```
                          _             _                                 _
  /\/\   __ _ ___ ___    /_\  ___ ___(_) __ _ _ __  _ __ ___   ___ _ __ | |_
 /    \ / _` / __/ __|  //_\\/ __/ __| |/ _` | '_ \| '_ ` _ \ / _ \ '_ \| __|
/ /\/\ \ (_| \__ \__ \ /  _  \__ \__ \ | (_| | | | | | | | | |  __/ | | | |_
\/    \/\__,_|___/___/ \_/ \_/___/___/_|\__, |_| |_|_| |_| |_|\___|_| |_|\__|
                                        |___/

   _   _   _             _
  /_\ | |_| |_ __ _  ___| | _____
 //_\\| __| __/ _` |/ __| |/ / __|
/  _  \ |_| || (_| | (__|   <\__ \
\_/ \_/\__|\__\__,_|\___|_|\_\___/
```

Overview

- Sometimes called an over-posting attack
- An attack where a malicious user sets a parameter on a particular model (object) that they weren’t supposed to be able to set
- Exploits the magic functions of MVC frameworks
- Subtle bug that isn’t often caught

ASP.NET MVC Example

- Say you have the following model:

```
public class User
{
    public string FirstName { get; set; }
    public bool IsAdmin { get; set; }
}
```

ASP.NET MVC Example

- You give the users the following form so they may update their names:

```
@using (Html.BeginForm()) {

     @Html.EditorFor(model => model.FirstName)
    <input type="submit" value="Save" />

}
```

How would you exploit this?

ASP.NET MVC Example

- What if you submitted this:
  - FirstName=Andrew&IsAdmin=true

- Well if you update a model blindly with the contents of the parameters of the request then it will set the IsAdmin property of the model

A Django Example

```
from django.shortcuts import render
from myapp.models import Whatzit

def create_whatzit(request):
    Whatzit.objects.create(**request.POST)
    return render(request, 'created.html')

def update_whatzit(request, id):
    whatzit = Whatzit.objects.filter(pk=id)
    whatzit.update(**request.POST)
    whatzit.save()
    return render(request, 'saved.html')
```

Where’s the vulnerable bit?

```
whatzit.update(**request.POST)
    whatzit.save()
```

A PHP Example

```php
<?php
class Model
{
    private $username = null;
    private $email = null;
    private $admin = false;

    /**
     * Populate the model with the given values
     * @param array $values Input data
     */
    public function values($values)
    {
        foreach($values as $name => $value)
        {
            if (property_exists(get_class($this), $name)) {
                $this->$name = $value;
            }
        }
    }

    /**
     * Save user to database
     */
    public function save() {}

    /**
     * Check to see if the user is an admin
     */
    public function isAdmin()
    {
        return ($this->admin == true) ? true : false;
    }
}
?>
```

Where’s the vulnerable bit?

```
foreach($values as $name => $value)
{
    if (property_exists(get_class($this), $name)) {
        $this->$name = $value;
    }
}
```

Real World Example

- A “researcher” added his public key to a Rails official organization user account
- Pushed a file to their repo to demonstrate the vulnerability
- “That’s why when Russian programmer Egor Homakov reported the “bug”, it was rejected. The argument was that this is behaving by design, and that while it was dangerous, it was prominently described in the ‘Ruby On Rails Security Guide’”
- “This gave Homakov a neat opportunity. He wanted the owners of the Ruby-on-Rails project to take him seriously, so what better way to do that than hack the very site hosting Ruby-on-Rails using the very bug he was talking about? He used the bug to hack into GitHub, and gave himself administrative control over the Ruby-on-Rails project. He used this hacked authority to then write a comment into the Ruby-on-Rails source code discussing the problem.”

Write-ups for Github Attack

- http://homakov.blogspot.com/2012/03/how-to.html
- https://gist.github.com/peternixey/1978249

Mitigation

- Whitelist properties that can be updated!
- Individually update each property with a separate update statement (basically whitelisting)

Mitigations

- ASP.NET
  - TryUpdateModel(user, includeProperties: new[] { "FirstName" });
  - [HttpPost]
    public ViewResult Edit([Bind(Include = "FirstName")] User user) {}
- Django

```
class WhatzitForm(forms.ModelForm):
    class Meta(object):
        model = Whatzit
        fields = ('foo', 'bar', 'baz’)
```

Resources

- http://odetocode.com/blogs/scott/archive/2012/03/12/complete-guide-to-mass-assignment-in-asp-net-mvc.aspx
- http://coffeeonthekeyboard.com/mass-assignment-security-part-10-855/
- http://websec.io/2013/02/04/Beware-the-Mass-Assignment.html
- http://www.sitepoint.com/rails-mass-assignment-issue-a-php-perspective/
- http://blog.mhartl.com/2008/09/21/mass-assignment-in-rails-applications/