Security Testing Software Made Easy

(or a reasonable facsimile thereof)

Presented by

Blake McNeill

mcneillb@ZingPow.ca

AKA Dr. Evil (the original)

About Blake McNeill

- First program written on punch cards in 1977
- Involved in software/product development for large corporations and a number of startups
- Principle languages used: Fortran, C, VB, ASP, Delphi, C#
- Principle databases used: SQL Server, Postgres, Oracle, Sybase, Access
- Principle OS's used: VMS, Unix, Linux, Windows including mobile devices
- Microsoft Windows Security MVP for four years
- Currently second year as Microsoft Secure Development MVP

Outline of Presentation

- Why should you care?
- Realities of Software Security
- Exercise in Evil Thinking
- Threat Modeling
- Pillars of Secure Software
- SQL Injection Attack
- Fiddler Attack
- Fuzz Testing
- Questions, Comments and Abuse

Why should you care?

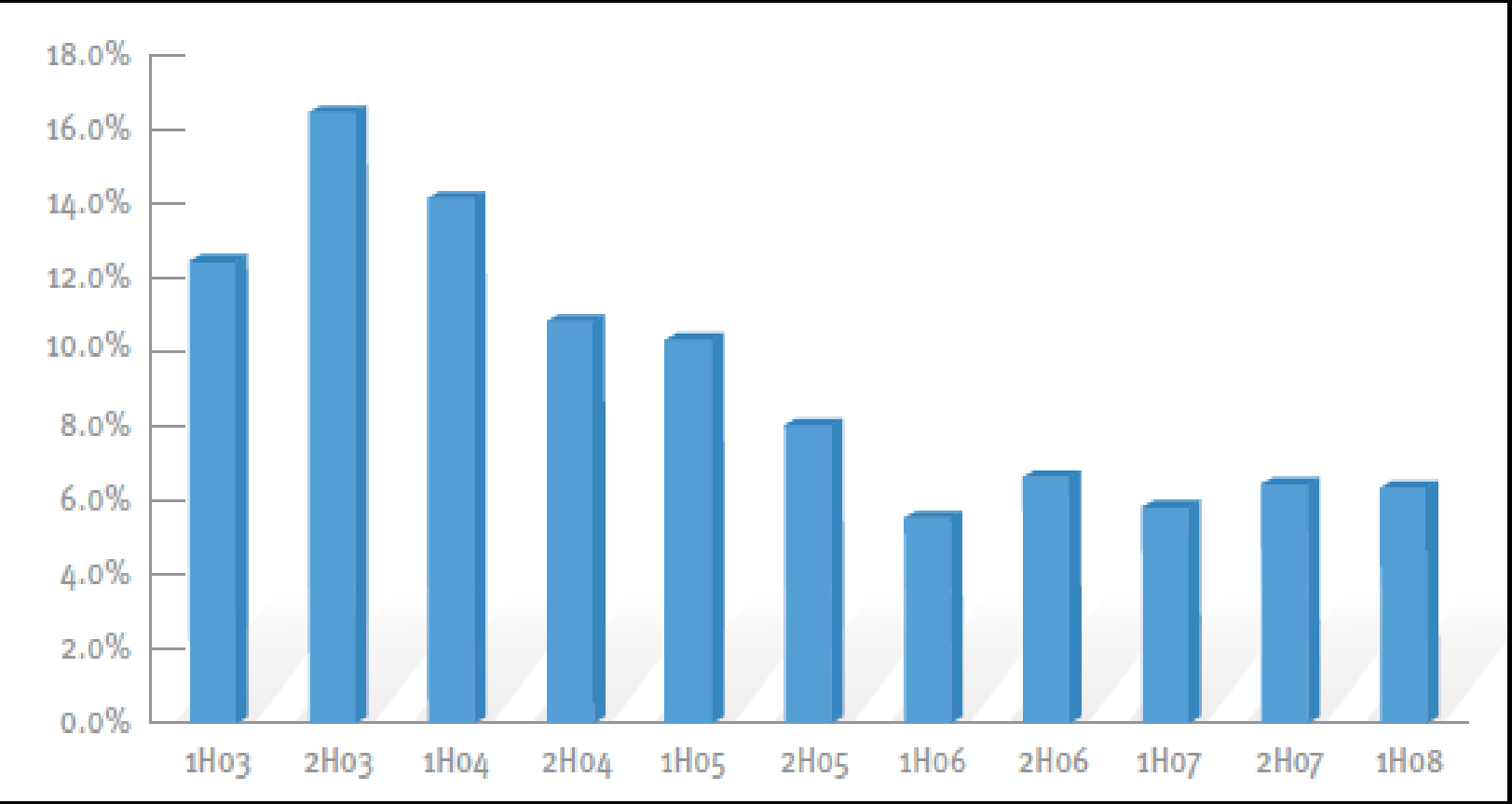
- Windows/IIS/etc are no longer viable targets for hackers, now your application is!!
  - Last major internet worm that relied on solely a Windows vulnerability was April 2004 (Sasser)
  - Microsoft has invested many millions of dollars into SDL (Security Development Lifecycle) to improve the security of their applications
    - Training of developers, managers, testers etc
    - Security now part of the design, coding, testing, deployment, operations etc has been indoctrinated into all areas of Microsoft
    - Security budget, department and resources
    - Security concerns can stop, cancel and even remove applications and projects
- Is your company doing this, not likely

Why should you care?

- Your applications (and Users) are now the low hanging fruit. Your app, your problem.
- Third party apps are typically developed without much consideration for security or secure development practices.
- Your in 'good' company with some very large software development companies (see http://www.us-cert.gov/cas/bulletins for lists of weekly vulnerabilities)
- The Good News is its not really that hard to raise your fruit.

Why should you care?

Operating system vulnerabilities as a percentage of all disclosures 1H03–1H08



Realities of Software Security

- You Will Never Get Your Code Right!.
  - "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 with a new type of exploit for example.
- Recognize The Asymmetry, the odds are against you
  - You have to get 100% of the features/code right 100% of the time, with many constraints including limited time
  - Hackers can spend as long as they want to find one bug or defect and that might be all they need
- The only man who never makes mistakes is the man who never does anything (Theodore Roosevelt)

Realities of Software Security

- Software Security ultimately breaks down into Risk Management
  - Risk = Probability x Consequence
  - Deal with the large risk items first and then work your way down to acceptable risk (nothing of value is really risk free).
  - Software industry has much to learn about managing risk
    - Pipeline Risk Analysis for example which have risk models that consider over 200 factors and give a fairly accurate risk assessment of the system, now granted they don't typically have such a strong 'human' component (human users, human attackers) which makes the problem somewhat more difficult

Exercise in Evil Thinking

- First a legal disclaimer
  - Some topics discussed could be considered illegal or used for illegal purposes, so this information is provided for educational purposes only, use at your own risk using good judgement and remember to play nice and fairly with others. If you are arrested or otherwise subjected to interrogation, torture and/or other unpleasantries, it's not my fault.

Excise in Evil Thinking

- How would you open a Combination Lock without knowing the combination, ie how would you hack a combination lock?

Excise in Evil Thinking

- Destructive Methods
  - Cut the shackle
  - Brute force the lock open
  - Remove the front and manually manipulate the lock mechanism
  - Etc etc etc
- Problems for the hacker
  - Hard to walk around unnoticed with a pair of bolt cutters/crowbar
  - Too much noise smashing stuff
  - The lock owner will know the lock has been compromised

Excise in Evil Thinking

- Non Destructive Methods
  - Try every possible combination
    - If it takes three seconds to input the first digit, two seconds for the second digit, and one second for the third digit, then the normal search time for a 60-number dial with three cams would be $(3 + 2 + 1) \times 60^3$ or about 360 hours to brute force.
  - Use an intelligent combination search
    - Typical padlocks are manufactured with generous tolerances, allowing two, three or even more digits of 'play' in the correct access sequence. Given a 60-number dial with three cams and three digits of play, the search space is reduced from 60 × 60 × 60 to 20 × 20 × 20, a 96% reduction in potential combinations or about 2.78 hours to brute force.

Excise in Evil Thinking

- Non Destructive Methods
  - Know something 'special' about the lock
    - For example early combination padlocks made by Master Lock could be cracked by pulling on the shackle of the lock and turning the dial until it stopped; each numeral in the combination could be revealed in this manner. More recent models of Master padlock with a 40-position dial have a mechanical weakness that can give away the last numeral in the combination, and the first two numerals have a mathematical relationship with the last number. This weakness reduces the number of possible combinations from 64,000 to a mere 100, which can be tried in a relatively short time.
  - Listen for the fence falling into the wheel notches (just like they do in the movies, if that even works)

Excise in Evil Thinking

- Non Destructive Methods
  - Problems for the hacker
    - Takes to long
    - Requires ‘special’ knowledge about the particular lock and model

Excise in Evil Thinking

- Non Destructive Methods
  - Use an injection attack
    - Get a pop can and a pair of scissors and make a shim

Excise in Evil Thinking

- Injection Attack on a Lock
  - No physical evidence that the lock has been compromised
  - Easy/cheap to make
  - No special skills required
  - Can be used on other lock types as well (keyed paddle locks etc).
    - Example of "Same Bug, Different App"

Excise in Evil Thinking

- Forensics/Why does this work?
  - Lock was designed for brute force attack
    - Harden steel shackle and case
    - Enough permutations in the combination
  - Is this a flaw in construction/coding?
    - No, I think its a flaw in design and a usability 'feature' that allows the lock to closed without knowing the combination which allowed for the injection as the designer never thought of the case where a lock would be 'closed' while locked which then allowed it to be ultimately unlocked (ie the latch disengaged from the hasp).
- Should have built a Threat Model!!
  - Hackers typically challenge your assumptions, better find them all and get them right as they will make you pay otherwise.

Threat Modeling

- Despite what some people think you can't just plug security in at the end of a project like something like Performance tuning.
- Threat modeling is like everything thing else, the idea is to get you to think about the whole problem before rushing out to solving the problem in parts.
- Threat Modeling, 10,000 hackers can't be wrong, if you don't Threat Model they will.

Threat Modeling

- The point here is to get a high-level overview of how scared you should be about your product and its features; to see what kind of attack surface it has.
  - Is it Notepad or is it IIS?
  - Is it a rich-client app or a network service?
  - What account does it run under?
  - Does it run in user mode or in kernel mode?
  - Is it restricted to fully trusted (native) code or does it allow partially-trusted callers?
  - Does it run by default every time the box boots, or is it a utility that the user runs once in a blue moon?
- You don't need to be an expert in threat modeling to get something positive from threat modeling.

Threat Modeling

- Three key components to threat modeling
  - Permissions / Roles
    - Anonymous / remote users
    - Privileges
  - Data
    - Remote or Local
    - Who put that data there and how
    - Where does it go and how is it used
    - Value (value to a hacker)
  - Components
    - Entry points
    - Trust Boundaries
      - definition of a trust boundary is simply where you don't trust what's on the other side

Threat Modeling

- Use STRIDE to test components, assumptions, etc

Threat	Property	Definition	Example
Spoofing	Authentication	Impersonating something or someone else.	Pretending to be any of these: billg, microsoft.com, or ntdll.dll.
Tampering	Integrity	Modifying data or code.	Modifying a DLL on disk or DVD, or a packet as it traverses the LAN.
Repudiation	Non-repudiation	Claiming to have not performed an action.	"I didn't send that e-mail," "I didn't modify that file," "I certainly didn't visit that Web site, dear!"
Information Disclosure	Confidentiality	Exposing information to someone not authorized to see it.	Allowing someone to read the Windows source code; publishing a list of customers to a Web site.
Denial of Service	Availability	Deny or degrade service to users.	Crashing Windows or a Web site, sending a packet and absorbing seconds of CPU time, or routing packets into a black hole.
Elevation of Privilege	Authorization	Gain capabilities without proper authorization.	Allowing a remote Internet user to run commands is the classic example, but going from a limited user to admin is also EoP.

Threat Modeling

- Microsoft Threat Analysis and Modeling Tool
  - http://www.microsoft.com/SDL
  - Free
  - Version 3.1.3.1 Beta recently released

Threat Modeling

- Microsoft Threat Analysis and Modeling Tool
  - Quick Overview

Security Testing

- If you don't do it, someone you might not like very much will.
- Hackers love assumptions, especially when they are WRONG!!
- Think about what the hacker has, controls or can influence
  - Code
  - Entry Points
  - Physical Device
  - Network Traffic
- Prioritize Testing based on Threat Modeling

The 5 Pillars of software security

1. Code and process
   - Code Access Security, Evidence, Policies, OS
2. In-memory data
   - DPAPI
3. Stored persisted data
   - Cryptography or just DPAPI
4. Access
   - Roles, AzMan, Claim-based Identities, AD, Certs
5. Communications
   - WCF (for WS-Security), S.S.C.Xml, S.S.C.Pkcs

Code and Process

There are two types of attacks:
- Injection
- Everything else

About 49% of security bugs tracked by CVE between 2001-2007 were due to too much trust in data
- Buffer overflows
- SQL Injection
- Script Injection
- Cononicalization
- HTML Injection
- Etc

Dynamically created anything should be suspect especially if it involves user input

Code Protection

Code Protection depends on Injection Projection

- You must validate
  - Input (All Input should be considered EVIL)
  - Parameters
  - Buffers / Pointers
- 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
- Digitally Sign Code

Simple Tests

Attack	Test Case
Cross-site scripting	Test<B>aaa Bill O'Henry aaa;
SQL Injection	Bill O'Henry 1;test
Buffer Overflows	AAAA…AAAA Ax65535 Ax65536 Etc…
User Interface Spoofing	Aaa\rbbb aaa\nbbb Aaa[0x0D,0x0A,0x00]bbb
Traversal	../../../../../../boot.ini ..%2f.. %2f.. %2f.. %2f.. %2f.. %2fboot.ini
Canonicalization	"Foo.exe." (trailing dot) "Foo.exe " (trailing space)

Security Testing

- Entry Points – consider all of them examples
  - User interfaces
  - Listening Sockets
  - Pipes
  - Files
  - Shared Sections
  - Protocol Handlers
  - ActiveX Controls
  - RPC
  - HTTP Requests
  - HTTP Responses
  - Databases
  - Message Pump
  - Registry
  - Email
- Error Messages / Logging
  - See if Error Messages Report 'too much' information
  - Check if logs/debugging/etc contain 'too much' information

Simple SQL Injection Example

```
private void cmdLogin_Click(object sender, System.EventArgs e) {

  string strCnx = "server=localhost;database=northwind;uid=sa;pwd=;";
  SqlConnection cnx = new SqlConnection(strCnx);

  cnx.Open();

  string strQry = "SELECT Count(*) FROM Users WHERE UserName='" + txtUser.Text + "' AND Password='" +
        txtPassword.Text + "'";
  int intRecs;
  SqlCommand cmd = new SqlCommand(strQry, cnx);
  intRecs = (int) cmd.ExecuteScalar();

  if (intRecs>0) {
    FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false);
  }
  else
  {
    lblMsg.Text = "Login attempt failed.";
  }
  cnx.Close();
}
```

Simple SQL Injection Example

User enters a userid of 'Admin' and a password of '1234' query becomes:

Select count(*) from Users
Where UserName = 'Admin'
and Password = '1234'

Evil User enters a userid of 'Admin' and a password of "1234' or 1=1 --"
and the query becomes

Select count(*) from Users
Where UserName = 'Admin'
and Password = '1234' or 1 = 1 --'

The Evil User is now in as Admin, which would be very bad.

I’ve Got Your Code

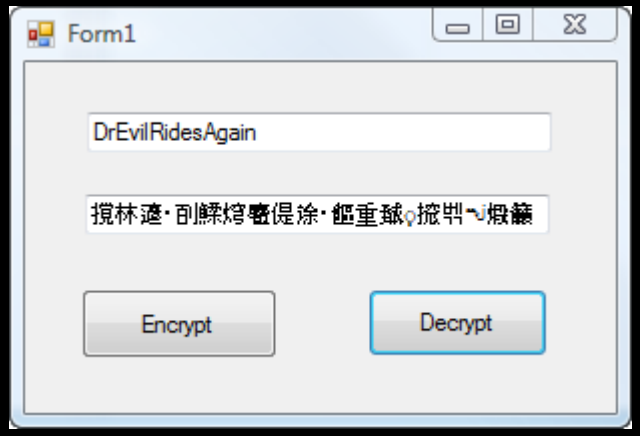
- Code
  - The bad guy has your code, be it HTML (ie view code in IE) or Source Code using .NET Reflector
    - Look for secrets etc hidden in code

Cracking code is trivial and is like standing on the train tracks and wondering where the train goes, just follow the tracks.

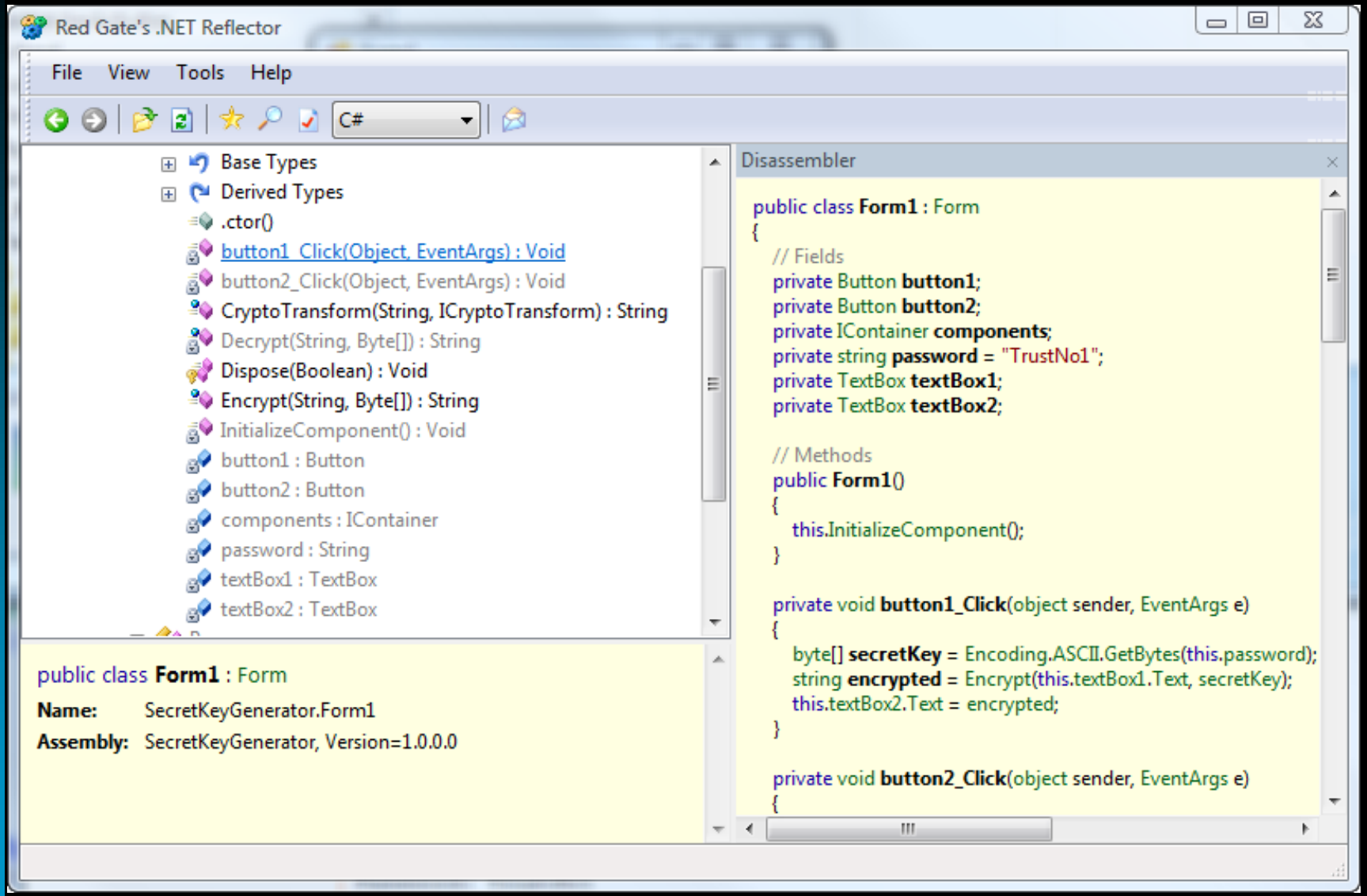
I’ve Got Your Code

- How can you tell if there are any secrets etc in the code?
  - .Net Code use .Net Reflector to decompile the code back to source code.
    - any program is vulnerable to being decompiled.

The Victim, simple app that encrypts / decrypts a string



Using Reflector we can see the source and what do we find?

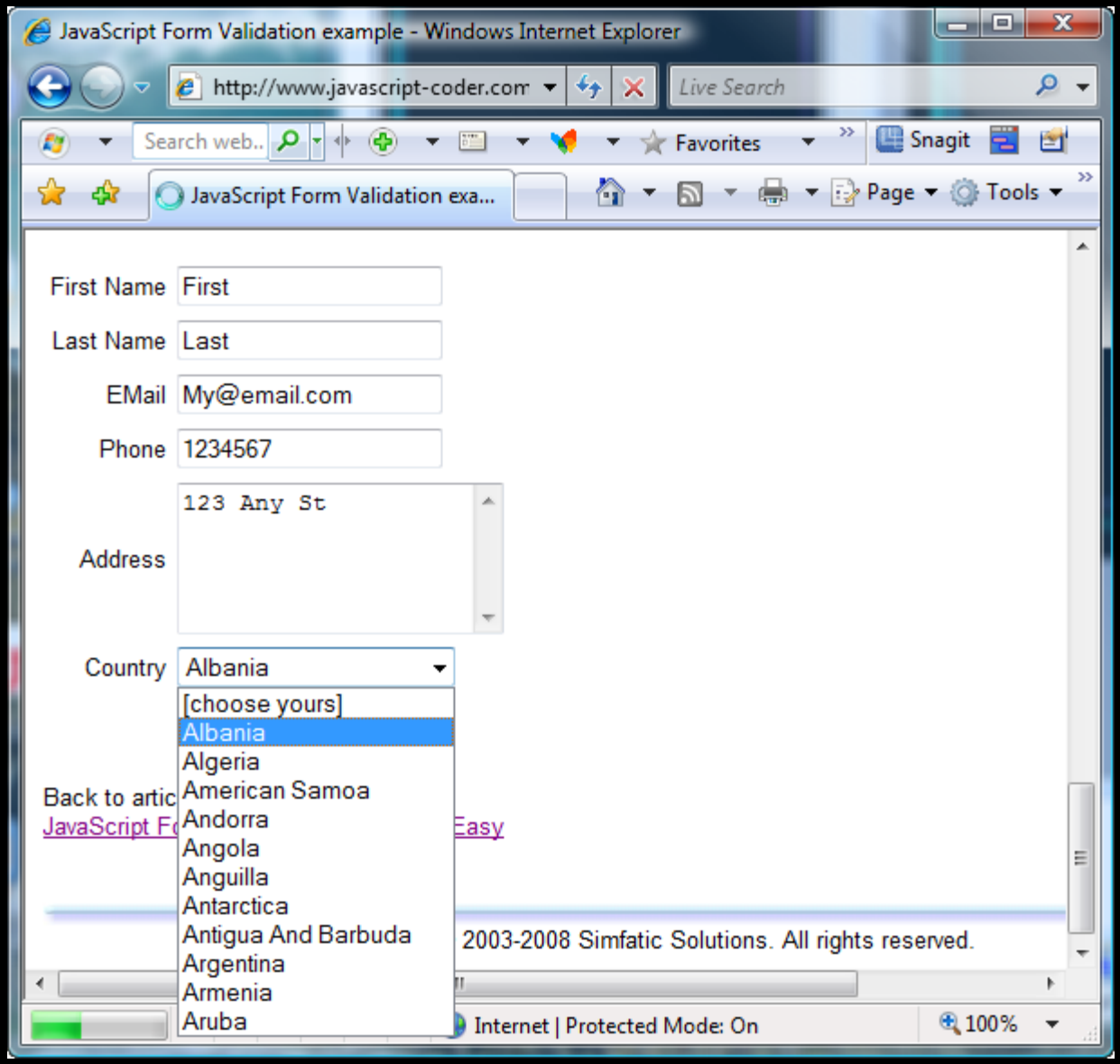

Red Gate's .NET Reflector
File View Tools Help
C#
Base Types
Derived Types
.ctor()
button1_Click(Object, EventArgs) : Void
button2_Click(Object, EventArgs) : Void
CryptoTransform(String, ICryptoTransform) : String
Decrypt(String, Byte[]) : String
Dispose(Boolean) : Void
Encrypt(String, Byte[]) : String
InitializeComponent() : Void
button1 : Button
button2 : Button
components : IContainer
password : String
textBox1 : TextBox
textBox2 : TextBox
public class Form1 : Form
Name: SecretKeyGenerator.Form1
Assembly: SecretKeyGenerator, Version=1.0.0.0
Disassembler
public class Form1 : Form
{
// Fields
private Button button1;
private Button button2;
private IContainer components;
private string password = "TrustNo1";
private TextBox textBox1;
private TextBox textBox2;
// Methods
public Form1()
{
this.InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{
byte[] secretKey = Encoding.ASCII.GetBytes(this.password);
string encrypted = Encrypt(this.textBox1.Text, secretKey);
this.textBox2.Text = encrypted;
}
private void button2_Click(object sender, EventArgs e)
{

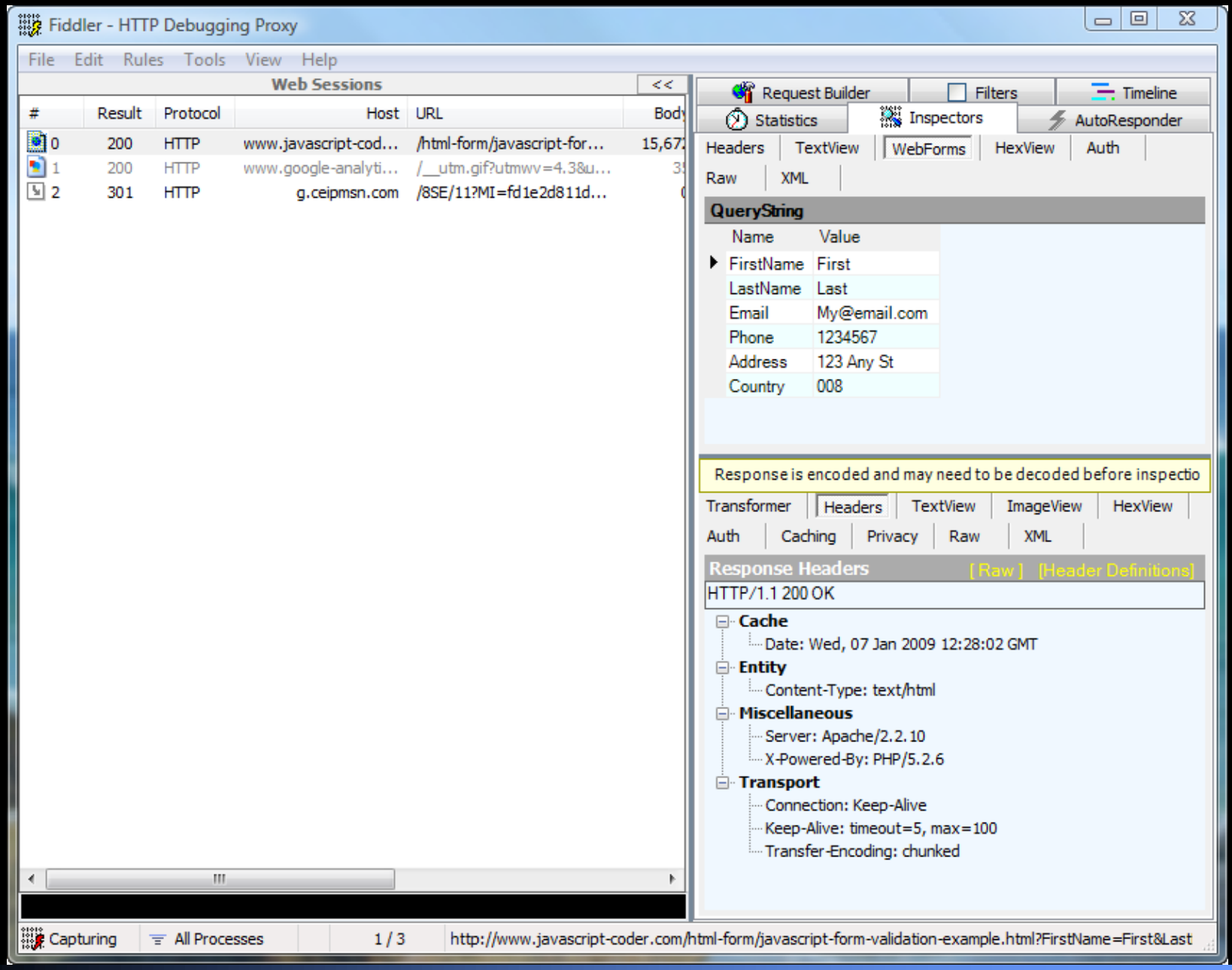
I’ve Got Your Code

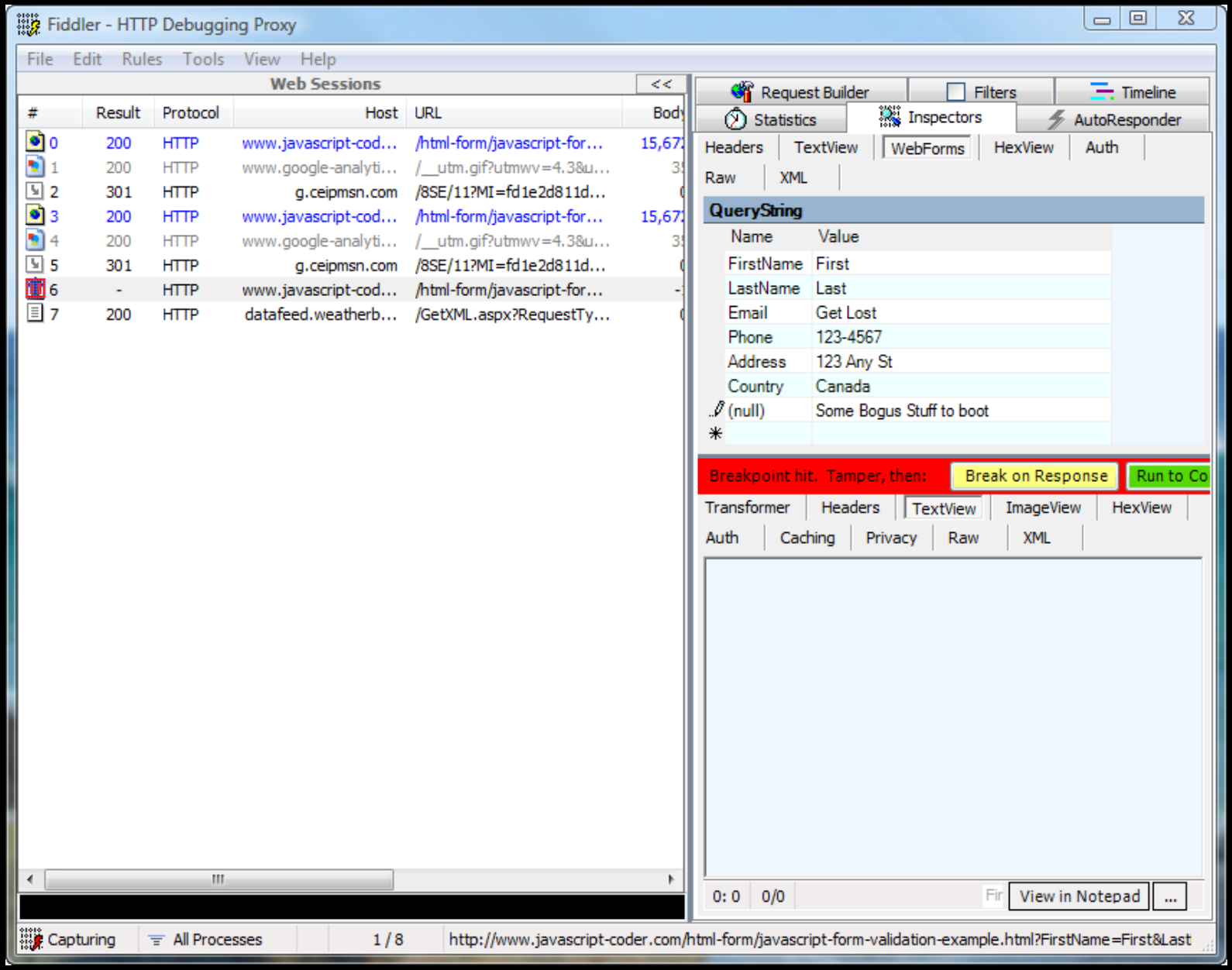
- You can do the same thing with web pages, simply view source, save it, modify it and reload it for example.
- Or you can use something Fiddler

Network Injection

- Say that is some nice client side validations you have there, too bad they don’t stop a hacker.
- The Victim limits my choices for country and email/phone formats etc, but does it really? I hope they have server side checks.


JavaScript Form Validation example - Windows Internet Explorer
http://www.javascript-coder.com
Live Search
Search web..
Favorites
Snagit
JavaScript Form Validation exa...
Page
Tools
First Name
First
Last Name
Last
EMail
My@email.com
Phone
1234567
Address
123 Any St
Country
Albania
[choose yours]
Albania
Algeria
American Samoa
Andorra
Angola
Anguilla
Antarctica
Antigua And Barbuda
Argentina
Armenia
Aruba
Back to artic
Easy
2003-2008 Simfatic Solutions. All rights reserved.
Internet | Protected Mode: On
100%


Fiddler - HTTP Debugging Proxy
File Edit Rules Tools View Help
Web Sessions
Result Protocol Host URL Body
0 200 HTTP www.javascript-cod... /html-form/javascript-for... 15,67
1 200 HTTP www.google-analyti... /__utm.gif?utmwv=4.3&u...
2 301 HTTP g.ceipmsn.com /8SE/11?MI=fd1e2d811d...
Request Builder
Filters
Timeline
Statistics
Inspectors
AutoResponder
Headers TextView WebForms HexView Auth
Raw XML
QueryString
Name Value
FirstName First
LastName Last
Email My@email.com
Phone 1234567
Address 123 Any St
Country 008
Response is encoded and may need to be decoded before inspectio
Transformer Headers TextView ImageView HexView
Auth Caching Privacy Raw XML
Response Headers [Raw] [Header Definitions]
HTTP/1.1 200 OK
Cache
Date: Wed, 07 Jan 2009 12:28:02 GMT
Entity
Content-Type: text/html
Miscellaneous
Server: Apache/2.2.10
X-Powered-By: PHP/5.2.6
Transport
Connection: Keep-Alive
Keep-Alive: timeout=5, max=100
Transfer-Encoding: chunked
Capturing
All Processes
1 / 3
http://www.javascript-coder.com/html-form/javascript-form-validation-example.html?FirstName=First&Last


Fiddler - HTTP Debugging Proxy
File Edit Rules Tools View Help
Web Sessions
Result Protocol Host URL Body
0 200 HTTP www.javascript-cod... /html-form/javascript-for... 15,67
1 200 HTTP www.google-analyti... /__utm.gif?utmwv=4.3&u... 3
2 301 HTTP g.ceipmsn.com /8SE/11?MI=fd1e2d811d...
3 200 HTTP www.javascript-cod... /html-form/javascript-for... 15,67
4 200 HTTP www.google-analyti... /__utm.gif?utmwv=4.3&u... 3
5 301 HTTP g.ceipmsn.com /8SE/11?MI=fd1e2d811d...
6 - HTTP www.javascript-cod... /html-form/javascript-for...
7 200 HTTP datafeed.weatherb... /GetXML.aspx?RequestTy...
Request Builder
Filters
Timeline
Statistics
Inspectors
AutoResponder
Headers TextView WebForms HexView Auth
Raw XML
QueryString
Name Value
FirstName First
LastName Last
Email Get Lost
Phone 123-4567
Address 123 Any St
Country Canada
(null) Some Bogus Stuff to boot
Breakpoint hit. Tamper, then:
Break on Response
Run to Co
Transformer Headers TextView ImageView HexView
Auth Caching Privacy Raw XML
0: 0
0/0
View in Notepad
...
Capturing
All Processes
1 / 8
http://www.javascript-coder.com/html-form/javascript-form-validation-example.html?FirstName=First&Last

Network Injection

- What about HTTPS?
- Since Fiddler (Burp, etc) are proxies they allow you to do a man in the middle attack and thus get around HTTPS (you will see a warning that the cert is invalid).

Protecting Communications

- WCF Security
- In .NET 3.5 secure communications with Windows Communication Foundation
  - Message Security
    - End-to-End using WS-Security
    - Protocol-independent
    - More secure
    - Slower
  - Transport Security
    - Point-to-Point using e.g. HTTPS
    - Widely adopted
    - Generally faster
  - Both (TransportWithMessageCredential)
- WCF can also be used for access control and auditing

Security Testing

- Network Traffic
  - Never assume that network traffic is golden as intercepting / tampering with network traffic is trivial.
    - WireShark / NetMon
    - Fiddler
    - Etc

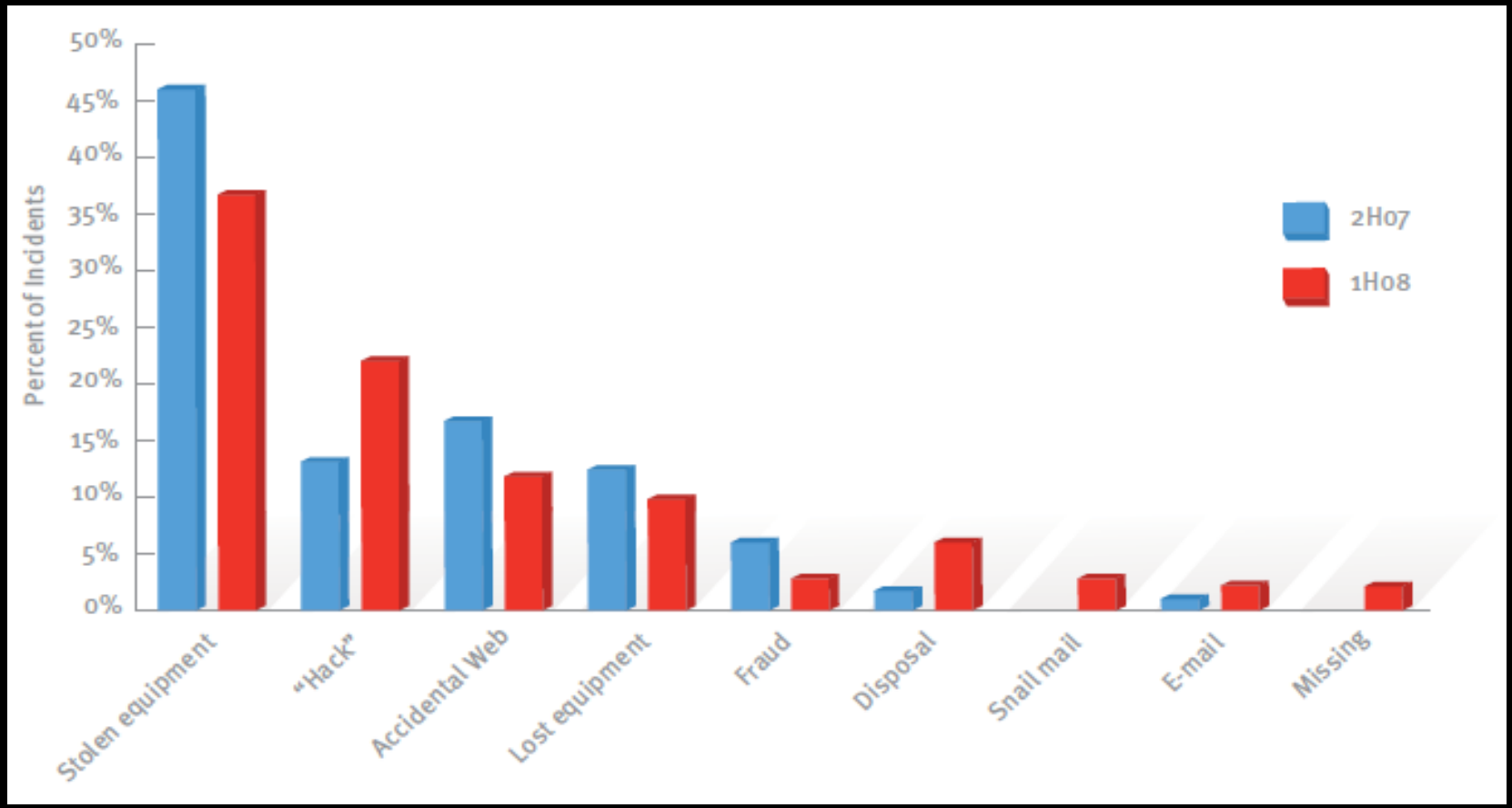
Data Protection is Key

- DP is at the heart of all defences
- It has to work when everything failed
- DP is typically the only defence when physical security has been broken
- You need Data Protection in your application's architecture!

- Hacking is responsible for less then 25% of Security Breaches, Stolen Equipment is the leading cause at over 35%

Data Protection is Key

Security breach incidents by type, expressed as percentages of the total, 2H07 and 1H08



Protecting Stored and Persisted Data

- Cryptography Next Generation
  - CAPI 1.0 has been deprecated
  - CNG is simpler than CAPI
  - Supports modern crypto (NSA Suite-B)
    - SHA-256, and AES with 128-bit keys - 'Secret' level
    - SHA-384, and AES with 256-bit keys - 'Top Secret' level
    - NSA Suite A - highly sensitive communication and critical authentication systems (unpublished and will likely remain that way)
  - Wrapped in System.Security.Cryptography in 3.5

Protecting Access

- Two models: Authentication and Authorization
- Some can be mixed
- Choice depends on deployment context
  - CardSpace + Claims + ADFS
  - Roles
  - AD/ADAM
  - AzMan
  - Certificates
  - Other Security Principals
- See 7 “Laws of Identity” on www.identityblog.com

- The only safe solution for the broad spectrum of computer users is one in which they cannot give away their secrets.

Fuzz Testing

- 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
- Tests your Injection protection
- Remember the 49% of all attacks were based on overly trusted data
- Over 70% of Security Vulnerabilities Microsoft patched in 2006 were found by fuzzing

Fuzz Testing

- 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, build or download fuzzers
  - Dumb
  - Smart
- You have to automate
  - Watch code coverage if you can
- Dedicate a computer or three

Fuzz Demo

Fuzz Testing

- Fuzz Testing Tools generally suck as they are poorly designed, poorly written, poorly tested and generally just suck.
  - You are entering the domain of hacking tools and most of them suck as they are poorly designed, poorly written, poorly tested and rather narrow in scope or target, but I believe they will get better.

Tools that can Help

- Microsoft SDL Tools Repository
  - Threat Modeling Tool
  - FxCOP
  - SiteLock
  - Code Analysis for C/C++
  - Anti-Cross Site Scripting
  - Code Analysis Tool for .Net
  - Banned.h
  - http://msdn.microsoft.com/en-us/security/cc421514.aspx
  - http://www.Microsoft.com/SDL
- Windows Sysinternals
  - http://technet.microsoft.com/en-us/sysinternals/default.aspx

Testing Tools that can Help

- .NET Reflector
  - http://www.red-gate.com/products/reflector/
- WireShark
  - http://www.wireshark.org
- Fiddler
  - http://www.fiddler2.com/fiddler2/
- OWASP (Open Web Application Security Project)
  - http://www.owasp.org/index.php/Main_Page
- The Art Of Fuzzing
  - http://www.theartoffuzzing.com
- Fuzzing
  - http://www.Fuzzing.org

Penetration Testing

- The Metasploit Project
  - http://www.metasploit.com
- Top 100 Network Security Tools (2006)
  - http://sectools.org/

Sites you should visit

- SDL Blog
  - http://blogs.msdn.com/sdl/default.aspx
- SecurityFocus
  - http://www.securityfocus.com/
- CERT Security Bulletins
  - http://www.us-cert.gov/cas/bulletins/

Sites you must visit

- Microsoft Security Intelligence Report
  - http://www.microsoft.com/security/portal/sir.aspx
- CWE/SANS TOP 25 Most Dangerous Programming Errors – And How to Fix Them
  - http://www.sans.org/top25errors/

Questions, comments and general abuse

Please get your evaluations in for Book draw

Security Testing Software Made Easy

(or a reasonable facsimile thereof)

Presented by

Blake McNeill

mcneillb@ZingPow.ca

AKA Dr. Evil (the original)