The Code Behind The Vulnerability (v3)

Barry Dorrans
.NET Security Czar
https://idunno.org

Microsoft

Who am I

.NET Security “Czar”

Which means I annoy everyone about coding securely

And I get to deal with it when I miss things

I also ramble and swear on twitter in a non-work approved manner
@blowdart

What you can expect

7 MSRC bulletins and what lay behind them

And a few things we caught before they caused a bulletin.

Unicode and how to abuse it

Misusing encryption

Zip files and runaway processes

Naïve cert comparisons

XSS via URL validation

7 Tales of MSRC Cases

The Microsoft Bulletin Process.

- Vulnerability reported
- MSRC open case
- Product Group Security emailed
- Reproduction attempted
- Reproduction tested against all supported OS/software configurations
- Fix developed
- Fix tested
- Bulletin written
- Fix deployed internally
- Bulletin and fix released

Security Bulletins

Hash DoS (MS11-100 / CVE-2011-3141)

Padding Oracle (MS10-070 / CVE-2010-3332)

SharePoint ViewState RCE (MS13-067 / CVE-2013-1330)

Infinite Regex DoS (MS15-101 / CVE-2015-2526)

XSS in ASP.NET Core with Raw HTML (CVE-2018-0784)

Griefing via CSRF (CVE-2018-0785)

Malware via WSDL (CVE-2017-8759)

Hash DoS (CVE-2011-3141)

A Denial of Service caused by parsing FORM inputs.

Caused by HashTable and Dictionary using a predictable hash algorithm for string keys.

Fixed by switching hash functions to one that is unpredictable or limiting dictionary size .

Hash DoS explained

All form fields beginning with A would go into slot A.

To get a value back you go to the slot and look through everything.
The more A fields there are the longer it takes.

If you can force everything into a single slot then lookups will take more and more CPU which leads to DoS.

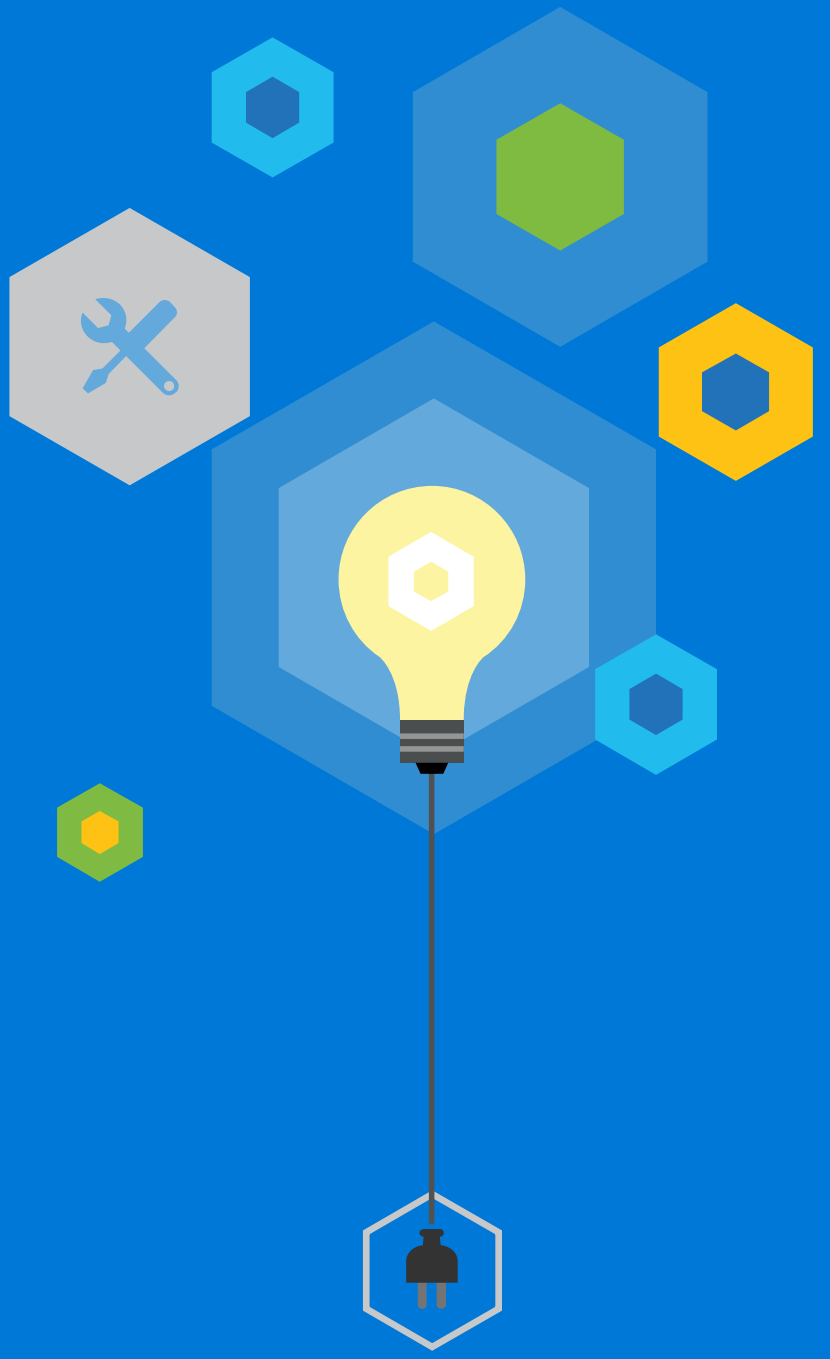
Originally presented at 28c8 (28th Chaos Communication Congress) 2011. Alexander “alech” Klink and Julian “zeri” Wälde exploited PHP, Java, Python, JavaScript.

DIY Hash DoS

Dictionary<TKey> fix only applies to string keys.

If you allow user input into a hash based collection which has a key which isn’t string and your GetHashCode() algorithm for the key class is predictable then you are vulnerable…

Hash DOS Demo



How can I fix this?

Don’t use user input as keys in dictionaries unless

The user input is a string OR

The HashCode for the input is strong and you implement a session key;
eg. SHA1HMAC with a random key set at runtime.

Padding Oracle (CVE-2010-3332)

Information disclosure via cryptographic attack.

An endpoint was exposed which validated if a string was encrypted correctly.

The attacker could use the yes or no result to build properly encrypted strings without knowing the key.

This then allowed forged requests to scriptresource.axd and serve up any file in the web application, like web.config, which contains machine keys, which you can forge auth tickets with.

Fixed by having scriptresource.axd validate signatures and serve JS only.

Padding Oracle Explained

Encrypted data is often padded to meet algorithm requirements.

Side channel attack can be used to discover the padding is correct.

This can then lead to fast decryption – you tweak a single byte and ask “Is this padding correct?”. From there you can calculate the original character.

Originally presented at EUROCRYPT 2002, Serge Vaudenay against CBC mode in SSL, IPSEC

Expanded to ASP.NET by Thai Duong, Juliano Rizzo at IEEE Symposium on Security and Privacy 2011.

How can I fix this?

Don’t expose padding oracles.

Add a authenticated signature to the encrypted data, and validate it.

SharePoint ViewState RCE (CVE-2013-1330)

Remote code execution via Serialization.

Made possible by SharePoint disabling ViewState MAC.

Fixed by turning ViewState MAC on.

Discovered by Alexandre Herzog, presented at Area41 in 2013.

Expanded by Alexandre to demonstrate RCE if machine key is known.

Expands on a remoting exploit discovered by James Forshaw and presented at BlackHat 2012.

RCE via ViewState

ViewState is, by default, signed.

If it is not signed, or the signing key is known you can forge ViewState to contain arbitrary objects.

ViewState is implicitly trusted and objects embedded in it get created when it’s parsed.

Deleting files with ViewState

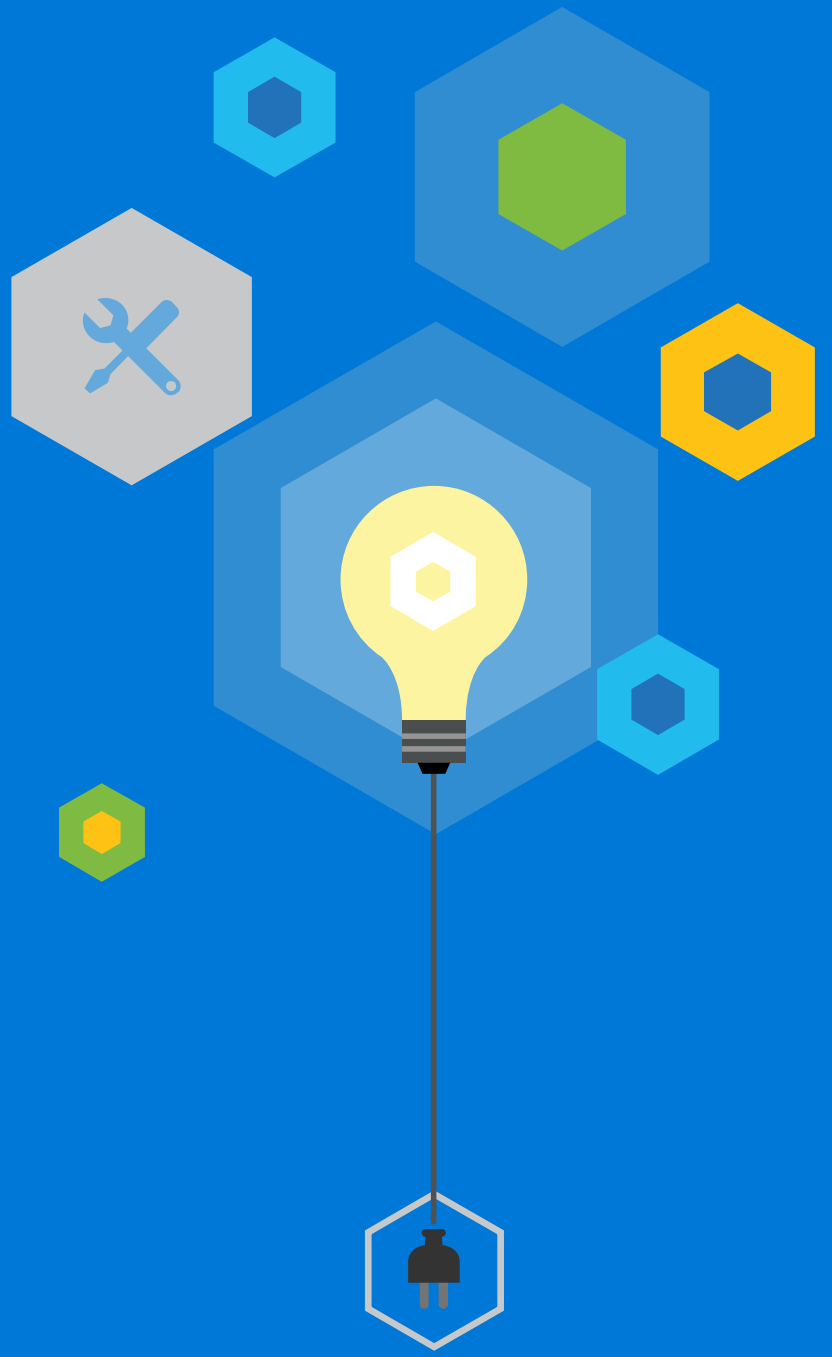
.NET contains a TempFileCollection class which is serializable.

Serializable classes can be stored in ViewState

The class cleans up files when the class is GCed.

You could set the file names in the collection from a serialized representation…

Viewstate / Serialization Demo



How can I fix this?

When you roundtrip state to a user controlled space sign the data.

Don’t publish your signing key.

Infinite Regex (CVE-2015-2526)

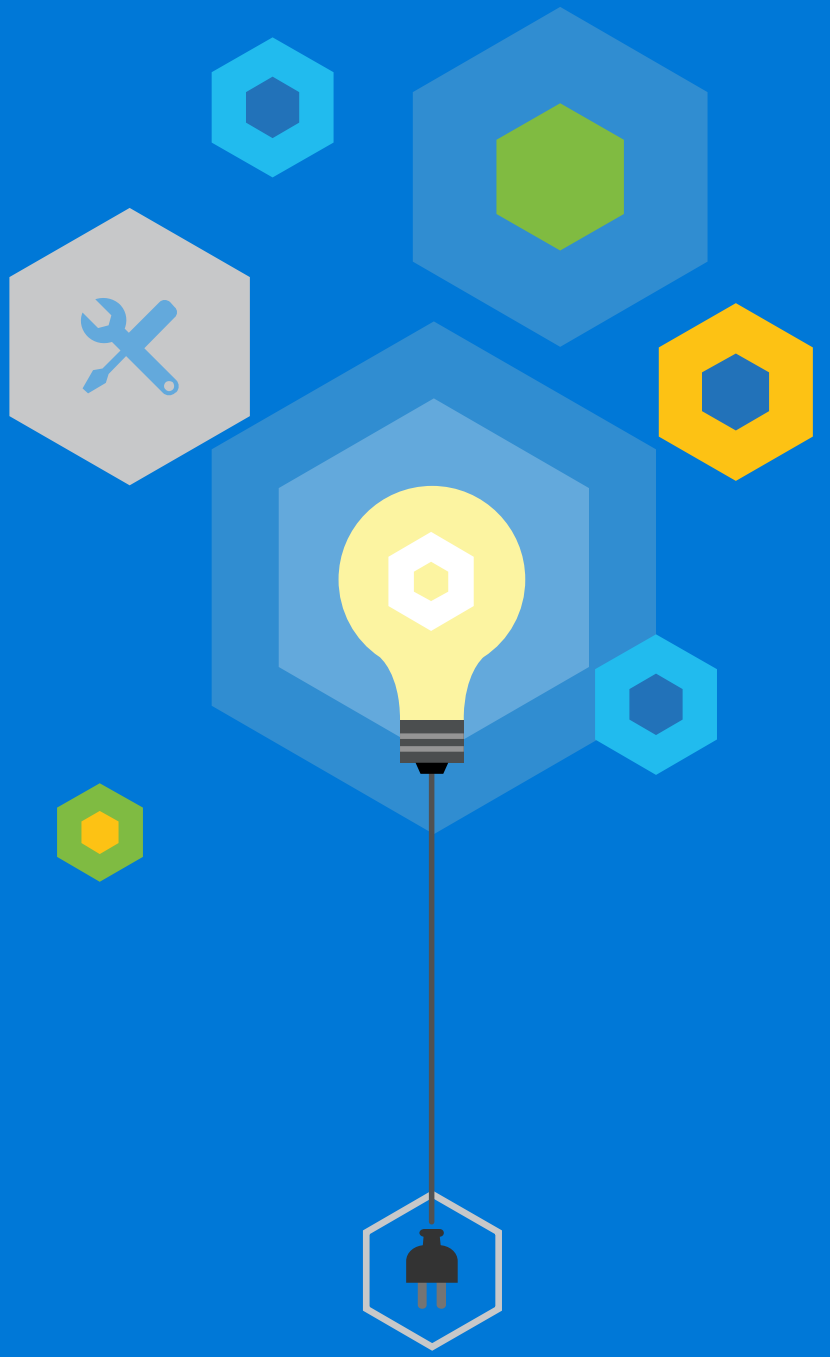
DoS against validators with crafted input.

Made possible due to a backtracking Regex and .NET not putting a timeout on regular expressions by default.

Fixed by adding timeouts.

Discovered by Roberto Suggi, reported in April 2015.

Infinite Regex Demo



How can I fix this?

Find a better way to validate than regular expressions.

Set timeouts on all regular expressions.

Set a global regex timeout.

```
AppDomain.CurrentDomain.SetData(
        "REGEX_DEFAULT_MATCH_TIMEOUT",
        TimeSpan.FromSeconds(1));
```

XSS in ASP.NET Core (CVE-2018-0784)

MVC encodes every piece of output by default.

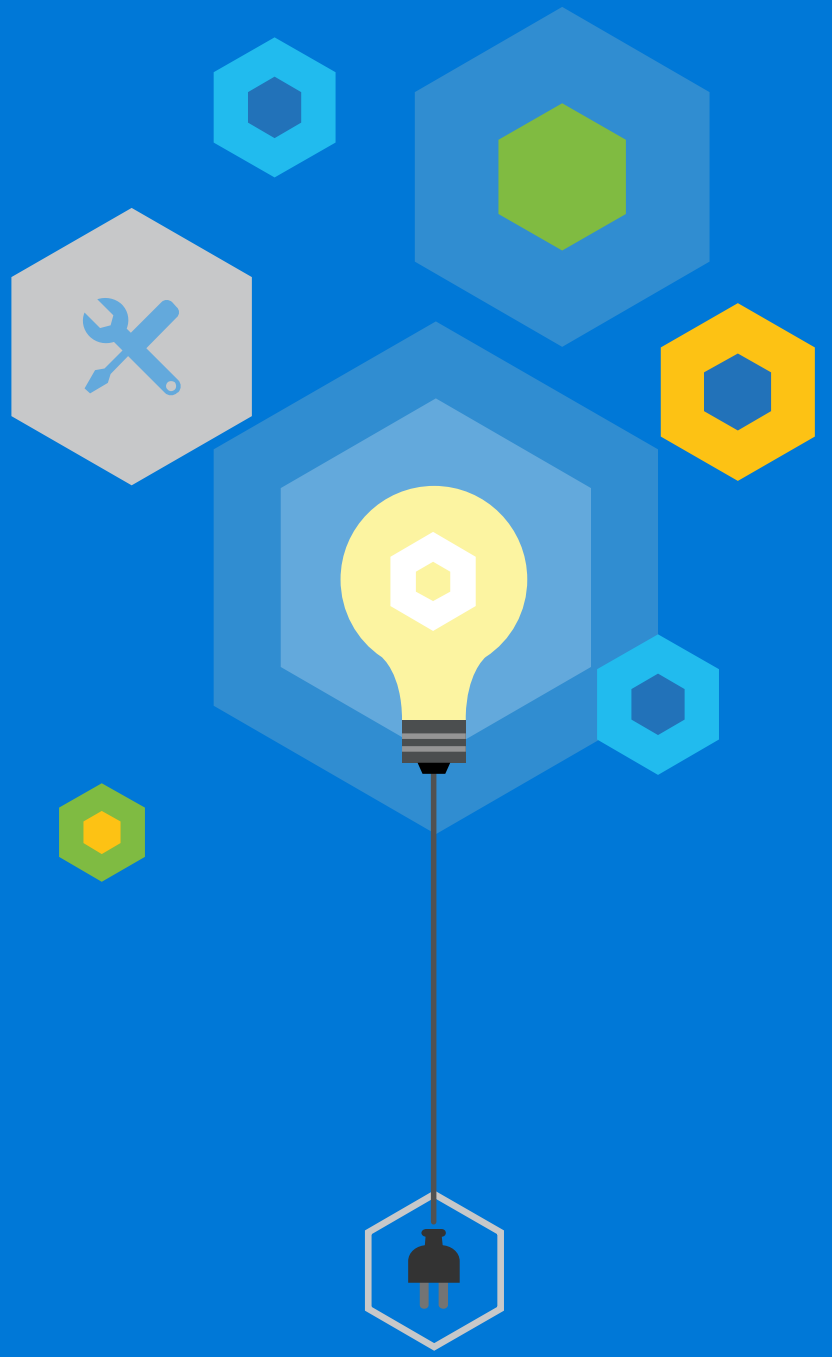
Except when you tell it not to.

And we told it not to.

Which would have been ok … but …

Discovered by Kévin Chalet, reported in 2017.

XSS via Overbinding Demo



How can I fix this?

Don’t use Html.Raw. Search your code for it.

For things should never come from user input apply [BindNever].

If something is generated server side, always generate server side, stop putting it in a hidden field to transport it between requests.

CSRF in ASP.NET Core (CVE-2018-0785)

CSRF allows another site to act on behalf of a logged in user.

ASP.NET Core MVC automatically adds tokens to all HTML forms to prevent this.

Which doesn’t help when it’s not a form.

Discovered by Kévin Chalet, reported in 2017.

How can I fix this?

GET should be idempotent.

(Resubmitting a GET shouldn’t change the results)

When you change state use a POST.

WSDL abuse (CVE-2017-8759)

Nation state attack.

WSDL is metadata for SOAP endpoints. Used to generate proxy objects.

Did you know .NET Framework has a WSDL parser which generates code?

Discovered by FireEye, reported in 2017.

WSDL abuse (CVE-2017-8759)

IsValidUrl() inside the parser had a bug.

This enabled a nation state targeting Russian speakers in another Eastern Europe country to compile code on their machine.

It was activated via Word's SOAP moniker functionality that starts off .NET with no user interaction, parses WSDL and then can connect to a SOAP endpoint.

How can I fix this?

Don’t be a country Russia wants to invade (or has partially invaded).

Don’t ever write code that generates code from user input, then compiles and runs it.

Mistakes we've seen

Bad encoding

What's wrong with?

```
string.Format(
    "onmouseover=\"DisplayTooltip('Issue: {0}');\"" +
    "onmouseout=\"DisplayTooltip('');\"",
     HttpUtility.HtmlEncode(d["Title"]);
```

Encoding is context specific and hard to get right once JavaScript is involved.

Better to set the value in a DIV HTML Encoded then pull out contents.

```
<div data-stuff="{@Title}" />
```

I İ Captain

What will the following print?

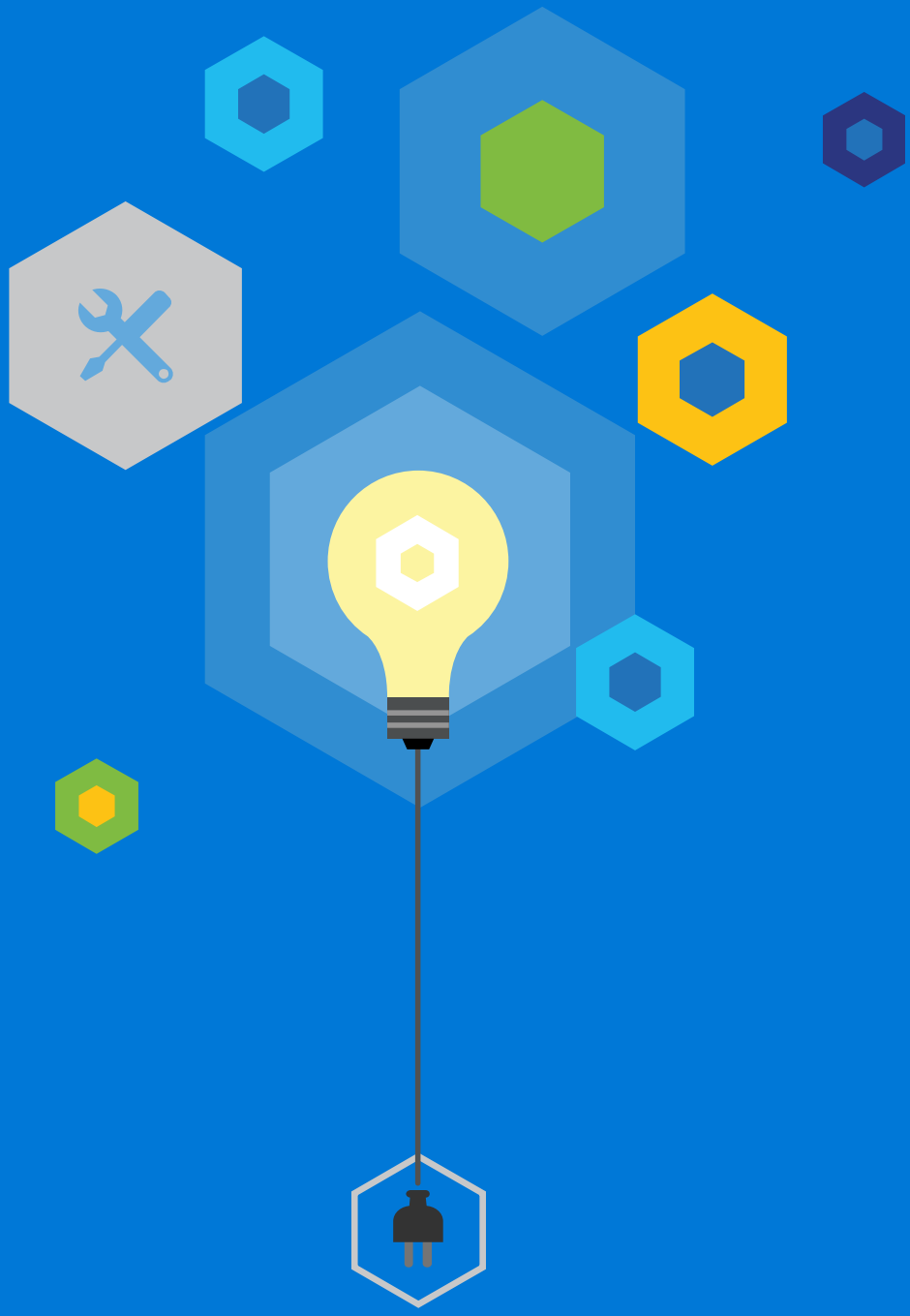
```
const string input = "interesting";
bool comparison = input.ToUpper() == "INTERESTING";
Console.WriteLine("These things are equal: " + comparison);
```

People make security decisions based on string comparisons.

Use `string.Compare(a, b, StringComparison.OrdinalIgnoreCase)`

There’s a code analysis rule for this – CA1309

Turkish I Demo



It’s not just Turkey

If you have a `CultureInfo` of `th_TH.UTF8`

`StartsWith(".")` crashes a process on Mac & Linux because Thai has no punctuation marks.

ISO/IEC 14651: http://unicode.org/L2/L2006/06105-02n3847-14651-2001-fdam3.pdf (Annex C 2.1)

“No syllable structure or word boundary analysis is required, as Thai lexicons are ordered alphabetically, not phonetically. Note that Thai normally does not use any word separator, except, and exceptionally, zero width space.“

Wide <

Unicode has “full width” characters

< becomes ＜

＜ is ＜

Browsers ignore this, so ＜ ｓ ｃ ｒ ｉ ｐ ｔ ＞ isn’t XSS

However if put ＜ into a SQL varchar field it gets converted to <

And now you have persisted XSS

So don’t use a non Unicode column to store Unicode characters.

(And don’t trust data coming from SQL – HtmlEncode it!)

Common cryptography mistakes

"This is base64 encrypted."

"I have this great new algorithm."

Key/IV reuse – when a document/resource changes, change Key/IV.

Keys have purposes. Don't reuse keys across purposes.

Encrypted data must have message authentication.

Hashed combined string errors

```
builtin+securely == built+insecurely
```

Prefer `hash(len(a) + hash(a) + len(b) + hash (b))`

Certificate verification by host name alone.

Path Transversal

File upload – RFC1867

The original local file name may be supplied as well, either as a 'filename' parameter either of the 'content-disposition: form-data' header or in the case of multiple files in a 'content-disposition:file' header of the subpart.

Of course if you edit the request before it leaves the browser …

Use a unique name. Generate it yourself.

If you have to keep it use something like

```
string filename =
    System.IO.Path.GetFileName(userProvidedFileName);
string fullPath = Server.MapPath(
    System.IO.Path.Combine(
       @"d:\inetpub\inbound\",filename));
```

Zip bombs

When accepting user files for parsing it’s easy to evil.

Zip bombs – tiny file which recursively uncompresses.
e.g. 42.zip - zip file consisting of 42 kilobytes of compressed data, containing five layers of nested zip files in sets of 16, each bottom layer archive containing a 4.3 gigabyte file for a total of 4.5 petabytes when uncompressed.

Billion laughs

XML has the same problem.

Billion Laughs defines 10 entities, each of which is defined as consisting of 10 of the previous entities etc. the document being a single instance of the largest entity, which contains one billion copes of the first entity.

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ELEMENT lolz (#PCDATA)>
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Preventable by turning off DTD Parsing

Things we do for you

and a couple of things we don’t.

Opt-in to protections

Back compat means you need to opt-in to certain breaking changes.

To opt in to ASP.NET 4.5 behaviors, set the following in web.config

```
<machineKey compatibilityMode="Framework45" />
```

Or (which is what the ASP.NET 4.5 project templates do)

```
<httpRuntime targetFramework="4.5" />
```

String hashcode randomisation

To help against HashDOS we can randomize hashcodes.

We don’t do this by default in the .NET Framework because people save hashcodes even though we never promised they’d be constant.

```
<configuration>
  <runtime>
    <UseRandomizedStringHashAlgorithm enabled=1 />
    ...
  </runtime>
</configuration>
```

DataProtector

ASP.NET 4.5 introduces MachineKey.Protect and Unprotect

This replaces Encode and Decode which allowed a developer to choose between Sign, Encrypt or Both.

It’s now always both.

Aimed at ephemeral data.

ASP.NET Core replaces it with DataProtector

Stop using this

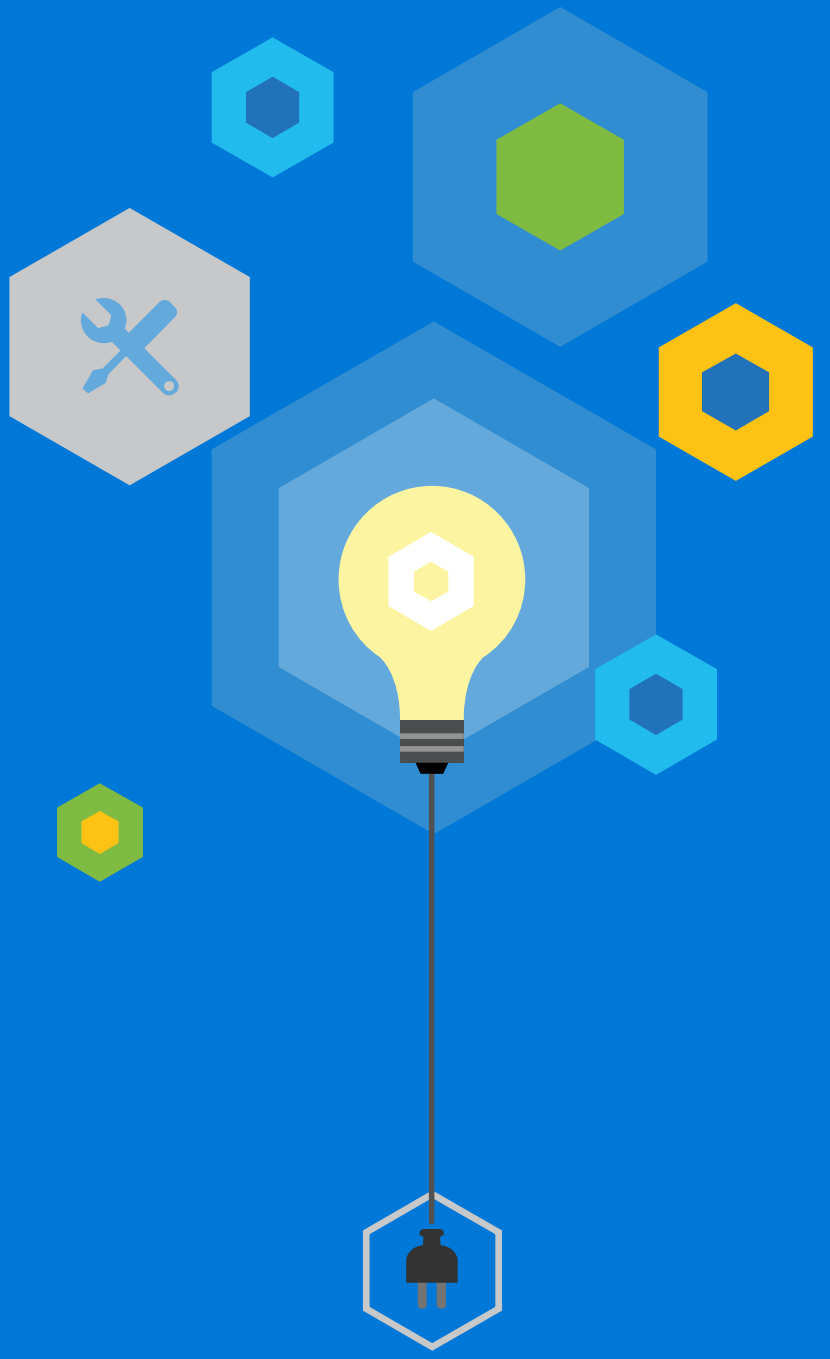
Partial Trust - https://support.microsoft.com/kb/2698981

Request Validation

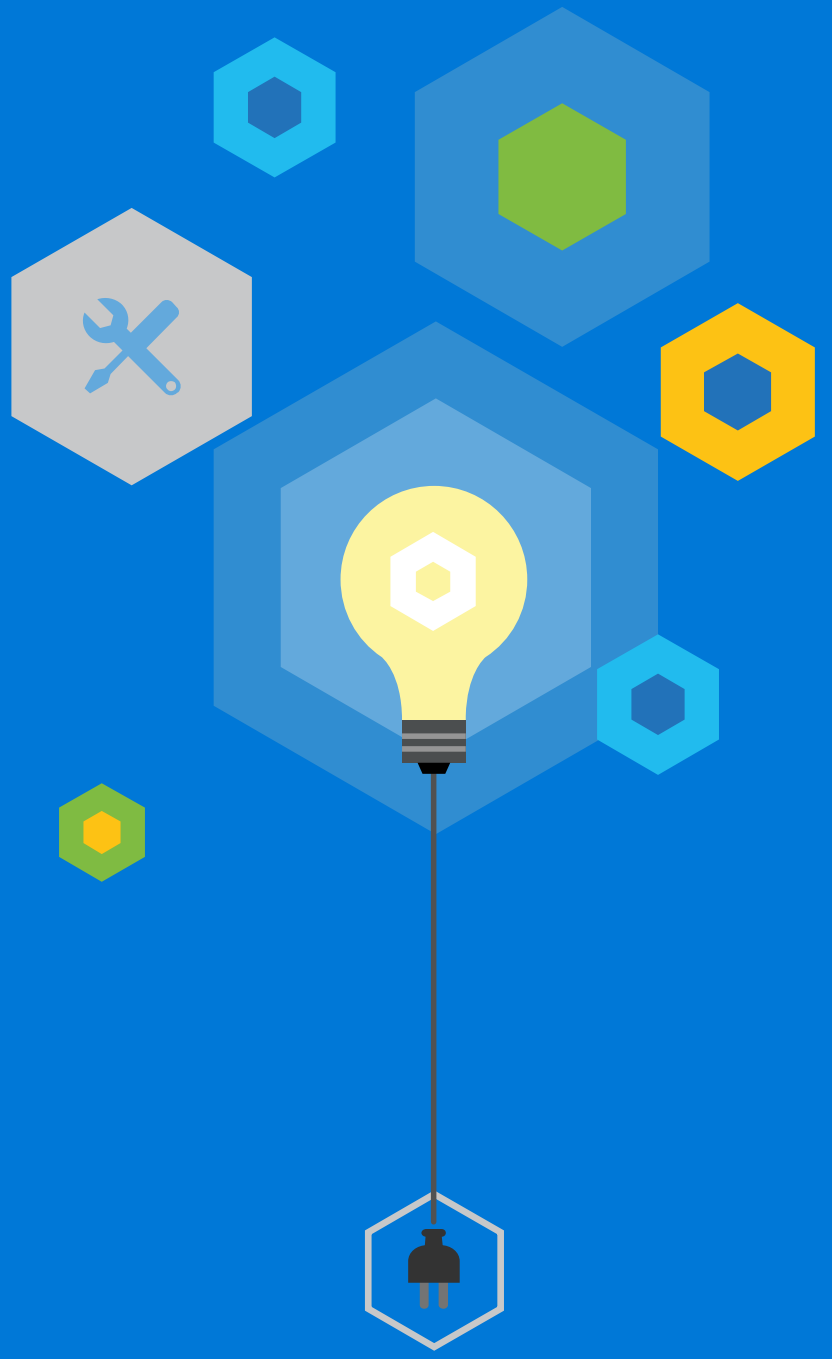
MachineKey.Encode & MachineKey.Decode

http://www.asp.net/aspnet/overview/web-development-best-practices/what-not-to-do-in-aspnet,-and-what-to-do-instead#security

Breaking partial trust demo



The scary demo



In summary

Sign your data, even when it’s encrypted

Don’t use regular expressions

Don’t use BinaryFormatter

Don’t overbind

Use the right verb

Opt-in to new protections