КАК ЗАЩИЩАЮТ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

...Важно не только научиться. Важно не научиться.
Не научиться бывает подчас трудней.
Нарушить стандарт, отвергнуть шаблон.
Не принять общепринятого.

Л. Озеров
«Выбор и предпочтение»

Потребность в защитных механизмах растет с каждым днем, но качество их реализации падает вниз стремительным домкратом. Давным-давно сломан HASP, FLEX LM, ASProtect и другие широко разрекламированные решения, лишний раз подтверждая, что вы не можете доверять никакому коду, кроме своего собственного, но что делать, если вы и в себе не уверены? Эта статья перечисляет основные ошибки, совершаемые разработчиками, и дает советы по усилению защиты.

КРИС КАСПЕРСКИ

В борьбе за выживание защитные механизмы обречены. Защита лишь продлевает мучения программы, отодвигая ее взлом на некоторый срок. Теоретически. Практически же ничего не стоит создать защиту, позволяющую программе дожить до преклонных лет и «умереть» собственной смертью, передавая наследие следующей версии. Главное – правильно забаррикадироваться. Бессмысленно вешать стальную дверь на картонный дом. Линия обороны должна быть однородна во всех направлениях, поскольку стойкость защиты определяется наиболее уязвимой ее частью.

Мы не будем касаться вопросов целесообразности защиты как таковой (это отдельный больной вопрос), а сразу перейдем к рекомендациям, следование которым усилит защиту настолько, насколько это возможно. Необходимо не только добиться экономической нецелесообразности взлома, но и убить моральный стимул к хакерству «на интерес» – для этого защита должна быть максимально «тупой», а ее взлом – рутинным.

Выбирайте только надежные, системно-независимые методики. Хитроумные антиотладочные приемы только разжигают хакерский интерес, и что еще хуже – выдают ложные, срабатывая у легальных пользователей, а этого допускать ни в коем случае нельзя. Делайте ставку на частый выход новых версий и техническую поддержку – это оттолкнет пользователей от «кракнутых» версий.

Джинн из бутылки, или недостатки решений из коробки

Зачем изобретать велосипед, когда на рынке представлено множество готовых решений – как программных (ASProtect, FLEX LM, Extreme Protector), так и аппаратных (HASP, Sentinel, Hardlock)? Ответ – стойкость защиты обратно пропорциональна ее распространенности, особенно если она допускает универсальный взлом (а все вышеперечисленные решения его допускают). Даже начинающий хакер, скачавший руководство по борьбе с HASP, за короткое время «отвяжет» программу, особенно если защита сводится к тривиальному:

```
if (IsHaspPresent() != OK) exit();
```

Защитный механизм должен быть глубоко интегрирован в программу, тесно переплетен с ней. Все, что быстро защищается, быстро и ломается!

Хуже всего то, что многие протекторы содержат грубые ошибки реализации, нарушающие работоспособность защищаемой программы или даже всей операционной системы вызывающим поведением (например, Armadillo замусоривает реестр, замедляя работу системы, и это никак не отражено в документации!) Защищая программу самостоятельно, вы тратите силы и время, но зато получаете предсказуемый результат. «Фирменный» защитный механизм – кот в мешке, и неизвестно, какие проблемы вас ждут.

От чего защищаться

Чаще всего программы защищают от несанкционированного копирования, ограничения времени испытательного использования, реконструкции оригинального алгоритма и модификации файла на диске и памяти.

Защита от копирования, распространения серийного номера

От несанкционированного копирования в принципе защищает привязка к машине (как это сделать с прикладного уровня, см. в статье [1]). «В принципе» потому, что пользователям очень не нравится, когда ограничивают их свободу, поэтому привязывайтесь только к носителю (лазерному диску). Нет никакой необходимости выполнять проверку при каждом запуске программы, требуя наличия диска в приводе (а сможет ли ваша программа «увидеть» его по сети?), достаточно проверки на этапе инсталляции. Не надо бояться, что это ослабит защиту – в любом случае при наличии ключевого диска, программа отвязывается элементарно. Цель «привязки» – противостоять не хакерам, а пользователям. О том, как создать диск, не копируемый копировщиками защищенных дисков (Alcohol, CloneCD), можно прочитать в [2] или [3] .

Пусть защита периодически «стучится» в Интернет, передавая вам серийный номер. Если один и тот же серийный номер будет поступать со многих IP-адресов, можно передать удаленную команду на дезактивацию «флага» ре-

гистрации (удалять файл с диска категорически недопустимо!). Только не возлагайте на этот механизм больших надежд – пользователь может поставить брандмауэр (правда, локальные брандмауэры легко обойти, см. [4]) или же вовсе работать на машине, изолированной от сети.

Защита серийным номером (далее по тексту – s/n) не препятствует несанкционированному копированию, но если все серийные номера различны, можно вычислить пользователя, выложившего свой s/n в сеть. То же самое относится и к парам имя пользователя/код активации (далее по тексту – u/r).

Идея: пусть сервер генерирует дистрибутивный файл индивидуально на основе регистрационных данных, переданных пользователям. Тогда его u/r не подойдет к чужим файлам и весь дистрибутив придется выкладывать в сеть целиком, что намного проблематичней, и к тому же далеко не каждый пользователь рискнет скачивать двоичный файл из ненадежных источников.

Защита испытательным сроком

При защите «испытательным сроком» никогда не полагайтесь на системную дату – ее очень легко перевести «назад». Сравнивайте текущую дату с датой последнего запуска программы, сохраненной в надежном месте, помня о том, что не всякое расхождение свидетельствует о попытке взлома – может, это пользователь поправляет неверно идущие часы. Используйте несколько независимых источников – отталкивайтесь от даты открываемых файлов (или в общем случае – файлов, обнаруженных на компьютере пользователя), подключайтесь к службе атомного времени (если доступен Интернет) и т. д.

Для предотвращения деинсталляции/повторной инсталляции никогда не оставляйте никаких «скрытых» пометок в реестре или файловой системе. Во-первых, «уходя, заметай следы», а во-вторых, пользователь может запускать вашу программу из-под эмулятора ПК (Microsoft Virtual PC, VM Ware), – современные аппаратные мощности это уже позволяют. Он просто переформатирует виртуальный диск, уничтожая все следы пребывания вашей программы. Сохраняйте количество запусков в обрабатываемых программой документах (естественно, в нетривиальном формате, который непросто подправить вручную). Противостоять этому очень трудно!

Вообще лучшая защита демонстрационных версий – физическое усечение функциональности с удалением программного кода, отвечающего, например, за печать или сохранение файла.

Защита от реконструкции алгоритма

Чтобы хакер не смог реконструировать алгоритм защитного механизма и слегка «доработать» его напильником (или же попросту «подсмотреть» правильный пароль), обязательно используйте многослойную динамическую шифровку по s/n или u/r. Тогда без s/n, u/r взлом программы станет невозможным. Не проверяйте CRC s/n! Чтобы убедиться в правильности ввода s/n, проверяйте CRC расшифрованного кода (восстановить исходный s/n по его CRC на достаточных аппаратных мощностях вполне возможно, но CRC расшифрованного кода криптоаналитику вообще ни о чем не говорит!). Игнорируйте первые четыре символа s/n, посадив на них подложный расшифровщик – обычно хакеры ставят точку останова на начало s/n, но не на его середину. Еще лучше, если каждый из нескольких расшифровщиков будет использовать «свою» часть s/n. Спрашивайте имя, компанию, прочие регистрационные данные, делайте с ними сложные манипуляции, но никак их не используйте – пусть хакер сам разбирается, какие из них актуальные, а какие нет.

Шифровка называется динамической, если ни в какой момент весь код программы не расшифровывается целиком, в противном случае хакер может снять с него дамп, и привет! Расшифровывайте функцию перед вызовом, а по выходу из нее – зашифровывайте вновь. Используйте несколько независимых шифровщиков и перекрытия шифроблоков, иначе хакер расшифрует программу «руками» самого расшифровщика, просто передавая ему номера/указатели зашифрованных блоков, и сбросит дамп. Многословная шифровка: делаем на каждом «слое» что-то полезное, затем расшифровываем следующий слой и т. д. Программный код как бы размазывается между слоями, вынуждая хакера анализировать каждый из них. Если же весь программный код сосредоточен в одном-единственном слое, количество слоев шифровки, «оборачивающих» его, не имеет никакого значения и ничуть не усложняет взлом.

Разбавляйте защитный код (и в особенности код расшифровщика) большим количеством мусорных инструкций – процедуру размером в 1 Мбайт за разумное время дизассемблировать практически нереально. Мусор легко генерировать и автоматически (ищите в вирусных журналах полиморфные движки) – следите только за тем, чтобы визуально он был неотличим от полезного кода. Еще лучший результат дают виртуальные машины, выполняющие элементарные логические операции (Сети Петри, Стрелка Пирса, Машина Тьюринга). Даже если хакер напишет декомпилятор, на реконструкцию алгоритма уйдет вся оставшаяся жизнь (подробнее см. [5]).

При использовании однослойной шифровки размазывайте расшифровщик по телу программы, никогда не располагайте весь код расшифровщика в конце модуля – тогда переход на оригинальную точку входа может быть распознан по скачкообразному изменению регистра EIP. Кстати, стартовый код, внедряемый компиляторами в программу, в большинстве случаев начинается с обращения к FS:[0] (регистрация собственного обработчика исключений) – почаще обращайтесь к этой ячейке из расшифровщика, не позволяя хакеру быстро определить момент завершения расшифровки (вызовы должны следовать из разных мест, иначе хакер просто наложит фильтр, благо современные отладчики это позволяют).

Обязательно привязывайтесь к начальному значению глобальных инициализированных переменных, т. е. поступайте так:

```
FILE *f = 0; main(){if (!f) f = fopen(,,)…}
```

тогда дамп, снятый вне оригинальной точки входа, окажется неработоспособным. Дизассемблируете стандартный «Блокнот» – он именно так и устроен.

Защита от модификации на диске и в памяти

Модификацию кода предотвращает проверка целостности, причем следует проверять как целостность самого файла, так и целостность дампа в памяти (хакер может модифицировать программу на лету, или даже ничего не модифицировать, а на время перехватить управление и проэмулировать несколько следующих команд, или же просто изменить значение регистра EAX после выполнения команды типа TEST EAX, EAX или подобной ей).

Перехвату управления/эмуляции противостоять практически невозможно, а вот предотвратить модификацию легко – используйте несистематические коды Рида-Соломона (см. [6]). Чтобы взломать программу, хакеру потребуется не только разобраться, какие именно байты следует подправить для взлома программы, но и рассчитать новые коды Рида-Соломона, а для этого ему придется написать собственный кодировщик, что не так-то просто (несистематическое кодирование изменяет все кодируемые байты, в отличие от систематического кодирования, где к программе просто дописывается «контрольная сумма», проверка которой может быть легко «отломана»). Опять-таки это актуально только для многослойной динамической шифровки, в противном случае хакер просто дождется завершения декодирования кодов Рида-Соломона и снимет беззащитный дамп.

Как вариант используйте несимметричные криптоалгоритмы. Это предотвратит модификацию файла на диске (но не в памяти!), и, что еще хуже, зашифровка функции по выходу из нее оказывается невозможной (мы ведь не хотим сообщать хакеру секретный ключ?), а значит, код программы рискует в какой-то момент оказаться расшифрованным целиком. Чтобы этого не произошло, расшифровывайте код функции во временный буфер (общий для всех функций), не трогая оригинал.

Проверяя целостность кода, не забывайте о перемещаемых элементах. Если программа/динамическая библиотека будет загружена по адресу, отличному от указанного в PE-заголовке, системный загрузчик автоматически скорректирует все ссылки на абсолютные адреса. Либо избавьтесь от перемещаемых элементов (ключ /FIXED линкера MS Link), либо, что лучше, проверяйте только ячейки, не упомянутые в relocation table.

Никогда не блокируйте некоторые пункты меню/кнопки для ограничения функциональности демонстрационной версии – их не разблокирует только ленивый! Лучше физически вырезайте соответствующий код или на худой конец время от времени проверяйте состояние заблокированных элементов управления, т.к. они могут быть разблокированы не только в ресурсах, но и динамически – отправкой сообщения окну.

От кого защищаться

Арсенал современных хакеров состоит преимущественно из: дизассемблера, отладчика, дампера памяти и монитора обращений к файлам/реестру. Это очень мощные средства взлома, но с ними все-таки можно справиться.

Антидизассемблер

Дизассемблер – в девяти из десяти случаев это IDA Pro. Существует множество приемов, приводящих ее в замешательство («ее» – потому, что Ида женское имя): множественные префиксы, искажение заголовка PE-файла и т. д., однако смысла в них немного, и многослойной шифровки на пару с мусорным кодом для ослепления дизассемблера вполне достаточно (правда, опытные хакеры могут написать плагин, автоматизирующий расшифровку и вычищающий мусорный код, но таких единицы, и вы можете гордиться, что ломались у них). Кстати говоря, активное использование виртуальных функций в C++ существенно затрудняет дизассемблирование программы, поскольку для определения эффективных адресов приходится выполнять громоздкие вычисления или «подсматривать» их в отладчике (но про борьбу с отладчиками мы еще поговорим). Только помните, что оптимизирующие компиляторы при первой возможности превратят виртуальные функции в статические.

Антиотладка

Еще ни одному из отладчиков не удалось полностью скрыть свое присутствие от отлаживаемой программы, и потому он может быть обнаружен. Чаще всего используется сканирование реестра и файловой системы на предмет наличия популярных отладчиков, проверка флага трассировки, чтение содержимого отладочных регистров/IDT, замер времени выполнения между соседними командами и т. д. (если хотите узнать больше, наберите «anti-debug» в Google). Однако запрещать пользователю иметь отладчик категорически недопустимо – защита должна реагировать лишь на активную отладку. К тому же все эти проверки элементарно обнаруживаются и удаляются. Надежнее трассировать самого себя, подцепив на трассировщик процедуру расшифровки или генерировать большое количество исключительных ситуаций, повесив на SEH-обработчики процедуры, делающие что-то полезное. Попутно: оставьте в покое soft-ice – в хакерском арсенале есть и альтернативные отладчики.

Эмулирующим отладчикам противостоять труднее, но ничего невозможного нет. Используйте MMX-команды, сравнивая время их выполнения со временем выполнения «нормальных» команд. На живом процессоре MMX-команды работают быстрее. Отладчики же либо вообще не эмулируют MMX-команд, либо обрабатывают их медленнее нормальных команд.

Антимонитор

Мониторы – очень мощное хакерское средство, показывающее, к каким файлам и ключам реестра обращалась защищенная программа. Активному мониторингу в принципе можно и противостоять, но лучше этого не делать, ведь существует и пассивный мониторинг (снятие слепка с реестра/файловой системы и сравнение его до и после запуска программы), а против него не пойдешь.

Не храните регистрационную информацию в явном виде. Забудьте о флагах регистрации! Вместо этого размазывайте ключевые данные маленькими кусочками по всему файлу, считывайте их в случайное время из случайных мест программы, расшифровывая очередной кусок кода.

Вместо fopen/fseek/fread используйте файлы, проецируемые в память, они намного сложнее поддаются мониторингу, но это уже тема отдельного разговора.

Антидамп

Дамперам противостоять проще простого. Их много разных, но нет ни одного по-настоящему хорошего. Затирайте PE-заголовок в памяти (но тогда не будут работать функции типа LoadResource) или по крайней мере его часть (о том, какие поля можно затирать, читайте в [7]). Выключайте временно неиспользуемые страницы функцией Virtual Protect(.,PAGE_NOACCES,), а перед использованием включайте их вновь. Впрочем, хакер может уронить NT в «синий экран», получив образ подопытного процесса в свое распоряжение. Однако при использовании динамической многослойной шифровки толку от этого образа будет немного.

Как защищаться

Никогда не давайте хакеру явно понять, что программа взломана! Тогда ему остается найти код, выводящий ругательное сообщение (а сделать это очень легко) и посмотреть, кто его вызвал – вот сердце защитного механизма и локализовано. Используйте несколько уровней защиты. Первый – защита от ввода неправильного s/n и непредумышленного нарушения целостности программы (вирусы, дисковые сбои и т. д.). Второй – защита от хакеров. Обнаружив факт взлома, первый уровень «ругается» явно, и хакер быстро его нейтрализует, после чего в игру вступает второй, время от времени вызывающий зависания программы, делающий из чисел «винегрет», подменяющий слова при выводе документа на принтер и т. д. При грамотной реализации защиты нейтрализация второго уровня потребует полного анализа всей программы. Да за это время можно десять таких программ написать! Второй уровень никогда не срабатывает у честных пользователей, а только у тех, кто купит «крак». Если же вы боитесь, что второй уровень случайно сработает в результате ошибки, лучше вообще не беритесь за программирование, это не для вас.

Не показывайте хакеру, каким путем регистрируется защита. Это может быть и ключевой файл, и определенная комбинация клавиш, и параметр командной строки. Ни в коем случае не считываете s/n или u/r через WM_GETTEXT/GetWindowText, вместо этого обрабатывайте нажатия одиночных клавиш (WM_CHAR, WM_KEYUP/WM_KEYDOWN), прямо из основного потока ввода данных, и тут же их шифруйте. Смысл шифровки в том, чтобы вводимая пользователем строка нигде не присутствовала в памяти в явном виде (тогда хакер просто поставит на нее точку останова, и могучий soft-ice перенесет его прямо в самый центр защитного механизма). Интеграция с основным потоком ввода предотвращает быстрый взлом программы. Точка останова на WM_XXX ничего не дает, поскольку не позволяет быстро отличить обычные вводимые данные от s/n.

Возьмите на вооружение генератор случайных чисел – пусть проверки идут с разной периодичностью из различных частей программы (использовать общие функции при этом недопустимо – перекрестные ссылки и регулярный поиск выдадут вас с головой!). Не используйте функцию rand() – вместо этого отталкивайтесь от вводимых данных, преобразуя в псевдослучайную последовательность задержки между нажатиями клавиш, коды вводимых символов, последовательность открытий меню и т. д.

Ни в коем случае не храните «ругательные» строки открытым текстом и не вызывайте их по указателю – хакер мгновенно найдет защитный код по перекрестным ссылкам. Лучше так: берем указатель на строку, увеличиваем его на N байт. Сохраняем указатель в программе, а перед использованием вычитаем N на лету (при этом вам придется сражаться с коварством оптимизирующих компиляторов, норовящих вычесть N еще на стадии компиляции).

Избегайте прямого вызова API-функций. Наверняка хакер поставит на них точку останова. Используйте более прогрессивные методики – копирование API-функций в свое тело, вызов не с первой машинной команды, распознание и дезактивация точек останова (подробности в «Записках мыщъх'а». Солон-Р, 2004 г.).

Разбросайте защитный механизм по нескольким потокам. Отладчики не выполняют переключение контекста, и остальные потоки просто не получат управление. Кроме того, очень трудно разобраться в защитном механизме, исполняющимся сразу в нескольких местах.

Создайте несколько подложных функций, дав им осмысленные имена типа CheckRegisters, – пусть хакер тратит время на их изучение!

Заключение

Можно уметь ломать программы, не умея их защищать, а вот обратное утверждение неверно. Это не вопрос морали, это вопрос профессиональных навыков (закон разрешает «взлом», если он не влечет за собой прочих нарушений авторских и патентных прав). Теория – это хорошо, но в машинных кодах все не так, как на бумаге, и большое количество взломов свидетельствует отнюдь не о всемогуществе хакеров, а о катастрофической ущербности защитных механизмов, многие из которых ломаются за считанные минуты!

Личное наблюдение – за последние несколько лет качество защит ничуть не возросло. Причем если раньше творчески настроенные программисты активно экспериментировали со своими идеями, то сейчас все больше склоняются к готовым решениям. Хотелось бы, чтобы эта статья послужила своеобразным катализатором и подтолкнула вас к исследованиям.

Литература:

1. Мешков В. Пакетные команды интерфейса ATAPI. – Журнал «Системный администратор», №9, сентябрь 2004 г.
2. Касперски К. Техника защиты CD. БХВ-Петербург, 2004 г.
3. Касперски К. Искажение TOC как средство борьбы с несанкционированным копированием диска. – Журнал «Системный администратор», №9, сентябрь 2003 г.
4. Касперски К. Побег через брандмауэр плюс терминализация всей NT. – Журнал «Системный администратор», №5, май 2004 г.
5. Касперски К. Техника и философия хакерских атак. Солон-Р, 2005 г.
6. Касперски К. Коды Рида-Соломона в практических реализациях, или информация, воскресшая из пепла III. – Журнал «Системный администратор», №11, октябрь 2003 г.
7. Касперски К. Путь воина – внедрение в pe/coff-файлы. – Журнал «Системный администратор», №6, июнь 2004 г.