АНАЛИЗ НЕСАНКЦИОНИРОВАННОГО КОДА

Газизов Альберт Артурович

студент МИРЭА - Российский технологический университет, РФ, г. Москва

Аннотация. В этой статье детализируется некоторые особенности остатков кода, которые можно проанализировать, а затем использовать для идентификации их авторов. Можно называть данный процесс судебной экспертизой программного обеспечения, аналогичной судебной медицинской экспертизе: необходимо изучить остатки, чтобы получить свидетельства об этих факторах.

1. **Анализ исполняемого кода**

Часто след атаки представляет собой часть исполняемого кода, такого как вирус или программа-червь. К сожалению, многие функции кода, которые могли быть использованы в анализе, были удалены во время компиляции. Комментарии и отступы были удалены, а идентификаторы были заменены адресами памяти или именами регистров.

Например, оптимизирующий компилятор может генерировать один и тот же исполняемый код для каждого из следующих фрагментов программы на языке C:

```
for (x = 0; x < 10; x++) {
func(x);
}
x = 0;
while (x < 10) {
func(x);
++x;
}
x = 0;
while (TRUE) {
func(x);
if (x++ == 10) break;
}
x = 1;
do {
func(x-1);
x++;
} while (x <= 10);
```

Газизов А.А. Анализ несанкционированного кода // Молодежный научный форум: электр. сб. ст. по мат. LI междунар. студ. науч.-практ. конф. № 21(51). URL: https://nauchforum.ru/archive/MNF_interdisciplinarity/21(51).pdf (дата обращения: 05.09.2025)

Каждый из этих различных сегментов исходного кода демонстрирует особенности, которые могут быть использованы при определении стиля программирования. Однако эти функции могут быть потеряны.

Например, во время анализа программы Internet Worm этот остаток был перепроектирован для программ на C, скомпилированных в идентичные двоичные версии. Во многих случаях аналитики выбирали произвольные имена для переменных и локальных подпрограмм - компилятор не сохранял значения, поэтому выбор не имел значения. Когда впоследствии дизассемблированный код сравнивался с копией «реального» исходного кода, было замечено много небольших различий с копиями с обратной инженерией, скомпилированными в один и тот же двоичный файл. Исполняемый код, даже если он оптимизирован, по-прежнему содержит множество функций, которые можно учитывать при анализе:

Компетенция структур данных и алгоритмов. Предпочтения для определенных структур данных могут быть извлечены из исполняемого кода. Это может дать ключ к пониманию автора кода. Например, необоснованно подозревать начинающего программиста в создании кода, который широко использовал B-tree для хранения данных. Точно так же выбор алгоритмов, используемых в программе, может быть функцией, достойной анализа. Кажется, вероятным сделать вывод, что программист будет продолжать использовать алгоритмы, с которыми они особенно удобны.

В качестве примера рассмотрим Интернет-червя. Код использовал связанные списки в качестве основной структуры данных для построения длинных списков, которые неоднократно подвергались поиску. Это был, безусловно, плохой подход, поскольку повторный поиск в длинных списках резко снижал эффективность программы. Такого совпадения, безусловно, недостаточно для обоснования какого-либо конкретного действия, оно может помочь укрепить другие доказательства, полученные другими способами.

С этим связан способ доступа к структурам данных. В языках с указателями и массивами выбор которых часто зависит от программиста. Аналогично, использование перекрывающихся структур (например, оператора EQUIVALENCE в Fortran и оператора union в C) обеспечивает индикатор. Некоторые программисты используют эти структуры, в то время как другие используют принудительные и побитовые операции.

Компилятор и исполняемая информация о системе. Код может содержать контрольные признаки его происхождения. Уникальный порядок команд может указывать на конкретный компилятор в качестве источника кода. Код может содержать функции системных вызовов, найденные только в определенных операционных системах. Эти биты информации могут исключать или подтверждать отдельных лиц как авторов кода.

В случае многих вирусов анализ двоичного кода может выявить, что он был написан на C или Pascal от определенного поставщика. Это можно определить потому, что в двоичном файле присутствуют подпрограммы поддержки (иногда называемые «thunks») и вызовы библиотек, уникальные для этого поставщика.

Навыки программирования и знание системы. Уровень знаний автора программы, как для рассматриваемой операционной системы, так и для компьютерного программирования в целом, можно оценить по исполняемому коду. Например, программирование, которое дублирует функциональность, уже обеспеченную стандартными системными вызовами, использует рекурсию или делает надлежащие вызовы расширенных системных функций, может указывать на различные уровни знаний и навыков.

Кроме того, включение или пропуск кода проверки ошибок также весьма показателен. Некоторые программисты редко (или никогда) не включают код обработки исключений в свои программы. Другие всегда включают такой код. В случаях, когда код иногда включается, это может обеспечить идентифицируемый набор подпрограмм, которые автор всегда проверяет (возможно, из-за прошлых сбоев программы с этими подпрограммами). Затем этот набор можно сравнить с набором из других известных программ как показатель сходства.

Выбор системных вызовов. Функции поддержки, используемые в коде, также могут

указывать на фон программиста. Например, в системе UNIX иногда есть два разностных вызова, чтобы найти первый экземпляр определенного символа в строке. Процедура индексирования получена из версии UNIX для Berkeley (BSD), а функция strchr - из версии UNIX для System V. Пользователи обычно проявляют явное предпочтение тому или иному вызову при программировании в среде, которая предоставляет обе функции. Опыт чтения и переноса кода убедил нас в том, что существует множество таких наблюдаемых предпочтений.

Ошибки Программисты, как правило, допускают ошибки во всех программах, кроме самых простых или тщательно закодированных. Некоторые программисты будут постоянно делать одни и те же типы ошибок, такие как ошибки «один за другим» в массивах обработки циклов. Каталогизация и сравнение этих ошибок может предоставить еще один показатель для определения авторства подозрительного кода.

Возможно, что таблица символов все еще может присутствовать в исполняемом файле, как это часто бывает, когда компилятору предписывается генерировать отладочную информацию. В этом случае некоторые из функций, обычно связанных с исходным кодом программы, также могут быть рассмотрены в исполняемом коде.

2. **Анализ исходных файлов**

Исходный код программы обеспечивает гораздо более богатую базу для специфики писателя функций программирования.

Язык. Пожалуй, самая непосредственная особенность кода - это язык программирования, выбранный автором. Причины выбора могут быть не очевидными, но могут включать в себя наличие и знания. Было бы неразумно подозревать человека в том, что он является автором программы, написанной на языке программирования, который он не знает.

Форматирование. Форматирование исходного кода часто демонстрирует очень личный стиль. Формат также имеет тенденцию быть согласованным между программами, облегчая автору читать то, что она написала. Эти факторы указывают на то, что стиль форматирования кода должен давать специфические особенности автора. Размещение разделителей составных операторов, несколько операторов в строке, формат объявлений типов, форматирование аргументов функций и многие другие характеристики могут быть определены в рассматриваемом коде. Это предполагает, что рассматриваемая среда программирования не имеет широко используемого жесткого средства форматирования кода («pretty-printer»), которое могло бы создать наблюдаемый стиль.

Другая часть информации, которая может стать доступной в этом анализе, выбор редактора. Например, может быть возможно распознать стили форматирования, созданные и редактором, такими как Emacs, или обнаружить встроенные команды настройки режима. Редакторы программ, ориентированные на синтаксис, могут также обеспечить особый и необычный стиль, если они станут более распространенными.

Специальные функции. Некоторые компиляторы поддерживают дерективы или специальные макросы, которых нет в каждой системе. Наличие любой из этих специальных функций может дать подсказки относительно среды разработки программного обеспечения автора. Включение конструкций условной компиляции, особенно тех, которые содержат файлы инициализации и декларации, также может предоставить аналогичную информацию о среде.

Стили комментариев. У пользователей часто свойственный способ комментирования своих программ. Некоторые используют строки графического символа для выделения комментариев из кода. Другие размещают заголовки комментариев над каждой функцией, описывая ее. Третьи избегают комментариев любой ценой.

Частота и детализация присутствующих комментариев также могут отличаться. Некоторые программисты комментируют короткими тегами, а другие пишут целые абзацы. Это может привести к измеримой модели.

Имена переменных. Выбор имен переменных - это еще один аспект программирования, который часто указывает на стиль автора. Некоторые программисты предпочитают соединять слова в идентификаторах с подчеркиванием, другие используют подход SmallTalk и используют первую букву каждого слова без разделителя. Ярые разработчики программного обеспечения могут использовать схему именования, такую как метапрограммирование, которая включает информацию о типе в имя переменной. Тем не менее, другие никогда не использовали более одного или двух символов в имени переменной. Полезной метрикой для анализа идентификаторов может быть что-то вроде распределения расстояний Хэмминга между именами. У большинства опытных программистов есть набор «служебных» имен переменных, которые они используют для локальных переменных при кодировании небольших сегментов. Общие примеры включают в себя мусор, ii и indx. Анализ этих имен может быть полезен при сопоставлении с другим кодом того же автора.

Правописание и грамматика. Многим программистам трудно писать правильную прозу. Имена переменных с ошибками (например, TransactoinReciept) и слова внутри комментариев могут быть довольно убедительными, если ошибка написана согласованно. Аналогично, небольшие грамматические ошибки внутри комментариев или печатных выражений, такие как неправильное использование или чрезмерное использование тире и точек с запятой, могут обеспечить небольшое дополнительное сходство между двумя программами.

Ошибки. Некоторые авторы постоянно допускают одинаковые ошибки в своем кодировании. Часто это ошибки, которые редко вызывают проблемы, и только с экстремальными значениями или при переносе на другое оборудование. Именно из-за того, что эти ошибки редко вызывают проблемы, пользователи склонны продолжать вводить их в свой код. Наличие идентифицирующих ошибок должно служить очень убедительным доказательством сходства между двумя частями кода.

Использование возможностей языка. Способ использования авторами язык программирования также может их дифференцировать. Некоторые авторы могут последовательно использовать подмножество доступных функций, в то время как другие могут более полно использовать все функции. Например, автор может последовательно использовать цикл while, даже если цикл for / do или repeat..until будет более подходящим. Точно так же использование вложенных операторов if вместо операторов case или (отсутствие) спецификации параметров по умолчанию в операторах case может отличать особенности кода.

Другие примеры, которые попадают в эту категорию, включают возвращаемые значения в параметрах процедуры в сравнении с возвращаемыми значениями функции, использование перечисляемых типов данных, использование типов поддиапазонов, использование побитовых логических операций, использование постоянных типов данных и использование структур и указателей. Средний размер подпрограмм также может использоваться в качестве идентифицирующей функции: некоторые программисты будут кодировать 300 линейных модулей, а другие никогда не будут иметь модуль больше, чем уместится на экране сразу. Один из аспектов использования языковых функций относится к компьютерным языкам, которые программист может знать лучше. Например, программисты, которые проводят большую часть своего времени на процедурных языках, редко используют рекурсию. Изучение программирования на языке, таком как Basic или Fortran, также может привести к сокращению использования while и do …till структуры. Дальнейшее изучение таких влияний может привести к заметной тенденции использовать или избегать определенных языковых особенностей.

Scoping. Соотношение глобальных и локальных идентификаторов может быть характерной чертой автора. Кроме того, объявление вспомогательных функций как доступных только в ограниченной области также может способствовать идентификации программиста.

Пути выполнения. Общим фактором, обнаруживаемым при анализе студенческих программ, а также при анализе некоторого вредоносного кода, является наличие кода, который не может быть выполнен. Код представлен либо как функция, которая никогда не была полностью включена, либо представлена как код, который присутствовал для отладки и не был удален. Это отличается от кода, который присутствует, но не выполняется из-за ошибки в

логическом условии - это полностью функциональный код, на который никогда не ссылается ни один путь выполнения.

Метрики. Программные метрики могут использоваться для определения средних характеристик человека. Некоторые применимые метрики могут включать число строк кода на функцию, отношение комментария к коду, меры сложности функций, меры Холстеда и метрики МакКейба.

Список литературы:

1. От Нестора до Фонвизина. Новые методы определения авторства. М.: Издат. группа Прогресс, 1994.

2. Charles Simonyi. Meta-programming: A software production technique. Byte, pages 34–45, September 1991

3. C.C. Zou, W. Gong, D. Towsley. Code Red Worm Propagation Modeling and Analysis. 9th ACM Conf. on Computer and Communication Security, 2002

4. M. Mannan and P.C. van Oorschot. On instant messaging worms, analysis and countermeasures. In Proceedings of the 2005 ACM Workshop on Rapid Malcode, pages 2-11, November 2005.