеете дело с «троянским конем» в компьютерном исполнении. Далеко не всегда обнаружить троянского коня бывает так просто - вероятность обнаружения тем больше, чем очевиднее результаты действия троянского коня. А если задачей троянского коня является не удаление ваших файлов, а изменение их защиты? Явно вы этого не увидите, но каково будет ваше удивление, когда через некоторое время выяснится, что конфиденциальные данные, содержавшиеся в одном из ваших файлов (например, проект важного документа), стали известны всем. Проверяйте наличие файла - на месте, проверяйте защиту файла - а оказывается, его может читать любой! Однако такие фиксированные результаты представляют очень опасную для троянского коня угрозу разоблачения его деятельности и, соответственно, обнаружения. Более сложный троянский конь может быть запрограммирован таким образом, чтобы по изменении защиты файлов (в нашем примере) подавать захватчику (лицу, заинтересованному в срабатывании троянского коня) некоторый условный сигнал о доступности файлов. После выдачи сигнала троянский конь некоторое время выжидает, а затем возвращает защиту файлов в исходное состояние. Такой алгоритм позволяет захватчику в течение некоторого времени делать с вашими файлами все, что угодно. И никаких следов атаки после этого не остается. Вы же остаетесь в полном счастливом неведении, то есть с носом.

**Скрытые каналы**

Рассмотрим программы, передающие информацию лицам, которые в обычных условиях эту информацию получать не должны. Называются такие необычные способы извлечения информации ***скрытыми каналами***.

В таких системах, где ведется обработка критической информации, программист не должен иметь доступа к обрабатываемым программой данным после начала эксплуатации этой программы. Например, банковский программист не должен иметь доступа к именам или счетам вкладчиков и клиентов, как не должен знать порядка обслуживания клиентов. В процессе отладки программы разработчику допускается предоставление ограниченного объема реальных данных для проверки работоспособности программы, но предоставление реальных данных после сдачи программы в эксплуатацию не имеет оправдания. Из факта обладания некоторой служебной информацией можно извлечь достаточно немалую выгоду, хотя бы элементарно продав эту информацию (например, список клиентов) конкурирующей фирме. Поэтому нетрудно предположить. Что кто-либо из сотрудников будет заинтересован в том, чтобы иметь возможность такую информацию получить. Достаточно квалифицированный программист всегда может найти способ скрытой передачи информации; при этом программа, предназначенная для создания самых безобидных отчетов может быть немного сложнее, чем того требует задача. Для скрытой передачи информации можно с успехом использовать различные элементы формата «безобидных» отчетов, например, разную длину строк, пропуски между строками, наличие или отсутствие служебных заголовков, управляемый вывод незначащих цифр в выводимых величинах, количество пробелов или других символов в определенных местах отчета и так далее. Например, определенную информацию может нести появление в отчете вместо служебного заголовка «TOTAL» другого заголовка - «TOTALS». Разница всего в один символ, но именно его наличие или отсутствие и может быть сигналом осведомленному о канале захватчику (например, о том в системе появился файл с определенным именем). В данном случае имеет место 1-битовый скрытый канал, так как за один раз передается один бит информации (наличие или отсутствие символа «S»). Однако возможна ситуация, когда

захватчик не будет иметь доступа ко всем отчетам, создаваемым «хитрой» программой. И таким образом он будет лишен возможности пользоваться своим скрытым каналом непосредственно. В таком случае программист может обеспечить, например, вызов в определенных ситуациях более безопасной программой, непосредственно обрабатывающей интересующие захватчика данные, менее безопасной программы, которой и будет переданы критичные данные. А уж вызванная программа может создавать замаскированный, внешне «безобидный» отчет, доступный для просмотра захватчику. Еще более сложным может быть случай, когда захватчик не имеет возможности просматривать замаскированные отчеты или обеспечивать вызов подпрограмм-шпионов, но имеет возможность доступа к компьютеру во время работы интересующей его программ. В такой ситуации возможно создание специального массива данных непосредственно в оперативной памяти компьютера, куда программа, содержащая скрытый канал, будет пересылать критичную информацию. Причем, опять же, информация может быть представлена в виде служебных символов, старт-стопных сигналов и так далее. Как видно из приведенных примеров, для получения относительно небольших объемов информации захватчик вынужден проделать достаточно большую работу. Поэтому скрытые каналы наиболее применимы в ситуациях, когда захватчика интересует даже не содержание информации, а, допустим, факт ее наличия (например, наличие в банке расчетного счета с определенным номером). Можно предположить и более изощренную ситуацию: с помощью скрытого канала захватчик получит сигнал о появлении в системе файла с определенным именем, что в свою очередь служит признаком работы в системе некоторого процесса, позволяющего провести атаку иного типа.

**Жадность - порок**

Большинство методов нарушения безопасности направлены на то, чтобы получить доступ к данным, не допускаемый системой в нормальных условиях. Однако не менее интересным для захватчиков является доступ к управлению самой компьютерной системой или изменение ее качественных характеристик. Это может потребоваться для того, чтобы явно использовать компьютерную систему в своих целях (хотя бы для бесплатного решения своих задач) либо просто заблокировать систему, сделав ее недоступной другим пользователям. Такой вид нарушения безопасности системы называется «***отказом в обслуживании***» или «***отказом от пользы***». Некоторые компьютеры, особенно в исследовательских центрах, имеют так называемые «фоновые» задачи, постоянно решаемые, но с очень маленьким приоритетом. Обычно это задачи, требующие для решения очень большого машинного времени, но не особенно срочные (типа вычисления чисел $e$, с очень большой степенью точности). Низкий приоритет делает возможным решения этих задач только при условии, что в системе нет задач с более высоким приоритетом. Однако по недосмотру либо в результате ошибки, либо умышленно приоритет такой «фоновой» задачи может быть существенно повышен: в этом случае бывшая «тихоходная» задача захватывает процессор в монопольное использование, блокируя выполнение всех других вычислений. Такого рода программы, захватывающие некоторый ресурс в монопольное использование, называются ***жадными программами***. Ведь чтобы остановить выполнение какого-либо задания, вовсе не обязательно захватывать именно процессор. Например, если известно, что на некотором этапе задача, которую нужно заблокировать, должна выполнять вывод на печать некоторого отчета, можно заблокировать доступ к принтеру. В таком случае оператор системы будет видеть, что обработка заданий проходит «нормально» - процессор-то не заблокирован и другие задачи нормально решаются - и так до тех пор, пока не прибежит разгневанный хозяин заблокированной задачи в сопровождении администратора системы. Изобразить подходящую «долгоиграющую» задачу для того, чтобы намертво заблокировать систему, несложно - достаточно запрограммировать в обычной программе явный или неявный бесконечный цикл. Насколько опасной может быть блокировка компьютерной системы? Прежде всего, спросите военных, насколько опасной может быть блокировка компьютера, управляющего различного рода системами раннего обнаружения или той же противовоздушной обороны. Можно поинтересоваться у врачей, насколько серьезными могут быть последствия блокировки компьютеров, осуществляющих мониторинг больных в реанимационном отделении. Уже из этих примеров становится очевидным, что «отказ от обслуживания» чрезвычайно опасен для так называемых систем реального времени - систем, управляющих некоторыми технологическими процессами, осуществляющих различного рода синхронизации и так далее.