Повышение безопасности решений KasperskyOS на основе анализа графа потока управления

Сорокин Игорь Александрович, Igor.Sorokin@kaspersky.com
АО «Лаборатория Касперского», Россия, Москва, 125212, Ленинградское шоссе, д.39А, стр.3, БЦ «Олимпия Парк»

1. Мотивация создания

Одним из элементов обеспечения безопасности решений являются механизмы контроля доступа к программно-аппаратным ресурсам, которые находятся под управлением ОС. Правила безопасности, которые выполняются такими механизмами (например, LSM – Linux Secure Module для Linux или KSS – Kaspersky Security System для KasperskyOS), позволяют предотвратить доступ к критическим (с точки зрения ИБ) данным и\или распространение атаки в случае успешного проникновения и активизации эксплойта в решение. Однако, отказ в доступе не позволяет однозначно сказать, что процесс, для которого произошел отказ, заражен эксплойтом, т.к. вычисление вердикта при попытке доступа осуществляется в соответствии со множеством условий и состоянием решения.

Очевидно, что для выполнения своих функций процесс должен взаимодействовать с ядром ОС и системными сервисами. С другой стороны, активация эксплойта также требует получения доступа к ресурсам системы путем обращения к ядру ОС и системным сервисам. С точки зрения механизмов безопасности контролируемый процесс выполняет определенную последовательность обращений к ядру ОС. Активизация эксплойта нарушает наблюдаемую последовательность обращений, что может являться индикатором нарушения работоспособности процесса при условии, что логика взаимодействия контролируемого процесса с ядром ОС и системными сервисами определена заранее.

Рассмотрим, как можно сформировать возможную последовательность обращений контролируемого процесса к ядру ОС (модель процесса) на основе анализа исходного кода приложения, разработанного на языках C\C++ для компиляторов gcc и clang. Покажем, как происходит вычисление вердикта о нарушении работоспособности при получении информации об обращении процесса к ядру и системным сервисам.

2. Построение модели процесса

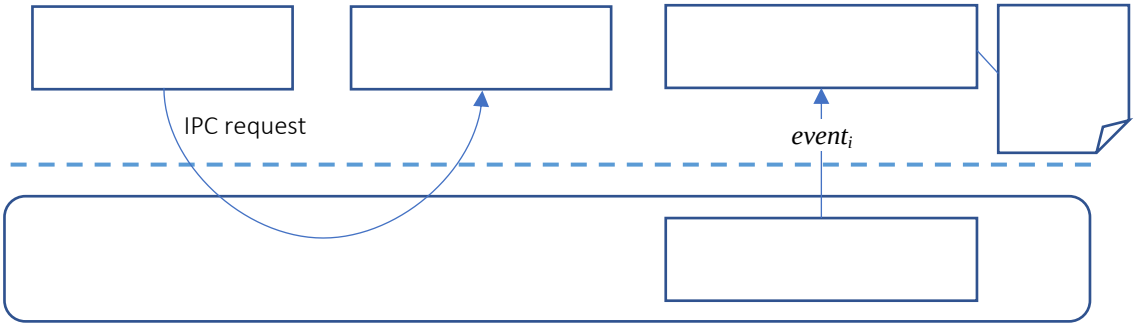
Работоспособность решения и его компонентов – это способность выполнят заявленные функции в соответствии с установленными требованиями. Отклонение от выполнения заявленных функций можно считать нарушением работоспособности. При этом отклонениями могут считаться как невыполнение заявленных функций, так и попытка выполнения каких-либо функций, не предусмотренных требованиями. Например, для приложения авторизации пользователя требуемой функции является сопоставления предъявленной информации аутентификации с записями зарегистрированного пользователя. Невыполнение этой функции означает нарушение работоспособности. Однако, попытка передачи информации аутентификации (например) по сетевому каналу также является отклонением от требований, а значит нарушением работоспособности.

Очевидно, что для выполнения функций процессу необходим доступ к аппаратно-программным ресурсам системы. При этом доступ к ресурсам необходим как для выполнения штатных функций, так и для выполнения паразитных функций (не предусмотренных требованиями), что может быть вызвано активацией внедренного в процесс эксплойта.

Выполнение функций процесса определяется исходным кодом приложения, которое компилируется и компонуется в бинарный файл. При этом в процессе компиляции

возможно построение графа потока управления (Control Flow Graph - CFG), который определяет множество всех возможных путей исполнения программы. Обычно CFG состоит из вершин (базовых блоков) и направленных дуг, которые показывают возможные пути перехода логики выполнения. Базовый блок представляет прямолинейный участок кода, не содержащий в себе ни операций передачи управления, ни точек, на которые управление передается из других частей программы. Если базовый блок включает вызов другой функции, то такая запись может быть легко идентифицирована. Обращение к ядру ОС с помощью системных вызовов также может быть легко найдена в базовом блоке. Таким образом, CFG приложения является полной моделью процесса с помощью которой возможно проследить в каком порядке будут происходить обращения к ядру ОС при выполнении логики программы.

Рассмотрим каким образом данная модель может быть использована для контроля работоспособности процесса KasperskyOS.



В KasperskyOS все межпроцессные взаимодействия в том числе и обращения к ядру выполняются через IPC-интерфейсы (Inter Process Communication), которые включают требуемый набор методов. Запрос на взаимодействие некоторого процесса App к любому другому процессу, например VFS, осуществляется через IPC под контролем ядра ОС. Системные вызовы (например, выделение памяти, создание потока и т.п.) также являются IPC-запросами, интерфейс для которых предоставляется ядром ОС. В ядро встроен монитор безопасности (KSS), который для каждого обращения формирует вердикт разрешен или запрещен запрос в соответствии с политиками безопасности. При этом KSS имеет встроенную систему логирования IPC запросов. KCFM (Kaspersky Control Flow Monitor) подключается к системе логирования KSS и получает набор событий $\{event_i\}$. Событие $event_i$ включает следующую информацию: источник IPC-запроса, приемник IPC-запроса, информация об IPC интерфейсе и идентификатор вызываемого IPC-метода. Таким образом, KCFM с помощью анализа событий $\{event_i\}$ может определить последовательность обращений контролируемого приложения App к ядру ОС.

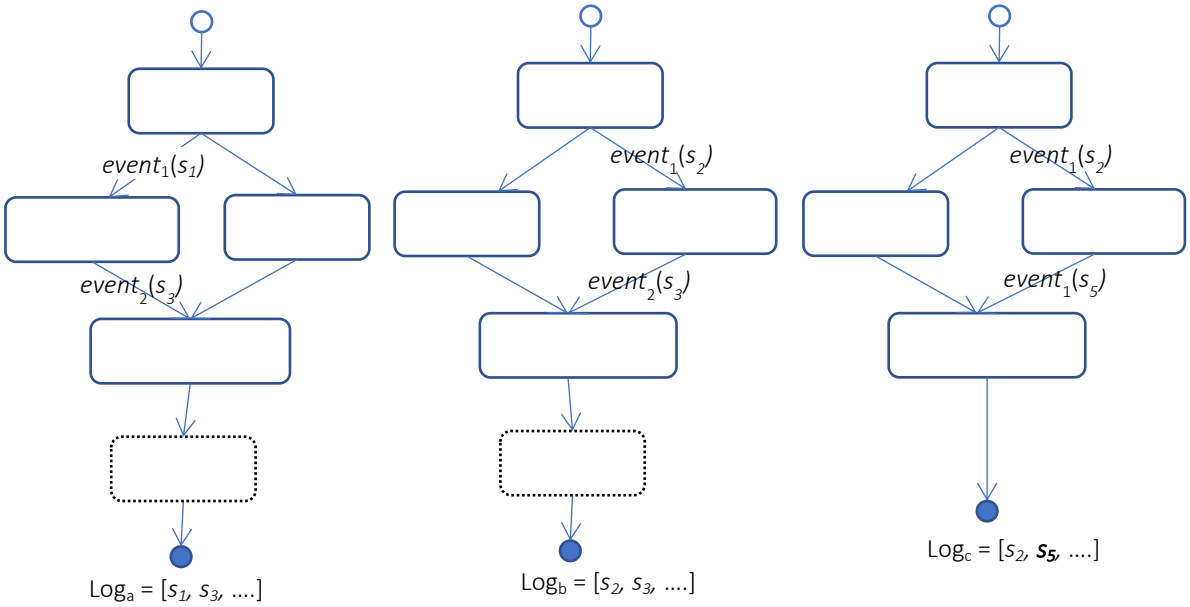
3. Вычисление вердикта контроля работоспособности

Пусть в процессе анализа исходного кода сформирован CFG приложения, который состоит из базовых блоков $\{b_1, b_2, b_3, \ldots\}$. Блоки соединены между собой направленными дугами, которые определяют логику выполнения программы. Блок может содержать обращение к ядру с помощью некоторого метода известного IPC-интерфейса – опорный вызов (s). Совокупность всех методов IPC-интерфейса обозначим $S=\{s_1,s_2,s_3,\ldots\}$. В случае, если базовых блок содержит опорный вызов обозначим такой блок как $b_n(s_i)$, где n – номер базового блока в CFG, i – номер опорного вызова множества S.

В процессе работы программы происходит обращения к ядру путем вызова определенных методов IPC-интерфейса ядра, что логируется системой KSS. Каждый

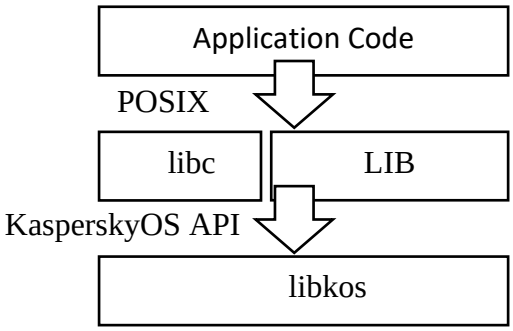
вызов s_i приводит к появлению события $event_j(s_i)$ (j – определяет порядковый номер события), которое передается в подсистему контроля работоспособности (Monitor).

В процессе инициализации Monitor загружает CFG граф процесса и устанавливает указатель p на первый базовый блок. При поступлении события $event_i$ алгоритм контроля выделяет опорный вызов s_i из события, после чего производит поиск базового блока, который включает данный вызов. Если такой базовый блок найден, то указатель p перемещается на найденный базовый блок. Если базовый блок не найден, то формируется решение о попытке несанкционированного обращения к ядру, то есть нарушении работоспособности процесса. Прием событий останавливается, вердикт передается в подсистему контроля, которая принимает решение: перезапуск всей системы, перезапуск контролируемого процесса. Пример, такого вычисления и перемещения указателя p для трех разных последовательностей событий (a, b, c) представлен на следующем рисунке.



4. Формирование модели приложения KasperskyOS

Для формирования модели приложения (CFG) необходимо получить CFG всех функций, используемых при компоновке приложения. Общая компонентная архитектура приложения для KasperskyOS выглядит следующим образом:



Код приложения (Application Code) реализует логику работу. При этом для общения с каналами вода-вывода и вычислительными ресурсами используется библиотека libc, которая предоставляет POSIX интерфейс. Реализация библиотеки libc базируется на библиотеке libkos. Библиотека libkos представляет KasperskyOS API, предназначенный для формирования запросов к операционной системе. При создании кода приложения

также используются и другие наборы библиотек (LIB), компилируемые для целевой платформы KasperskyOS.

В процессе компиляция исходного кода библиотек и приложения для каждой единицы трансляции формируется CFG функций данной единицы трансляции. Полный состав функций может быть определен с помощью map-файла.

При использовании компилятора gcc формируемые CFG не позволяют в дальнейшем получить полный CFG приложения, т.к. в CFG отсутствует информация о манглировании функций. Решение данной задачи осуществляется с помощью gcc-plugin, которые собирает информацию манглирования. Более удобным представляется использование clang-компиляторов, которые формируют bite-код каждой единицы трансляции. Анализ bite-кода позволяет получить CFG-используемых в приложении функций, а также определить связи между функциями. В результате такого анализа формируется модель приложения, которая представляет собой набор CFG-функций, описание связей между функциями, а также определение точки старта, с которой начинается выполнение кода контролируемого процесса.

5. Заключение

Предложенный подход позволяет реализовать систему контроля работоспособности, которая по взаимодействиям процесса с операционной системой позволяет однозначно определить нарушение логики работы процесса в случае успешной эксплуатации уязвимости или выполнения некоторого недокументированного кода. При этом не требуются дополнительные наложенные средства безопасности, что особенно важно в автономных устройствах, IOT-устройствах и т.п.