NTNU

Norwegian University of Science and Technology

Assuring trust in .NET assemblies by instrumentation

Hans Oluf Hagen

Master in Information Security
Submission date: June 2017
Supervisor: Basel Katt, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Preface

The master thesis took part during the spring semester in 2017 at NTNU. The thesis was suggested by my employeer SuperOffice AS. SuperOffice develops and sells SuperOffice CRM. The solution has been traditionally an onsite solution, but is now available as a Software-as-a-Service-solution. Unfortunately, the move to the cloud reduced the level of customization options available to the customers. Is it possible that SuperOffice can trust assemblies written by partners adequately to be executed in-process on the multi-tenant environment and if so, how?

The reader should be familiar with programming and especially .NET programming. A short description of high level C# to Common Intermediate Language is given.

01.06.2017

Acknowledgment

I would like to thank SuperOffice AS for supporting me during the master's programme and for suggesting the thesis. SuperOffice partners contributed with samples of their real-world customizations so I could assess the applicability (and limitations) of my thesis.

Finally, I wish to give a special thanks for ideas, discussions, quarrels and feedback to Jens Glattetre and Dr Marek Vokáč, both experienced architects and wizards at SuperOffice AS. Cake rights are hereby granted.

HOH.

Abstract

SuperOffice is a software company developing SuperOffice CRM software. The traditional hosting option is the on-site solution where the customer is responsible for hosting and maintenance. SuperOffice CRM is quite extensible due to differences between enterprises' requirements and processes. The move from an on-site installation to an online installation reduces the level of customization available. Especially, executing custom code written by third parties in on-site solutions was the customer's responsibility. The shift to online moves this responsibility to SuperOffice, resulting in unacceptable risk towards the installation, other installations and the online environment. Is it possible to trust the custom code written by third-parties? If so, how? This thesis looks at how instrumentation techniques can be used for analyzing and instrumenting .NET assemblies in order to get assurance they do behave in a predictable manner and with acceptable risk to the customer installation, other installations and the environment. Analyzing the custom assemblies with static analysis techniques reveal the potential interactions between the custom assembly, the .NET runtime and the rest of the system. Runtime enforcers can be added to calls to methods which can only be conditionally executed.

However, there are several threats to an instrumentation engine such as this. There are indeed many ways of fooling it; Platform Invoke, ForwardedTypes and Mixed-Mode assemblies to mention a few.

Contents

List of Figures

List of Tables

1 Introduction

SuperOffice is a software company developing SuperOffice CRM software. The traditional hosting option is the on-site solution where the customer is responsible for hosting and maintenance. SuperOffice CRM is quite extensible due to differences between enterprises' requirements and processes. The move from an on-site installation to an online installation reduces the level of customization available. Especially, executing custom code written by third parties in on-site solutions was the customer's responsibility. The shift to online moves this responsibility to SuperOffice, resulting in unacceptable risk towards the installation, other installations and the online environment. Is it possible to trust the custom code written by third-parties? If so, how? This thesis looks at how instrumentation techniques can be used for analyzing and instrumenting .NET assemblies in order to get assurance they do behave in a predictable manner and with acceptable risk to the customer installation, other installations and the environment.

Analyzing the Common Intermediate Language (CIL) generated by a compliant compiler reveal which features, assemblies, types, methods, properties and fields are being used by the custom assembly. Re-writing assemblies with inline reference monitors are demonstrated in the thesis. An user-defined policy sets the rules for what is allowed and what is denied including support for black- and white-listing. Violations of the rules can lead to reduction of a trust score. The final trust score is calculated when all rules have been evaluated. Unfortunately, we were unable to validate how the trust score should be modified.

Sets of policies can be created by comparing real-world samples, collected from SuperOffice partners and subsidiaries alongside with the API and the threats to the system. Unfortunately, the API made available to partners is quite large, spanning over almost 12 000 classes and over 64 000 methods. Creating policies by manually inspecting types and methods is a daunting and time-consuming task yielding little in return. Policies should be written with clear boundaries of what is accepted and what is not.

2 Methodology

The main research method is a qualitative exploratory case study with an experiment. A case study is suitable when we want to examine data within a specified limited context [1, 2].

In our project, we wish to see how instrumentation techniques can be used to give assurance and trust to .NET assemblies. In the thesis we limit the scope to assemblies created by SuperOffice partners and subsidiaries.

The project started with a literature review which formed the basis of the test application GuardiNET. A focus group was assembled and a workshop was held. The purpose of the workshop was to identify the security challenges of running code from partners in SuperOffice CRM Online. The preliminary application was demonstrated to the focus group for further input in the same workshop. This preliminary application was capable of detecting only external method calls (explained in 4.2.6). It also included a trust evaluation model where violations would decrease a trust score, and compliance could raise the trust score. However, the group raised questions about HOW trust scores should be modified.

To validate the application, real world customizations from partners were collected and analyzed. Identification of features which would break the requirements for reference monitors were performed.

3 Background

3.1 Abbreviations and acronyms

CIL	Common Intermediate Language - part of ECMA-335
CLI	Common Language Infrastructure - part of ECMA-335
CLS	Common Language Specification - part of ECMA-335
CRM	Customer Relationship Management
CTS	Common Type System - part of ECMA-335
IRM	Inline reference monitor
JSON	JavaScript Object Notation
SaaS	Software-as-a-Service - cloud computing delivery model
SuO	SuperOffice AS
VES	Virtual Execution System - abstract machine executing CLI compliant code (ECMA-335)

3.2 Microsoft .NET

Microsoft .NET framework is an implementation of Common Language Infrastructure defined in ECMA-335[3]. High-level languages such as C#, Visual Basic and F# are compiling their source code to CLI compliant assemblies. Assemblies written in Visual Basic can therefore be used in C# source. The standard defines Common Type System, Virtual Execution System and metadata[1]. The standard includes verification rules. Assemblies that are not verifiable according to CLI will result in a runtime exception if executed. CLI is a managed framework which guarantees type safety and garbage collection. A type can not be casted to a non-compatible type. Vulnerabilities such as Use-After-Free is therefore not possible in this framework.

3.2.1 From high-level to Common Intermediate Language

Listing 3.1: Program which prints Hello to standard output

```
static class Program
{
    static void Main(string[] args)
    {
        var name = args.FirstOrDefault();
        SayHello(name);
    }

```

[1]Common Language Specification is also defined, but not applicable in this thesis.

```
    static void SayHello(string name)
    {

        Console.WriteLine($"Hello {name}");
    }
}
```

Listing 3.2: 3.1 after compilation

```
.class private abstract auto ansi sealed beforefieldinit Hello.Program
       extends [mscorlib]System.Object
{
  .method private hidebysig static void  Main(string[] args) cil
      managed
  {
    .entrypoint
    // Code size       16 (0x10)
    .maxstack  1
    .locals init ([0] string name)
    IL_0000:  nop
    IL_0001:  ldarg.0
    IL_0002:  call       !!0 [System.Core]System.Linq.Enumerable::
        FirstOrDefault<string>(class [mscorlib]System.Collections.
        Generic.IEnumerable`1<!!0>)
    IL_0007:  stloc.0
    IL_0008:  ldloc.0
    IL_0009:  call       void Hello.Program::SayHello(string)
    IL_000e:  nop
    IL_000f:  ret
  } // end of method Program::Main

  .method private hidebysig static void  SayHello(string name) cil
      managed
  {
    // Code size       19 (0x13)
    .maxstack  8
    IL_0000:  nop
    IL_0001:  ldstr      "Hello {0}"
    IL_0006:  ldarg.0
    IL_0007:  call       string [mscorlib]System.String::Format(string
        ,
                                                                 object
                                                                     )
    IL_000c:  call       void [mscorlib]System.Console::WriteLine(
        string)
    IL_0011:  nop
    IL_0012:  ret
  } // end of method Program::SayHello

} // end of class Hello.Program
```

Listing 3.1 lists the source for a program writing Hello and the first argument given to the program. Upon execution of the program, it will call the method SayHello with the first argument as parameter. The result after compilation can be seen in a human-readable IL format in listing 3.2. Readers familiar with assembly will see many differences between assembly and CIL. Assembly have registers for general use and stack registers. Registers do not exists in CIL. Stacks do exist, but in another form than assembly. Every methods have their own evaluation stacks with an optional limit. As we can see from the listing, SayHello has a maxstack value set to 8 and Main has 1. Pushing more values to the stack in these methods will result in a runtime exception. The instructions "ldarg.0", "ldloc.0" and "ldstr" push a value onto the evaluation stack. Arguments and local variables are not part of the evaluation stack unless their values are pushed onto the stack. ldarg.index pushes the argument with index onto the stack. "ldloc.index" pushes the value of the local variable at index onto the stack. In this example, ldloc.0, is the variable "name". Arguments to methods are pushed from left to right order. Looking at SayHello, we can see the first push is the "Hello 0" string followed up by the first argument to SayHello. Calling methods are quite declarative in CIL. The call to "string [mscorlib]System.String::Format(string, object)" is decomposed to:

- ReturnType: string
- Assembly: mscorlib
- Type: System.String
- Method: Format
- Arguments: string, object

Upon return, the arguments are popped from the evaluation stack, and the return value is pushed onto the evaluation stack of the calling method. The stack transition for calling the format function is: ..., "Hello 0", name $\rightarrow$ ..., "Hello name" (where the rightmost element is the top of the stack). Calls to instance methods (non-static methods) pushes the instance object as the first argument.

An assembly contains metadata describing it. The metadata includes information about which assembly references are used, which external types are referenced and which methods are used. We will, in this thesis, be using Mono.Cecil designed with an easy to use API for reading and writing .NET assemblies.

3.3 Reference monitors

Reference monitors is a model where an abstract machine mediates access decisions between subjects and objects. Anderson [4] formulated three requirements for a reference monitor:

1. Complete mediation - All access must be through the reference monitor
2. Tamper resilient
3. Verifiable - The RM must be small enough for proper inspection and tests to ensure correctness

3.4 Assurance and trust

The proliferation and adoption of software library repositories such as NPM, RubyGems, Cargo and NuGet have made third party software repositories an integral part of the tool box to any modern software developer today. Historically, software developers have been re-inventing the wheel instead of re-using other technologies/methodologies. Collaboration across companies, teams and projects have become a part of the workflow. In some cases, maybe even unintentionally. StackOverflow and StackExchange are rallying points for developers where questions are asked and answered with varying degree of quality. Sample code from these sources are even promoted to production code without transformation. With these changes, software assurance or due dilligence of the third-party libraries are lacking behind. NuGet, which is the main repository used for .NET libraries, have few or no support for publisher verification or ensuring the integrity and provenance of packages. The NuGet team has indeed identified these gaps in security and started implementing better verification system in Spring 2017 [5].

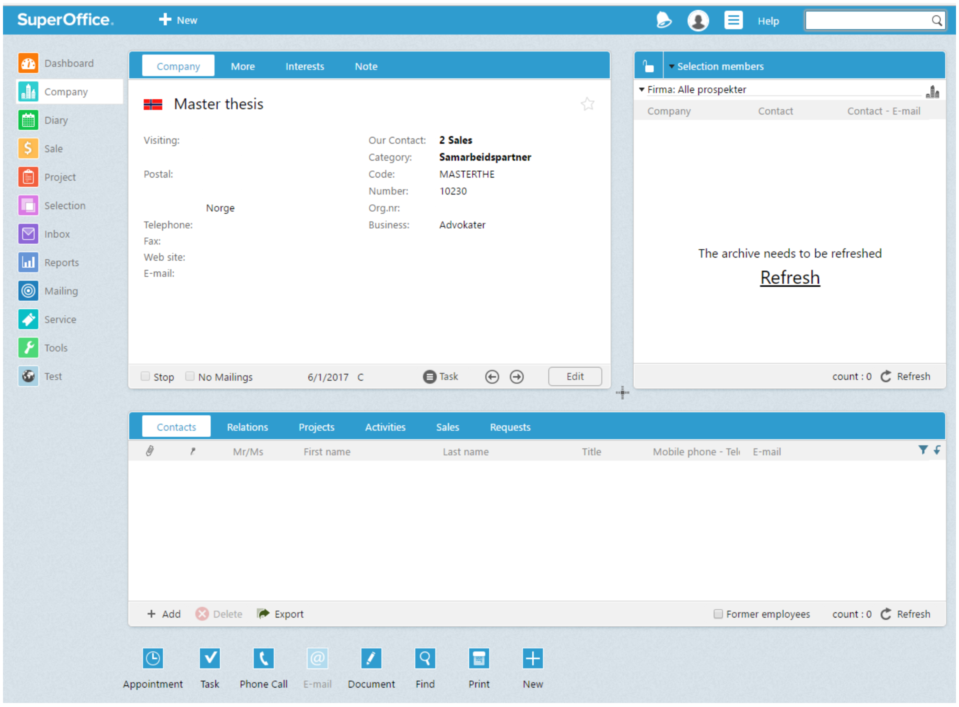
How do developers do due dilligence or software assurance on third-party libraries? After interviewing and browsing the web for answers, the main answer is clear: Developers are positive trusters, looking for traits increasing or confirming trust. By looking at the reputation and popularity given by other developers, developers decide to trust a package. The same application of reputation-based trust can be found in other areas of the Internet, for instance reviews on market places and even on StackOverflow. Evidence-based trust for instance by decompiling and manually inspecting packages are rarely performed by the focus group.

3.5 SuperOffice

SuperOffice is a Norwegian-based software company specializing in Customer Relationship Management (CRM) software. The main solution, SuperOffice CRM, is facilitating marketing, sales and service processes. SuperOffice CRM is highly extensible to align the software to the companies' requirements and processes.

Traditionally, SuperOffice CRM was installed on servers on customer premises (onsite installation) and maintained by the customer. With the introduction of cloud computing, SuperOffice made it's CRM solution available as a Software-as-a-Service (online installation) solution where SuperOffice is maintaining hardware, operating system and SuperOffice CRM.

Figure 1 shows a SuperOffice CRM application without any customizations as presented to the end-user.



Master thesis

Visiting:		Our Contact:	**2 Sales**
		Category:	**Samarbeidspartner**
Postal:		Code:	MASTERTHE
		Number:	10230
	Norge	Org.nr:	
Telephone:		Business:	Advokater
Fax:			
Web site:			
E-mail:			

6/1/2017 C

Firma: Alle prospekter

Company	Contact	Contact - E-mail

The archive needs to be refreshed

Refresh

count : 0 Refresh

Mr/Ms	First name	Last name	Title	Mobile phone - Tel	E-mail

count : 0 Refresh

Figure 1: SuperOffice CRM Web client focused on the contact card.

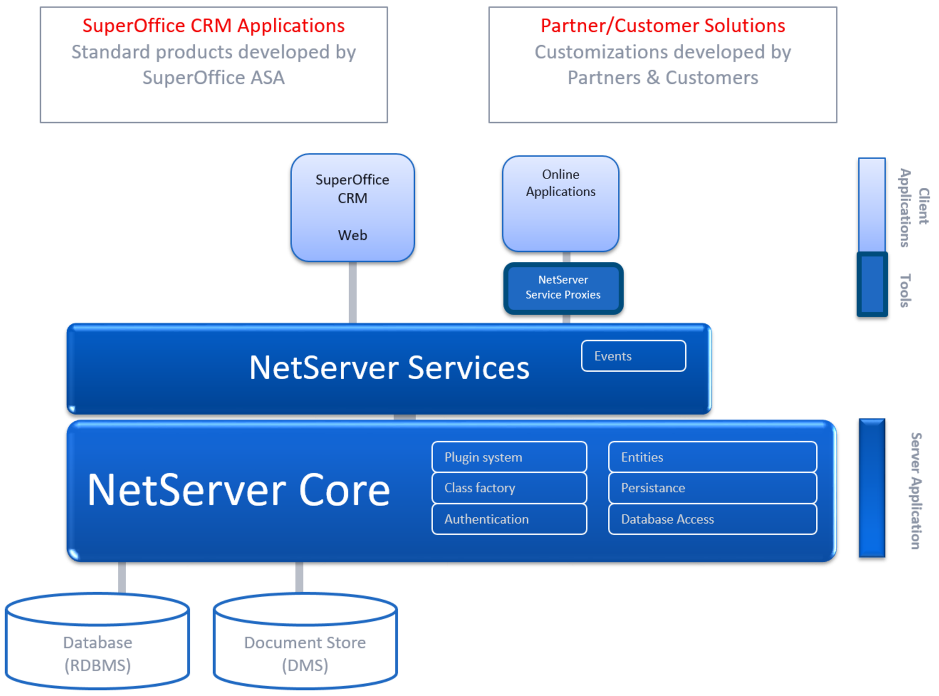


Figure 2: SuperOffice CRM architecture

3.5.1 Architecture

The SuperOffice CRM application is a multi-tiered web application running on IIS and developed in .NET. Figure 2 depicts a simplified model of the architecture. At the top, you have the frond-end client, SuperOffice CRM Web. The front-end client can be customized, but this is out of scope for this thesis. The client contains only front-end specific logic. The next layer is NetServer Services. This layer contains facades called agents which are used to further communicate down the stack. For instance, the agent called ContactAgent can be used for fetching and updating the contacts (companies) in the installation. In this layer there are also support for event-based scripting where scripts can be triggered on events. NetServer core is the main component and holds all business logic and persistence.

The architecture makes extensive use of a plugin-based system called ClassFactory. ClassFactory is an implementation of the Service Locator pattern where the implementation asks for an instance of a type or abstraction/interface, but can get a derived instance of the type. Adding new document-plugins give the customer the ability to store and get documents from other sources than the built-in typem, for instance through Microsoft Azure, Google or Office 365. Adding faulty authentication plugins may result in missing access checks. All loaded assemblies are registered in the ClassFactory (including assemblies from partners).

3.5.2 Differences between on-site and online

On-site installations have no restrictions in what partners can customize in the solution. They are free to add code to any layer in the architecture from top to bottom. Customizations in Online-installations, called Online apps, can only communicate with SuperOffice through Netserver Service Proxies, which are HTTP-based services. All online apps require registration in SuperOffice AppStore and have restrictions in which agents they are allowed to use. Registrations require information about the partner and application. Before online apps are published to the AppStore, they must be certified. The certification process includes a security check from a third-party security firm. Finally, the online apps also require the consent from the customer to be accessible from the SuperOffice CRM client.

3.6 Related work

Instrumentation has been widely used in software engineering to verify the correctness of the software, performance measurements or tracing. It has additionally been used to protect against typical programming errors such as poor memory management which lead up to buffer overflows or Use-After-Free vulnerabilities [6, 7] which again result in remote-code-executions. Instrumentation techniques such as Code-Flow-Integrity/Control-Flow-Guard analyzes the call-graphs in the binaries to see which methods are allowed to call a method, and then enforce this policy during run-time execution. The program terminates if there are a policy violation [8]. The focus on instrumenting native binaries is protecting against these memory corruptions.

Instrumentation can be approached as statically or dynamically. Static instrumentation analyzes the binaries or assemblies on disk. Dynamic instrumentation analyzes the assemblies/binaries during execution. Both approaches are complimentary to each other. The static instrumentation approach is not precise, but does not have the performance degradation imposed by dynamic instrumentation[6, 7]. Dynamic instrumentation is known to be more precise than static approaches, especially towards obfuscation techniques.

Java and .NET are not susceptible to memory corruption bugs (except when p/invoking or using unsafe methods). These are managed platforms with built-in memory management. Instrumentation on these platforms are not focused on memory corruption, but rather on access policies.

Research on instrumentation techniques on Android and iOS are popular in the academic community. [9] has created "AspectDroid". The system use both static and dynamic instrumentation to prevent the android apps for accessing sensitive APIs. [10] created an enterprise-enforcing policy layer to Android, giving enterprises better control over the security policies through a central policy center for all devices. The approach required however the devices to be rooted. Apps in Android and iOS are given access to resources based on the permissions in the manifest (which the user need to give consent too). However, many of the apps might get more privilege access than they need, or they can consume third-party libraries which then inherit the same permissions as the app. As explained by [11], the main revenue for app-developers are by ads. Unfortunately, many of the ad-libraries do not respect the privacy of the user, and misuse the permissions granted to the original app, to collect and send information to their own servers. [11] focused the research on how to de-privilege the ad-libraries to either deny the API calls, or to return false information. One example of false information is to return an incorrect location than the correct on the location services.

The concepts used on Android and iOS can be used to determine what type of policies are applicable for .NET assemblies.

Instrumentation techniques on .NET have received little attention from the academic community. There are however, two papers which stand out. [12] designed and implemented a library called RAIL for analyzing and weaving code into an assembly. [13] designed "Mobile", an inline reference monitor. Policies in Mobile are expressed by w-regular expressions representing the call graph both on global and object level. This allows for policies such as "you can call downloadFile, but you can't call uploadFile afterwards). The expressions did not inspect the parameters to the methods.

[14] tested naive malware samples written in .NET on popular anti-virus programs. The malware took screenshots, received binaries from Command & Control servers and executed programs locally. The malware were configured to startup when the user logged-in. The majority of anti-virus programs failed to detect even the simplest samples. Obfuscation or any anti-detection techniques were not employed.

4 GuardiNET

GuardiNET is the instrumentation tool developed during the thesis. The tool has both static and dynamic instrumentation capabilities. The core engine uses extractors to find relevant features from the assembly. One example of extractors is the ExternalMethodInvocation extractor. EMI locates all calls to external .NET methods from within the assembly. The decision whether the extracted feature is violating the policy or not, is decided by evaluators configured by the security policy.

4.1 Policies

GuardiNET is configured by a user-defined policy file describing the name, description, default access and rule definitions. Default access determines in which order the rules will be evaluated. If default access is set to Deny, all rules where access is set to deny, will be evaluated first, before any allow-rules are evaluated. Each evaluator type may create a default rule depending on the default access. For instance, EMI will create a default deny rule denying all external method invocations if default access is set to deny. This is therefore black- and white-listing respectively. The policy in listing 4.1 only allows external method calls to the String class in the mscorlib assembly. Since the default access is set to "Deny", all other extracted features where the evaluator creates a default deny rule, will result in a policy violation.

Access levels are defined to be Allow, Deny and DontCare. Severity is of the ordinal scale: Critical, High, Medium, Low and None with decreasing severity.

Listing 4.1: String only policy (Trust section omitted)

numbers

```
{
  "Name": "StringOps only",
  "Description": "Allow only string methods",
  "DefaultAccess": "Deny",
  "Rules": [
    {
      "Name": "Allow all System.String",
      "Type": "ExternalMethod",
      "Access": "Allow",
      "Severity": "None",
      // GenerateEnforcers: true,
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*",
        "ExternalMethodName": "System.String.*",
        "IsRegex": true
```

```
            }
        }
    ]
}
```

4.2 Extractors and evaluators

Table 1 shows a summary of all supported extractors and evaluators including default allow and deny rules.

Evaluators decide whether a feature is matching the rule or not. The decision consists of a severity level, access verdict and a comment. If the feature is matching the criteria set, the evaluator must return the access and severity set in the policy. Non-matched features will return DontCare as access.

An allow rule have access set to allow and severity set to none. Deny rules have access set to deny and severity set to an appropriate severity level. By convention, default deny rules have severity set to critical.

4.2.1 AssemblyReference

Extractor extracts all assembly references from the assembly file. *Evaluator* is configured with a regular expression of the assembly name.

4.2.2 Attribute

Extractor extracts all custom attributes on assembly, modules, classes and methods in the assembly. *Evaluator* can be configured based on the attribute full type name and target. Custom attributes are typically used in IoC[1]-containers and/or reflection-based type invocations. SuperOffice CRM uses custom attributes in class-, plugin and injection factories. Assemblies using custom attributes can therefore change intended behaviour depending on context.

4.2.3 DigitalSignature

Extractor extracts if the assembly is signed or not. The evaluator can be configured to accept or reject signed/non-signed assemblies. Digital signatures are used for verifying the provenance and integrity of assemblies. AppLocker and Microsoft SmartScreen may deny or warn the user of unsigned applications.

4.2.4 ExternalDelegate

Extractor extracts all usage of external delegates. External delegates are identified by the CIL instructions ldsd* and ldf*. *Evaluator* has the same criteria as EMI.

[1]Inversion-of-Control

4.2.5 ExternalField

Extractor extracts all usage of fields in external assemblies. External fields are identified by the CIL instructions "Ldfld", "Ldsfld", Stfld", "Stsfld" with a field reference to an external assembly. *Evaluator* can be configured with the name of the assembly and the type name of the field.

4.2.6 ExternalMethodInvocation

Extractor extracts all calls to external assemblies. External calls are identified by the CIL instructions (call, calli and callvirt) with a method specifier to an external assembly. *Evaluator* can be configured with the external assembly name and the type name including the method signature.

4.2.7 ForwaredTypes

Extractor extracts exported types marked as forwarders in an assembly. *Evaluator* evaluates the forwarded assembly and type name. Malicious assemblies may forward critical types to other assemblies trying to circumvent any naive EMI rules. Default rule is to deny this functionality.

4.2.8 InternalCalls

Extractor extracts all internal calls to the runtime. *Evaluator* evaluates the name of the internal runtime call. Based on experiments, the runtime only accepts internal calls from system libraries. However, the method on how the runtime verifies the origin of the call has not been determined. It may be possible to spoof this origin.

4.2.9 MixedMode

Extractor extracts non-managed code from the assembly. *Evaluator* evaluates if mixed mode is acceptable or not. Accepting mixed mode will effectively bypassing any security controls imposed by GuardiNET. A mixed-mode assembly may have a DllMain method which will be executed when the assembly is loaded.

4.2.10 ModuleInitializer

Extractor extracts module initializers from the assembly. Module initializers are identified by the module name "<Module>" and a static constructor named ".cctor" and the flags special name and runtime special name. *Evaluator* has no criteria except for access and severity. Since high level languages such as C#, F# and Visual Basic do not support module initializers, any presence of these markers can be interpreted as unwanted behaviour. Presence of module initializers has to be explicitly approved by the operator.

4.2.11 PInvoke

Extractor extracts methods marked with PInvokeImpl. *Evaluator* evaluates the module name and the entry point of the platform invocation. Module name is a dynamic loadable library (DLL) and the entry point is a function in this DLL. For instance, ..::.. Platform invocations is the lowest type (hierarchically) of method invocations possible in .NET. Failure to adequately control platform invocations may lead to total loss of security.

4.2.12 StaticField

Extractor extracts all static non-constant and non-compiler generated fields in the assembly. *Evaluator* can use the assembly name, type name, and the declaring type name to evaluate whether the static field is accepted or not. Static fields can leak information in a multi-tenant environment.

4.2.13 String

Extractor extracts all ldstr instructions. *Evaluator* can use base64decode and use regular expressions on the strings. Strings may reveal secrets or intentions to use communication, file access or encryption. There are however no default allow nor deny policies of this type.

4.2.14 Type

The type feature allows the policy writer to have a specific policy for a type. The evaluators available in for rules in this policy are a subset of the standard ones, except those who are only checking assemblies. ExternalMethodInvocations, static fields, attributes and so on can be used in this type of policy.

Extractor extracts all types in the assembly. *Evaluator* can be configured to apply a specific policy for type belonging to an assembly, inheriting from a given type, implements an interface or have certain custom attributes. The idea behind this rule type is that not all interactions goes from the untrusted assembly to the trusted assemblies, but the interactions can be bidirectional. The trusted API may call into the untrusted assembly, for instance through Inversion-of-Control/Dependency Injection/Service Locator patterns. The plugin-based architecture in SuperOffice CRM will call into other assemblies as described in 3.5.1. For instance types marked with the SoCredentialPluginAttribute and implementing ISoCredentialPlugin can circumvent the authentication process in SuperOffice. When a ExternalMethodExtractor is running in this mode, it will try to find all candidates matching any calls if the opcode callvirt to an external assembly is found.

4.3 Dynamic instrumentation

Name	Description	Default deny
AssemblyReference	References to external assemblies	Deny All
Attribute	Custom attributes	Deny all
DigitalSignature	Verification of any digital signature	NA
ExternalDelegate	Delegate targeting external methods	Deny all
ExternalField	Reference to external fields	Deny all
ExternalMethodInvocation	Invocations to methods in external assemblies	Deny all
ForwardedTypes	Usage of forwarded types	Deny all
InternalCalls	Calls to internal methods in runtime	Deny all
MixedMode	Mixing native and managed assemblies	Deny all
ModuleInitializer	Module initializer code	Deny all
PInvoke	Platform method invocations	Deny all
StaticField	Detection of static fields in assembly	Deny all
String	String detection	NA
Type	Enforce specific policy for given type	NA

Table 1: Supported enforceable features

Listing 4.2: Generated enforcer source file

```
using System;
using System.Linq;
namespace GuardiNet.Instrumentation
{

  internal class GuardEnforcerAttribute : Attribute
  {
    public string AssemblyName { get; set; }
    public string MethodName { get; set; }
    public bool WarnOnSimilar { get; set; }

    public GuardEnforcerAttribute(string assemblyName, string
        methodName, bool warnOnSimilar)
    {
        AssemblyName = assemblyName;
        MethodName = methodName;
        WarnOnSimilar = warnOnSimilar;
    }
  }

  public static class Enforcer
  {
    [GuardEnforcer("System, Version=4.0.0.0, Culture=neutral,
        PublicKeyToken=b77a5c561934e089", "System.Net.WebRequest
        System.Net.WebRequest::Create(System.String)", true)]
    public static System.Net.WebRequest Guard_WebRequest_Create(System
        .String p0)
    {
      //TODO: Add check and call the original method here
      throw new NotImplementedException();
    }


    [GuardEnforcer("System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089", "System.Net.WebResponse System.Net.WebRequest::GetResponse()", true)]
    public static System.Net.WebResponse Guard_WebRequest_GetResponse(System.Net.WebRequest originalObject)
    {
      //TODO: Add check and call the original method here
      throw new NotImplementedException();
    }


    [GuardEnforcer("System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089", "System.Net.HttpStatusCode System.Net.HttpWebResponse::get_StatusCode()", true)]
    public static System.Net.HttpStatusCode Guard_HttpWebResponse_get_StatusCode(System.Net.HttpWebResponse originalObject)
    {
      //TODO: Add check and call the original method here
      throw new NotImplementedException();
    }


    [GuardEnforcer("System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089", "System.Threading.Tasks.Task`1<System.Net.WebResponse> System.Net.WebRequest::GetResponseAsync()", true)]
    public static System.Threading.Tasks.Task`1<System.Net.WebResponse> Guard_WebRequest_GetResponseAsync(System.Net.WebRequest originalObject)
    {
      //TODO: Add check and call the original method here
      throw new NotImplementedException();
    }
  }
}
```

Listing 4.3: Manually added enforcer code

```
using System;
using System.Linq;
namespace GuardiNet.Instrumentation
{

  internal class GuardEnforcerAttribute : Attribute
  {
    public string AssemblyName { get; set; }
    public string MethodName { get; set; }
    public bool WarnOnSimilar { get; set; }

    public GuardEnforcerAttribute(string assemblyName, string methodName, bool warnOnSimilar)
    {
        AssemblyName = assemblyName;
        MethodName = methodName;
        WarnOnSimilar = warnOnSimilar;
    }
  }

  public static class Enforcer
  {
    [GuardEnforcer("System, Version=4.0.0.0, Culture=neutral,
        PublicKeyToken=b77a5c561934e089", "System.Net.WebRequest
        System.Net.WebRequest::Create(System.String)", true)]
    public static System.Net.WebRequest Guard_WebRequest_Create(System
        .String p0)
    {
        if (!string.IsNullOrWhitespace(p0) && p0.StartsWith("https://
            www.ntnu.no"))
        {
            return System.Net.WebRequest.Create(p0);
        }

        throw new SecurityException("Not allowed to call " + p0);
    }
  }
}
```

GuardiNET has the capability of rewriting assemblies to add runtime enforcers to calls specified by the policy. Listing 4.2 shows the generated source file after using GuardiNET with a policy set to generate enforcers on System.Net.WebRequest methods and an assembly file using these methods. The policy writer may now add any runtime checks to the source file in hers/his favorite IDE. Listing 4.3 shows the modified source file. Only web request calls to https://www.ntnu.no is allowed, other arguments result in a SecurityException. GuardiNET is now executed again with the policy file, assembly and the source file containing any enforcement code. The enforcer source file is compiled into an assembly. All method calls matching the assembly and method name from the GuardEnforcerAttribute will be replaced with the call to the enforcer code. The enforcer code will call the original method[2]. The code for the runtime weaver is available in appendix .1.6.

IRMs listed in [13, 15, 16] are based on traces of method calls and these traces may violate the policy. For instance, the program is not allowed to send data after reading sensitive data or the program is just allowed to use a method n times. GuardiNETs CallSequenceTracker has the same functionality. First, the tracker is initialized with the sequence/trace according to policy. The tracker creates a simplified deterministic finite automaton where the states are the instrumented calls and the edges are the allowed calls from that state including the number of allowed calls to this method. The code for CallSequenceTracker is listed in appendix .1.4.

Type- and InstanceTracker can be used instead of a CallSequenceTracker in situations where only method-invocations to one specific type or instance are interesting. Code is listed in the appendix. Example of usage where only five invocations

[2]which is actually up to the policy writer

of System.Net.WebRequest.Create is allowed, is listed in appendix .1.2.

One problem with tracking instances and the methods they are using is the circular ownership preventing the garbage collector to release this memory when the instance is no longer used (due to the reference from the TypeTracker). By using weak references (ConditionalWeakTable) instead of standard references, there are no longer cyclic ownership issue with the tracking.

4.4 Trust levels

The concept of trust levels was added to GuardiNET. The policy can be configured with trust levels and modifiers. The assembly starts with an initial trust score. Matching rules will lead to the addition of trust score and the match modifier for that severity. For no-matching rules, the no-match modifier is used. The levels, modifiers and scoring mechanism stand as unvalidated in the thesis. It is based on the notion that negative actions have greater negative impact, than positive actions have a positive impact.

Listing 4.4: Trust level section in policy file)

numbers

```
"Trust": {
    "Levels": {
      "Untrusted": {
        "From": 0,
        "To": 10
      },
      "Low": {
        "From": 10,
        "To": 40
      },
      "Medium": {
        "From": 40,
        "To": 80
      },
      "High": {
        "From": 80,
        "To": 100
      }
    },
    "InitialScore": 100,
    "Modifiers": {
      "None": {
        "MatchModifier": 0,
        "NoMatchModifier": 0
      },
      "Low": {
        "MatchModifier": -5,
        "NoMatchModifier": 0
      },
      "Medium": {
        "MatchModifier": -10,
        "NoMatchModifier": 0
      },
      "High": {
        "MatchModifier": -50,
        "NoMatchModifier": 0
      },
      "Critical": {
        "MatchModifier": -100,
        "NoMatchModifier": 0
      }
    }
}
```

5 Results

5.1 Custom assemblies for SuperOffice CRM

The results are based on evaluation of the samples collected from the SuperOffice partners and subsidiaries. The sample size is too small to make any significant conclusions.

The samples range from quite simplistic to complex customizations. Most of the invocations to external assemblies were to APIs in SuperOffice. Some samples communicated with external web services over known .NET HTTP and WCF[1] classes. Any needs of persistence were solved by adding database tables to the existing database using the SuperOffice APIs. Compared with online applications which use the NetServer proxies, the samples tend to use APIs found in a lower layer.

Reflection was used on most of the samples.

SuperOffice CRM exports close to 12000 classes which are accessible to partner code. Together, these types exports close to 64 000 methods. The complete statistics are listed in appendix 2

5.2 Analyzing popular NuGet packages

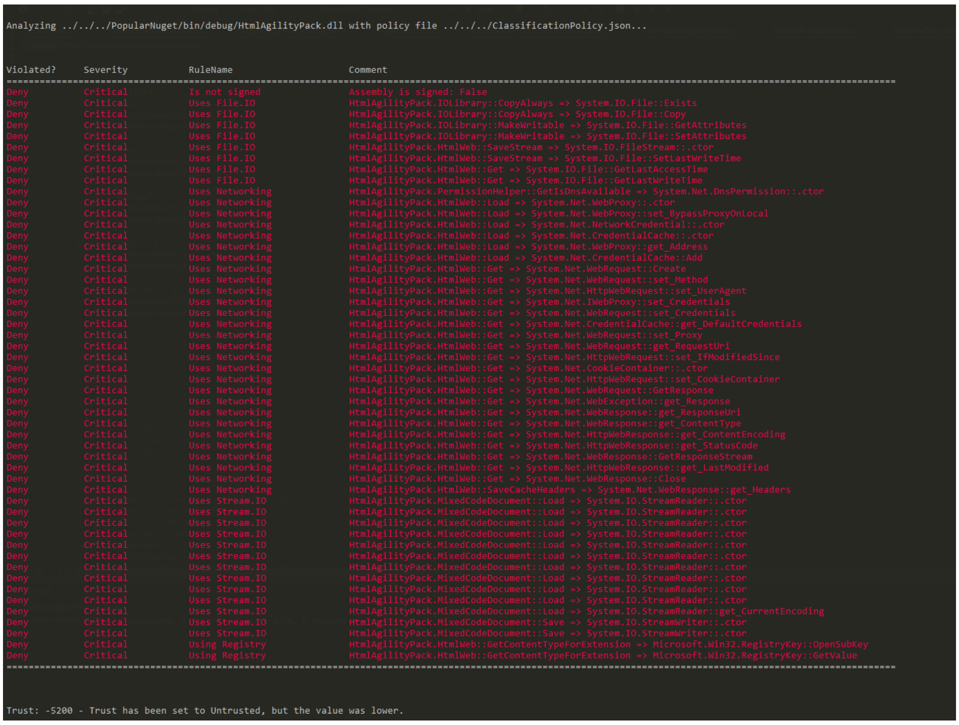
60 NuGet packages were downloaded and analyzed to see if the known features that could be used for circumventing the detection mechanism in GuardiNET are prevalent. There were no occurences of neither module initializer nor mixed mode assemblies. Reflection were used in almost all of the assemblies in varying degree. Few samples emitted code dynamically using the classes in the System.Reflection.Emit[2]. These samples were creating dynamic proxies for types and the existence of emit was expected in these cases.

Assemblies from open-source projects tend to be unsigned unless they come from Microsoft.

Figure 3 shows GuardiNET executing the classification policy listed in appendix .3.1. The intention of HtmlAgilityPack is to parse a DOM tree. As we can see, it has code for accessing registry, file and network operations.

[1]Window Communication Foundation

[2]This is advanced use of .NET. Were no occurences of this in SuperOffice assemblies



```
Analyzing ../../../PopularNuget/bin/debug/HtmlAgilityPack.dll with policy file ../../../ClassificationPolicy.json...

Violated?     Severity           RuleName                     Comment
=================================================================================================================================================================
Deny          Critical           Is not signed                Assembly is signed: False
Deny          Critical           Uses File.IO                 HtmlAgilityPack.IOLibrary::CopyAlways => System.IO.File::Exists
Deny          Critical           Uses File.IO                 HtmlAgilityPack.IOLibrary::CopyAlways => System.IO.File::Copy
Deny          Critical           Uses File.IO                 HtmlAgilityPack.IOLibrary::MakeWritable => System.IO.File::GetAttributes
Deny          Critical           Uses File.IO                 HtmlAgilityPack.IOLibrary::MakeWritable => System.IO.File::SetAttributes
Deny          Critical           Uses File.IO                 HtmlAgilityPack.HtmlWeb::SaveStream => System.IO.FileStream::.ctor
Deny          Critical           Uses File.IO                 HtmlAgilityPack.HtmlWeb::SaveStream => System.IO.File::SetLastWriteTime
Deny          Critical           Uses File.IO                 HtmlAgilityPack.HtmlWeb::Get => System.IO.File::GetLastAccessTime
Deny          Critical           Uses File.IO                 HtmlAgilityPack.HtmlWeb::Get => System.IO.File::GetLastWriteTime
Deny          Critical           Uses Networking              HtmlAgilityPack.PermissionHelper::GetIsDnsAvailable => System.Net.DnsPermission::.ctor
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Load => System.Net.WebProxy::.ctor
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Load => System.Net.WebProxy::set_BypassProxyOnLocal
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Load => System.Net.NetworkCredential::.ctor
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Load => System.Net.CredentialCache::.ctor
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Load => System.Net.WebProxy::get_Address
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Load => System.Net.CredentialCache::Add
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebRequest::Create
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebRequest::set_Method
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.HttpWebRequest::set_UserAgent
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.IWebProxy::set_Credentials
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebRequest::set_Credentials
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.CredentialCache::get_DefaultCredentials
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebRequest::set_Proxy
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebRequest::get_RequestUri
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.HttpWebRequest::set_IfModifiedSince
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.CookieContainer::.ctor
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.HttpWebRequest::set_CookieContainer
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebRequest::GetResponse
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebException::get_Response
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebResponse::get_ResponseUri
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebResponse::get_ContentType
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.HttpWebResponse::get_ContentEncoding
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.HttpWebResponse::get_StatusCode
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebResponse::GetResponseStream
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.HttpWebResponse::get_LastModified
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::Get => System.Net.WebResponse::Close
Deny          Critical           Uses Networking              HtmlAgilityPack.HtmlWeb::SaveCacheHeaders => System.Net.WebResponse::get_Headers
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Load => System.IO.StreamReader::get_CurrentEncoding
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Save => System.IO.StreamWriter::.ctor
Deny          Critical           Uses Stream.IO               HtmlAgilityPack.MixedCodeDocument::Save => System.IO.StreamWriter::.ctor
Deny          Critical           Using Registry               HtmlAgilityPack.HtmlWeb::GetContentTypeForExtension => Microsoft.Win32.RegistryKey::OpenSubKey
Deny          Critical           Using Registry               HtmlAgilityPack.HtmlWeb::GetContentTypeForExtension => Microsoft.Win32.RegistryKey::GetValue
=================================================================================================================================================================


Trust: -5200 - Trust has been set to Untrusted, but the value was lower.
```

Figure 3: GuardiNET analyzing HtmlAgilityPack

6 Discussion

6.1 .NET analysis and instrumentation

Static analysis combined with runtime instrumentation will detect violations of the known features in the security policy. .NET may have other features, which hides in the darkest corner in the runtime, for circumventing the reference monitor. Reflection-based programming is used heavily in .NET programming as the analysis of NuGet-packages and SuperOffice CRM SDK show. What are the consequences for prohibiting the use of reflection on code written by third-parties?

Rewriting assemblies adding reference monitors in the assembly will invalidate any prior testing done by the partner on their local installations. Improper rewriting will introduce bug and runtime errors.

A common pattern used in .NET programming is Dependency Injection, which is often accompanied by Inversion-of-Control containers, such as Autofac, nHibernate, nInject and StructureMap. Many IoC-containers use reflection for finding type candidates.

6.1.1 Security in .NET

A trend can be extrapolated based on the removal or deprecation of security features in .NET. New features will be added, but they will unlikely be adding features for securing execution of untrusted code. As mentioned in the background chapter, .NET core is currently not adding support for AppDomains, Security-Transparent or Code-Access-Policy. The recommendation from the .NET core team is to use operating system security primitives or other means of process isolation [1].

6.2 Security policies in legacy systems

Defining adequate security policies was one of the biggest challenges in the thesis due to the size and organization of the SDK. Table 2 shows the number of exported types, properties (with setters) and fields from a typical SuperOffice CRM installation. Manually considering whether an exported feature is a security property or not, is a time-consuming and daunting task. Imagine going through over 18 000 entities to assess whether access by custom code should be allowed, denied or conditionally allowed/denied.

As development of a product continues, new APIs are added as a natural part of development. Using this approach requires a well-organized API, with delineations

[1] https://blogs.msdn.microsoft.com/dotnet/2016/02/10/porting-to-net-core/

based on the security threats and trust levels on third parties. Guidelines must be known to the developers and should be enforced as part of automation tests.

6.3 Choice of policy language/representation

The combined use of JSON and C# source file was selected due to the familiarity for the intended audience and the context. Developers and architects in SuperOffice are used to both JSON and C#. Using another language would require development on an implementation that would have to properly concretize an abstract specification into CIL or another intermediate language before compilation to CIL.

6.4 Limitations

Inserting runtime enforcers when the opcode callvirt or a delegate is being used proved to be challenging. This has been resolved by others researchers [16]. Instead of inserting a call to an external assembly, they are inserting the checks directly into the calling method.

7 Conclusion and further work

GuardiNET can not be used as a standalone utility for assuring trust in .NET assemblies. Untrusted assemblies may contain malicious code using .NET/CLI features not detected by the analysis tool.

SuperOffice partners delivering solutions on the online platform, must already be registered and have a frequent communication including certification with the AppStore-team. GuardiNET can be added to the certification process to analyze assemblies from known third-parties.

This tool can be used for allowing trusted partners to run their code in SuperOffice Online, if adequate policies can be created and accepted. Currently, the existing layering in SuperOffice CRM can define the security boundaries defining the security policy. This leaves partners with a reduced API, which have proven to be enough in some of the collected samples from partners.

7.1 Further work

An appropriate trust model for SuperOffice partners should be developed and combined with GuardiNET for deciding whether the assembly is accepted to be executed in the environment.

Bibliography

[1] Zainal, Z. & Malaysia, U. T. 2007. Case study as a research method. *Jurnal Kemanusiaan*, 1–6.

[2] Yin, R. K. 2013. *Case study research: Design and methods*. Sage publications.

[3] International, E. 2017. Standard ecma-335. https://www.ecma-international.org/publications/standards/Ecma-335.htm. Accessed: 2017-02-17.

[4] Anderson, J. P. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, U.S. Air Force Electronic Systems Division, 10 1972.

[5] NuGet. 2017. Nuget package identity and trust. http://blog.nuget.org/20170417/Package-identity-and-trust.html. Accessed: 2017-04-20.

[6] Gosain, A. & Sharma, G. *Static Analysis: A Survey of Techniques and Tools*, 581–591. Springer India, New Delhi, 2015. URL: http://dx.doi.org/10.1007/978-81-322-2268-2_59, doi:10.1007/978-81-322-2268-2_59.

[7] Gosain, A. & Sharma, G. *A Survey of Dynamic Program Analysis Techniques and Tools*, 113–122. Springer International Publishing, Cham, 2015. URL: http://dx.doi.org/10.1007/978-3-319-11933-5_13, doi:10.1007/978-3-319-11933-5_13.

[8] Microsoft. 2016. Control Flow Guard. https://msdn.microsoft.com/en-us/library/windows/desktop/mt637065(v=vs.85).aspx. [Online; accessed 01-december-2016].

[9] Ali-Gombe, A., Ahmed, I., Richard, III, G. G., & Roussev, V. 2016. Aspectdroid: Android app analysis system. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY '16, 145–147, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2857705.2857739, doi:10.1145/2857705.2857739.

[10] Wang, X., Sun, K., Wang, Y., & Jing, J. 2015. Deepdroid: Dynamically enforcing enterprise policy on android devices. In *NDSS*.

[11] Liu, B., Liu, B., Jin, H., & Govindan, R. 2015. Efficient privilege de-escalation for ad libraries in mobile apps. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, 89–103, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2742647.2742668, doi:10.1145/2742647.2742668.

[12] Cabral, B., Marques, P., & Silva, L. 2004. Rail: Code instrumentation for .net. In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '04, 210–211, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/1028664.1028754, doi:10.1145/1028664.1028754.

[13] Hamlen, K. W., Morrisett, G., & Schneider, F. B. 2006. Certified in-lined reference monitoring on .net. In *Proceedings of the 2006 Workshop on Programming Languages and Analysis for Security*, PLAS '06, 7–16, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/1134744.1134748, doi:10.1145/1134744.1134748.

[14] Thamsirarak, N., Seethongchuen, T., & Ratanaworabhan, P. June 2015. A case for malware that make antivirus irrelevant. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2015 12th International Conference on*, 1–6. doi:10.1109/ECTICon.2015.7206972.

[15] Vanoverberghe, D. & Piessens, F. 2008. A caller-side inline reference monitor for an object-oriented intermediate language. In *International Conference on Formal Methods for Open Object-Based Distributed Systems*, 240–258. Springer.

[16] Vanoverberghe, D. & Piessens, F. 2015. Policy ignorant caller-side inline reference monitoring. *International Journal on Software Tools for Technology Transfer*, 17(3), 291–303.

Appendix

.1 Sample source code

.1.1 Source code for TypeTracker

```
public class TypeTracker
  {
      private static Dictionary<string, TypeTracker> _entityMappings = new Dictionary<string, TypeTracker>();

      public static TypeTracker<TType> GetTypeTracker<TType>()
          where TType : class
      {
          var typeTracker = GetOrCreateTypeTracker<TType>();
          return typeTracker;
      }

      public static int Increment<TType>(string methodName)
          where TType : class
      {
          var typeTracker = GetOrCreateTypeTracker<TType>();
          return typeTracker.Increment(methodName);
      }

      public static InstanceTracker<TEntity> GetTrackInstance<TEntity>()
          where TEntity : class
      {
          // Get the dictionary with entitites
          var typeTracker = GetTypeTracker<TEntity>();
          return typeTracker.InstanceTracker;
      }


      private static TypeTracker<TType> GetOrCreateTypeTracker<TType>()
          where TType : class
      {
          var typeName = typeof(TType).FullName;

          lock (_entityMappings)
          {
              TypeTracker<TType> typeTracker = null;
              if (_entityMappings.ContainsKey(typeName))
              {
                  typeTracker = _entityMappings[typeName] as TypeTracker<TType>;
              }
              else
              {
                  typeTracker = new TypeTracker<TType>();
                  _entityMappings[typeName] = typeTracker;
              }

              return typeTracker;
          }
      }
  }

  public class TypeTracker<TType> : TypeTracker
      where TType : class
  {

      private MethodData _methodData = new MethodData();

      public TypeTracker()
      {
          InstanceTracker = new InstanceTracker<TType>();
      }

      public InstanceTracker<TType> InstanceTracker { get; }

      public int Increment(string methodName)
      {
          return _methodData.Increment(methodName);
      }
  }
```

.1.2 Example of usage for TypeTracker

```
using System;
using System.Linq;
namespace GuardiNet.Instrumentation
{

  internal class GuardEnforcerAttribute : Attribute
  {
    public string AssemblyName { get; set; }
    public string MethodName { get; set; }
    public bool WarnOnSimilar { get; set; }

    public GuardEnforcerAttribute(string assemblyName, string
        methodName, bool warnOnSimilar)
    {
        AssemblyName = assemblyName;
        MethodName = methodName;
        WarnOnSimilar = warnOnSimilar;
    }
  }

  public static class Enforcer
  {
    [GuardEnforcer("System, Version=4.0.0.0, Culture=neutral,
        PublicKeyToken=b77a5c561934e089", "System.Net.WebRequest
        System.Net.WebRequest::Create(System.String)", true)]
    public static System.Net.WebRequest Guard_WebRequest_Create(System
        .String p0)
    {
        var result = TypeTracker.Increment<System.Net.WebRequest>("
            Create");
        if (result == 5)
        {
            throw new SecurityException("Called web request too many
                times");
        }

        if (!string.IsNullOrWhitespace(p0) && p0.StartsWith("https://
            www.ntnu.no"))
        {
            return System.Net.WebRequest.Create(p0);
        }

        throw new SecurityException("Not allowed to call " + p0);
    }
  }
}
```

.1.3 Source code for InstanceTracker

```
  public class InstanceTracker<TEntity>
        where TEntity : class
    {
        private readonly ConditionalWeakTable<TEntity, MethodData>
            _instanceMapping = new ConditionalWeakTable<TEntity,
            MethodData>();

        /// <summary>
        /// Increment the method count and return the current method
            count
        /// </summary>
        /// <param name="entity"></param>
        /// <param name="methodName"></param>
        /// <returns></returns>
        public int Increment(TEntity entity, string methodName)
        {
            var methodData = _instanceMapping.GetOrCreateValue(entity)
                ;
            return methodData.Increment(methodName);
        }
    }
```

.1.4 Source code for CallSequenceTracker

```
public class CallSequenceTracker
{
    private static readonly Dictionary<string, CallSequenceTracker>
        SequenceTrackers = new Dictionary<string, CallSequenceTracker
        >();
    public static CallSequenceTracker GetSequenceTracker(string
        ruleName)
    {
        return SequenceTrackers.GetOrCreate(ruleName, () => new
            CallSequenceTracker(ruleName));
    }

    private readonly  string _ruleName;
    private List<CallSequenceSpec> _sequence = new List<CallSequenceSpec>();

    public CallSequenceTracker(string ruleName)
    {
        _ruleName = ruleName;

        // Add a default non-empty sequence.
        _sequence.Add(new CallSequenceSpec(false, 0, ".*", ".*", ".*"));
    }

    public void AddSpecification(string assemblyName, string typeName, string methodName, bool isRequired = true, int maxInvocations = 0)
    {
        var spec = new CallSequenceSpec(isRequired, maxInvocations, assemblyName, typeName, methodName);
        _sequence.Add(spec);
    }

    private int[][] _dag; // First dimension is the state we are currently in, secondary dimension is the edges to other states.
    private int _currentIndex;


    public const int FinishedState = -1;

    public void CreateDFA()
    {
        // int -> [ints]
        // -1 = final (accept)
        // no edges = []
        var edges = new List<int[]>();
        for (var index = 0; index < _sequence.Count; index++)
        {
            var indexEdges = new List<int>();
            var leftovers = _sequence.Count - index - 1;
            var current = _sequence[index];

            if (leftovers > 0)
            {
                for (var next = 0; next < leftovers; next++)
                {
                    var nextIndex = index + next + 1;
                    var nextSequence = _sequence[nextIndex];
                    indexEdges.Add(nextIndex);

                    if (nextSequence.IsRequired)
                        break;
                }
            }
            else
            {
                indexEdges.Add(FinishedState);
            }

            if (current.MaxInvocations >= 0)
            {
                indexEdges.Add(index);
            }

            edges.Add(indexEdges.ToArray());
        }

        _dag = edges.ToArray();
        _currentIndex = 0;
    }


    public bool Allow(string assemblyName, string typeName, string
        methodName)
    {
        var edges = _dag[_currentIndex];

        foreach (var edgeIndex in edges)
        {
            // Check if we can move to this node
            if (edgeIndex == -1)
            {
                // Accepted state
                _currentIndex = 0;
                return true;
            }

            if (edgeIndex >= 0)
            {
                var invocation = _sequence[edgeIndex];
                var decision = invocation.Decide(assemblyName,
                    typeName, methodName);
                if (decision == CallSequenceSpec.Decision.Deny)
                {
                    return false;
                }

                if (decision == CallSequenceSpec.Decision.Allow)
                {
                    // Successful traversal to next
                    _currentIndex = edgeIndex;
                    return true;
                }
            }
        }
        return false;
    }

    private class CallSequenceSpec
    {
        public bool IsRequired { get; }

        public int MaxInvocations { get; }

        private readonly Matcher _assemblyMatcher;

        private readonly Matcher _typeMatcher;

        private readonly Matcher _methodMatcher;

        private int _currentInvocationCount;
        public CallSequenceSpec(bool isRequired, int maxInvocations,
            string assemblyName, string typeName,
            string methodName)
        {
            IsRequired = isRequired;
            MaxInvocations = maxInvocations;

            _assemblyMatcher = new Matcher(assemblyName);
            _typeMatcher = new Matcher(typeName);
            _methodMatcher = new Matcher(methodName);
            _currentInvocationCount = 0;

        }

        public Decision Decide(string assemblyName, string typeName,
            string methodName)
        {
            var isMatch = _assemblyMatcher.IsMatch(assemblyName) &&
                _typeMatcher.IsMatch(typeName) &&
                                    _methodMatcher.IsMatch(methodName)
                                        ;

            if (isMatch == false)
                return Decision.DontCare;

            _currentInvocationCount++;

            if (MaxInvocations == -1 || (MaxInvocations > 0 &&
                _currentInvocationCount >= MaxInvocations))
            {
                return Decision.Deny;
            }

            return Decision.Allow;
        }

        public enum Decision
        {
            DontCare,
            Deny,
            Allow,
        }

    }

}
```

.1.5 Source code for PolicyManager

```
public class RuleResult
{
    public Rule Rule { get; set; }

    public Decision Decision { get; set; }
}

public class PolicyResult
{
    public RuleResult[] Results { get; set; }
    public string EnforcerFile { get; set; }

    public int Score { get; set; }

    public string FinalTrust { get; set; }
}

public class PolicyManager : PolicyManagerBase
{
    class LastResult
    {
        public Rule Rule;
        public Decision Decision;


        public void Set(Rule rule, Decision decision)
        {
            Rule = rule;
            Decision = decision;
        }
    };

    public bool ReportFirstDenyOnly { get; set; }

    private readonly string _outputDirectory;
    public PolicyManager(string policy, bool isFileName, string
        outputDirectory)
        : base(policy, isFileName)
    {
        _outputDirectory = outputDirectory;
    }



    public PolicyResult AnalyzeAssembly(string assemblyFileName)
    {
        var assemblyDefinition = AssemblyDefinition.ReadAssembly(
            assemblyFileName);
        var result = new PolicyResult();
        var ruleResults = new List<RuleResult>();
        var generator = new RuntimeEnforcerStubGenerator();

        foreach (var rule in OrderedRules)
        {
            var evaluators = CreateEvaluators(rule.Value);
            var extractor = _featureExtractorManager.Create(rule.Key, null);
            Logging.LogInformation(LogCategory.Policy, $"Created extractor for {rule.Key}");

            foreach (var feature in extractor.Extract(assemblyFileName, assemblyDefinition))
            {
                var lastResult = new LastResult();
                Logging.LogInformation(LogCategory.Extractor, $"Extracting new feature...");
                for (var index = 0; index < evaluators.Count(); index++)
                {
                    var evaluator = evaluators[index];
                    var decision = evaluator.Decide(feature);
                    var currentRule = rule.Value[index];
                    Logging.LogInformation(LogCategory.Evaluator, $"'{currentRule.Name}' => '{decision}'");

                    if (decision.Access != Access.DontCare)
                    {
                        lastResult.Set(currentRule, decision);
                        if (currentRule.GenerateEnforcers)
                        {
                            // Convert it to an analyzer which potentially can generate weaver code
                            var dynamicEnforcer = evaluator as IRuleEnforcer;
                            if (dynamicEnforcer == null)
                            {
                                throw new InvalidAnalyzerOperationException(evaluator.GetType().Name, "does not support generating enforcers");
                            }

                            Logging.LogInformation(LogCategory.Evaluator, $"Creating enforcers...");
                            var invocations = dynamicEnforcer.GetEnforcers(feature);
                            foreach (var invocation in invocations)
                            {
                                var calledMethod = invocation.CallInstruction.Operand as MethodReference;
                                generator.Add(calledMethod);
                            }
                        }
                    }
                }

                if (lastResult.Decision == null)
                {
                    Logging.LogInformation(LogCategory.Policy, "No decision was made for feature --> ignoring.");
                    continue;
                }

                Logging.LogInformation(LogCategory.Policy, $"'{feature}' was finally evaluated to '{lastResult.Decision}'");

                var severityName = lastResult?.Rule?.Severity.ToString();
                // Update the score.
                if (!string.IsNullOrWhiteSpace(severityName) && _policy.Trust.Modifiers.ContainsKey(severityName))
                {
                    var modifiers = _policy.Trust.Modifiers[severityName];
                    result.Score += lastResult.Decision.Access == Access.Deny ? modifiers.MatchModifier : modifiers.NoMatchModifier;
                }

                ruleResults.Add(
                    new RuleResult
                    {
                        Decision = lastResult.Decision,
                        Rule = lastResult.Rule
                    });

                if (ReportFirstDenyOnly && lastResult.Decision.Access == Access.Deny)
                    break;
            }
        }

        result.Results = ruleResults.ToArray();

        var dynamicSource = generator.ToString();
        if (dynamicSource.Length > 0)
        {
            var assemblyFileNameWithoutPath = Path.GetFileName(assemblyFileName);

            var enforcerPath = Path.Combine(_outputDirectory, assemblyFileNameWithoutPath + ".cs");
            DirectoryHelper.EnsureOutputDirExists(_outputDirectory);
            result.EnforcerFile = enforcerPath;
            File.WriteAllText(enforcerPath, dynamicSource);
        }

        // Generate the final description.
        result.FinalTrust = GenerateDescription(result);

        return result;
    }

    public void CompileEnforcers(string assemblyFile, string[] sourceFiles)
    {
        var runtimePolicyCompiler = new RuntimePolicyEnforcerCompiler();
        var assemblyDefinition = AssemblyDefinition.ReadAssembly(assemblyFile);


        var compiledBytes = runtimePolicyCompiler.Compile(sourceFiles, assemblyDefinition);
        using (var ms = new MemoryStream(compiledBytes))
        {
            var weaverAssembly = AssemblyDefinition.ReadAssembly(ms);
            RuntimeWeaver.Weave(assemblyDefinition, weaverAssembly);

            assemblyDefinition.MainModule.AssemblyReferences.Add(weaverAssembly.Name);

            weaverAssembly.Write(Path.Combine(_outputDirectory, weaverAssembly.Name.Name + ".dll"));
            assemblyDefinition.Write(Path.Combine(_outputDirectory, assemblyDefinition.Name.Name + ".dll"));
        }
    }

    private string GenerateDescription(PolicyResult result)
    {
        var trustLevel = _policy.Trust.Levels.Where(l => l.Value.From <= result.Score && l.Value.To >= result.Score).Select(l => new
        {
            Level = l.Key
        }).FirstOrDefault();

        if (trustLevel == null)
        {
            // Check min
            // Find the lowest min:
            var minRule = _policy.Trust.Levels.OrderBy(l => l.Value.From).FirstOrDefault();
            var maxRule = _policy.Trust.Levels.OrderByDescending(l => l.Value.To).FirstOrDefault();


            if (result.Score < minRule.Value.From)
            {
                return $"Trust has been set to {minRule.Key}, but the value was lower. ";
            }
            else
            {
                return $"Trust has been set to {maxRule.Key}, but the value was higher.";
            }
        }

        return $"Trust has been set to: {trustLevel.Level}";
    }
}
```

.1.6 Source code for RuntimeWeaver

```
public class RuntimeWeaver
{

    public static void Weave(AssemblyDefinition originalAssembly,
        AssemblyDefinition weaverAssembly)
    {
        // Find all the usage of GuardEnforceAttribute in
            weaverAssembly
        var typeWalker = new TypeWalker(weaverAssembly);
        var rtEnforcerList = new Dictionary<AssemblyAndMethodData,
            MethodDefinition>();


        foreach (var type in typeWalker)
        {
            foreach(var method in type.Methods)
            {
                var enforcerAttribute = method.CustomAttributes.Where(
                    c => c.AttributeType.FullName == "GuardiNet.
                    Instrumentation.GuardEnforcerAttribute").
                    FirstOrDefault();
                if (enforcerAttribute != null)
                {
                    // we need to take a closer look at the
                        constructor arguments
                    var assemblyName = enforcerAttribute.
                        ConstructorArguments[0].Value as string;
                    var methodName = enforcerAttribute.
                        ConstructorArguments[1].Value as string;
                    var warnOnSimilar = (bool) enforcerAttribute.
                        ConstructorArguments[2].Value;

                    var entry = new AssemblyAndMethodData
                    {
                        AssemblyName = assemblyName,
                        MethodName = methodName
                    };

                    rtEnforcerList.Add(entry, method);
                }
            }
        }

        var externalInvocationExtractor = new ExternalMethodExtractor
            ();
        var invokes = externalInvocationExtractor.Extract(string.Empty
            , originalAssembly).ToArray();

        foreach (var inv in invokes)
        {
            var invocation = TypeHelper.Cast<
                ExternalMethodCallInstructionFeature, IFeature>(inv);

            // Check if this is on our list

            // Replace the call instruction with our own
            var reference = AssemblyAndMethodData.CreateReference(
                invocation.CallInstruction);

            if (rtEnforcerList.ContainsKey(reference))
            {
                var enforcer = rtEnforcerList[reference];
                var methodReference = invocation.Method.Resolve();

                // Weave in the reference and replace the call
                var processor = methodReference.Body.GetILProcessor();
                var importedReference = originalAssembly.MainModule.
                    Import(enforcer);
                processor.Replace(invocation.CallInstruction,
                    CreateCallInstruction(importedReference));

            }
        }
    }

    private static Instruction CreateCallInstruction(MethodReference
        method)
    {
        var instruction = Instruction.Create(OpCodes.Call, method);
        return instruction;
    }
}
```

.2 Statistics from SuperOffice CRM

Table 2: The table shows all exported classes/types with methods, properties (with setters) and fields.

AssemblyName	Namespaces	Types	Methods	Properties	Fields
SOCore	36	475	1455	237	53
SoDataBase	58	4896	29683	6635	5819
SuperOffice.Services	4	378	2911	2106	0
SuperOffice.Services.Implementation	12	224	2307	32	0
SuperOffice.DCFWeb	25	183	688	350	18
SuperOffice.CRMWeb	23	673	2302	933	111
SuperOffice.Plugins	12	161	528	334	20
SoLicense	2	6	30	0	0
SuperOffice.Contracts	5	61	90	141	67
SuperOffice.DCF.Services	1	21	130	22	0
SuperOffice.DCF.Services.Implementation	4	30	143	13	0
SuperOffice.Mime	4	76	340	85	0
SuperOffice.Services.Versioned.Contract	6	2346	8135	13562	0
SuperOffice.Services.Versioned	12	2220	14846	0	0
SoLicense	2	6	30	0	0
Total	206	11756	63618	24450	6088

.3 Example policies

.3.1 Classification policy

numbers

```
{
        "Name": "Classification of capabilities",
        "Description": "",
        "DefaultAccess": "Allow",
        "Trust": {
                "Levels": {
                        "Untrusted": {
                                "From": 0,
                                "To": 10
                        },
                        "Low": {
                                "From": 10,
                                "To": 40
                        },
                        "Medium": {
                                "From": 40,
                                "To": 80
                        },
                        "High": {
                                "From": 80,
                                "To": 100
                        }
                },
                "InitialScore": 100,
                "Modifiers": {
                        "None": {
                                "MatchModifier": 0,
                                "NoMatchModifier": 0
                        },
                        "Low": {
                                "MatchModifier": -5,
                                "NoMatchModifier": 0
                        },
                        "Medium": {
                                "MatchModifier": -10,
                                "NoMatchModifier": 0
                        },
                        "High": {
                                "MatchModifier": -50,
                                "NoMatchModifier": 0
                        },
                        "Critical": {
                                "MatchModifier": -100,
                                "NoMatchModifier": 0
                        }
                }
        },
  "Rules": [
    {
      "Name": "Uses File.IO",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*",
        "ExternalMethodName": "System.IO.File.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Uses Stream.IO",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*",
        "ExternalMethodName": "System.IO.Stream.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Emitting code",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.Reflection.*",
        "ExternalMethodName": "System.Reflection.Emit.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Loading assemblies",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.Reflection.*",
        "ExternalMethodName": "System.Reflection.Assembly::Load.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Invoking methods dynamically",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.Reflection.*",
        "ExternalMethodName": "System.Reflection.[^<>]*::Invoke.*|
            System.Reflection.PropertyInfo::setValue.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Using WPF/WinForms",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.*",
        "ExternalMethodName": "System.Windows.*|System.Drawing.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Using DirectoryServices",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.*",
        "ExternalMethodName": "System.DirectoryServices.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Using Process",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.*",
        "ExternalMethodName": "System.Diagnostics.Process.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Using Registry",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.*",
        "ExternalMethodName": "Microsoft.Win32.Registry.*",
        "IsRegex": true
      }
    },

    {
      "Name": "Is not signed",
      "Type": "DigitalSignature",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "Signed": false

      }
    },

    {
      "Name": "Uses Threads",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.Reflection.*",
        "ExternalMethodName": "System.Threading.Thread.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Uses Crypto",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*",
        "ExternalMethodName": "System.Security.Cryptography.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Uses Networking",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.*",
        "ExternalMethodName": "System.Net.*|System.ServiceModel.
            ChannelFactory.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Uses PInvoke",
      "Type": "PInvoke",
      "Access": "Deny",
      "Severity": "Critical",

      "Properties": {
        "Import": ".*",
        "IsRegex": true
      }
    },

    {
      "Name": "Uses Module initializer",
      "Type": "ModuleInitializer",
      "Access": "Deny",
      "Severity": "Critical"
    },

    {
      "Name": "Is mixed mode",
      "Type": "MixedMode",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "IsMixedMode": "true"
      }
    },

    {
      "Name": "Internal/runtime calls",
      "Type": "InternalCall",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "InternalCallPattern": ".*"
      }
    }


  ]
}
```

.3.2 SoPartnerWebPolicy

numbers

```
{
  "Name": "SuperOffice Online Partner Policy",
  "Description": "Policy for analyzing and instrumenting partner code
      .",
  "DefaultAccess": "Deny",
  "Trust": {
    "Levels": {
      "Untrusted": {
        "From": 0,
        "To": 10
      },
      "Low": {
        "From": 10,
        "To": 40
      },
      "Medium": {
        "From": 40,
        "To": 80
      },
      "High": {
        "From": 80,
        "To": 100
      }
    },
    "InitialScore": 100,
    "Modifiers": {
      "None": {
        "MatchModifier": 0,
        "NoMatchModifier": 0
      },
      "Low": {
        "MatchModifier": -5,
        "NoMatchModifier": 0
      },
      "Medium": {
        "MatchModifier": -10,
        "NoMatchModifier": 0
      },
      "High": {
        "MatchModifier": -50,
        "NoMatchModifier": 0
      },
      "Critical": {
        "MatchModifier": -100,
        "NoMatchModifier": 0
      }
    }
  },
  "Rules": [
    {
      "Name": "Allow all System.Object",
      "Type": "ExternalMethod",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*",
        "ExternalMethodName": "System.Object.*|System.
            NotImplementedException.*|System.Enum.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Deny unsigned assemblies",
      "Type": "DigitalSignature",
      "Access": "Deny",
      "Severity": "Critical"
    },
    {
      "Name": "Allow inheritance from System.Object",
      "Type": "Inheritance",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "InheritsFromAssemblyName": "mscorlib.*",
        "InheritsFromTypeName": "System.Object.*|System.
            NotImplementedException.*|System.Enum.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow inheritance from SuperOffice.IPlugin",
      "Type": "Inheritance",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "InheritsFromAssemblyName": "SuperOffice.Plugins.*",
        "InheritsFromTypeName": "SuperOffice.Factory.IPlugin.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow inheritance from SuperOffice.DCF",
      "Type": "Inheritance",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "InheritsFromAssemblyName": "SuperOffice.DCFWeb.*",
        "InheritsFromTypeName": "SuperOffice.DCF.Web.UI.Validations.
```

```
            ValidationBase.*|SuperOffice.DCF.Web.Factory.IWebObject.*|
            SuperOffice.DCF.Web.Data.DataHandlerBase.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow inheritance from SuperOffice.CRMWeb",
      "Type": "Inheritance",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "InheritsFromAssemblyName": "SuperOffice.CRMWeb.*",
        "InheritsFromTypeName": "SuperOffice.CRM.Web.UI.Controls.
            IArchiveControlDataFetcher.*|SuperOffice.CRM.Web.UI.
            Controls.SoArchiveFetcherBase.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow inheritance from SoDatabase",
      "Type": "Inheritance",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "InheritsFromAssemblyName": "SoDatabase.*",
        "InheritsFromTypeName": "SuperOffice.CRM.ArchiveLists.
            IArchiveProvider.*|SuperOffice.CRM.Tooltips.
            TooltipPluginBase.*|SuperOffice.CRM.Lists.LiteralsOnlyBase
            .*|SuperOffice.CRM.ArchiveLists.TableExtenderBase.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow inheriting from common collections and patterns",
      "Type": "Inheritance",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "InheritsFromAssemblyName": "mscorlib.*",
        "InheritsFromTypeName": "System.Collections.*|System.
            IDisposable.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow common .NET calls",
      "Type": "ExternalMethod",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.*",
        "ExternalMethodName": "System.Collections.*|System.String.*|
            System.Array.*|System.Comparison.*|System.Boolean.*|System
            .Int16.*|System.Int32.*|System.Int64.*|System.Double.*|
            System.Exception.*|System.NotSupportedException.*|System.
            Argument(Null)?Exception.*|System.IDisposable.*|System.
            Decimal.*|System.Linq.*|System.DateTime.*|System.
            Globalization.(CultureInfo|Calendar).*|System.
            Globalization.DateTimeFormatInfo.*|System.Text.
            RegularExpressions.*|System.Text.StringBuilder.*|System.
            Convert.*|System.Text.Encoding.*|System.Predicate.*|System
            .Func.*|System.Environment::get_NewLine",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow data dispatcher calls",
      "Type": "ExternalMethod",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "ExternalAssemblyName": "SuperOffice.DCFWeb.*",
        "ExternalMethodName": "SuperOffice.DCF.Web.DataDispatcher.*|
            SuperOffice.Data.DataDispatcher.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow control and datahandler calls",
      "Type": "ExternalMethod",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "ExternalAssemblyName": "SuperOffice.DCFWeb.*",
        "ExternalMethodName": "SuperOffice.DCF.Web.UI.Controls.*|
            SuperOffice.DCF.Web.Data.DataHandler.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow archiverestriction info calls",
      "Type": "ExternalMethod",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "ExternalAssemblyName": "SoDataBase.*|SoCore.*|SuperOffice.
            Services.*",
        "ExternalMethodName": "SuperOffice.CRM.ArchiveLists.*|
            SuperOffice.CRM.Services.ArchiveList(Result|Item).*|
            SuperOffice.Data.QueryExecutionHelper.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow access to Rows and Services",
      "Type": "ExternalMethod",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "ExternalAssemblyName": "SoDataBase.*|SoCore.*|SuperOffice.
            Services.*",
        "ExternalMethodName": "SuperOffice.Data.SoDataReader.*|
            SuperOffice.Data.SQL.*|SuperOffice.Data.TablesInfo.*|
            SuperOffice.CRM.Security.TableRight.*|SuperOffice.CRM.Rows
            .*|SuperOffice.CRM.Services.(Person|Contact|Sale|Project|
            List|MDO|Associate|Appointment|Batch)(Agent)?.*|
            SuperOffice.CRM.Services.*Entity.*|SuperOffice.CRM.Data.(
            Phone|Person|Email|Sale|SaleStakeholder|
            SaleStakeholderRole|StakeholderRole|Contact|Associate|
            UDPersonSmall|ForeignDevice|ForeignApp|ForeignKey)
            TableInfo.*|SuperOffice.Data.S::Parameter.*|SuperOffice.
            Data.S::NewSelect|SuperOffice.CRM.Services.Carrier.*|
            SuperOffice.CRM.Services.ColumnDataDictionary.*|
            SuperOffice.Data.NestedCollectionPersist.*|SuperOffice.CRM
            .Lists.LiteralsOnlyBase.*|SuperOffice.CRM.Tooltips.
            TooltipPluginBase.*|SuperOffice.Util.ParameterBuilder.*|
            SuperOffice.CRM.Lists.I?SoListItem.*|SuperOffice.Data.
            Dictionary.SoTable.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow calls to SuperOffice.Globalization",
      "Type": "ExternalMethod",
      "Access": "Allow",
      "Severity": "None",
      "Properties": {
        "ExternalAssemblyName": "SoDataBase.*|SoCore.*|SuperOffice.
            Services.*|SuperOffice.Plugins.*",
        "ExternalMethodName": "SuperOffice.CRM.Globalization.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Uses File.IO",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*",
        "ExternalMethodName": "System.IO.File.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Uses Reflection",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.Reflection.*",
        "ExternalMethodName": "System.Reflection.*|System.Type.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Uses threads or locking mechanisms",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*",
        "ExternalMethodName": "System.Threading.Thread.*|System.
            Threading.Monitor.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Uses crypto",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*",
        "ExternalMethodName": "System.Security.Cryptography.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Uses Networking",
      "Type": "ExternalMethod",
      "Access": "Deny",
      "Severity": "Critical",
      //"GenerateEnforcers": true,
      "Properties": {
        "ExternalAssemblyName": "mscorlib.*|System.*|System.
            ServiceModel.*",
        "ExternalMethodName": "System.Net.*|System.ServiceModel.
            Channel.*",
        "IsRegex": true
      }
    },
    {
      "Name": "Allow common .NET attributes",
      "Type": "Attribute",
      "Access": "Allow",
      "Severity": "None",

      "Properties": {
        "AttributeTarget": "All",
        "AttributePartialName": "System.Runtime.CompilerServices.
            CompilerGeneratedAttribute|System.Reflection.Assembly.*
            Attribute",
        "IsRegex": true
      }
    },
    {
      "Name": "Add instrumentation to ClassFactory.Create",
      "Type": "ExternalMethod",
      "Access": "Allow",
      "GenerateEnforcers": true,
      "Severity": "Low",
      "Properties": {
        "ExternalAssemblyName": "SoDataBase.*|SoCore.*|SuperOffice.
            Services.*",
        "ExternalMethodName": "SuperOffice.Factory.ClassFactory.*",
        "IsRegex": true
      }
    }

  ]
}
```