11. Инкремент/декремент свойств модели с учетом многопоточности;

12. Возможность использовать таблицы «только для чтения»;

13. Возможность использовать «Fluent Interface», т. е. цепочки вызовов наподобие нижеприведенной:

$user = User::model() -> findByAttributes(array('username' => 'andrey')) → populate($data) -> save()[3].

Также следует заметить, что применение ActiveModel делает процесс разработки более гибким. Эта гибкость характеризуется тем, что при изменении таблиц в БД нет необходимости переписывать все функции, работающие с ними, а нужно всего лишь переписать функцию __construct() модели. Таким образом, применение паттерна ActiveModel может значительно ускорить как процесс разработки, так и процесс сопровождения кода веб-приложения и обезопасить веб-приложение уже на этапе разработки.

СПИСОК ЛИТЕРАТУРЫ:

1. Миф о надежности open-source развеян: обнаружена 1 ошибка на каждую 1000 строк // Softodrom.ru. URL: http://news.softodrom.ru/ap/b2388.shtml.
2. *Веллинг Л., Томсон Л.* Разработка Web-приложений с помощью PHP и MySQL. Вильямс, 2009.3. *Дюбуа П.* MySQL. Сборник рецептов. Символ, 2007.
4. *Фаулер М.* Архитектура корпоративных программных приложений. Вильямс, 2009.
5. *Хоп Г., Вульф Б.* Шаблоны интеграции корпоративных приложений. Вильямс, 2009.

[3] Представлен код на языке веб-программирования PHP.

Ю. М. Туманов, С. В. Гаврилюк

ИСПОЛЬЗОВАНИЕ ПОВЕДЕНЧЕСКОГО АНАЛИЗА С ПРИМЕНЕНИЕМ ПОВЕДЕНЧЕСКИХ СИГНАТУР ДЛЯ ВЫЯВЛЕНИЯ ВРЕДОНОСНОГО КОДА НА ПРИМЕРЕ JAVASCRIPT-СЦЕНАРИЕВ

Интернет построен на протоколе передачи данных HTTP, который позволяет передавать HTML-текст страницы интернет-браузерам пользователей. Данная технология повсеместно используется в сети Интернет и постоянно развивается. Очередным этапом развития технологии HTTP в 1995 г. стало создание языка JavaScript.

На данный момент количество интернет-страниц, использующих технологию JavaScript, составляет 75 % [1]. В свою очередь, около 80 % интернет-страниц, использующих JavaScript, содержат в себе различные уязвимости [2].

В антивирусной индустрии широко используются два подхода к детектированию вредоносного программного обеспечения — детектирование вредоносного кода по его сигнатуре либо по его поведению.

Сигнатурный анализ является простейшим вариантом детектирования вредоносного кода. На данный момент этот подход наиболее распространен и исследован.

Поведенческий анализ предоставляет возможность детектировать вредоносный код не на основании его свертки и сравнения с сигнатурой, а на основании математических алгоритмов, позволяющих описать поведение программы, проанализировать воздействие программы на ее окружение. Поведенческий анализ является перспективным направлением исследований.

Модель угрозы вредоносного использования языка JavaScript можно представить следующим образом:

- находящийся в теле интернет-страницы JavaScript-код не был реализован как злонамеренный, но нарушитель внедрил в код исходной интернет-страницы сценарий, выполнение которого может привести к нежелательным результатам;
- интернет-ресурс или конкретная интернет-страница были реализованы изначально с целью совершать несанкционированные действия на ЭВМ просматривающего интернет-ресурс или интернет-страницу пользователя.

В случае реализации одной из вышеописанных угроз возможны атаки на:

- пользователя, использующего интернет-страницу или интернет-ресурс, содержащий вредоносный JavaScript-код;
- интернет-сервис, на котором располагается код вредоносной интернет-страницы или код страницы, в которую вредоносный JavaScript-код был внедрен.

Для представления программы с точки зрения взаимодействия различных наборов функций, ее составляющих, программа представляется в виде графа потока управления. Граф потока управления рассматривается как представление всех путей выполнения кода программы и ее состояний во время ее выполнения. Каждая вершина в графе — линейный участок кода без переходов. Переходы представляются в виде дуг графа или объединения вершин и дуг в путь в графе. Направленные ребра используются, чтобы представить переходы в потоке выполнения [3].

На основании алгоритма построения графа потока управления программы создается база данных с проанализированными программами, в которую вносятся те, поиск поведения которых нас впоследствии будет интересовать.

Каждая конкретная вершина графа потока управления анализируемой программы или находящаяся в базе данных с уже проанализированными вершинами описывается следующим образом:

- идентификатор — каждой вершине присваивается идентификатор, позволяющий впоследствии обращаться к данным, описывающим рассматриваемую вершину;
- список функций вершины — в данный список входят все стандартные функции языка JavaScript, вызов которых произведен в ходе выполнения JavaScript-инструкций, принадлежащих к линейному участку программы, соответствующему данной вершине;
- список наследных вершин — в данный список входят вершины, в которые возможен переход из рассматриваемой вершины после выполнения содержащихся в данной вершине функций.

Для описания поведенческой сигнатуры программы, реализованной на языке JavaScript, были введены следующие понятия, описывающие программу:

- состояние программы — функции программы, исполняемые в текущей вершине графа потока управления программы;
- поведение программы — последовательность состояний, необходимых для перехода из одного состояния в другое.

Пусть набор всех функций языка JavaScript описывается алфавитом Ω из ω символов, ω_t — t-й символ данного алфавита, являющийся определенной стандартной функцией языка JavaScript.

Пусть Alyz — граф потока управления анализируемой программы, состоит из n вершин, a_q — q-я вершина в Alyz, $q=\overline{1,n}$, Sig — граф потока управления поведенческой сигнатуры, состоит из m вершин, s_g — какая-либо конкретная вершина графа поведенческой сигнатуры, $g=\overline{1,m}$, $n<m$. Пусть a_q, s_g состоят из k и l — стандартных функций языка JavaScript, входящих в каждый конкретный список функций вершины; ρ, n, m, k, $l \in N$, и пусть гомоморфизм рассматривается как функция Ψ, которая отображает Sig в Alyz, такая, что если в графе Sig существует путь, соединяющий вершины s_i и s_j, то и в графе Alyz существует путь, соединяющий вершины $\Psi(s_i)$ и $\Psi(s_j)$.

Пусть слова wa_q и ws_g определены как объединения всех символов (которыми являются функции языка JavaScript), входящих в эти слова.

Пусть ρsydr — вероятность существования гомоморфизма Ψ графа Sig в граф Alyz.

cros(wa_q,ws_g) — функция «схожести» слов wa_q и ws_g, принимающая значения «0» или «1», DLD — функция, вычисляемая на основании алгоритма Дамерау—Левенштейна для всех вершин графа потока управления анализируемой программы и графа потока управления поведенческой сигнатуры [4].

В случае, если существует f(ρsydr,m) вершин $a_q \in$ Alyz, для которых существует $s_g \in$ Sig, таких, что cros(wa_q,ws_g) = 1, то принимается решение о проверке существования гомоморфизма Ψ графа Sig в граф Alyz. Все обнаруженные вершины a_q с cros(wa_q,ws_g) = 1 в Alyz помечаются. На основании алгоритма «поиска простого пути» [5] проверяется, есть ли путь в Alyz, позволяющий соединить вершины a_q графа Alyz с cros(wa_q,ws_g) = 1 в той же последовательности, что и вершины s_g в графе Sig. В случае существования подобного пути принимается решение, что код вредоносен.

СПИСОК ЛИТЕРАТУРЫ:

1. Blended Attacks and Web 2.0 Threats: Are You Ready for 2009? URL: http://securitylabs.websense.com/content/Alerts/3277.aspx.
2. Ask MAMA what the Web is. URL: http://www.opera.com/press/releases/2008/10/15/.
3. *Вояковская Н. Н., Москаль А. Е., Булычев Д. Ю., Терехов А. А.* Анализ потока управления. URL: http://www.intuit.ru/department/sa/compilersdev/12/.
4. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. АН СССР. 1965. Т. 163. № 4. С. 845—848.
5. *Кнут Э. Д.* Искусство программирования. Том 1. Основные алгоритмы. М.: Издательский дом «Вильямс», 2000. — 832 с.

В. В. Филатов

СИСТЕМЫ УПРАВЛЕНИЯ СОБЫТИЯМИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

В условиях сложившейся ситуации в области защиты информации проблема сбора информации о событиях становится наиболее актуальной. В организациях участились инциденты информационной безопасности, связанные с несовершенством систем защиты информации, используемых для защиты автоматизированных систем.

В крупных организациях используются сложные информационные системы, состоящие из большого числа сетевых устройств, операционных систем, антивирусного программного