Microsoft

Microsoft®

tech·ed

Developers | 2008

Be a part of the experience.

DVP321

Top Ten Strategies To Secure Your Code

Michael Howard
mikehow@microsoft.com
Principal Security Program Manager
Microsoft Corp.

Microsoft® tech·ed Developers | 2008

Take these strategies to heart, and you'll do just fine!

All these strategies are based on real-world experience!

Microsoft® tech·ed Developers | 2008

“A Secure System does what it's supposed to do, and no more.”

Strategy #1

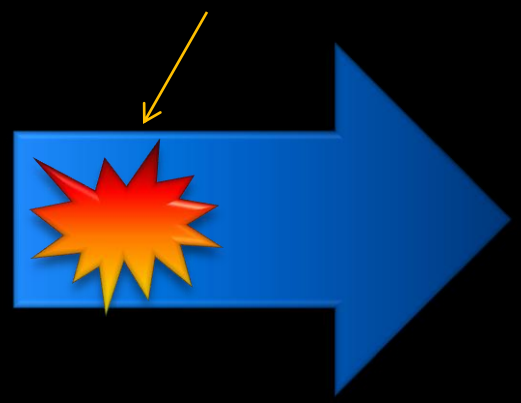
Remember: You will never get your code right!

Remember: You Will Never Get Your Code Right!

- Why?
  - "Attacks only get better, not worse"
  - You are not perfect (even if you think you are)
- Your code might be secure today, but that could all change tomorrow

Example: “Attacks Only Get Better”

MS03-047
XSS in Exchange 5.5 OWA
Oct 15, 2003



Example: "Attacks Only Get Better"

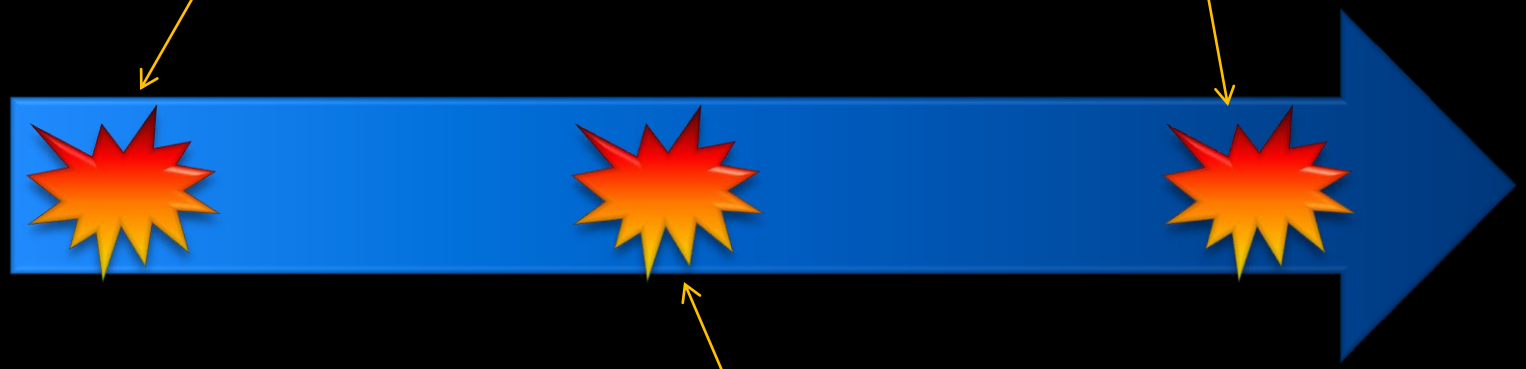
MS03-047
XSS in Exchange 5.5 OWA
Oct 15, 2003

"Divide & Conquer"
Response Splitting paper.
Sanctum
Mar 4, 2004

Example: "Attacks Only Get Better"

MS03-047
XSS in Exchange 5.5 OWA
Oct 15, 2003

MS04-026
Response Splitting in
Exchange 5.5 OWA
Aug 10, 2004



"Divide & Conquer"
HTTP Response Splitting paper.
Sanctum
Mar 4, 2004

Action

- Reduce your attack surface
  - Least privilege
  - Require authenticated connections by default
  - Disable less-used functionality
    - Apply the 80/20 rule
  - Use as many defenses as possible…
    - … which leads me into…

Strategy #2

Use all possible defenses

Microsoft® tech·ed Developers | 2008

Use All Possible Defenses

- A large portion of the Security Development Lifecycle (SDL) focuses on defenses
- Extra defenses help protect customers in the event they are attacked
- Defenses either
  - Utterly stop an attack (they offer a security guarantee)
    - E.g., Firewall
  - Make life harder for an attacker
    - E.g., randomization

Example: Protected Customers

- Blaster took advantage of a buffer overflow in RPCSS
- In Windows Server 2003, RPCSS is compiled with /GS
  - A defense that detects stack-based buffer overruns at runtime
- On Windows Server 2003 the attack was detected by the /GS code
  - RPCSS was killed rather than running the Blaster exploit code
  - A remote elevation was turned into a DoS

Action

- Add XSS Defenses (ASP.NET, ASP, Gadgets)
  - Always HTMLEncode output
    - … or better, use the Microsoft Anti-Cross Site Scripting Library
- Add SQL injection Defenses (any language)
  - Grant access to sprocs
  - Deny access to all underlying tables
- Add Buffer Overrun Defenses (C and C++)
  - Use VC++ 2005 SP1 or later
  - Compile with/GS
  - Link with /NXCOMPAT /DYNAMICBASE /SAFESEH

Strategy #3

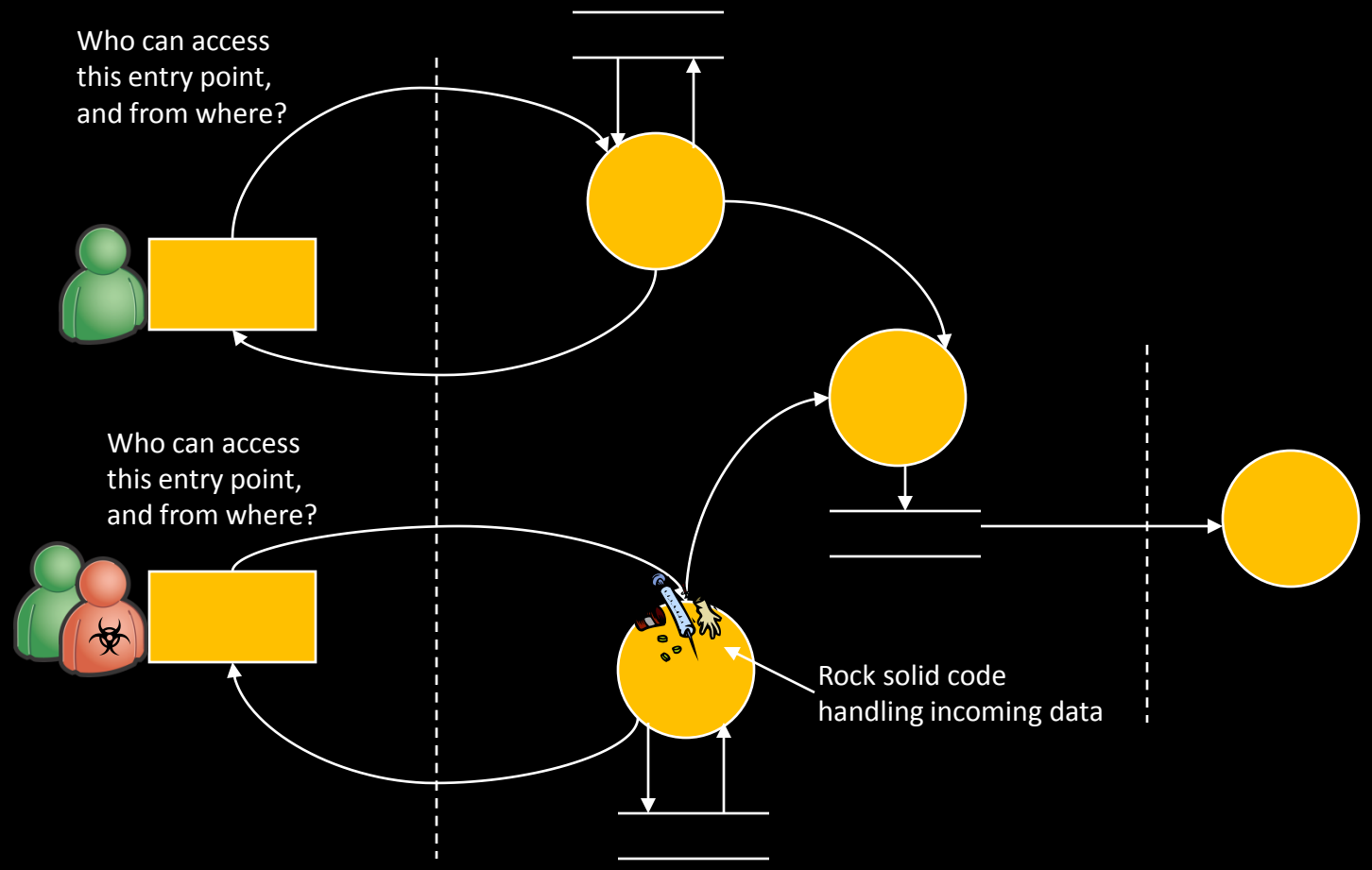
Leverage Threat Models

Microsoft® tech·ed Developers | 2008

Leverage Threat Models

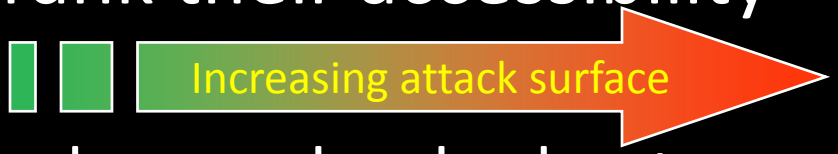
- Threat models not only benefit design
- Understand your code more
  - Where does the data come from (local, remote, local subnet)
  - What trust level is required to communicate with your code (anonymous, user, admin)
  - Pay special attention to external dependencies and assumptions
  - Make sure the correct defenses are in place
- What data are you storing?
  - Privacy concerns: Is the data personally identifiable or confidential?

Leverage Threat Models



Action

- Build effective threat models
- Identify all entry points into the system, and rank their accessibility



  - Local versus local subnet versus remote
  - Admin versus user versus anonymous
- Higher attack surface == better be good code!
- Consider reducing attack surface (see strategy #1)
- Review code along the anonymous data paths

Strategy #4

Never Trust Data

Microsoft® tech·ed Developers | 2008

Never Trust Data

- “All data is evil, until proven otherwise”
- The most heinous bugs are because of too much trust in data
  - Buffer overruns
  - Cross-site scripting (XSS)
  - SQL injection
  - Command injection
  - Etc.

Evidence

- ~49% of security bugs tracked by CVE between 2001-2007 were due to too much trust in data
- Stragglers include
  - Breaking a sandbox, poor crypto, information disclosure etc

Action

- Don’t solely use “blocklists”
- Constrain
  - Only allow what you know to be good
  - E.g., constrain to only a valid email address
- Reject
  - Reject that which you know is bad
  - E.g,, reject bad characters, often environment specific (Web etc) such as <>& etc
- Sanitize
  - Encode if possible
    - E.g., HTML encode

Crystal Reports MS04-017

```
public class CrystalImageHandler : WebControl {
  private string tmpdir = null;
  protected override void Render(HtmlTextWriter writer) {
    string filepath;
    string dynamicImage =
      (string)Context.Request.QueryString.Get("dynamicimage");
    if (tmpdir == null) {
        tmpdir = ViewerGlobal.GetImageDirectory();
    }
    filePath = tmpdir + dynamicImage;
    FileStream imagestream =
      new FileStream (filePath, FileMode.Open, FileAccess.Read);
    ...
    // stream file to user^h^h^h^h bad guy
    ...

    File.Delete (filePath);
  }
}
```

(1) Get filename from querystring

(2) Concat temp dir and filename

(3) Open the file

(4) Delete the file!

MS04-017 The Lesson & The Fix

- Trusting an untrusted filename – not good!
- Fix was:
  - Constrain
    - Extension must be “.jpg” or “.png”
    - Filename must be a GUID
      - xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
  - Reject
    - Path must be devoid of ‘:’, ‘/’ and ‘\’
- On failure (ANY FAILURE!) – simply return 404

Strategy #5

Fuzz!

Fuzz

- Fuzzing was designed to find reliability bugs
  - It turns out many reliability bugs are actually security bugs
- A buffer overrun defect might crash an app
- The right payload could execute malicious code
- This strategy is to prove that your messed up on Strategy #4!

Why Fuzz?

- XLS (MS06-012)
- BMP (MS06-005, MS05-002)
- TNEF (MS06-003)
- EOT (MS06-002)
- WMF (MS06-001, MS05-053)
- EMF (MS06-053)
- PNG (MS05-009)
- GIF (MS05-052, MS04-025)
- JPG (MS04-028)
- ICC (MS05-036)
- ICO (MS05-002)
- CUR (MS05-002)
- ANI (MS05-002)
- DOC (MS05-035)
- ZIP (MS04-034)
- ASN.1 (MS04-007)
- Etc…

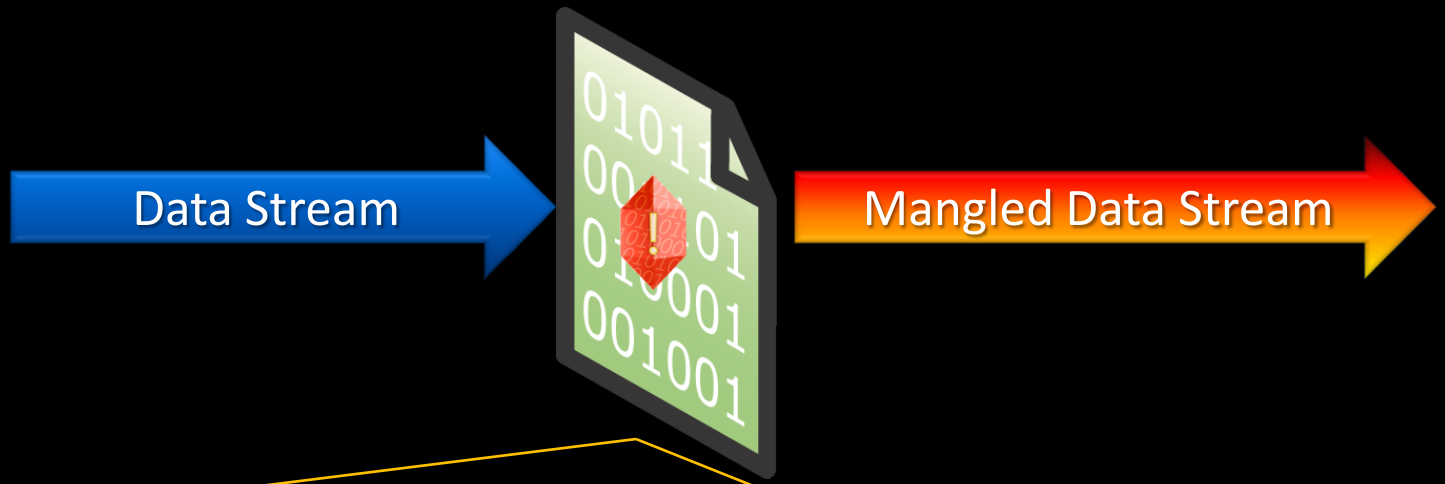
Fuzzers

- Two major types, both can effective
  - Dumb
    - Effective in early fuzz testing
    - Randomly change data
  - Smart
    - Effective in early and advanced fuzz testing
    - Know of the data format

Action

- If you consume files or network data you MUST fuzz!
  - File
    - SDL mandates minimum 100,000 iterations per file format
  - Network end points
    - Especially anonymous and remote end points
- Buy or build fuzzers
- You have to automate
- Dedicate a computer or three
- Add a “layer of pain” to your app

Action: Layer of Pain




```
#if _FUZZ_
   Malform pain = new Malform();
   fileData = pain.Munge(fileData);
#endif
```

Set NULL to non-NULL
Flip adjacent bytes
Toggle upper bits
Write junk
Truncate stream
Extend stream

One byte
Entire stream
Range of bytes

Strategy #6

Stop the Bleeding

Microsoft® tech·ed Developers | 2008

Stop The Bleeding

- Attackers attack all code
- Don’t add insecure code to your product
- “Friends don’t let friends write insecure code”

Action

- Security training for all engineers
  - Instructor-led classes
  - “just in time” training
    - About to build a Web app? Read up on XSS issues
    - About to do database access? Read up on SQL injection
    - Using C or C++? Learn about buffer overruns and integer overflow bugs
    - Doing crypto? Read about common crypto failures.
- Create a “Security Quality Gate”
  - No banned APIs
  - No lousy crypto
  - All code analysis tools pass
- VSTS can have a check-in rule run

Strategy #7

Recognize the asymmetry

Microsoft® tech·ed Developers | 2008

Recognize The Asymmetry

- "The Attacker's Advantage, the Defender's Dilemma"
- The odds are against you
  - You have to get 100% of the features right 100% of the time, with many constraints
- The odds are stacked in favor of the bad guy
  - They can spend as long as they want to find one bug

Welcome to the Internet Grudge Match

Action

- I already said, “Stop the bleeding”
  - But cure the patient too!
- Bad guys whack all code
- Review and fix legacy code too
- Deprecate old features

Strategy #8

Use the best tools
at your disposal

Microsoft®
tech·ed
Developers | 2008

Use The Best Tools At Your Disposal

- Tools are no panacea
- But they do help
- Tools do something humans cannot do
  - They scale
- Tools can give you a feel for how bad code might be
  - A lot of warnings may indicate poor quality code

Action:Use The Best Tools At Your Disposal

- Use static analysis tools on every build
  - FxCop
  - /analyze
  - lint
  - Etc.
- /W4
- Run the tools on all code

Strategy #9

Stay one Step Ahead

Microsoft® tech·ed Developers | 2008

Stay One Step Ahead

- The security landscape constantly changes
- Not only might there be bugs in your code
- But patches required in systems you use

Action: Stay One Step Ahead

- Learn from past mistakes
- Read! Read! Read!
- Nominate a security champ
  - Distill and disseminate security intelligence
- Sign up for bugtraq
  - www.securityfocus.com
  - Create an inbox rule!!

Strategy #10

Security of the Software is up to (singular) you

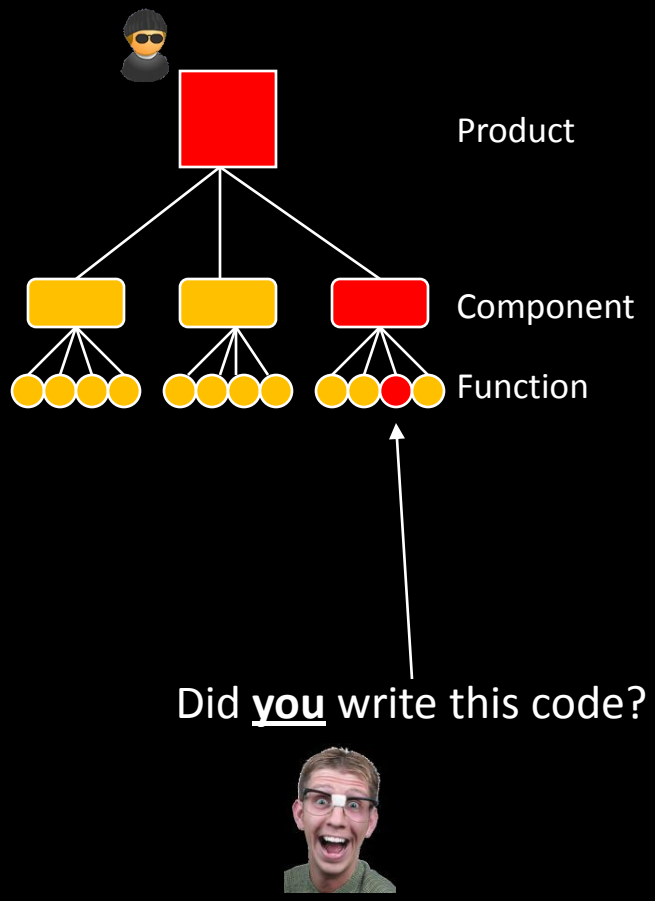
Microsoft® tech·ed Developers | 2008

Security Of The Software Is Up To (Singular) You

- It's totally up to you (yes, YOU!) to get security right
  - Secure products are made from secure code
  - Secure code is written by individuals
  - You cannot be replaced by a tool
  - Therefore, the security of the product is totally up to you. QED
- Remember, your code will be scrutinized and attacked
- But will it be compromised?
- The difference is totally up to you

Evidence

- Blaster was due to a bug in DCOM
  - Only one person wrote the function containing the flaw
- CodeRed was due to a bug in Index Server ISAPI
  - Only one person wrote the function containing the flaw
- Debian/Ubuntu rand number bug
  - Only one person made the error



Action

- Take pride in your code
- Use all the tools at your disposal
- Take advantage of every language construct that leads to greater safety and security
- Have it peer-reviewed
- Don’t be scared or too vain to ask for help
- No passing the buck

Summary

- Remember: You will never get your code right!
- Use all possible defenses
- Leverage Threat Models
- Never Trust Data
- Fuzz!
- Stop the Bleeding
- Recognize the asymmetry
- Use the best tools
  at your disposal
- Stay one Step Ahead
- Security of the Software is up to (singular) you

Security Resources

- The SDL 3.2 documentation
  - http://go.microsoft.com/?linkid=8685076
- The SDL Book (Howard and Lipner)
- The SDL Blog
  - http://blogs.msdn.com/sdl

Resources for Developers

Microsoft® tech·ed Online

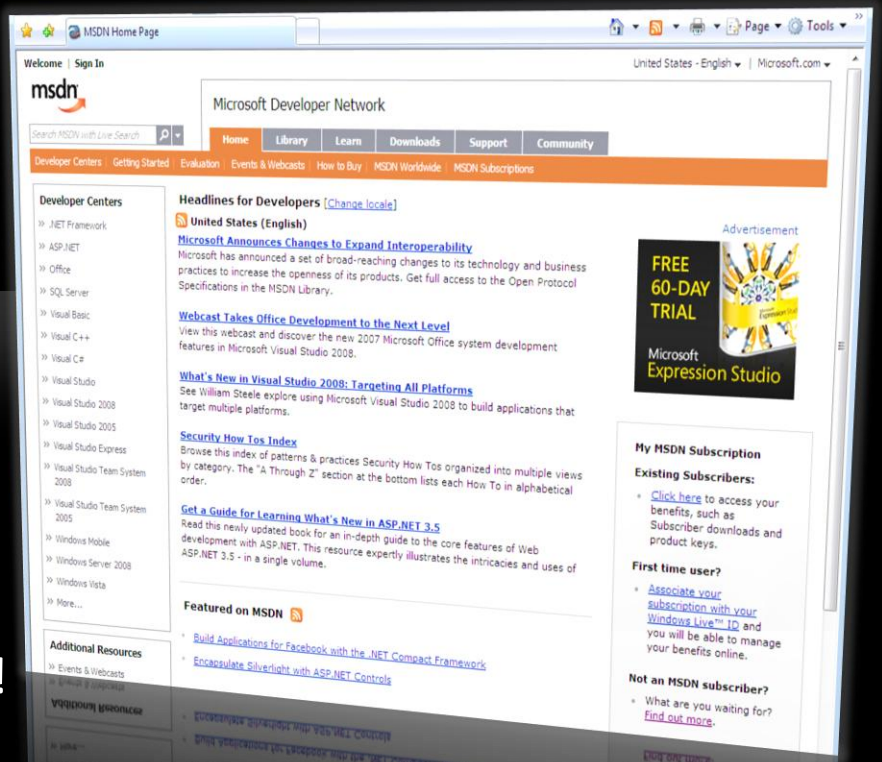
www.microsoft.com/teched

Tech·Talks
Tech·Ed Bloggers
Live Simulcasts
Virtual Labs

msdn®

http://microsoft.com/msdn

Developer's Kit, Licenses, and MORE!



Development Practices Resources

- Be sure to visit the *ALM & Development Practices* booth in the TLC (RED section)
- Visit the TechEd Online Stage:
  - Visual Studio Team System Panel – Meet the Team Tuesday at 3:00 PM
  - Let's Talk Application Lifecycle Management Panel Wednesday at 9:00 AM
- Michael Howard Book Signing
  - Thursday from 4:15 – 5:15 *Developer Tools & Languages* (Blue) section of the TLC

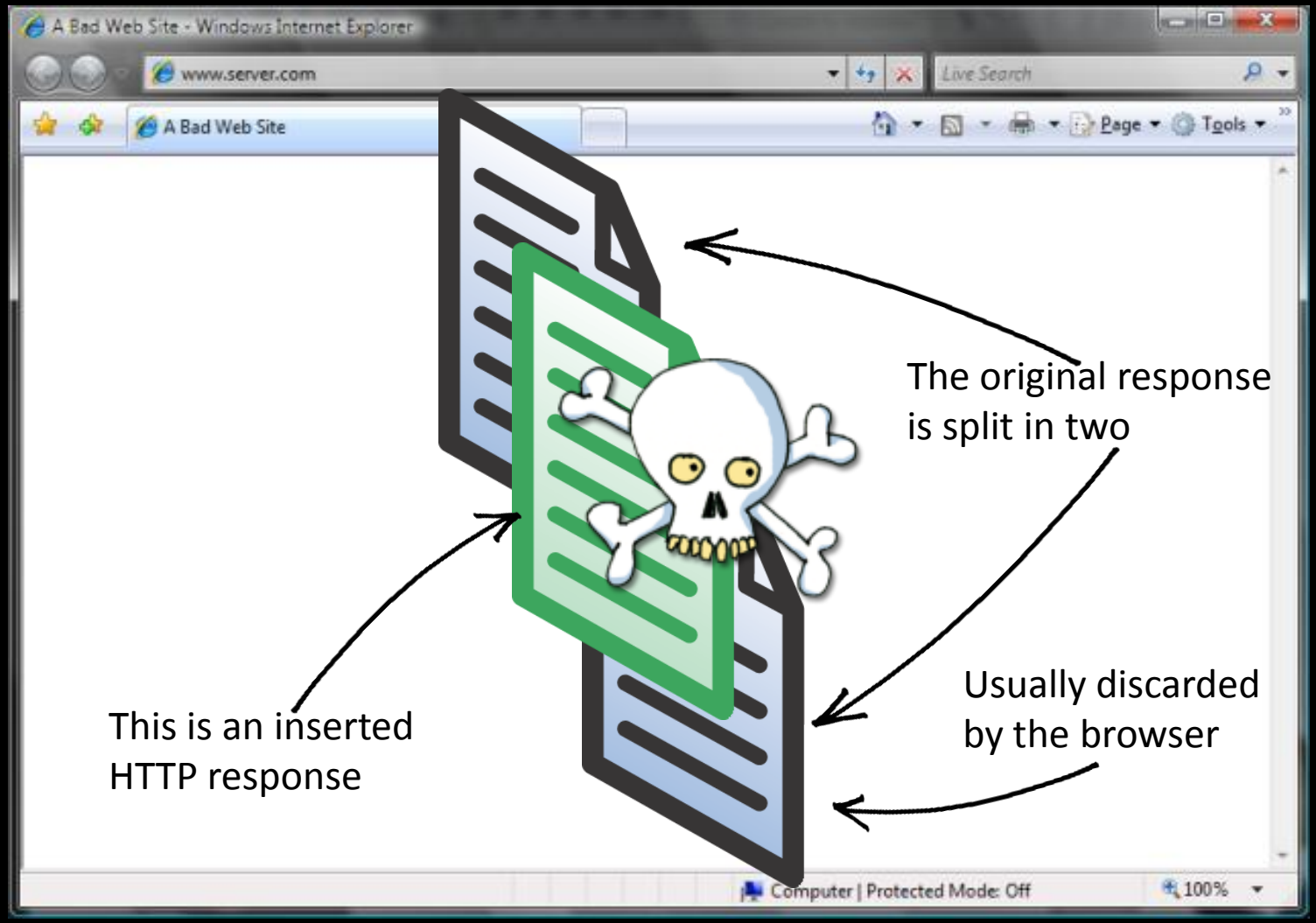
1 Year
Subscription!

msdn

Complete an evaluation on CommNet and enter to win!

Email: mikehow@microsoft.com
Blog: http://blogs.msdn.com/michael_howard/

Response Splitting in Pictures



Response Splitting in Code (1 of 2)

get.aspx

```
<% Response.Redirect("/getitem.aspx?item="
       + Request.QueryString("item")) %>
```

Vulnerable code

```
/get.aspx?item=TwasMidnightInTheSchool
```

'Normal' request

```
/getitem.aspx?item=TwasMidnightInTheSchool
```

Which redirects to

```
/get.aspx?item=
  foo%0d%0aContent-Length:%200%0d%0a
  HTTP/1.1%20200%20OK%0d%0a
  Content-Type:%20text/html%0d%0a
  Set-Cookie:%20xyzzy%0d%0a
  Content-Length:%2020%0d%0a
  <html>Gotcha!</html>
```

An attack, poisons a user's cookie with xyzzy (or worse!)

Response Splitting in Code (2 of 2)

```
HTTP/1.1 302 Moved Temporarily
Date: Wed, 20 Apr 2004 15:00:11 GMT
Location: http://foo.com/getitem.aspx?item=foo
Content-Length: 0
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: xyzzy
Content-Length: 20
<html>Gotcha!</html>
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Content-Type: text/html
…
```

Completes the first response

The forged HTTP Response from the Response.Redirect