ДИНАМИЧЕСКАЯ БИНАРНАЯ ИНСТРУМЕНТАЦИЯ ПРИЛОЖЕНИЙ С ПОМОЩЬЮ PIN

Андросова Татьяна Евгеньевна

студент 4 курса, кафедра геоинформатики и информационной безопасности, Самарский университет, РФ, г. Самара

Курочкин Владислав Михайлович

студент 4 курса, кафедра геоинформатики и информационной безопасности, Самарский университет, РФ, г. Самара

Болдырев Артем Сергеевич

студент 4 курса, кафедра геоинформатики и информационной безопасности, Самарский университет, РФ, г. Самара

Чернов Роман Вячеславович

студент 4 курса, кафедра геоинформатики и информационной безопасности, Самарский университет, РФ, г. Самара

Процесс анализа современных бинарных приложений постоянно усложняется. Связано это напрямую с возрастающей сложностью самих программных обеспечений: они становятся больше и динамичнее. Старые методы уже не столь эффективны и во многих случаях на смену им приходят новые способы анализа. Одними из современных и наиболее перспективными являются методы динамической бинарной инструментации (DBI).

Сегодня динамические методы вытесняют методы статические, хотя они и продолжают обладать рядом преимуществ. Рассмотрим последние в качестве сравнения.

Статический анализ выполняется до запуска самого приложения, до момента его компиляции, в этом его сильная и слабая сторона одновременно. Благодаря выполнению анализа кода до его компиляции удаётся избежать многих ошибок, получить часть информации о запускаемом приложении ещё до старта, что очень сильно помогает при отладке. Инструменты для проведения статического анализа имеются сейчас в составе большинства популярных IDE. Так же к его преимуществам многие исследователи относят полноценный обхват исследуемого кода. Однако истинность последнего заявления в настоящее время не соответствует действительности. Дело в том, что большое множество современных программ генерируют код во время исполнения, используя различные динамические приёмы. И в таком случае статический анализ уже бессилен: он не способен дать полный обхват кода, не способен предоставить максимум информации о запускаемом приложении. И это при условии, что есть исходные коды. А что делать, если исходные коды отсутствуют? Что делать, например, исследователям зловредов? Статический анализ без наличия исходных кодов приложения в разы усложняется и даёт ложные сведения т.к. возникает проблема отделения данных от кода. Выходом в таких случаях служит динамическая инструментация, которая приобретает всю большую популярность у исследователей.

Концепция динамической бинарной инструментации главным образом заключается сборе информации о приложении в момент его исполнения, то есть непосредственной работы.

Андросова Т.Е., Курочкин В.М., Болдырев А.С. [и др.] Динамическая бинарная инструментация приложений с помощью PIN // Молодежный научный форум: Технические и математические науки: электр. сб. ст. по мат. XLI междунар. студ. науч.-практ. конф. № 1(41). URL: https://nauchforum.ru/archive/MNF_tech/1(41).pdf (дата обращения: 14.04.2025)

Достигается это посредством вставки заданных исследователем процедур в определённые участки кода, например, до или после вызова функции, или даже вместо неё.

На сегодняшний день существуют и продолжают развиваться довольно большое количество фреймворков, работа которых основана на применении методов DBI. Можно выделить следующие: Pin (http://www.pintool.org), Frida (http://www.frida.re), DynamoRIO (http://dynamorio.org), Dyninst (http://www.dyninst.org), Valgrind (http://valgrind.org). В этой статье остановимся поподробнее на Pin.

Pin – это фреймоворк по динамическому оснащению кода для платформ Windows и Unix от компании Intel. Он может использоваться, как отдельно, так и быть интегрированным в среду разработки Visual Studio от корпорации Microsoft. Pin представляет собой программное обеспечение с закрытыми исходными кодами, однако его можно абсолютно бесплатно скачать для использования с официального сайта по приведённой выше ссылке.

При работе с Pin необходимо учитывать так называемые уровни гранулярности. В Pin их существуют три: подпрограмма, инструкция и образ (routine, instruction и image). Эти уровни выбраны для удобства выбора между уровнем детализации и производительностью, написанного инструмента.

Отличный пример по работе с Pin описал в своей статье Хади Браис (ссылка приведена в списке используемой литературы). Его и рассмотрим в качестве основного. Созданный инструмент (Pintool), будет помогать выслеживать утечки памяти в программах написанных на языке C/C++, определяя утечки памяти (при выделении, но не освобождении памяти), двойном освобождении (неоднократное освобождение) и освобождение памяти, которая не выделялась (при передачи функции free() значения NULL). Pintool будет работать с функциями free(), malloc() (следить за операторами new и delete не имеет смысла т.к. в их основе также лежат вызовы free() и malloc()). Алгоритм работы очень простой: при вызове malloc() будет записываться возвращаемое значение, а при вызове free() будет просиходить сверка адрес памяти с заранее записанными. При выделении, но не освобождении – будет происходить отметка о том, что область памяти освобождена. Но, если она была освобождена и выделена, то это будет сигнализировать о повторном освобождении. Операторы new и delete не нуждаются в проверки т.к. их функционал реализован так же на вызовах функций malloc() и free().

Для начала необходимо определится с уровнем гранулярности для нашего pintool. Хади Браис выбирает уровень образа, аргументируя это тем, что это повлечёт лишь небольшие издержки производительности.

Переходим к исходному коду. Воспроизведём его в статье лишь частично, он будет доступен полностью по приведённый ссылке. Опишем в виде таблицы глобальные переменные.

Record	MallocMap	ProgramImage	KNOB	OutFile
Флаг нахождения внутри функции main	Отвечает за состояние памяти	Содержит название образа	Содержит в себе параметр командной строки	Имя выходного файла для логгирования

Рисунок 1. Таблица описания некоторых переменных

Работа всех функций будет производиться только при условии нахождении в функции main(). Если условие не выполняется, то будет выполнен оператор return.

Рассмотрим последовательно используемые Хади функции. После выполнения вышесказанной проверки в функции RecordMalloc() происходит определение наличия этого адреса в MallocMap. Если адрес отсутствует в контейнере, то происходит его вставка со значением FALSE (то есть он занят). В ином случае для этого адреса была выделена и впоследствии освобождена память (значение TRUE). Аналогично будет работать процедура для функции free(), её описывать не будем.

Любой инструмент должен понимать, когда он должен производить оснащение программы. Для этой цели определяют так называемые функции оснащения. В данном случае это функция Image(). Переменная Img – это образ исполняемого файла. Имена всех функций Pin начинаются с префикса: в случае образа IMG_, в случае процедур RTN_. При условии, что необходимая процедура находится в образе, RTN - будет валидным. После нахождения можно перейти к вставке функции анализа. Для этого будет осуществляться вызов RTN_InsertCall(), которая в качестве первого аргумента имеет оснащаемую процедуру, второго – перечисление, которое показывает куда необходимо вставить функцию анализа и третьего – сама процедура анализа. Кроме этого нужно обязательно написать функцию завершения: она будет вызываться после завершения программы.

Скажем пару слов о функции main(). Сначала происходит вызов PIN_Init, для инициализации переменных KNOB. Далее – PIN_InitSymbols(). Она необходима загрузки таблицы символов. В main() необходимо также прописывать функции регистрации. И обязательно завершить функции оператором return (рис. 2).

```
int main(int argc, char *argv[]) {
  PIN_Init(argc, argv);
  ProgramImage = argv[6]; // предполагаем, что имя образа всегда
находится по индексу 6
  PIN_InitSymbols();
  OutFile.open(OutFileName.Value().c_str());
  IMG_AddInstrumentFunction(Image, NULL);
  PIN_AddFiniFunction(Fini, NULL);
  PIN_StartProgram();
  return 0;
}
```

Рисунок 2. Функция main()

В итоге мы получили полноценный инструмент для динамического анализа приложений. Для полного ознакомления с особенностями его написания, а также с исходным кодом, советуем ознакомиться со статьёй Хади Брайса по приведённой ниже ссылке в списке литературы.

Список литературы:

1. Инструментация — эволюция анализа – [Электронный ресурс] – https://xakep.ru/2013/09/11/61232/.

2. Application Instrumentation: Application Analysis with Pin – [Электронный ресурс] – https://msdn.microsoft.com/en-us//magazine/dn818497.aspx.