NANYANG TECHNOLOGICAL UNIVERSITY

Lecture 7: From Meta-Characters to Injection Attacks

presented by

Li Yi
Assistant Professor
SCSE

N4-02b-64
yi_li@ntu.edu.sg

Change of Scene

- So far, we have examined vulnerabilities in the operating system
- The problems have not been 'solved', but there has been much change to the better (DEP, ASLR, …)
  - Known problems keep appearing in other areas
  - "IoT devices are programmed by people without a proper software engineering background"
- Attacks have moved up to the application layer
- We will follow and move to 'application layer' vulnerabilities in browsers and web servers

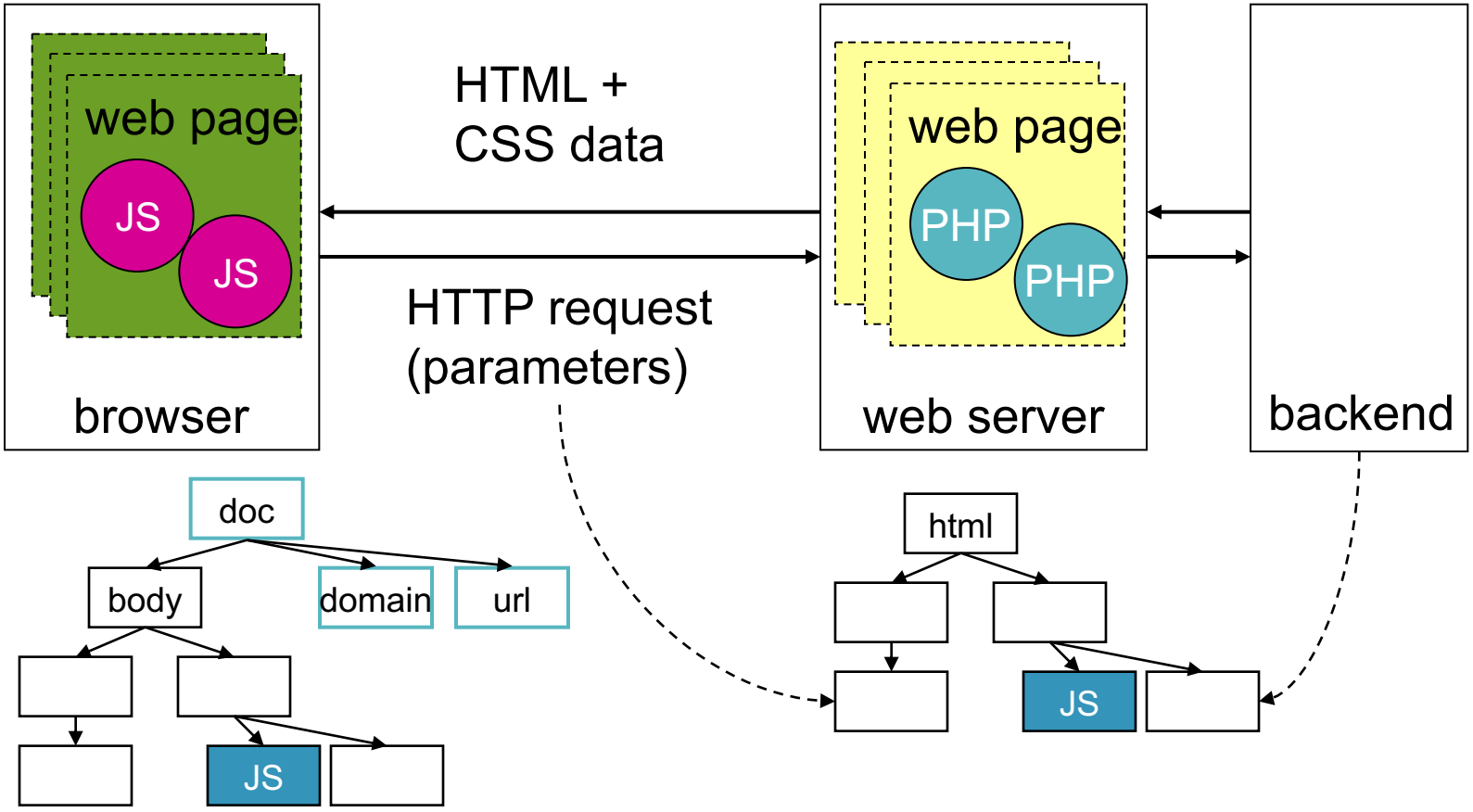
Agenda

- Dynamic web pages
- Meta-characters
- Character encodings
- XML External Entity attacks
- HTTP Parameter Pollution attacks
- Conclusions

Web Applications – Brief Overview

- At the client-side, interaction with a web application is handled by the browser
- At the server-side, a web server receives the client requests
- Scripts at web server extract input from client data, may construct queries to a backend server, e.g. a database server
- Web server receives query result from backend server; returns HTML result pages to client

Dynamic Web Pages



Dynamic Web Pages

- Server-side scripts build response page using request parameters and data from backend servers as inputs
  - Slide 5 uses PHP as a shorthand for server-side scripts
- Client-side scripts can manipulate the DOM in the browser using data in the page received and other DOM objects as inputs
  - Slide 5 uses JS as a shorthand for client-side scripts

Transport Protocol

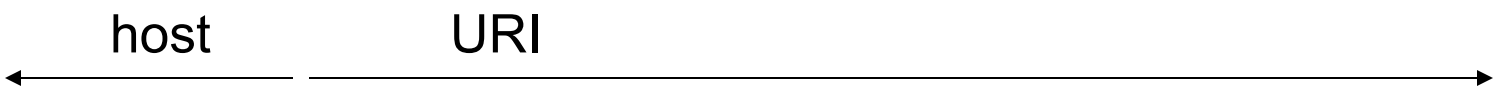
- Transport protocol used between client and server: HTTP (hypertext transfer protocol)
  - HTTP/1.1 specified in RFC 2616
- HTTP located in the application layer of the Internet protocol stack
- Client sends HTTP requests to server
- A request states a method to be performed on a resource held at the server

GET & POST Methods

- GET method retrieves information from a server
  - Resource given by Request-URI (Uniform Resource Identifier) and Host fields in the request header
- POST method specifies the resource in the Request-URI and puts the action to be performed on it into the body of the HTTP request
  - POST intended for posting messages, annotating resources, and sending large data volumes that would not fit into the Request-URI
- In principle POST can be used for any action that can be requested by GET; side effects may differ

URI

- Parsing URI and Host:



www.wiley.com/WileyCDA/Section/id-302475.html?query=computer\%20security

- Attack:
  - Create host name containing a character similar to a slash
  - A user looking at the browser bar may take the string to the left of this character as the host name
  - Actual delimiter used by the browser is too far out to the right to be seen by the user

Defences

- Block "semantically dangerous" characters
  - Characters that can be easily confused with other characters
- Display to the user where the browser splits host name from URI
  - Render (parts of) the host name in bold
  - Aligns the user's view with the browser's view

HTML

- Server sends HTTP responses to the client
- Web pages in a response are written in HTML (HyperText Markup Language)
- Elements that may appear in a web page include frame (subwindow), iframe (in-lined subwindow), img (embedded image), script, form, div, …
- Form: interactive element specifying action to be performed on a resource when triggered by a particular event; onclick is such an event
- Cascading Style Sheets (CSS) for giving further information on how to display a web page

Web Browser

- Functions performed by client browser:
  - Displays web pages: may use the Document Object Model (DOM) as an internal representation of a web page; required by JavaScript
  - Manages sessions
  - Access control when executing scripts in a web page; sandbox, origin-based security policies
  - Reference monitor in the browser is today more relevant than the reference monitor in the operating system

Document Object Model (DOM)

- When the browser receives an HTML page it parses the HTML into the document.body of the DOM
- Origin of a web page (presumably the host it has been loaded from) is stored in document.domain
- Elements in a web page may have origins different from the parent page
  - It depends on the type of an element whether it is loaded in its own context or in the context of its parent page
- Objects like document.URL, document.location, and document.referrer get their values according to the browser's view of the current page

Web Page

- For the user: what you see on the display
  - Accepts user input: text fields, mouse clicks
- For the browser: what is stored in the DOM
  - Actions triggered by user input
  - Page may contain code (JavaScript, Java)
- For the web server: what is stored at the URI
  - May refer to code (PHP) that accepts browser input, builds queries for backend server, and constructs response page

Meta-Characters

Meta-Characters

- Characters with special meanings
- Examples:
  - String terminators
  - End-of-line terminators
  - Command-line terminators
  - Separators for elements in a list
  - Characters for creating name-value pairs
  - Characters for composing commands
  - Characters for traversing directory trees
  - Characters for constructing XML tags

Meta-Characters

- Meta-characters differ between network protocols
- Meta-characters differ between mark-up languages
- Meta-characters differ between programming languages
- Meta-characters are security relevant
  - Meta-characters included in user input can cause unexpected behavior

Escaping

Escape Characters

- Special meta-characters that change the meaning of the characters that follow
- Used when meta-characters should be interpreted "at face value", i.e. not in their special meaning
  - Several programming languages, e.g. PHP, use backslash (\) to neutralize the effect of certain meta-characters
  - In XML and HTML, & (ampersand) declares the beginning of an entity reference
  - Windows command-line interpreter uses ^ (caret) as the escape character

HTML Entities

- HTML has five predefined character entity references used to escape sensitive meta-characters
  - & for & (ampersand, U0026)
  - < for < (less-than sign, U003C)
  - > for > (greater-than sign, U003E)
  - " for " (quotation mark, U0022)
  - ' for ' (apostrophe, U0027)
- Other character entity references can be defined

Escaping in PHP

- **`addslashes`** function adds backslashes (\) to a string in front of single quote ('), double quote ("), backslash (\), and NULL
  - **`addslashes`** changes *Isn't this nice* into *Isn\'t this nice*
  - Applied to all GET, POST, and COOKIE data by default; not to be used on strings that have already been escaped
- **`mysql_real_escape_string()`** escapes the characters **`\x00`**, **`\n`**, **`\r`**, **`\`**, **`'`**, **`"`**, **`\x1a`** ("Ctrl+Z")

http://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string

Interaction Between Layers

- GBK: character set for Simplified Chinese
  - Has one-byte and two-byte characters
- **0xbf27** is not a valid GBK multi-byte character; interpreted as two single-byte characters: **¿'**
  - Note: **0x27** is the single quote; **0x5c** is the slash
- Add a slash in front of the single quote: **0xbf5c27** 縗'
- Valid two-byte character **0xbf5c** followed by a single quote; single quote has survived unguarded!
- Lesson: Danger of abstraction – manipulation at lower layer does not have the desired effect

Using Meta-Characters in Exploits

Programming Language

- Famous typo in a FORTRAN program

  DO 20 I = 1. 100 (instead of DO 20 I = 1, 100)

- Intended as a loop, but gets parsed as assignment of value 1.1 to an implicitly declared variable DO20I
  - Space characters are skipped when parsing variable names
  - Dot after digit interpreted as decimal dot
  - Comma after digit would be a separator between two digits
- Urban myth (don't trust your inputs!) blames this typo for the loss of the Mariner 1 Venus space probe

Unix *rlogin* – Combining Commands

- Unix *login* command:
  - **`login [[-p] [-h<host>] [[-f]<user>]`**
  - **`-f`** option 'forces' log in: user is not asked for password
- Unix *rlogin* command for remote login:
  - **`rlogin [-l<user>] <machine>`**
  - *rlogin* daemon sends a login request for <*user*> to <*machine*>

Unix *rlogin* – Combining Commands

- Attack (some versions of Linux, AIX):
  - `% rlogin -l -froot <machine>`
- Result: forced login as root at designated machine
  - `% login -froot <machine>`
- Cause: user input to one command interpreted as control data by another command
  - **`-f`** has a special meaning for **`login`** but not for **`rlogin`**

End-of-line: CRLF Injection

- CRLF (carriage return-line feed, `\r\n`, `%0d%0a`) is a common end-of-line indicator
- Attack: insert CRLF in input that expects a single line
- E.g., log file where entries are separated by CRLF
  - Entries are recorded with timestamp and sender identity
  - Attacker's entry contains CRLF, followed by fake entries with fake timestamps and spoofed identities

HTTP: CRLF Injection Attack

- HTTP header is list of "key: value" pairs, each terminated by CRLF
- Take a script that constructs a HTTP redirect from input $url by creating the pair "Location: $url%0d%0a"
- Malign value for $url: http://www.redirect-to.org/%0d%0aSet-Cookie:Authenticated=yes%0d%0aReferer: www.spoofed.org
  - Malign input pretends to be re-direct from an authenticated session with website www.spoofed.org

Directory Traversal

- Fragment of a script index.cgi:

```
&ReadParse(*input);
$filename = $input{page};
$filename = "usr/local/apache/htdocs/" .
  $filename;
While (<FILE>){print $_;}
Close(FILE);
```

- Input: `index.cgi?page=/../../../../../etc/passwd`
- Result?

The Attack

- `../` traverses a directory tree one step upwards
- Intention of the script: only provide information about entries in directory `usr/local/apache/htdocs/`
- Attack: use `../` several times to step up in the directory tree to the root; print the password file
- Countermeasure: input validation, filter out `../`

  (as you will see in a moment, life isn't that easy)
- Don't trust your inputs

Character Encodings

Filtering

- You have seen some examples of attacks where meta-characters are inserted in user input
  - More examples are too follow in this course
- Defence: don't trust your inputs; filter your inputs
- You need to know about all relevant meta-characters and about their encodings
  - In the following, hexadecimal characters in C/C++ code are indicated by 0x, hexadecimal characters in URLs by %

UTF-8 Encoding

- UTF-8 encoding of the Unicode character defined for using Unicode on systems designed for ASCII
- The encoding:
  - U000000 - U00007F: 0xxxxxxx
  - U000080 - U0007FF: 110xxxxx 10xxxxxx
  - U000800 - U00FFFF: 1110xxxx 10xxxxxx 10xxxxxx
  - U010000 - U10FFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
- ASCII characters (U0000–U007F) represented by ASCII bytes (0x00–0x7F)
- All non-ASCII characters represented by sequences of non-ASCII bytes (0x80–0xF7)

UTF-8 Encoding [RFC 2279]

- The xxx bits are the least-significant bits of the binary representation of the Unicode number
- For example, U00A9 = 1010 1001 (copyright sign) is encoded in UTF-8 as

  11000010 10101001 = 0xC2 0xA9

- Only the shortest possible UTF-8 sequence is valid for any Unicode character, but many UTF-8 decoders also accept longer variants

Multiple Representations

- Multi-byte UTF-8 formats: a character gets more than one representation
- Example: slash "/"

	format	binary	hex
• 1 byte	0xxx xxxx	0010 1111	2F
• 2 byte	110x xxxx	1100 0000	C0
	10xx xxxx	1010 1111	AF
• 3 byte	1110 xxxx	1110 0000	E0
	10xx xxxx	1000 0000	80
	10xx xxxx	1010 1111	AF

Exploit (historic) – 'Unicode Bug'

- Vulnerability in IIS; URL starting with {address}/scripts/..%c0%af../winnt/system32/
- Translated to directory C:\winnt\system32
  - Because %c0%af is the 2 byte UTF-8 encoding of /
  - ..%c0%af../ becomes ../../
  - ../../ steps up two levels in the directory
  - The /scripts/ directory is usually C:\inetpub\scripts
- IIS did not filter illegal Unicode representations of single byte characters by multi-byte UTF-8 formats

Double Decode

- Consider URL starting with {addr.}/scripts/..%25%32%66../winnt/system32/
- This URL is decoded to {addr.}/scripts/..%2f../winnt/system32/
  - Convert %25%32%66 to ASCII:

    00100101 00110010 01100110 → %2f
- If the URL is decoded a second time, it gets translated to directory C:\winnt\system32
- Characters change meaning as they are processed

Lessons

- Unicode attacks show how an attacker might disguise dangerous inputs even if developers try to have defences in their code
- Canonicalization is a useful first step before filtering when a character has multiple representations
- Meta-characters are of particular interest for attackers and defenders
- Parsing strings is a major software security challenge

Limits of Filtering

- You must know all relevant meta-characters
  - Differ between protocols, programming languages, data formats, products
- You must know about all character encodings an application accepts
  - Google was hit by an attack using UTF-7 encoding in 2005
- You must know how data are processed after filtering
  - Safe values may get converted to unsafe values!
  - Helpful application replaces non-ASCII characters by similar ASCII characters, e.g. 〈 (U304F) or ‹ (U2039) with <

HTTP Parameter Pollution Attacks

Background

- Do you know where your inputs come from?
- In web applications, the browser may send inputs to the server as URI parameters in a GET request
- Values for predefined parameters may be entered by the user in a web form
- Don't trust your inputs!
- The user may provide more inputs than asked for

URI Parameters – Meta-characters

- URI parameters: field=value pairs, separated by : or &
- Meta-characters are url-encoded using %:
  - '=' as %3D
  - '&' as %26
  - ':' as %3A
  - '%' as %25
- To experiment with URL-encoding, go e.g. to https://www.urlencoder.org/

HTTP Parameter Pollution

- When a value is taken from user input, a malicious user may provide more parameters than intended
  - Use meta-characters in user input to inject more parameters
  - How will a server script react to inputs it doesn't expect?
- Case 1: server-side script handling the request has set a parameter to a 'secure' default value; attack sends a new value for that parameter
  - Will the script update the parameter to the more recent value?
  - Will the script block attempts to update the parameter?

HTTP Parameter Pollution

- Case 2: attack provides a value for a parameter the server-side script handling the request has not yet set
  - Will this parameter be ignored by the script?
  - Does the logic of the script distinguish between the case where the parameter has been set and where it has not?

http://blog.mindedsecurity.com/2009/05/client-side-http-parameter-pollution.html

Case Study: Webmail

- HTTP GET request to get first page of a user's inbox in the Yahoo! Classic Mail webmail system:
  - showFolder?fid=Inbox&order=down&tt=245&pSize=25&startMid=0
  - The precise meanings of these fields do not matter for us
- PHP stores the (url-decoded) parameters from GET requests in the $_GET superglobals array
- Authenticator stored in .rand parameter by browser
  - Included by the browser in all requests sent to the mail server

Deleting Emails

- Deleted emails are moved into a trash folder
- To completely delete email, delete it from trash folder
- Goal of attack: erase the victim's inbox
  - The attack will require two steps to reach its goal
- Attacker crafts special value for startMid parameter:
  - 0%2526cmd=fmgt.emptytrash%26DEL=1%26DelFID=Inbox%26cmd=fmgt.delete
  - %25 is the url-encoding of %
  - %26 is the url-encoding of &

First GET Request

- Server receives URI

  showFolder?fid=Inbox&order=down&tt=245&pSize=25&startMid=0%2526cmd=fmgt.emptytrash%26DEL=1%26DelFID=Inbox%26cmd=fmgt.delete&.rand=1076957714

  from victim; PHP passes GET parameters through urldecode(); the value of startMid is decoded to

  0%26cmd%3Dfmgt.emptytrash&DEL=1&DelFID=Inbox&cmd=fmgt.delete

- Value of startMid now contains URI meta-character '&' and the server sees three more GET parameters

Dynamically Created Response Page

- These three new parameters instruct the server to delete all entries in the inbox:
  - DEL=1, DelFID=Inbox, cmd= fmgt.delete
- If request comes with a valid authenticator, all emails from folder Inbox are moved into the trash folder
  - User would only notice once view on the inbox is refreshed
- Server script next copies parameters submitted in the request into links in the response page, in particular the new value of startMid

Attack – Second Step

- Link to show a message might predefine the value for startMid; authenticator .rand included at client side

  showMessage?sort=date&order=down&startMid=0%26cmd%3Dfmgt.emptytrash&DEL=1&DelFID=Inbox&cmd=fmgt.delete&.rand=1076957714

- When the user clicks on this link, the server accepts the request as authenticated, decodes

  0%26cmd%3Dfmgt.emptytrash to 0&cmd=fmgt.emptytrash

  and empties the trash folder

Spoofing Authentication

- How can the victim's browser be made to submit the requests with parameters polluted by the attacker but with the genuine authenticator attached?

Attack Flows

- Option #1:
  - Attacker sends email to victim with the first URL in a link
  - Victim clicks on link, gets inbox page, inbox is emptied
  - Victim clicks a link to open a message, trash folder is emptied
- Option #2:
  - Lure victim to attacker's page
  - Attacker checks if user is logged in on webmail, then redirects victim to first URL, inbox page sent to victim, inbox is emptied
  - Victim clicks to open a message, trash folder is emptied

Countermeasures – Encoding

- Urlencode parameters taken from HTTP requests; undoes automatic urldecoding; no additional fields can be inserted

```
<a href="/?startmid=”
<?=urlencode($_GET['startMid'])?> &id=4"> View
</a>
```

- Mistake: translate request parameters to HTML entities; fields added by attacker will survive encoding; do not use

```
<a href="/?startmid="
<?=htmlspecialchars($_GET['startMid'])?> &id=4">View
</a>
```

- This attack inserts HTTP parameters, not HTML elements!

Remarks

- HTTP parameter pollution is a parameter injection attack, not a code injection attack
  - The fact that some parameters in the web mail example are function names is accidental
- Client-side defences can stop users from inserting malign inputs via the browser
- They are insufficient as an attacker could modify parameters directly in the request that is being sent
- You still need server-side defences for dealing with malign parameters

Conclusions

Summary – Meta-Characters

- Meta-characters are control data
  - Meta-characters in user input can change the logic of an application
- What constitutes a meta-character depends on the application that processes the user input
  - E.g., which character terminates a filename?
  - What is the extension of foo.txt .exe?
  - Meta-characters are protocol and product specific!

Escaping & Encoding

- To secure an application, remove dangerous meta-characters from user input before further processing
- Escaping encodes meta-characters in a way so that they are no longer interpreted as control data
- To spot all meta-characters in user input, you have to know all character encodings you are accepting
- You have to know which encoding to apply to defend against a given class of attacks

Postel's Law & Software Security

- "Be conservative in what you send; be liberal in what you accept" [Postel's Law]
  - Robustness principle that has contributed to the success of the internet
  - but makes life more difficult for software security
- Strategies for dealing with accidental flaws may open the door for attackers with intent