Практические аспекты сетевой безопасности

Уязвимости в веб-приложениях

Часть I: Введение

Making things clear

- Недостатки (weakness) и уязвимости
  - переполнение буфера в ls – не уязвимость!
    - http://en.wikipedia.org/wiki/Vulnerability_(computing)
- Атака: <цель, уязвимость, метод>
  - цели: технические и нетехнические
    - технологический шпионаж, причинение вреда репутации
    - повышение привилегий, нарушение работоспособности
  - уязвимость –> техническая цель атаки – арность 1 к N
  - нетехническая цель –> метод атаки – арность 1 к N
  - уязвимости client-side и уязвимости server-side
  - атаки на клиента и атаки на сервер
  - софистика: XSS в интерфейсе администратора

Уязвимости в ПО

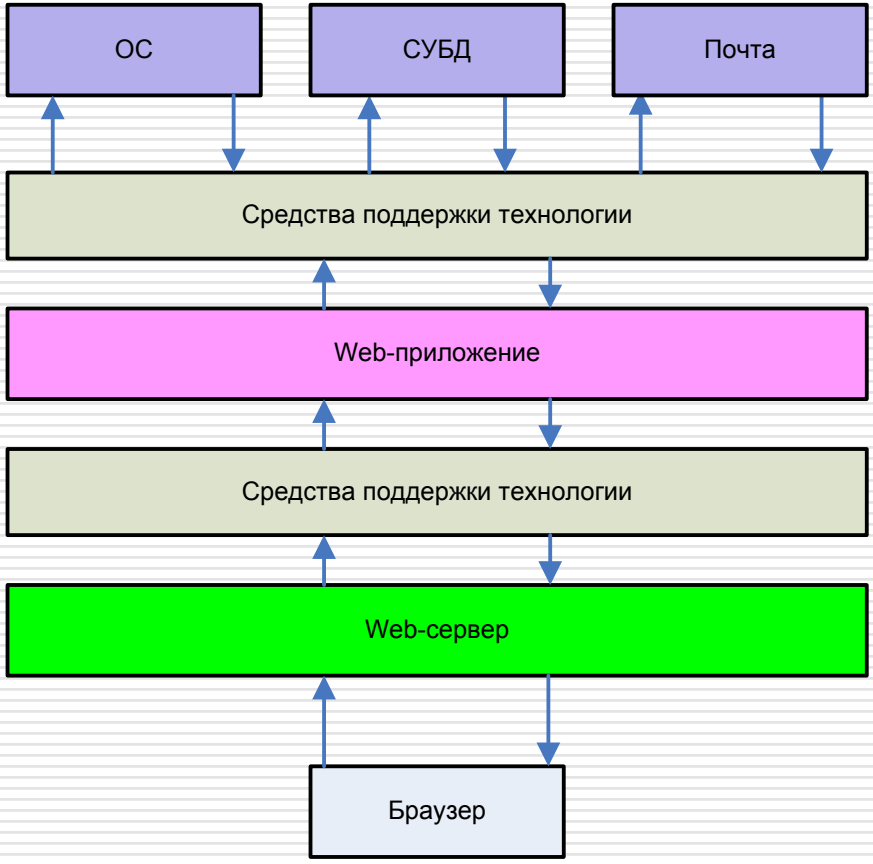
- Уязвимость – свойство ПО
  - => адаптируем существующие подходы к анализу ПО
  - надо различать методы обнаружения и методы эксплуатации уязвимостей
- Классификация подходов к анализу свойств ПО
  - по возможностям оператора-аналитика
    - доступ только к внешним интерфейсам ПО
    - доступ к внутреннему представлению ПО
  - по состоянию объекта анализа
    - анализ во время выполнения
    - анализ без выполнения
  - статический анализ, динамический анализ и тестирование черного ящика

Типичные причины уязвимостей

- Некорректная обработка входных данных
  - полное доверие
    - заголовки, cookies, скрытые поля и т.п.
  - проверка на стороне клиента
  - проверка на основе черных списков
  - ошибки при реализации белых списков
    - нормализация данных (например, кривой HTML)
- Отсутствие или некорректная реализация механизма безопасности:
  - аутентификация, авторизация, безопасный транспорт
- Небезопасные настройки
  - в т.ч. пароли по умолчанию

Некорректная обработка входных данных

Схема работы модуля типичного веб-приложения



Взаимодействие с внешними компонентами

- Веб-приложение взаимодействует с внешними подсистемами: СУБД, LDAP, интерпретатор ОС, почтовый демон, браузер, файловая система и т.п.
- У большинства подсистем свой язык: SQL, Shell/CMD, SMTP, HTML и Javascript у браузера и т.п.
- Обращения веб-приложения во внешние подсистемы параметризуются
  - есть константная часть
  - есть переменная часть, значение которой формируется динамически с участием пользовательских данных
  - обработка запроса http://newsfeed.us.to/news.php?id=17 может содержать оператор вида

    mysql_query(“SELECT * FROM news WHERE id = ”.$id);

Некорректная обработка входных данных

- В каждом языке определены ключевые слова, разделители, правила оформления секций данных:
  - SELECT * FROM news WHERE author = '**petand**'
  - View first
  - grep -R "**search string**" *
- В каждом запросе или команде к внешней подсистеме есть контекст команд и есть контекст данных
- Injection-атаки: внешний пользователь может выйти из контекста данных в контекст команд:
  - SELECT * FROM news WHERE author = '**petand' or sleep(5) --** '
  - <a href="/show?id="**1"><script>alert(1)</script>View first
  - grep -R "**search string" 1; echo "p0wned" #**" *

Примеры Injection-атак

- **SQL-операторы**
  - данные не обрабатываются при построении SQL-запроса
- **HTML-разметку**
  - данные не обрабатываются при построении HTML
- **Javascript-операторы**
  - данные не обрабатываются при построении JSON-объекта
- **CSS-директивы**
  - данные не обрабатываются при построении CSS-правил
- **HTTP-заголовки**
  - данные не обрабатываются при построении ответного заголовка (например, Location)
- **XPath-операторы**
  - данные не обрабатываются при построении XPath-запроса

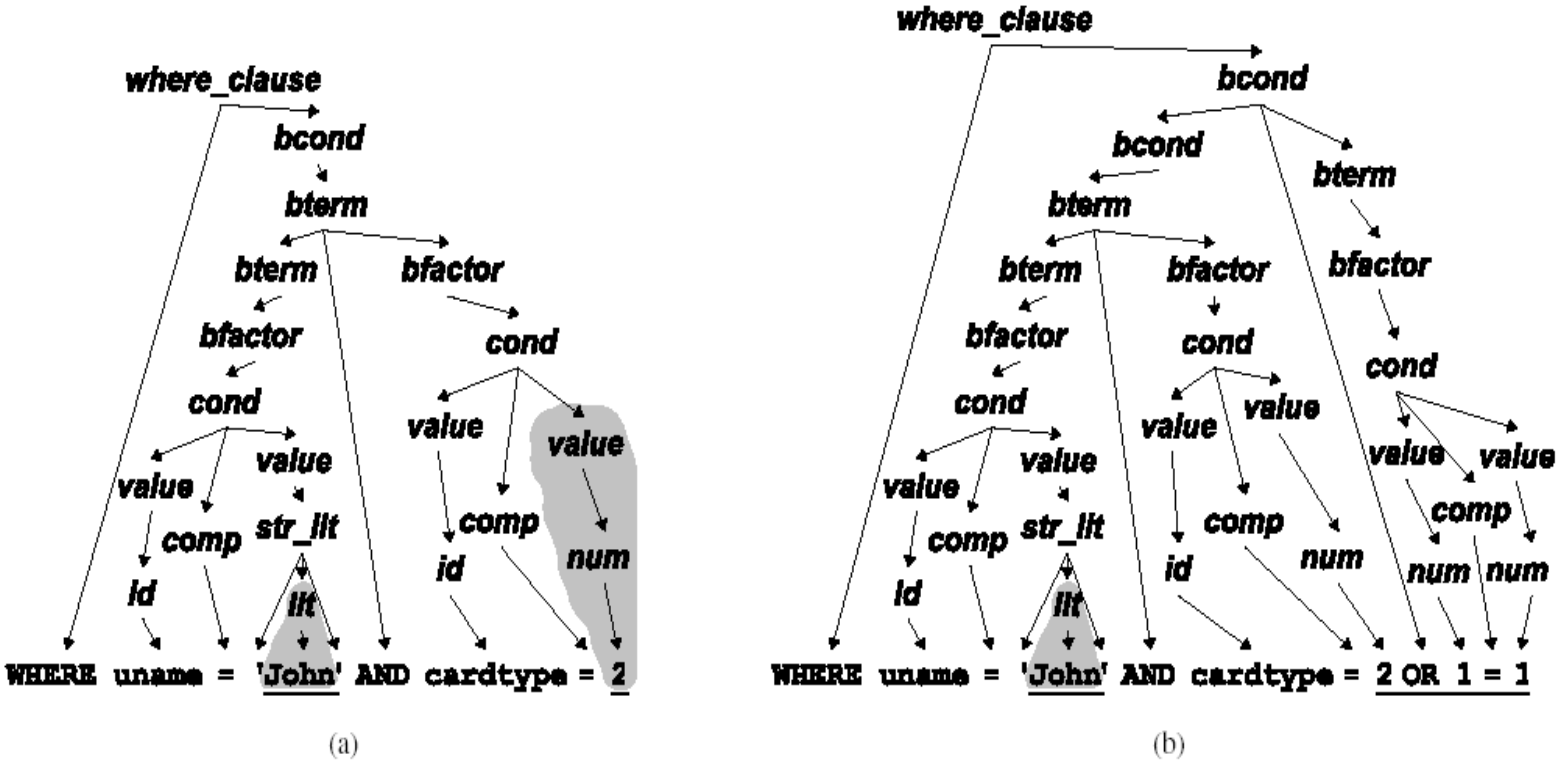
SQL injection

- Терминология
  - SQL injection – это название *атаки*
  - недостаток: возможность внедрения операторов SQL
  - причина: некорректная обработка входных данных
- Определение
  - пользователь может изменить структуру SQL-запроса
- Возможные последствия
  - нарушение конфиденциальности/целостности данных
  - нарушение доступности (DROP DATABASE …)
  - повышение привилегий на сервере
  - обход аутентификации

Наглядный пример

SELECT * FROM DB WHERE uname=‘John’ AND cardtype=2

SELECT * FROM DB WHERE uname=‘John’ AND cardtype=2 OR 1



Корректная обработка пользовательских данных

- Проверки на клиенте должны дублироваться на сервере
- Любые данные, контролируемые клиентом, являются потенциально опасными
- Валидация. Единственно верным подходом к обработке входных данных является подход “accept known good”
  - black-listing’у отказать!!!!
  - в частном случае – приведение к типу (дата, число, boolean)
- Валидация входных данных относительно белого списка не всегда может гарантировать безопасность
  - входные данные после валидации могут содержать служебные символы и разделители языка служебного компонента
  - => необходимо осуществлять экранирование или кодирование
  - например, O’Neal в SQL-операторе должен стать O\’Neal
  - (a + b) < c в HTML должно быть (a + b) < c

Поиск уязвимостей

Тестирование черного ящика

- Что мы можем делать?
  - получать «нормальные» HTTP-запросы
    - переходить по ссылкам, заполнять формы и т.п.
  - манипулировать полученными HTTP-запросами
    - проигрывать их as-is, меняя порядок, от лица другого user’а
    - изменять методы и заголовки запросов; изменять URL
    - удалять и добавлять параметры, дублировать параметры
    - модифицировать значения параметров
- Что доступно для анализа?
  - код и заголовки HTTP-ответа
  - тело ответа
  - время ответа

Статический анализ

- Программа не выполняется, просматриваются все пути
- Варианты представлений:
  - текст
  - лексемы
  - дерево синтаксического разбора (AST)
  - граф потоков управления (CFG)
  - графы зависимостей (DFG/PDG/SDG)
  - множества состояний программы (см. абстрактная интерпретация)
  - множества возможных значений переменных в точках программы

Статический анализ

- Можно поискать наличие в представлении фрагмента, соответствующего заданному шаблону
  - “сигнатурный” метод
  - варианты: grep, регулярные выражения над потоком лексем, регулярные выражения над AST, поиск подграфа в графах зависимостей (taint-анализ)
  - так делают все известные статические анализаторы
- Построить гипотезу спецификаций, неявно предполагаемых разработчиками, и проверить соответствие ПО им

Динамический анализ

- Анализ программы во время ее выполнения
- Варианты данных для анализа:
  - переменные программы
  - системные вызовы
  - вызовы различных API
  - изменение окружения
- Может быть:
  - частью среды выполнения (модифицируется интерпретатор)
  - частью окружения (системный монитор)
  - частью программы (инструментирование)

Модель невмешательства

- Программа получает данные из high-level (доверенных) и low-level (недоверенных) каналов
- Программа генерирует вывод в high-level (критические) и low-level (обычные) каналы
- Вывод low-level (недоверенных) данных в high-level (критический) канал считается нарушением принципа невмешательства

Реализация в виде taint-анализа

- Данные, полученные из HTTP-запросов, считаются ненадежными; это low-level каналы ввода
- Данные, полученные из локального хранилища данных (файловая система, СУБД), считаются надежными; это high-level каналы ввода
- Ненадежные данные могут стать надежными в следствие специальной обработки (валидации, кодирования, экранирования)
- Ненадежные данные не должны попадать в критические операции (запросы к СУБД, вывод HTML, системные вызовы, eval'ы и т.д.)
- Taint-анализ можно проводить статически или динамически
  - важно понимать ограничения такого анализа; это самостоятельно

Вопросы?