Обнаружение уязвимостей в механизме авторизации веб-приложений

Андрей Петухов, Георгий Носеевич
Лаборатория вычислительных комплексов
Факультет ВМиК МГУ имени М. В. Ломоносова

Мотивация

По данным статистики WASC за 2008 год:

- Insufficient authorization:
  - В автоматическом режиме найдена в 0.13% сайтов
  - В ручном режиме найдена в 14.82% сайтов
  - Разница – в 100 раз!
- XSS
  - В автоматическом режиме найдена в 37.66% сайтов
  - В ручном режиме найдена в 56.41% сайтов
- SQLI
  - В автоматическом режиме найдена в 11.65% сайтов
  - В ручном режиме найдена в 16.16% сайтов
- Коррелирует со статистикой WhiteHat

В чем же трудность?

- Механизм авторизации выполняет задачу по защите данных и/или функциональности веб-приложения от несанкционированного доступа
- В идеале:
  - понятие несанкционированного доступа определяется явно
  - описывается в политике разграничения доступа
  - входит в состав исходных требований к веб-приложению
  - понятно, как формализовать уязвимость механизма авторизации
- В реальности:
  - политика разграничения доступа отсутствует
  - механизм авторизации реализуется на основе здравого смысла
  - как формализовать уязвимость механизма авторизации - вопрос

Рабочее предположение

- Веб-интерфейс является неявным источником информации о правилах разграничения доступа
- Уязвимостью контроля доступа считается ситуация, когда пользователь может успешно произвести HTTP-запрос, не предусмотренный интерфейсом веб-приложения
- Аргументы в пользу адекватности предположения:
  - корректность пользовательского интерфейса проверяется на этапе функционального тестирования и этапе сдачи-приемки
  - можно ожидать, что все ссылки/формы, которые «не должны быть» в интерфейсе того или иного пользователя, будут убраны

Пример уязвимости

- Приложение банк-клиент
- Имеет страницу «детализация транзакций по счету»
- В пользовательском интерфейсе переход на страницу реализован через ссылки вида: /TransactionDetails.aspx?account_no=5204320422040001
- Подменив значение параметра account_no, получаем информацию по счетам других пользователей. Налицо горизонтальная эскалация привилегий
- Заметим, что пример удовлетворяет определению, данному ранее: пользователи могут сделать HTTP-запросы, не предусмотренные их интерфейсом

Существующий метод

Единственный известный метод для проверки механизма авторизации без политики разграничения доступа – «differential analysis»

- Пусть даны два пользователя: user1 и user2
- Построим «карту ссылок» веб-приложения, видимых пользователю user1 – Sitemap1, и карту ссылок, видимых пользователю user2 – Sitemap2
- Проверим возможность доступа:
  - к ресурсам Sitemap1\Sitemap2 от имени пользователя user2
  - к ресурсам Sitemap2\Sitemap1 от имени пользователя user1
- Если доступ успешен, то сообщаем о наличии уязвимости:
  - горизонтальная эскалация привилегий, если роли одинаковые
  - вертикальная эскалация привилегий, если роли разные

Ограничение существующего метода

- «Карта ссылок» – динамическая сущность, которая зависит от состояния веб-приложения, Sitemap(state)
- В сложных приложениях не существует такого состояния state, что из интерфейса Sitemap(state) можно было бы выполнить любой сценарий использования
- Для обеспечения полноты анализа необходимо:
  - найти последовательность состояний $\{state_i\}$ и переходов между ними такую, что для любого сценария использования найдется интерфейс Sitemap($state_j$), из которого он может быть запущен
  - уметь строить карту ссылок для каждого состояния приложения
- После применения «differential analysis» для каждого состояния $\{state_i\}$ анализ будет полным

Предлагаемый метод (1 из 2)

- Найти последовательность состояний $\{state_i\}$ и переходов между ними такую, что для любого сценария использования найдется интерфейс Sitemap($state_j$), из которого он может быть запущен
- Идея решения:
  - построить граф зависимостей между сценариями использования
  - линеаризовать граф в последовательность состояний и переходов между ними
  - интерфейсные элементы, реализующие переходы между состояниями, должны использоваться только после построения карты ссылок в текущем состоянии

Предлагаемый метод (2 из 2)

- Построить карту ссылок для заданного состояния приложения
- Как автоматически определять во время crawling'а, какие интерфейсные элементы изменяют состояние веб-приложения, а какие – нет?
  - Статический анализ
  - Ручная разметка
- Мы выбрали ручную разметку

Автоматизация метода

- Подготовка разметки
  - Оператор осуществляет навигацию по веб-приложению при помощи браузера FF с установленным расширением
  - Оператор помечает: ссылки и формы, которые изменяют состояние веб-приложения, формы аутентификации, ссылки logout, формы с анти-автоматизацией (CAPTCHA)
  - Оператор может разбить работу с приложением на сценарии использования и указать зависимости между ними
- Проведение «differential analysis» с использованием полученной разметки
  - Web-crawler строит карту ссылок для текущего состояния
  - Анализатор использует полученную карту ссылок для проведения «differential analysis»
  - Осуществляется переход к очередному состоянию

Community

- PoC-реализация метода была сделана во время OWASP Summer of Code 2008
  - PoC-реализация доступна здесь: http://code.google.com/p/accorute/downloads/list
- Следующую версию инструмента планируется представить этим летом на конференции OWASP AppSec Research 2010 в Стокгольме
  - Обо всех новостях будет сообщаться в рассылке owasp-access-control-rules-tester-project@lists.owasp.org
  - На рассылку можно подписаться на странице проекта

Контактная информация

- Андрей Петухов: petand@lvk.cs.msu.su
- Георгий Носеевич: ngo@lvk.cs.msu.su
- Тел. +7 (495) 939 46 71
- Москва, 119899 Ленинские горы вл. 1/52,
  факультет ВМК МГУ имени М. В. Ломоносова, к. 764

Спасибо за внимание!

Вопросы?