Ю. М. Туманов

ОБНАРУЖЕНИЕ ВРЕДОНОСНЫХ СЦЕНАРИЕВ JAVASCRIPT НА ОСНОВЕ ПОВЕДЕНЧЕСКИХ СИГНАТУР

Интернет построен на протоколе передачи данных HTTP, который позволяет передавать интернет-браузерам пользователя посредством сети Интернет информацию в формате HTML текст страницы. Одним из этапов 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 связи с вышеизложенным остро встает проблема детектирования вредоносных JavaScript скриптов в контексте интернет-страниц, представленных в свободный доступ в сети Интернет.

Сигнатурный анализ широко используется в антивирусных программах — является простейшим вариантом детектирования вредоносного кода. На данный момент этот подход наиболее распространен и исследован [3].

Поведенческий анализ на сегодня является более перспективным направлением исследований. Он предоставляет возможность детектировать вредоносный код не только на основании его свертки и сравнения с сигнатурой, но и на основании математических алгоритмов, позволяющих описать поведение программы, проанализировать воздействие программы на ее окружение [4].

Представление программы и поведенческой сигнатуры

Для того чтобы впоследствии было удобно обращаться к различным участкам кода программы, представляя их в виде взаимодействующих определенным образом блоков, было выбрано представление 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 — целостная часть 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 [5].

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

1. идентификатор — каждой конкретной вершине присваивается определенный идентификатор, позволяющий впоследствии обращаться к данным, описывающим рассматриваемую вершину;

2. список функций вершины — в данный список входят все функции, вызов которых произведен в ходе выполнения JavaScript инструкций, принадлежащих к данной вершине;

3. список наследных вершин — в данный список входят все 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:

1. состояние программы — функции программы, исполняемые в текущей вершине графа потока управления программы;

2. поведение программы — последовательность состояний, необходимых для перехода из одного состояния в другое.

Сама поведенческая сигнатура программы строится следующим образом:

1. в ходе выполнения каждой конкретной анализируемой программы происходит построение графа потока управления программы;

2. на основании построенного графа потока управления строится матрица достижимости состояний;

3. каждая вершина графа потока управления описывается на основании представленных выше характеристик;

4. определенная последовательность состояний помечается как сигнатура поведения.

Алгоритм детектирования вредоносного поведения

Пусть набор всех функций языка JavaScript описывается алфавитом Ω из ρ символов, ρ_t — t-й символ данного алфавита, являющийся определенной 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

$$\Psi : Sig \rightarrow Alyz, \quad (1)$$

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 $\Psi(s_j)$.

Тогда слова wa_q и ws_g определены как:

$$wa_q = \bigcup_{\substack{p(v)\in P,\\ v=\overline{1,k}}} p(v), \quad (2)$$

$$ws_g = \bigcup_{\substack{p(v)\in P,\\ v=\overline{1,l}}} p(v). \quad (3)$$

Пусть $\rho sydr$ — вероятность существования гомоморфизма Ψ графа Sig в граф Alyz.

Зададим $f(\rho sydr,m)$ как

$$f(\rho sydr,m) = \lfloor (1 - psydr)\, m \rfloor \quad (4)$$

$cros(wa_q,ws_g)$ — 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 [6]. Тогда пусть

$$cros(wa_q,ws_g) = DLD(wa_q \cap ws_g). \quad (5)$$

В случае, если существует $f(\rho sydr,m)$ 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» [7] 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. *Касперски К.* Компьютерные вирусы изнутри и снаружи. СПб.: Питер, 2006. — 527 с.

4. *Касперски К.* Записки исследователя компьютерных вирусов. СПб.: Питер, 2006. — 316 с.
5. *Вояковская Н. Н., Москаль А. Е., Булычев Д. Ю., Терехов А. А.* Анализ потока управления. URL: http://www.intuit.ru/department/sa/compilersdev/12/.
6. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады АН СССР. 1965. Т. 163. Вып. 4. С. 845—848.
7. *Кнут Э. Д.* Искусство программирования. Том 1. Основные алгоритмы. М.: Издательский дом «Вильямс», 2000. — 832 с.