Московский Государственный Университет имени М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Автоматизации Систем Вычислительных Комплексов

Чиботару Виктор Дорианович

Исследование способов эксплуатации недостатков веб-приложений, связанных с возможностью изменения области видимости переменных объектной модели веб-страницы

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
младший научный сотрудник
А.А. Петухов

Москва, 2016

Аннотация

В данной работе рассматривается метод автоматического анализа веб-приложений с целью выявления способов эксплуатации недостатков типа DOM Clobbering, связанных с возможностью изменения области видимости переменных объектной модели веб-страницы. Идея метода состоит в отслеживании и дальнейшем анализе потоков данных в коде веб-страницы, исполняемого на стороне клиента. В работе приводится обзор существующих методов анализа кода, написанного на языке JavaScript и предлагается использование метода тейнтирования (taint analysis). Предложенные идеи реализованы в виде дополнения к браузеру Mozilla Firefox. Для проверки корректности его работы было проведено тестирование как на синтетических наборах тестов, так и на реальных веб-приложениях.

Оглавление

Оглавление **1**

1 Введение **2**

2 Постановка задачи **4**
2.1 Неформальная постановка задачи 4
2.2 Терминология предметной области 6
2.3 Формальное определение DOM Clobbering 6
2.4 Постановка задачи 6

3 Анализ задачи **7**
3.1 Исследование возможных направлений атаки 7
Вызов ошибки в программе 7
Обход критических состояний 7
Выполнение произвольного кода 8

1 Введение

В период зарождения всемирной паутины веб-страницы представляли собой статические документы, сверстанные на языке HMTL. Но, с течением времени, страницы становились все более и более динамическими. Сегодняшние приложения обычно содержат большое количество кода, исполняемого на стороне клиента (в веб-браузере). Стоит отметить, что несмотря на существование конкурентов, язык JavaScript используется в подавляющем большинстве веб-приложений для написания кода для клиентских частей. Об этом факте свидетельствует бурный рост популярности данного языка и технологий, которые его используют (например, NodeJS, AngularJS, AJAX и т. д.)

Активное использование языка JavaScript привело к появлению новой парадигмы создания веб-приложений, называемой Single Page Application (Одностраничное приложение). Используя такие приложения, пользователь все время находится в рамках одной веб-страницы, в коде которой и реализована большая часть бизнес логики. Всё общение между клиентом и веб-сервером осуществляется с помощью AJAX запросов без необходимости перезагрузки или перехода на другие страницы.

Высокая популярность Single Page Application наглядно демонстрирует тенденцию к всё большему усложнению кода клиентской части. Однако, хорошо известно, что рост сложности системы часто ведёт к снижению её безопасности. В случае веб-приложений этот факт доказывается большим количеством недостатков на стороне клиента. Вот самые опасные и распространенные из них:

1. DOM Based XSS
2. DOM Redirection
3. Некорректное использование механизмов Same Origin Policy
4. DOM Clobbering

Самым распространенным из них является недостаток DOM Based XSS. В рейтинге недостатков веб-приложений OWASP (Open Web Application Security Project) Top Ten она занимает третье место. Он заключается в том, что код веб-страницы обрабатывает пользовательские данные и модифицирует её содержимое, позволяя злоумышленнику исполнять произвольные команды. Этот тип недостатков достаточно хорошо исследован и для него были разработаны эффективные автоматические средства обнаружения. В виду некоторых особенностей языка JavaScript (слабая типизация, возможность динамического исполнения кода) статические методы анализа кода оказываются непригодными для решения подобных проблем. Поэтому, разработанные решения опираются на такие методы, как динамический тейнт анализ и фаззинг.

Уязвимость, рассматриваемая в данной работе носит название DOM Clobbering (от англ. DOM - Document Object Model, Объектная Модель Документа и Clobber - перезаписывать). Суть DOM Clobbering заключается в возможности подмены объектов (переменных) веб-страницы, с помощью изменения их области видимости. Как и DOM-based XSS, DOM Clobbering - это недостаток целиком на стороне клиента. Стоит отметить, что в реальных сайтах DOM Clobbering встречается редко,

что и является причиной её недостаточной изученности. Однако, высокая степень опасности, которую таят в себе недостатки данного типа, указывает на необходимость разработки автоматизированного средства их поиска.

2 Постановка задачи

2.1. Неформальная постановка задачи

Рассмотрим на небольшом примере причины возникновения недостатка DOM Clobbering.

Пример 1:

```
<form name="form_name">
<script>
  var form = document.form_name; // указывает на <form name="
  form_name">
</script>
```

Для представления содержимого веб-страницы в виде объектов на языке JavaScript веб-браузеры используют технологию DOM (Объектная Модель Документа). В рамках DOM HTML-странице ставится в соответствие объект document, а окну веб-браузера - объект window. После загрузки и обработки страницы эти объекты заполняются различным свойствами (например, document.cookie - объект, представляющий идентификатор сессии пользователя).

Так же дело обстоит и с тэгами страницы: после создания HTML элемента он становится доступен по указателям document.element_name, window.element_name (верно для элементов одного из типов <img>, <form>, <embed>, <object> и <applet>) и window.element_id (верно для всех элементов), где element_name - это имя (атрибут name) созданного элемента, а element_id - это идентификатор элемента (атрибут id). Однако, в случае, если document.element_name или window.element_name указывали на какой-нибудь другой объект, указатель на этот объект заменяется указателем на HTML элемент (неверно для window.element_id). В этом заключается вся суть эксплуатации недостатка DOM Clobbering: злоумышленник подбирает имя HTML элемента таким образом, чтобы заменить какой-нибудь важный объект в объектной модели документа страницы.

Рассмотрим пару типичных примеров недостатка DOM Clobbering.

Пример 2:

```
<form name“=”querySelector>
<script>
  var element = document.querySelector“(”a); // ошибка: document.
  querySelector указывает на <form name“=”querySelector
</script>
```

В данном примере пользователь контролирует атрибут name (имя) у формы на странице и изменяет его на “querySelector”. Далее, в коде вызывается вызывается функция document.querySelector, но, так как document.querySelector теперь указывает на HTML форму, которая не является функцией, при таком вызове произойдет ошибка. То есть, действуя подобным образом, злоумышленник может добиться нарушения работоспособности кода на стороне клиента.

Пример 3:

```
<form name“=”security_flag>
<script>
  if (securityCheck()) {
    security_flag = true;
  }
  if (security_flag) {
    // критически важная с точки зрения безопасности секция кода
  }
</script>
```

В данном примере пользователь контролирует атрибут name (имя) у формы на странице и изменяет его на “security_flag”. Далее вызывается функция securityCheck, суть которой заключается, например, в проверке обладает ли пользователь достаточными правами для выполнения критически важной секции кода. Если пользователь проходит проверку, то глобальной переменной security_flag присваивается значение true, иначе она остается неинициализированной. Далее, в зависимости от значения переменной security_flag выполняется или не выполняется некоторый критически важный код. Но, так как пользователь изменил имя формы на “security_flag”, то во втором условном операторе security_flag будет указывать на HTML форму, следовательно, критический код будет выполнен даже если пользователь не прошел проверку securityCheck. Заметим, что данный пример является несколько вырожденным, но он ясно дает понять, что в некоторых случаях злоумышленники могут обойти логику работы кода на стороне клиента с помощью недостатка DOM Clobbering.

Пример 4:

```
<a href“=plugins/preview/preview.html#<svg onload=alert(1)>” id“=
  ”_cke_htmlToLoad target“=”_blank>
  Click me!
</a>

файл /plugins/preview/preview.html:
<script>
  ...
  document.write(window.opener._cke_htmlToLoad);
  ...
</script>
```

Данный пример взят из работы реального веб-приложения. На одной из его страниц размещалась ссылка (элемент <a>), с идентификатором равным “_cke_htmlToLoad”, указывающая на страницу “plugins/preview/preview.html#<svg onload=alert(1)>”. Далее, после того, как пользователь переходил по этой ссылке, на странице plugins/preview/preview.html отрабатывал код, записывающий строку window.opener._cke_htmlToLoad в конец веб-страницы. Но, так как window.opener указывал на ту страницу, с которой был осуществлен переход на текущую, а в ней _cke_htmlToLoad указывал на контролируемый элемент <a>, в конец документа записывалась строка “plugins/preview/preview.html#<svg onload=alert(1)>”. А дописывание в документ строки <svg onload=alert(1)> означало создание HTML элемента типа <svg>, при завершении загрузки которого выполнялся код alert(1). Таким образом, злоумышленник получал возможность внедрять и исполнять произвольный код на стороне клиента, что может привести к плохим последствиям.

2.2. Терминология предметной области

Введем некоторые определения, используемые далее:

- **DOM** - Document Object Model или Объектная Модель Документа - универсальный интерфейс для представления и взаимодействия с HTML, XHTML и XML документами.
- **Уязвимость** - недостаток веб-приложения, позволяющий злоумышленникам намеренно нарушить его целостность и вызвать его неправильную работу.
- **Клиентская сторона веб-приложения** - исполнимый код, содержащийся в веб-страницах, получаемых от веб-сервера.
- Сюда буду еще дописывать по мере работы.

2.3. Формальное определение DOM Clobbering

Используя введенные выше термины дадим формальное определение DOM Clobbering: DOM Clobbering - это недостаток клиентской стороны веб-приложения, заключающийся в подмене объектов (переменных) веб-страницы с помощью изменения имен и/или идентификаторов некоторых HTML элементов на веб-странице.

2.4. Постановка задачи

Сформулировать методику и разработать инструментальное средство для определения возможности (и последствий) подмены объектов DOM для заданной веб-страницы.

3 Анализ задачи

3.1. Исследование возможных направлений атаки

После исследования реальных примеров веб-страниц, обладающих недостатков DOM Clobbering, авторами были выявлены три возможных направления атаки.

Вызов ошибки в программе

Во-первых, злоумышленник может вызвать возникновение ошибки в коде клиентской стороны. Самый простой пример - это замена указателя на стандартную функцию, предоставляемую интерфейсом браузера, указателем на HTML элемент (пример 2 из главы Постановка задачи). Последствиями такой атаки является некорректная дальнейшая работа веб-приложения.

Обход критических состояний

Для описания второго направления, проанализируем пример 3 из главы Постановка задачи:

```
<form name“=”security_flag>
<script>
  if (securityCheck()) { // точка 1
    security_flag = true; // точка 2
  }
  if (security_flag) { // точка 3
    criticalFunction(); точка// 4
  }
</script>
```

Предположим, что функция securityCheck() осуществляет посылку запроса на веб-сервер с целью выяснения прав текущего пользователя на выполнение функции criticalFunction(). Будем считать, что данную проверку могут пройти только определенные пользователи (назовем их администраторами), и что злоумышленник не принадлежит к их числу.

Введем пару определений:

- *Состояние программы в точке* i - это тройка $< i, Vars, Values >$, где i - точка программы, $Vars$ - множество переменных, доступных в i, $Values$ - множество двоек $< var, value >$, где $var \in Vars$, а $value$ - значение var в i.
- *Критическое состояние* - это такое состояние, в которое может перейти страница только при условии выполнения определенных требований на серверной стороне приложения.
- *Некритическое состояние* - это состояние, не являющееся критическим.
- *Допустимое для точки* i *состояние* - это состояние $< i, Vars, Values >$, в котором программа может оказаться.

Например, состояние $< 4, security\_flag, < security\_flag, true > >$ является критическим, так как для того, чтобы программа перешла в него, необходима принадлежность пользователя к группе администраторов.

С другой стороны, состояние $< 4, security\_flag, < security\_flag, HTMLForm > >$ также является допустимым для точки 4, но не является критическим.

Другими словами, для точки 4 существуют два допустимых состояния, одно из которых дает нарушителю возможность совершения действий, на которые у него не хватает прав. Очевидно, такая ситуация является угрозой безопасности приложения.

С помощью введенных выше определений второе направление атаки можно описать так: *С помощью недостатка DOM Clobbering нарушителям удается перевести программу в некритическое состояние в некоторой точке, для которой все прочие допустимые состояния являются критическими*

Выполнение произвольного кода

В-третьих, злоумышленники могут добиться исполнения произвольного кода в веб-браузере жертвы. Реальным примером такой атаки является пример 4 из главы Постановка задачи. Стоит отметить, что в данном случае методы эксплуатации идентичны методам эксплуатации недостатка DOM Based XSS. Данная атака является самой опасной из приведенных трех, так как она позволяет нарушителям осуществлять любые действия от имени жертвы.