## Подходы к защите разрабатываемых программ от автоматической генерации инструментальными средствами программных закладок.

Если целью атаки является нанесение как можно большего вреда, то заманчивой целью для нарушителя, пытающегося внедрить РПС, являются программы, которые используют много различных пользователей, например, компиляторы. Для того, чтобы понять, как это можно сделать рассмотрим следующую упрощенную структуру компилятора, которая дает представление об общих принципах его работы:

compile:

get (line);

translate(line);

Согласно этой структуры компилятор сначала "получает строку", а затем транслирует ее. Конечно, настоящий компилятор устроен намного сложнее, чем эта схема, но этой иллюстрации отдано предпочтение потому, что она в виде некой модели разъясняет фазы лексического анализа трансляции компилятора. Целью РПС будет поиск новых текстовых участков во входных программах, которые будут транслироваться, и вставление в эти участки различного кода. В примере, представленном ниже, компилятор ищет текстовый участок "read\_pwd(p)", наличие которого в функции входа в данную компьютерную систему известно, как мы предполагаем, нападающей программе. Когда этот участок будет найден, компилятор не будет транслировать "read\_pwd(p)", а вместо этого буде транслировать вставку из РПС, которая может устанавливать "лазейку", которая потом позволит злоумышленнику легко получить доступ к системе. Измененный код компилятор а следующий:

compile;

get (line)

if line="readpwd(p)" then

translate (destructive means insertion);

else

translate(line);

fi;

В этом измененном коде, компилятор получает строку, ищет нужный текст, и если находит то транслирует код РПС. Код РПС может включать в себя простую проверку пароля "черного входа" (например, может признаваться правильный пароль "12345" для любого пользователя). Это особенно опасно, поскольку код источника больше не отражает того, что находится в объектном коде и просмотр кода источника (не смотря на то, что проверяется и компилятор) никогда не позволит уловить эту атаку (см рис.2.12).

Заметим, чти на рис.2.12 исходный текст или исполняемый код, включающий только те выражения, которые предлагались его разработчиком назван чистым, а код, содержащий РПС, - грязным. Далее заметим, что если компилируется атакованный компилятор и грязный исполняемы код устанавливается код в какой-либо рабочий директорий (так обычно и бывает), то компилятор с внедренным РПС может быть обнаружен, только если кто-нибудь вернется к источнику компилятора и проверит его (что редко случается). Но настоящий источник может быть восстановлен злоумышленником после компилирования грязного источника и создания грязного исполняемого компилятора. Это, вообще говоря, потом поможет восстановить настоящий исполняемый компилятор при рекомпиляции источника, но это редкий случай.

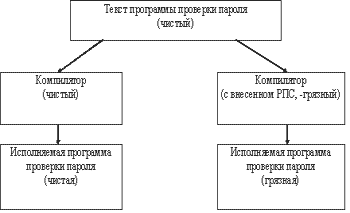


Рис.2.12. Работа компилятора с привнесенным РПС

## Методы идентификации программ и их характеристик.

##### 2.6. МЕТОДЫ ИДЕНТИФИКАЦИИ ПРОГРАММ И ИХ ХАРАКТЕРИСТИК

###### 2.6.1. Идентификация программ по внутренним характеристикам

Обеспечение безопасности программ, когда их исходные тексты попадают в руки злоумышленников, которые стремятся привнести в код программы РПС до того как программы подвергнутся компиляции, может заключаться в использовании методов идентификации программ и их характеристик.

При установления степени подобия исходной и исследуемой программы целесообразнее всего выбрать критерий, который насколько это возможно, не зависит от маскировок, вносимых в исходный текст программ нарушителем. Для этого необходимо выбрать параметры, характеризующие собственно программу и связанные с такими ее свойствами, которые трудно изменить и которые сохраняются в машинном коде программы. К таким параметрам может относится, например, распределение операторов по тексту программы, которое сложно изменить нарушителю, не искажая назначения программы. Такие изменения требуют глубокого понимания текста программы и логики вносимых изменений, что сопряжено с огромной работой по преобразованию программы.

Сначала рассмотрим вопросы анализа подобия последовательностей операторов в программе, поскольку этот подход не чувствителен к поверхностной маскировке, которую мог бы попытаться внести нарушитель, изменяя некоторые атрибуты программы, например, имена переменных, нумерацию строк и т.п. Для этого необходимо написать программу - анализатор, которая будет тестировать исследуемую программу, и выделять операторы, накапливая их в файле как данные, отражающие порядок их использования. Введем для последовательности операторов программы с номером n обозначение seq n. Тогда последовательность операторов для программ 1 и 2 будет обозначаться seq1 и seq2 соответственно. Одна из характеристик последовательности операторов - частота появления отдельного оператора. Анализ последовательности операторов оказывается эффективным в тех случаях, когда нарушитель изменяет или перемещает отдельные части программы, добавляет дополнительные операторы или погружает скопированную программу в некоторый модуль. При таких манипуляциях значительные участки последовательности операторов сохраняются неизменными, так как попытка изменить их равносильна переписыванию программы с сопутствующей ой трудоемкой операцией отладки. Рассмотрим распределение частот появления операторов в программе. Если программа скопирована целиком, но при этом замаскирована, число появлений каждого оператора в копии будет аналогично числу появлений в оригинале. Нарушитель может изменить некоторые операторы и добавить новые, но в целом процент изменений в программе, вероятно, будет мал, и распределение частот появления операторов ожидается одинаковым как для копии, так и для оригинала.

В то же время, если программа (или отдельный программный модуль) включена в большую программу необходимо рассматривать другую характеристику, связанную с сохранением структуры последовательности операторов и определяемую некоторой функцией. Такое подобие структур может быть выражена как максимум взаимной корреляцией функций двух программ, положение которого зависит от размещения модуля в программе. Интересен вопрос, будет ли заимствованная программа, откомпилированная в машинный код, обеспечивать достаточное значение корреляционной функции, чтобы выделить модуль, включенный в состав программы, а также будет ли взаимная корреляционная функция машинного кода соответствовать взаимной корреляционной функции исходной программы на языке высокого уровня.

Другая характеристика программы - автокорреляционная функция, определяющая меру соответствия, с которой одни и те же последовательности операторов повторяются в самой программе. По всей видимости, корреляционная функция должна быть чувствительна к добавлению, удалению или перемещению операторов, чем гистограмма частот появления операторов в программе, поскольку при сокращении последовательности значение корреляционной может существенно уменьшаться.

###### 2.6.2. Способы оценки подобия целевой и исследуемой программ с точки зрения наличия программных дефектов

Частоту появления операторов в программе можно изобразить в виде гистограммы. Для этого достаточно написать подпрограмму, которая будет подсчитывать каждое появление операторов в последовательности зафиксированных операторов программы. На рис.2.13 приведена гистограмма операторов для информационно - поисковой системы (ИПС) [[15]](http://citforum.ru/security/articles/kazarin/4.shtml#15); на абсциссе графика отмечено количество возможных операторов интерпретатора языка, используемого в этих текстах. Было выявлено , что некоторые операторы очень часто встречаются в программах различного назначения, некоторые редко, а некоторые вообще не встречаются. Для сравнения на рис.2.14 приведена гистограмма для программы редактирования текстов (РТ) , которая отличается от гистограммы на рис.2.13. Повторяемость некоторых операторов делает степень подобия программ визуально видимой, особенно для программ с одинаковым функциональным назначением, поскольку целевая направленность часто определяет и выбор операторов. Глаз плохо различает относительные значения амплитуд на различных гистограммах, поэтому удобнее изображать частоту появления операторов в одной программе в зависимости от частоты появления в другой. В этом случае для программ одного вида подобия точки будут размещаться на биссектрисе первого квадранта под углом 45o . Если программы существенно различаются по частоте вхождения операторов в последовательности операторов, тогда точки будут иметь существенный разброс.

Визуальное восприятие можно выразить математически, используя понятие корреляции. Простую меру оценки подобия можно получить, подсчитывая для каждого оператора с номером n среднее значение частоты его появления Dn в каждой программе. Математически эта мера подобия An для одного оператора записывается в виде

http://citforum.ru/security/articles/kazarin/53.gif

где fn(1),fn(1) - частоты появления оператора с номером n в программах 1 и 2 соответственно; Sn средняя сумма частот появления операторов с номером n в обеих программах; Dn - разность между частотами появления оператора с номером n в программах 1 и 2.

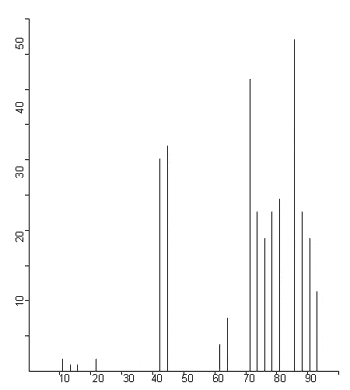


Рис.2.13. Гистограмма частоты появления операторов в программе для информационно-поисковой системы

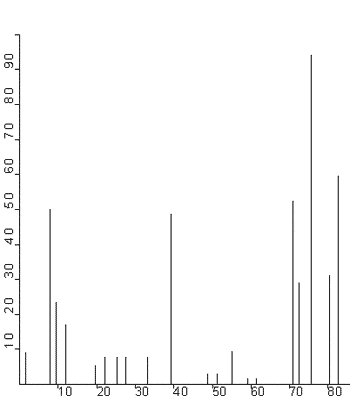


Рис.2.13. Гистограмма частоты появления операторов в программе для редакторов текста

Мера для всей последовательности операторов получается путем суммирования по всем операторам и нормировки (чтобы снять зависимость от длины программы, сумму делят на полное число операторов в обеих программах). Такая мера подобия A(1,2) для программ 1 и 2 имеет вид:

http://citforum.ru/security/articles/kazarin/56.gif

где N - число операторов в языке.

Если частота появления операторов в обеих программах одинакова, мера подобия A(1,2)=1, если операторы, присутствующие в программах, образуют пересекающие множества A(1,2)=-1. Для рассмотренных последовательностей операторов мера подобия равна A(ИПС,РТ)=-0,8. Статистическая формула для корреляции имеет вид:

http://citforum.ru/security/articles/kazarin/57.gif

где http://citforum.ru/security/articles/kazarin/58.gif - средние значения частот появления всех операторов в программах 1 и 2 соответственно.

Значения коэффициентов корреляции, вычисленных по формуле (2.6.2) всегда находятся в пределах от 0 до 1. Если программы имеют почти один и тот же вид подобия коэффициент корреляции близок к 1, в случае вычисления коэффициента корреляции для программ ИПС и РТ коэффициент корреляции равен C(ИПС,РТ)=0,56.

Вклад в значение коэффициента корреляции частоты появления операторов зависит от популярности применения некоторых операторов в программах разного типа. Этот вклад приводит к большему значению коэффициента корреляции по сравнению с мерой подобия. Это означает, что пороговое значение коэффициента корреляции при оценке подобия программ должен быть увеличен или, наоборот, соответствующие измерения должны быть скорректированы на величину, учитывающую степень подобия программ.

Распределение частот появления операторов наиболее полезно при сравнении двух программ, поскольку не зависит от порядка следования программ. Однако оно мало пригодно, когда программа встроена в программный продукт в сочетании с другими программными модулями, частотный спектр операторов которых может его поглотить. Для выявления присутствия модулей в большой программе более удобны коэффициенты взаимной корреляции.

Взаимная корреляция может быть использована для оценки взаимосвязи операторов в различных точках программы. Сравнивая последовательности операторов двух программ, можно достаточно просто проверить взаимную корреляцию операторов по их местоположению в этих последовательностях. Каждый раз, когда в последовательностях встречаются одинаковые операторы, это событие фиксируется и их общее число суммируется. В том случае, когда одна программа короче другой, более короткая продлевается циклическим повтором. Если обе программы идентичны, отношение числа зафиксированных операторов к общему числу операторов в программе равно 1.

Прямая корреляция между элементами множества не является оптимальной мерой, поскольку фоновая корреляция, обусловленная случайностью, может оказаться очень высокой, и поэтому необходимо переходить от анализа отдельных элементов к анализу групп элементов. Улучшенную корреляцию можно получить, если рассматривать группу элементов. Поэтому при поиске в некоторой последовательности операторов совпадающих элементов следует проверять, является следующий элемент совпадающим, и если это так, то совпадение фиксируется и этот процесс продолжается до окончания сравниваемой последовательности. Если последовательность имеет длину n, объем выборки, отнесенный к первому элементу, увеличивается с 1 до n.

Расчет корреляции (которая в данном случае называется взвешенной взаимной корреляцией) продолжается путем перехода к следующему (второму) элементу последовательности. В этом случае соответствующая выборка увеличивается с 2 до n-1.

Затруднения, связанные с использованием метода простой корреляции для последовательностей машинных команд, состоят в определении длины последовательности, поскольку длина должна быть различна для разных операторов высокого уровня. Метод взвешенной корреляции, когда устанавливается высокий порог повторяемости, решает эту проблему, поскольку, если и теперь отмечаются совпадающие последовательности, то весьма вероятно, что последовательность действительно выявляет совпадающие операторы языка высокого уровня, а случайные совпадения, имеющие корреляцию ниже установленного порога, во внимание не принимаются. Выше мы предполагали, что программы обработаны одним и тем же компилятором. В то случае, когда компилятор не известен, возможно, следует провести тестирование с различными компиляторами, пока корреляция не будет выявлена.

Для анализа корреляции внутри самой программы вводится автокорреляционная функция. Для этого необходимо воспользоваться подпрограммами для определения взаимных корреляций, если в качестве двух тестовых использовать одну и ту же последовательность. Автокорреляция представляет значительный интерес, поскольку дает некоторую числовую характеристику программы. По всей вероятности автокорреляционные функции различного типа можно использовать и тестировании программ на технологическую безопасность, когда разработанную программу еще не с чем сравнивать на подобие с целью обнаружения программных дефектов.

Таким образом, программы имеют целую иерархию структур, которые могут быть выявлены, измерены и использованы в качестве характеристик последовательности данных. При этом в ходе тестирования, измерения не должны зависеть от типа данных, хотя данные, имеющие структуру программы, должны обладать специфическими параметрами, позволяющими указать меру распознавания программы. Поэтому указанные методы позволяют в определенной мере выявить те изменения в программе, которые вносятся нарушителем либо в результате преднамеренной маскировки, либо преобразованием некоторых функций программы, либо включением модуля, характеристики которого отличаются от характеристик программы, а также позволяют оценить степень обеспечения безопасности программ при внесении программных закладок.