ЗАЩИТА ОТ НЕСАНКЦИОНИРОВАННОГО ИСПОЛЬЗОВАНИЯ ИНТЕГРИРУЕМЫХ HTML5 ПРИЛОЖЕНИЙ МЕТОДОМ ОБФУСКАЦИИ JAVASCRIPT КОДА

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Бартошик М. А.

Ярмолик В. Н. – д-р. техн. наук, профессор

В настоящее время широкое развитие получили технологии создания кросс-платформенных веб-приложений с использованием HTML5, CSS и JavaScript[1]. Обратное проектирование исходного кода позволяет анализировать и модифицировать механизмы защиты приложений с целью их несанкционированного использования, что указывает на необходимость разработки технических методов защиты веб-приложений от взлома[2].

С развитием мобильных технологий большую популярность получили магазины приложений: Apple App Store, Windows Marketplace, Android Market. В связи с этим имеет смысл рассматривать идею создания магазина HTML5 приложений, которые можно интегрировать (встроить) в веб-сайт организации-покупателя.

Самым простым способом интеграции является предоставление организации небольшого фрагмента кода, отвечающего за загрузку и запуск соответствующего приложения. Такой подход не требует модификации серверного кода и снижает издержки на интеграцию до минимума. Аналогичный способ используется многими медиа-сервисами: YouTube, Vimeo и пр. Однако данный способ предполагает бесплатное и свободное использование интегрируемого приложения и не обладает техническими средствами защиты от несанкционированного использования.

В работе предполагается предоставление приложений для использования в домене организации-покупателя. Организации предоставляется код интеграции на языке JavaScript, содержащий URL приложения с данными об идентификаторе приложения, идентификаторе организации, домене организации, защищенными цифровой подписью. Цифровая подпись используется для обеспечения целостности URL приложения и позволяет избегать подмены параметров. Подпись формируется согласно стандарту OAuth (RFC5849). Используемые алгоритмы – SHA-1 и RSA (RFC3447).

Каждое приложение содержит в себе код авторизации, также имеющий информацию о том, в каком домене это приложение может работать. Процесс авторизации основан на использовании возможности передачи сообщений между веб-страницами с различных доменов (функции postMessage и addEventListener). Задачей кода авторизации является проверка того, что интеграционный код находится в правильном домене.

Так как JavaScript является динамическим языком, исполняемым на стороне клиента, любой человек имеет возможность просмотреть код авторизации, разобраться в алгоритме его работы и попытаться обойти защитные механизмы. Современные браузеры позволяют подменить стандартную реализацию функций postMessage и addEventListener. Для защиты от подобной атаки был разработан кросс-браузерный метод определения подмены данных функций. Это позволяет избежать несанкционированного использования приложения при условии невозможности модификации кода авторизации.

Однако у злоумышленника имеется возможность использования прокси-сервера, задачей которого будет удаление либо модификация кода авторизации в автоматическом режиме. Для защиты от подобного рода атак был предложен метод, заключающийся в использовании случайных запутывающих преобразований (обфускации) кода авторизации при каждом запросе приложения.

Для искусственного увеличения объема кода авторизации и затруднения процесса обратного проектирования применятся синтаксическая (запутывания потока управления) и лексическая (модификация имен идентификаторов и строк) обфускация[3]. Формально процесс синтаксической обфускации можно представить как:

$$P \xrightarrow{T_{\text{синт.}}} P' \tag{1}$$

где P - исходный граф программы, P' - преобразованный граф программы, $T_{\text{синт.}}$- множество синтаксических преобразований.

Множество синтаксических преобразований можно представить как:

$$T_{\text{синт.}} \in \{S, C, E\} \tag{2}$$

где S - добавление селективных выражений, C - добавление циклов, E - использование динамического выполнения кода. Добавление селективных выражений представляет собой запутывание исходного кода с использованием дополнительных блоков ветвления потока управления (конструкции *if-else*, *switch-case*). Добавление циклов также вносит избыточную сложность в исходный код (например, расчет числовых констант в цикле). Использование динамического выполнения кода подразумевает использование функций *eval*, *setTimeout*, *setInterval* и конструктора *Function*.

Для усложнения восприятия кода авторизации и его автоматического анализа производится лексическая обфускация. Формально данный процесс может быть представлен в следующем виде:

$$\{V, S\} \xrightarrow{T_{\text{лекс.}}} \{V', S'\} \tag{3}$$

где V - множество идентификаторов программы, S - множество строковых констант, используемых в программе, V' - множество преобразованных идентификаторов программы, S' - множество преобразованных строковых констант, $T_{\text{лекс.}}$ - множество лексических преобразований.

Множество лексических преобразований представлено следующим образом:

$$T_{\text{лекс.}} \in \{E, R, I\} \quad (4)$$

где E - использование различных кодировок, R - использование регулярных выражений, I - использование индексаторов для сборки строк. Использование различных кодировок позволяет именовать идентификаторы случайным образом и представлять их различными символами (ASCII, UNICODE, HEX, ОСТ). Использование регулярных выражений позволяет получать строки необычным и неочевидным образом. Использование индексаторов позволяет конструировать строки с использованием встроенных в JavaScript возможностей, например:

```
_ = ({}+[])[1];
```

Результатом выполнения данного кода является символ “o”, который можно использовать для конструирования строк. Преобразование строк является необходимым для защиты приложения, так как разрешенный домен и токены доступа хранятся в строковом виде.

Вид преобразований и их комбинация обновляется случайным образом с периодом $T_{\text{обн.}}$. Вместе с обновлением способа обфускации приложению выдается временные токены доступа к серверу данных (с периодом действия $T_{\text{обн.}}$), необходимых для работы приложения. Таким образом, потратив на взлом приложения время $T_{\text{взл.}}$, злоумышленник сможет пользоваться приложением лишь ограниченное время, равное $T_{\text{обн.}} - T_{\text{взл.}}$. Соответственно, метод защиты можно считать надежным при выполнении условия:

$$T_{\text{обн.}} \leq T_{\text{взл.}} \quad (5)$$

В общем случае, использование преобразований (1) и (3) не гарантирует полную защиту от несанкционированного использования, так как обфускация – это способ обеспечения безопасности через неясность. Однако благодаря использованию временных токенов доступа и постоянному обновлению запутывающих преобразований можно добиться выполнения условия (5) и, соответственно, защиты приложения от взлома. Таким образом, разработанный способ технической защиты позволяет избежать несанкционированного использования интегрируемых приложений.

Список использованных источников:

1. Building Cross-Platform Apps with HTML5 [Electronic resource]. – 2013. – Mode of access: http://software.intel.com/en-us/articles/building-cross-platform-apps-with-html5
2. Chikofsky, E.J.; J.H. Cross II (January 1990). "Reverse Engineering and Design Recovery: A Taxonomy in IEEE Software". IEEE Computer Society: 13–17.
3. Heiderich M. Web application obfuscation / M.Heiderich. – Syngress, 2007.