Fortify Extension for Visual Studio

# Developer Workbook

## SharkCage

# Table of Contents

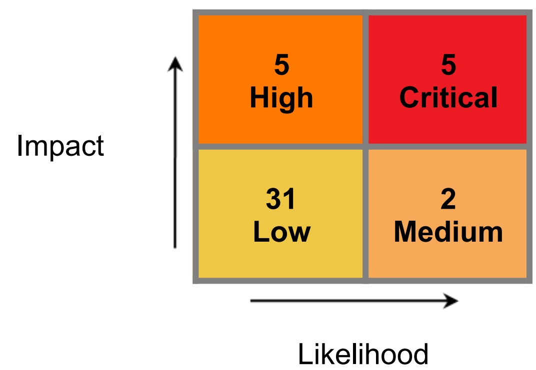
# Executive Summary

This workbook is intended to provide all necessary details and information for a developer to understand and remediate the different issues discovered during the SharkCage project audit. The information contained in this workbook is targeted at project managers and developers.

This section provides an overview of the issues uncovered during analysis.

| | |
|---|---|
| **Project Name:** | SharkCage |
| **Project Version:** | |
| **SCA:** | Results Present |
| **WebInspect:** | Results Not Present |
| **WebInspect Agent:** | Results Not Present |
| **Other:** | Results Not Present |

**Issues by Priority**



# Project Description

This section provides an overview of the Fortify scan engines used for this project, as well as the project meta-information.

**SCA**

| | | | |
|---|---|---|---|
| **Date of Last Analysis:** | Jul 24, 2018, 11:27 AM | **Engine Version:** | 18.10.0187 |
| **Host Name:** | itsec-08 | **Certification:** | VALID |
| **Number of Files:** | 271 | **Lines of Code:** | 26,377 |

# Issue Breakdown by Fortify Categories

The following table depicts a summary of all issues grouped vertically by Fortify Category. For each category, the total number of issues is shown by Fortify Priority Order, including information about the number of audited issues.

| Category | Fortify Priority (audited/total) | | | | Total Issues |
|---|---|---|---|---|---|
| | Critical | High | Medium | Low | |
| ASP.NET Bad Practices: Leftover Debug Code | 0 | 0 | 0 | 0 / 1 | 0 / 1 |
| Command Injection | 0 / 3 | 0 | 0 | 0 / 1 | 0 / 4 |
| Dead Code | 0 | 0 | 0 | 0 / 12 | 0 / 12 |
| Dead Code: Unused Field | 0 | 0 | 0 | 0 / 1 | 0 / 1 |
| Null Dereference | 0 | 0 / 2 | 0 | 0 | 0 / 2 |
| Path Manipulation | 0 / 2 | 0 | 0 / 2 | 0 / 5 | 0 / 9 |
| Poor Error Handling: Empty Catch Block | 0 | 0 | 0 | 0 / 1 | 0 / 1 |
| Poor Error Handling: Overly Broad Catch | 0 | 0 | 0 | 0 / 3 | 0 / 3 |
| Poor Style: Variable Never Used | 0 | 0 | 0 | 0 / 7 | 0 / 7 |
| Privacy Violation: Heap Inspection | 0 | 0 / 1 | 0 | 0 | 0 / 1 |
| Type Mismatch: Signed to Unsigned | 0 | 0 / 1 | 0 | 0 | 0 / 1 |
| Unsafe Native Invoke | 0 | 0 / 1 | 0 | 0 | 0 / 1 |

# Results Outline

## ASP.NET Bad Practices: Leftover Debug Code (1 issue)

### Abstract

Debug code can create unintended entry points in a deployed web application.

### Explanation

A common development practice is to add "back door" code specifically designed for debugging or testing purposes that is not intended to be shipped or deployed with the application. When this sort of debug code is accidentally left in the application, the application is open to unintended modes of interaction. These back door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the application. The most common example of forgotten debug code is a `Main()` method appearing in a web application. Although this is an acceptable practice during product development, classes that are part of a production ASP.NET application should not define a `Main()`.

### Recommendation

Remove debug code before deploying a production version of an application. Regardless of whether a direct security threat can be articulated, it is unlikely that there is a legitimate reason for such code to remain in the application after the early stages of development.

### Issue Summary

### Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| ASP.NET Bad Practices: Leftover Debug Code | 1 | 0 | 0 | 1 |
| **Total** | **1** | **0** | **0** | **1** |

| ASP.NET Bad Practices: Leftover Debug Code | Low |
|---|---|
| Package: CageConfigurator | |
| CageConfigurator/CageConfigurator.cs, line 12 (ASP.NET Bad Practices: Leftover Debug Code) | Low |

**Issue Details**

## ASP.NET Bad Practices: Leftover Debug Code — Low

### Package: CageConfigurator

### CageConfigurator/CageConfigurator.cs, line 12 (ASP.NET Bad Practices: Leftover Debug Code) — Low

**Kingdom:** Encapsulation
**Scan Engine:** SCA (Structural)

### Sink Details

**Sink:** Function: Main
**Enclosing Method:** Main()
**File:** CageConfigurator/CageConfigurator.cs:12
**Taint Flags:**

```
9  /// The main entry point for the application.
10 /// </summary>
11 [STAThread]
12 static void Main(string[] parameter)
13 {
14 Application.EnableVisualStyles();
15 Application.SetCompatibleTextRenderingDefault(false);
```

## Command Injection (4 issues)

### Abstract

Executing commands from an untrusted source or in an untrusted environment can cause an application to execute malicious commands on behalf of an attacker.

### Explanation

Command injection vulnerabilities take two forms: - An attacker can change the command that the program executes: the attacker explicitly controls what the command is. - An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means. In this case we are primarily concerned with the first scenario, the possibility that an attacker may be able to control the command that is executed. Command injection vulnerabilities of this type occur when: 1. Data enters the application from an untrusted source. 2. The data is used as or as part of a string representing a command that is executed by the application. 3. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have. **Example 1:** The following code from a system utility uses the system property `APPHOME` to determine the directory in which it is installed and then executes an initialization script based on a relative path from the specified directory.

```
...
string val = Environment.GetEnvironmentVariable("APPHOME");
string cmd = val + INITCMD;
ProcessStartInfo startInfo = new ProcessStartInfo(cmd);
Process.Start(startInfo);
...
```

The code in Example 1 allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property `APPHOME` to point to a different path containing a malicious version of `INITCMD`. Because the program does not validate the value read from the environment, if an attacker can control the value of the system property `APPHOME`, then they can fool the application into running malicious code and take control of the system. **Example 2:** The following code is from an administrative web application designed to allow users to kick off a backup of an Oracle database using a batch-file wrapper around the `rman` utility and then run a `cleanup.bat` script to delete some temporary files. The script `rmanDB.bat` accepts a single command line parameter, which specifies the type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

```
...
string btype = BackupTypeField.Text;
string cmd = "cmd.exe /K \"c:\\util\\rmanDB.bat"
              + btype + "&&c:\\util\\cleanup.bat\""));
Process.Start(cmd);
...
```

The problem here is that the program does not do any validation on `BackupTypeField`. Typically the `Process.Start()` function will not execute multiple commands, but in this case the program first runs the `cmd.exe` shell in order to run multiple commands with a single call to `Process.Start()`. Once the shell is invoked, it will allow for the execution of multiple commands separated by two ampersands. If an attacker passes a string of the form `"&& del c:\\dbms\\*.*"`, then the application will execute this command along with the others specified by the program. Because of the nature of the application, it runs with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well. **Example 3:** The following code is from a web application that gives users access to an interface through which they can update their password on the system. Part of the process for updating passwords in this network environment is to run an `update.exe` command, as shown below.

```
...
Process.Start("update.exe");
...
```

The problem here is that the program does not specify an absolute path and fails to clean its environment prior to executing the call to `Process.start()`. If an attacker can modify the `$PATH` variable to point to a malicious binary called `update.exe` and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's `update.exe` will now be run with these privileges, possibly giving the attacker complete control of the system.

## Recommendation

Do not allow users to have direct control over the commands executed by the program. In cases where user input must affect the command to be run, use the input only to make a selection from a predetermined set of safe commands. If the input appears to be malicious, the value passed to the command execution function should either default to some safe selection from this set or the program should decline to execute any command at all. In cases where user input must be used as an argument to a command executed by the program, this approach often becomes impractical because the set of legitimate argument values is too large or too hard to keep track of. Developers often fall back on blacklisting in these situations. Blacklisting selectively rejects or escapes potentially dangerous characters before using the input. Any list of unsafe characters is likely to be incomplete and will be heavily dependent on the system where the commands are executed. A better approach is to create a whitelist of characters that are allowed to appear in the input and accept input composed exclusively of characters in the approved set. An attacker may indirectly control commands executed by a program by modifying the environment in which they are executed. The environment should not be trusted and precautions should be taken to prevent an attacker from using some manipulation of the environment to perform an attack. Whenever possible, commands should be controlled by the application and executed using an absolute path. In cases where the path is not known at compile time, such as for cross-platform applications, an absolute path should be constructed from trusted values during execution. Command values and paths read from configuration files or the environment should be sanity-checked against a set of invariants that define valid values. Other checks can sometimes be performed to detect if these sources may have been tampered with. For example, if a configuration file is world-writable, the program might refuse to run. In cases where information about the binary to be executed is known in advance, the program may perform checks to verify the identity of the binary. If a binary should always be owned by a particular user or have a particular set of access permissions assigned to it, these properties can be verified programmatically before the binary is executed. Although it may be impossible to completely protect a program from an imaginative attacker bent on controlling the commands the program executes, be sure to apply the principle of least privilege wherever the program executes an external command: do not hold privileges that are not essential to the execution of the command.

## Issue Summary

## Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Command Injection | 4 | 0 | 0 | 4 |
| **Total** | **4** | **0** | **0** | **4** |

## Command Injection — Critical

## Package: CageChooser

### CageChooser/CageChooserForm.cs, line 177 (Command Injection) — Critical

#### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

#### Source Details

**Source:** Microsoft.Win32.Registry.GetValue()
**From:** CageChooser.CageChooserForm.openCageConfiguratorButton_Click
**File:** CageChooser/CageChooserForm.cs:168

```
165  private void openCageConfiguratorButton_Click(object sender, EventArgs e)
166  {
167  const string registry_key = @"HKEY_LOCAL_MACHINE\SOFTWARE\SharkCage";
168  var install_dir = Registry.GetValue(registry_key, "InstallDir", "") as
string;
169
170  if (install_dir.Length == 0)
171  {
```

#### Sink Details

**Sink:** System.Diagnostics.ProcessStartInfo.set_FileName()
**Enclosing Method:** openCageConfiguratorButton_Click()
**File:** CageChooser/CageChooserForm.cs:177
**Taint Flags:** REGISTRY

```
174  }
175
176  var p = new System.Diagnostics.Process();
177  p.StartInfo.FileName = $@"{install_dir}\CageConfigurator.exe";
178  p.StartInfo.Arguments = $@"""{configPath.Text}""";
179  p.Start();
180  }
```

### CageChooser/CageChooserForm.cs, line 178 (Command Injection) — Critical

#### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

#### Source Details

## Command Injection — Critical

### Package: CageChooser

### CageChooser/CageChooserForm.cs, line 178 (Command Injection) — Critical

**Source:** System.Windows.Forms.TextBox.get_Text()
**From:** CageChooser.CageChooserForm.openCageConfiguratorButton_Click
**File:** CageChooser/CageChooserForm.cs:178

```
175
176  var p = new System.Diagnostics.Process();
177  p.StartInfo.FileName = $@"{install_dir}\CageConfigurator.exe";
178  p.StartInfo.Arguments = $@"""{configPath.Text}""";
179  p.Start();
180  }
181
```

#### Sink Details

**Sink:** System.Diagnostics.ProcessStartInfo.set_Arguments()
**Enclosing Method:** openCageConfiguratorButton_Click()
**File:** CageChooser/CageChooserForm.cs:178
**Taint Flags:** GUI_FORM

```
175
176  var p = new System.Diagnostics.Process();
177  p.StartInfo.FileName = $@"{install_dir}\CageConfigurator.exe";
178  p.StartInfo.Arguments = $@"""{configPath.Text}""";
179  p.Start();
180  }
181
```

### Package: CageConfigurator

### CageConfigurator/CageConfiguratorForm.cs, line 370 (Command Injection) — Critical

#### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

#### Source Details

**Source:** Microsoft.Win32.Registry.GetValue()
**From:** CageConfigurator.CageConfiguratorForm.openCageChooserButton_Click
**File:** CageConfigurator/CageConfiguratorForm.cs:361

```
358  private void openCageChooserButton_Click(object sender, EventArgs e)
359  {
360  const string registry_key = @"HKEY_LOCAL_MACHINE\SOFTWARE\SharkCage";
361  var install_dir = Registry.GetValue(registry_key, "InstallDir", "") as string;
362
363  if (install_dir.Length == 0)
```

## Command Injection — Critical

### Package: CageConfigurator

### CageConfigurator/CageConfiguratorForm.cs, line 370 (Command Injection) — Critical

```
364 {
```

#### Sink Details

**Sink:** System.Diagnostics.ProcessStartInfo.set_FileName()
**Enclosing Method:** openCageChooserButton_Click()
**File:** CageConfigurator/CageConfiguratorForm.cs:370
**Taint Flags:** REGISTRY

```
367 }
368
369 var p = new System.Diagnostics.Process();
370 p.StartInfo.FileName = $@"{install_dir}\CageChooser.exe";
371 p.Start();
372 }
373
```

## Command Injection — Low

### Package: CageChooser

### CageChooser/CageChooser.cs, line 25 (Command Injection) — Low

#### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Semantic)

#### Sink Details

**Sink:** set_Arguments(0)
**Enclosing Method:** Main()
**File:** CageChooser/CageChooser.cs:25
**Taint Flags:**

```
22 var rootDir =
System.IO.Directory.GetParent(System.IO.Directory.GetCurrentDirectory()).Parent;
23 var scriptDir = rootDir.FullName + "\\install_service.ps1";
24
25 p.StartInfo.Arguments = "-ExecutionPolicy Unrestricted -File \"" + scriptDir + "\" -
DontStartNewContext";
26 try
27 {
28 p.Start();
```

## Dead Code (12 issues)

### Abstract

This statement will never be executed.

### Explanation

The surrounding code makes it impossible for this statement to ever be executed. **Example:** The condition for the second `if` statement is impossible to satisfy. It requires that the variable `s` be non-null, while on the only path where `s` can be assigned a non-null value there is a `return` statement.

```
String s = null;

if (b) {
   s = "Yes";
   return;
}

if (s != null) {
   Dead();
}
```

### Recommendation

In general, you should repair or remove unused code. It causes additional complexity and maintenance burden without contributing to the functionality of the program.

### Issue Summary

### Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Dead Code | 12 | 0 | 0 | 12 |
| **Total** | **12** | **0** | **0** | **12** |

| Dead Code | Low |
|---|---|
| Package: SharedFunctionality | |
| SharedFunctionality/json.hpp, line 8316 (Dead Code) | Low |

**Issue Details**

| Dead Code | Low |
|---|---|

## Package: SharedFunctionality

### SharedFunctionality/json.hpp, line 8316 (Dead Code) — Low

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8316
**Taint Flags:**

```
8313 {
8314 case value_t::object:
8315 {
8316 if (val.m_value.object->empty())
8317 {
8318 o->write_characters("{}", 2);
8319 return;
```

### SharedFunctionality/json.hpp, line 8389 (Dead Code) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8389
**Taint Flags:**

```
8386
8387 case value_t::array:
8388 {
8389 if (val.m_value.array->empty())
8390 {
8391 o->write_characters("[]", 2);
8392 return;
```

### SharedFunctionality/json.hpp, line 8456 (Dead Code) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8456
**Taint Flags:**

```
8453
```

## Dead Code | Low

### Package: SharedFunctionality

### SharedFunctionality/json.hpp, line 8456 (Dead Code) | Low

```
8454 case value_t::boolean:
8455 {
8456 if (val.m_value.boolean)
8457 {
8458 o->write_characters("true", 4);
8459 }
```

### SharedFunctionality/json.hpp, line 3968 (Dead Code) | Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** iter_impl()
**File:** SharedFunctionality/json.hpp:3968
**Taint Flags:**

```
3965 {
3966 case value_t::object:
3967 {
3968 m_it.object_iterator = typename object_t::iterator();
3969 break;
3970 }
3971
```

### SharedFunctionality/json.hpp, line 8448 (Dead Code) | Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8448
**Taint Flags:**

```
8445
8446 case value_t::string:
8447 {
8448 o->write_character('\"');
8449 dump_escaped(*val.m_value.string, ensure_ascii);
8450 o->write_character('\"');
8451 return;
```

## Dead Code — Low

### Package: SharedFunctionality

### SharedFunctionality/json.hpp, line 8469 (Dead Code) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8469
**Taint Flags:**

```
8466
8467  case value_t::number_integer:
8468  {
8469  dump_integer(val.m_value.number_integer);
8470  return;
8471  }
8472
```

### SharedFunctionality/json.hpp, line 8475 (Dead Code) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8475
**Taint Flags:**

```
8472
8473  case value_t::number_unsigned:
8474  {
8475  dump_integer(val.m_value.number_unsigned);
8476  return;
8477  }
8478
```

### SharedFunctionality/json.hpp, line 8481 (Dead Code) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8481
**Taint Flags:**

## Dead Code — Low

### Package: SharedFunctionality

#### SharedFunctionality/json.hpp, line 8481 (Dead Code) — Low

```
8478
8479 case value_t::number_float:
8480 {
8481 dump_float(val.m_value.number_float);
8482 return;
8483 }
8484
```

#### SharedFunctionality/json.hpp, line 8487 (Dead Code) — Low

##### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

##### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8487
**Taint Flags:**

```
8484
8485 case value_t::discarded:
8486 {
8487 o->write_characters("<discarded>", 11);
8488 return;
8489 }
8490
```

#### SharedFunctionality/json.hpp, line 8493 (Dead Code) — Low

##### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

##### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8493
**Taint Flags:**

```
8490
8491 case value_t::null:
8492 {
8493 o->write_characters("null", 4);
8494 return;
8495 }
8496 }
```

## Dead Code | Low

### Package: SharedFunctionality

### SharedFunctionality/json.hpp, line 8322 (Dead Code) | Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8322
**Taint Flags:**

```
8319  return;
8320  }
8321
8322  if (pretty_print)
8323  {
8324  o->write_characters("{\n", 2);
8325
```

### SharedFunctionality/json.hpp, line 8395 (Dead Code) | Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8395
**Taint Flags:**

```
8392  return;
8393  }
8394
8395  if (pretty_print)
8396  {
8397  o->write_characters("[\n", 2);
8398
```

## Dead Code: Unused Field (1 issue)

### Abstract

This field is never used directly or indirectly by a public method.

### Explanation

This field is never accessed, except perhaps by dead code. Dead code is defined as code that is never directly or indirectly executed by a public method. It is likely that the field is simply vestigial, but it is also possible that the unused field points out a bug. **Example 1:** The field named `glue` is not used in the following class. The author of the class has accidentally put quotes around the field name, transforming it into a string constant.

```
public class Dead {

  string glue;

  public string GetGlue() {
    return "glue";
  }

}
```

**Example 2:** The field named `glue` is used in the following class, but only from a method that is never called by a public method.

```
public class Dead {

  string glue;

  private string GetGlue() {
    return glue;
  }

}
```

### Recommendation

In general, you should repair or remove dead code. To repair dead code, execute the dead code directly or indirectly through a public method. Dead code causes additional complexity and maintenance burden without contributing to the functionality of the program.

### Issue Summary

## Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Dead Code: Unused Field | 1 | 0 | 0 | 1 |
| **Total** | **1** | **0** | **0** | **1** |

## Dead Code: Unused Field — Low

## Package: CageServiceInstaller

### CageServiceInstaller/ServiceInstaller.cs, line 10 (Dead Code: Unused Field) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** Field: STANDARD_RIGHTS_REQUIRED
**File:** CageServiceInstaller/ServiceInstaller.cs:10
**Taint Flags:**

```
7
8  public static class ServiceInstaller
9  {
10  private const int STANDARD_RIGHTS_REQUIRED = 0xF0000;
11  private const int SERVICE_WIN32_OWN_PROCESS = 0x00000010;
12
13  [StructLayout(LayoutKind.Sequential)]
```

## Null Dereference (2 issues)

### Abstract

The program can potentially dereference a null pointer, thereby raising a `NullException`.

### Explanation

Null pointer errors are usually the result of one or more programmer assumptions being violated. Most null pointer issues result in general software reliability problems, but if an attacker can intentionally trigger a null pointer dereference, the attacker may be able to use the resulting exception to bypass security logic or to cause the application to reveal debugging information that will be valuable in planning subsequent attacks.
**Example 1:** In the following code, the programmer assumes that the system always has a property named "`cmd`" defined. If an attacker can control the program's environment so that "`cmd`" is not defined, the program throws a null pointer exception when it attempts to call the `Trim()` method.

```
string cmd = null;
...
cmd = Environment.GetEnvironmentVariable("cmd");
cmd = cmd.Trim();
```

### Recommendation

Security problems caused by dereferencing null pointers are almost always related to the way in which the program handles runtime exceptions. If the software has a solid and well-executed approach to dealing with runtime exceptions, the potential for security damage is significantly diminished.

### Issue Summary

### Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Null Dereference | 2 | 0 | 0 | 2 |
| **Total** | **2** | **0** | **0** | **2** |

**Null Dereference** **High**

**Package: CageChooser**

**CageChooser/CageChooserForm.cs, line 170 (Null Dereference)** **High**

**Issue Details**

| Null Dereference | High |
| --- | --- |

## Package: CageChooser

### CageChooser/CageChooserForm.cs, line 170 (Null Dereference) — High

**Kingdom:** Code Quality
**Scan Engine:** SCA (Control Flow)

#### Sink Details

**Sink:** install_dir.get_Length() : install_dir is not checked for null value before being dereferenced
**Enclosing Method:** openCageConfiguratorButton_Click()
**File:** CageChooser/CageChooserForm.cs:170
**Taint Flags:**

```
167  const string registry_key = @"HKEY_LOCAL_MACHINE\SOFTWARE\SharkCage";
168  var install_dir = Registry.GetValue(registry_key, "InstallDir", "") as string;
169
170  if (install_dir.Length == 0)
171  {
172  MessageBox.Show("Could not read installation directory from registry, opening
CageConfigurator not possible", "Shark Cage", MessageBoxButtons.OK,
MessageBoxIcon.Information);
173  return;
```

## Package: CageConfigurator

### CageConfigurator/CageConfiguratorForm.cs, line 363 (Null Dereference) — High

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Control Flow)

#### Sink Details

**Sink:** install_dir.get_Length() : install_dir is not checked for null value before being dereferenced
**Enclosing Method:** openCageChooserButton_Click()
**File:** CageConfigurator/CageConfiguratorForm.cs:363
**Taint Flags:**

```
360  const string registry_key = @"HKEY_LOCAL_MACHINE\SOFTWARE\SharkCage";
361  var install_dir = Registry.GetValue(registry_key, "InstallDir", "") as string;
362
363  if (install_dir.Length == 0)
364  {
365  MessageBox.Show("Could not read installation directory from registry, opening CageChooser
not possible", "Shark Cage", MessageBoxButtons.OK, MessageBoxIcon.Information);
366  return;
```

## Path Manipulation (9 issues)

### Abstract

Allowing user input to control paths used in file system operations could enable an attacker to access or modify otherwise protected system resources.

### Explanation

Path manipulation errors occur when the following two conditions are met: 1. An attacker is able to specify a path used in an operation on the file system. 2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted. For example, the program may give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker. **Example 1:** The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker may provide a file name like "`..\\..\\Windows\\System32\\krnl386.exe`", which will cause the application to delete an important Windows system file.

```
String rName = Request.Item("reportName");
...
File.delete("C:\\users\\reports\\" + rName);
```

**Example 2:** The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with adequate privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension ".txt".

```
sr = new StreamReader(resmngr.GetString("sub")+".txt");
while ((line = sr.ReadLine()) != null) {
Console.WriteLine(line);
}
```

### Recommendation

The best way to prevent path manipulation is with a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list. With this approach the input provided by the user is never used directly to specify the resource name. In some situations this approach is impractical because the set of legitimate resource names is too large or too hard to keep track of. Programmers often resort to blacklisting in these situations. Blacklisting selectively rejects or escapes potentially dangerous characters before using the input. However, any such list of unsafe characters is likely to be incomplete and will almost certainly become out of date. A better approach is to create a whitelist of characters that are allowed to appear in the resource name and accept input composed exclusively of characters in the approved set.

### Issue Summary

## Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Path Manipulation | 9 | 0 | 0 | 9 |
| **Total** | **9** | **0** | **0** | **9** |

## Path Manipulation — Critical

### Package: CageConfigurator

### CageConfigurator/CageConfiguratorForm.cs, line 476 (Path Manipulation) — Critical

#### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

#### Source Details

**Source:** System.Windows.Forms.TextBox.get_Text()
**From:** CageConfigurator.CageConfiguratorForm.saveButton_Click
**File:** CageConfigurator/CageConfiguratorForm.cs:419

```
416  writer.WritePropertyName("has_signature");
417  writer.WriteValue(IsFileSigned(applicationPath.Text));
418  writer.WritePropertyName("binary_hash");
419  writer.WriteValue(GetSha512Hash(applicationPath.Text));
420  writer.WritePropertyName(TOKEN_PROPERTY);
421  writer.WriteValue(GetBase64FromImage(tokenBox.Image));
422  writer.WritePropertyName(ADDITIONAL_APP_NAME_PROPERTY);
```

#### Sink Details

**Sink:** System.IO.File.OpenRead()
**Enclosing Method:** GetSha512Hash()
**File:** CageConfigurator/CageConfiguratorForm.cs:476
**Taint Flags:** GUI_FORM

```
473
474  private static string GetSha512Hash(string file_path)
475  {
476  using (var bs = new BufferedStream(File.OpenRead(file_path), 1048576))
477  {
478  var sha = new SHA512Managed();
479  byte[] hash = sha.ComputeHash(bs);
```

### CageConfigurator/CageConfiguratorForm.cs, line 499 (Path Manipulation) — Critical

#### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

#### Source Details

**Source:** System.Windows.Forms.TextBox.get_Text()

## Path Manipulation — Critical

### Package: CageConfigurator

### CageConfigurator/CageConfiguratorForm.cs, line 499 (Path Manipulation) — Critical

**From:** CageConfigurator.CageConfiguratorForm.saveButton_Click
**File:** CageConfigurator/CageConfiguratorForm.cs:417

```
414 writer.WritePropertyName(APPLICATION_PATH_PROPERTY);
415 writer.WriteValue(applicationPath.Text);
416 writer.WritePropertyName("has_signature");
417 writer.WriteValue(IsFileSigned(applicationPath.Text));
418 writer.WritePropertyName("binary_hash");
419 writer.WriteValue(GetSha512Hash(applicationPath.Text));
420 writer.WritePropertyName(TOKEN_PROPERTY);
```

#### Sink Details

**Sink:** System.Security.Cryptography.X509Certificates.X509Certificate.CreateFromSignedFile()
**Enclosing Method:** IsFileSigned()
**File:** CageConfigurator/CageConfiguratorForm.cs:499
**Taint Flags:** GUI_FORM

```
496 {
497 try
498 {
499 X509Certificate.CreateFromSignedFile(file_path);
500 return true;
501 }
502 catch
```

## Path Manipulation — Medium

### Package: CageChooser

### CageChooser/CageChooserForm.cs, line 91 (Path Manipulation) — Medium

#### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

#### Source Details

**Source:** System.Windows.Forms.TextBox.get_Text()
**From:** CageChooser.CageChooserForm.configPath_Leave
**File:** CageChooser/CageChooserForm.cs:81

```
78
79 private void configPath_Leave(object sender, EventArgs e)
80 {
81 CheckConfigPath(configPath.Text, addToLRUconfigs);
82 }
83
```

## Path Manipulation — Medium

### Package: CageChooser

#### CageChooser/CageChooserForm.cs, line 91 (Path Manipulation) — Medium

```
84  private void CheckConfigPath(string config_path, Action<string> onSuccess)
```

##### Sink Details

**Sink:** System.IO.File.Exists()
**Enclosing Method:** CheckConfigPath()
**File:** CageChooser/CageChooserForm.cs:91
**Taint Flags:** GUI_FORM

```
88  return;
89  }
90
91  if (config_path.EndsWith(".sconfig") && File.Exists(config_path))
92  {
93  onSuccess(config_path);
94  }
```

### Package: CageConfigurator

#### CageConfigurator/CageConfiguratorForm.cs, line 90 (Path Manipulation) — Medium

##### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

##### Source Details

**Source:** System.Windows.Forms.TextBox.get_Text()
**From:** CageConfigurator.CageConfiguratorForm.applicationPath_Leave
**File:** CageConfigurator/CageConfiguratorForm.cs:146

```
143
144  private void applicationPath_Leave(object sender, EventArgs e)
145  {
146  CheckPath(applicationPath.Text, ".exe", (string path) =>
147  {
148  SetUnsavedData(true);
149  });
```

##### Sink Details

**Sink:** System.IO.File.Exists()
**Enclosing Method:** CheckPath()
**File:** CageConfigurator/CageConfiguratorForm.cs:90
**Taint Flags:** GUI_FORM

```
87  return path.EndsWith(type);
88  });
89
```

## Path Manipulation — Medium

### Package: CageConfigurator

#### CageConfigurator/CageConfiguratorForm.cs, line 90 (Path Manipulation) — Medium

```
90 if (matching_type && File.Exists(path))
91 {
92 onSuccess?.Invoke(path);
93 }
```

## Path Manipulation — Low

### Package: CageChooser

#### CageChooser/CageChooserForm.cs, line 29 (Path Manipulation) — Low

##### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

##### Source Details

**Source:** CageChooser.Properties.Settings.get_PersistentConfigPath()
**From:** CageChooser.CageChooserForm.CageChooser_Load
**File:** CageChooser/CageChooserForm.cs:29

```
26
27 private void CageChooser_Load(object sender, EventArgs e)
28 {
29 if (!File.Exists(Settings.Default.PersistentConfigPath))
30 {
31 configPath.Text = String.Empty;
32 }
```

##### Sink Details

**Sink:** System.IO.File.Exists()
**Enclosing Method:** CageChooser_Load()
**File:** CageChooser/CageChooserForm.cs:29
**Taint Flags:** PROPERTY

```
26
27 private void CageChooser_Load(object sender, EventArgs e)
28 {
29 if (!File.Exists(Settings.Default.PersistentConfigPath))
30 {
31 configPath.Text = String.Empty;
32 }
```

#### CageChooser/CageChooserForm.cs, line 37 (Path Manipulation) — Low

##### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

## Path Manipulation — Low

### Package: CageChooser

### CageChooser/CageChooserForm.cs, line 37 (Path Manipulation) — Low

#### Source Details

**Source:** CageChooser.Properties.Settings.get_PersistentLRUConfigs()
**From:** CageChooser.CageChooserForm.CageChooser_Load
**File:** CageChooser/CageChooserForm.cs:35

```
32 }
33 if (Settings.Default.PersistentLRUConfigs != null)
34 {
35 foreach (string lruConfig in Settings.Default.PersistentLRUConfigs)
36 {
37 if (File.Exists(lruConfig))
38 {
```

#### Sink Details

**Sink:** System.IO.File.Exists()
**Enclosing Method:** CageChooser_Load()
**File:** CageChooser/CageChooserForm.cs:37
**Taint Flags:** PROPERTY

```
34 {
35 foreach (string lruConfig in Settings.Default.PersistentLRUConfigs)
36 {
37 if (File.Exists(lruConfig))
38 {
39 lruConfigs.Items.Add(lruConfig);
40 }
```

### CageChooser/CageChooserForm.cs, line 37 (Path Manipulation) — Low

#### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

#### Source Details

**Source:** System.Configuration.ApplicationSettingsBase.get_Item()
**From:** CageChooser.Properties.Settings.get_PersistentLRUConfigs
**File:** CageChooser/Properties/Settings.Designer.cs:42

```
39 [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
40 public global::System.Collections.Specialized.StringCollection
PersistentLRUConfigs {
41 get {
42 return ((global::System.Collections.Specialized.StringCollection)
(this["PersistentLRUConfigs"]));
43 }
```

| Path Manipulation | Low |
|---|---|

## Package: CageChooser

### CageChooser/CageChooserForm.cs, line 37 (Path Manipulation) — Low

```
44 set {
45 this["PersistentLRUConfigs"] = value;
```

### Sink Details

**Sink:** System.IO.File.Exists()
**Enclosing Method:** CageChooser_Load()
**File:** CageChooser/CageChooserForm.cs:37
**Taint Flags:** PROPERTY

```
34 {
35 foreach (string lruConfig in Settings.Default.PersistentLRUConfigs)
36 {
37 if (File.Exists(lruConfig))
38 {
39 lruConfigs.Items.Add(lruConfig);
40 }
```

### CageChooser/CageChooserForm.cs, line 29 (Path Manipulation) — Low

### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

### Source Details

**Source:** System.Configuration.ApplicationSettingsBase.get_Item()
**From:** CageChooser.Properties.Settings.get_PersistentConfigPath
**File:** CageChooser/Properties/Settings.Designer.cs:31

```
28 [global::System.Configuration.DefaultSettingValueAttribute("")]
29 public string PersistentConfigPath {
30 get {
31 return ((string)(this["PersistentConfigPath"]));
32 }
33 set {
34 this["PersistentConfigPath"] = value;
```

### Sink Details

**Sink:** System.IO.File.Exists()
**Enclosing Method:** CageChooser_Load()
**File:** CageChooser/CageChooserForm.cs:29
**Taint Flags:** PROPERTY

```
26
27 private void CageChooser_Load(object sender, EventArgs e)
28 {
29 if (!File.Exists(Settings.Default.PersistentConfigPath))
```

## Path Manipulation — Low

### Package: CageChooser

#### CageChooser/CageChooserForm.cs, line 29 (Path Manipulation) — Low

```
30  {
31  configPath.Text = String.Empty;
32  }
```

### Package: CageConfigurator

#### CageConfigurator/CageConfiguratorForm.cs, line 90 (Path Manipulation) — Low

##### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

##### Source Details

**Source:** Main(0)
**From:** CageConfigurator.Program.Main
**File:** CageConfigurator/CageConfigurator.cs:12

```
9   /// The main entry point for the application.
10  /// </summary>
11  [STAThread]
12  static void Main(string[] parameter)
13  {
14  Application.EnableVisualStyles();
15  Application.SetCompatibleTextRenderingDefault(false);
```

##### Sink Details

**Sink:** System.IO.File.Exists()
**Enclosing Method:** CheckPath()
**File:** CageConfigurator/CageConfiguratorForm.cs:90
**Taint Flags:** ARGS

```
87  return path.EndsWith(type);
88  });
89
90  if (matching_type && File.Exists(path))
91  {
92  onSuccess?.Invoke(path);
93  }
```

## Poor Error Handling: Empty Catch Block (1 issue)

### Abstract

Ignoring an exception can cause the program to overlook unexpected states and conditions.

### Explanation

Just about every serious attack on a software system begins with the violation of a programmer's assumptions. After the attack, the programmer's assumptions seem flimsy and poorly founded, but before an attack many programmers would defend their assumptions well past the end of their lunch break. Two dubious assumptions that are easy to spot in code are "this method call can never fail" and "it doesn't matter if this call fails". When programmers ignore exceptions, they implicitly state that they are operating under one of these assumptions. **Example 1:** The following code excerpt ignores a rarely-thrown exception from `DoExchange()`.

```
try {
  DoExchange();
}
catch (RareException e) {
  // this can never happen
}
```

If a `RareException` were to ever be thrown, the program would continue to execute as though nothing unusual had occurred. The program records no evidence indicating the special situation, potentially frustrating any later attempt to explain the program's behavior.

### Recommendation

At a minimum, log the fact that the exception was thrown so that it will be possible to come back later and make sense of the resulting program behavior. Better yet, abort the current operation. **Example 2:** The code in Example 1 could be rewritten in the following way:

```
try {
  DoExchange();
}
catch (RareException e) {
  Log.Error("This can never happen: " + e);
}
```

### Issue Summary

## Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Poor Error Handling: Empty Catch Block | 1 | 0 | 0 | 1 |
| **Total** | **1** | **0** | **0** | **1** |

## Poor Error Handling: Empty Catch Block — Low

## Package: CageChooser

### CageChooser/CageChooser.cs, line 31 (Poor Error Handling: Empty Catch Block) Low

#### Issue Details

**Kingdom:** Errors
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** CatchBlock
**Enclosing Method:** Main()
**File:** CageChooser/CageChooser.cs:31
**Taint Flags:**

```
28  p.Start();
29  p.WaitForExit();
30  }
31  catch { /* not accepting the admin prompt causes an exception*/ }
32 #endif
33
34  // check if service is running
```

# Poor Error Handling: Overly Broad Catch (3 issues)

## Abstract

The catch block handles a broad swath of exceptions, potentially trapping dissimilar issues or problems that should not be dealt with at this point in the program.

## Explanation

Multiple catch blocks can get ugly and repetitive, but "condensing" catch blocks by catching a high-level class like `Exception` can obscure exceptions that deserve special treatment or that should not be caught at this point in the program. Catching an overly broad exception essentially defeats the purpose of .NET's typed exceptions, and can become particularly dangerous if the program grows and begins to throw new types of exceptions. The new exception types will not receive any attention. **Example:** The following code excerpt handles three types of exceptions in an identical fashion.

```
try {
  DoExchange();
}
catch (IOException e) {
  logger.Error("DoExchange failed", e);
}
catch (FormatException e) {
  logger.Error("DoExchange failed", e);
}
catch (TimeoutException e) {
  logger.Error("DoExchange failed", e);
}
```

At first blush, it may seem preferable to deal with these exceptions in a single catch block, as follows:

```
try {
  DoExchange();
}
catch (Exception e) {
  logger.Error("DoExchange failed", e);
}
```

However, if `DoExchange()` is modified to throw a new type of exception that should be handled in some different kind of way, the broad catch block will prevent the compiler from pointing out the situation. Further, the new catch block will now also handle exceptions of types `ApplicationException` and `NullReferenceException`, which is not the programmer's intent.

## Recommendation

Do not catch broad exception classes like `Exception`, , or  except at the very top level of the program or thread.

## Issue Summary

## Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Poor Error Handling: Overly Broad Catch | 3 | 0 | 0 | 3 |
| **Total** | **3** | **0** | **0** | **3** |

## Poor Error Handling: Overly Broad Catch — Low

### Package: CageConfigurator

### CageConfigurator/CageConfiguratorForm.cs, line 249 (Poor Error Handling: Overly Broad Catch) — Low

#### Issue Details

**Kingdom:** Errors
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** CatchBlock
**Enclosing Method:** GetImageFromBase64()
**File:** CageConfigurator/CageConfiguratorForm.cs:249
**Taint Flags:**

```
246
247   return Image.FromStream(ms, true);
248   }
249   catch (Exception)
250   {
251   return null;
252   }
```

### CageConfigurator/CageConfiguratorForm.cs, line 502 (Poor Error Handling: Overly Broad Catch) — Low

#### Issue Details

**Kingdom:** Errors
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** CatchBlock
**Enclosing Method:** IsFileSigned()

## Poor Error Handling: Overly Broad Catch — Low

### Package: CageConfigurator

### CageConfigurator/CageConfiguratorForm.cs, line 502 (Poor Error Handling: Overly Broad Catch) — Low

**File:** CageConfigurator/CageConfiguratorForm.cs:502
**Taint Flags:**

```
X509Certificate.CreateFromSignedFile(file_path);
return true;
}
catch
{
return false;
}
```

### CageConfigurator/CageConfiguratorForm.cs, line 219 (Poor Error Handling: Overly Broad Catch) — Low

#### Issue Details

**Kingdom:** Errors
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** CatchBlock
**Enclosing Method:** LoadConfig()
**File:** CageConfigurator/CageConfiguratorForm.cs:219
**Taint Flags:**

```
current_config_name = config_path;
Text = $"Cage Configurator - {current_config_name}";
}
catch (Exception e)
{
MessageBox.Show($"Could not load config: {e.ToString()}");
}
```

# Poor Style: Variable Never Used (7 issues)

## Abstract

This variable is never used.

## Explanation

This variable is never used. It is likely that the variable is simply vestigial, but it is also possible that the unused variable points out a bug. **Example:** In the following code, a copy-and-paste error has led to the same loop iterator (`i`) being used twice. The variable `j` is never used.

```
int i,j;

for (i=0; i < outer; i++) {
  for (i=0; i < inner; i++) {
    ...
```

## Recommendation

In general, you should eliminate unused variables in order to make the code easier to understand and maintain.

## Issue Summary

## Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Poor Style: Variable Never Used | 7 | 0 | 0 | 7 |
| **Total** | **7** | **0** | **0** | **7** |

**Poor Style: Variable Never Used** — **Low**

**Package: <none>**

**CageManager/SecuritySetup.cpp, line 67 (Poor Style: Variable Never Used)** — **Low**

**Issue Details**

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

**Sink Details**

## Poor Style: Variable Never Used — Low

### Package: <none>

### CageManager/SecuritySetup.cpp, line 67 (Poor Style: Variable Never Used) — Low

**Sink:** Variable: group_name_buf
**Enclosing Method:** CreateSID()
**File:** CageManager/SecuritySetup.cpp:67
**Taint Flags:**

```
64 DWORD buffer_size = 0;
65
66 // create a group
67 std::vector<wchar_t> group_name_buf(group_name.begin(), group_name.end());
68 group_name_buf.push_back(0);
69 localgroup_info.lgrpi0_name = group_name_buf.data();
70 ::NetLocalGroupAdd(NULL, 0, reinterpret_cast<LPBYTE>(&localgroup_info), NULL);
```

### Package: SharedFunctionality

### SharedFunctionality/json.hpp, line 8335 (Poor Style: Variable Never Used) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** Variable: cnt
**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8335
**Taint Flags:**

```
8332
8333 // first n-1 elements
8334 auto i = val.m_value.object->cbegin();
8335 for (std::size_t cnt = 0; cnt < val.m_value.object->size() - 1; ++cnt, ++i)
8336 {
8337 o->write_characters(indent_string.c_str(), new_indent);
8338 o->write_character('\"');
```

### SharedFunctionality/json.hpp, line 8364 (Poor Style: Variable Never Used) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** Variable: cnt
**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8364
**Taint Flags:**

```
8361
8362 // first n-1 elements
```

## Poor Style: Variable Never Used — Low

### Package: SharedFunctionality

### SharedFunctionality/json.hpp, line 8364 (Poor Style: Variable Never Used) — Low

```
8363 auto i = val.m_value.object->cbegin();
8364 for (std::size_t cnt = 0; cnt < val.m_value.object->size() - 1; ++cnt, ++i)
8365 {
8366 o->write_character('\"');
8367 dump_escaped(i->first, ensure_ascii);
```

### SharedFunctionality/json.hpp, line 8334 (Poor Style: Variable Never Used) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** Variable: i
**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8334
**Taint Flags:**

```
8331 }
8332
8333 // first n-1 elements
8334 auto i = val.m_value.object->cbegin();
8335 for (std::size_t cnt = 0; cnt < val.m_value.object->size() - 1; ++cnt, ++i)
8336 {
8337 o->write_characters(indent_string.c_str(), new_indent);
```

### SharedFunctionality/json.hpp, line 8363 (Poor Style: Variable Never Used) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** Variable: i
**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8363
**Taint Flags:**

```
8360 o->write_character('{');
8361
8362 // first n-1 elements
8363 auto i = val.m_value.object->cbegin();
8364 for (std::size_t cnt = 0; cnt < val.m_value.object->size() - 1; ++cnt, ++i)
8365 {
8366 o->write_character('\"');
```

## Poor Style: Variable Never Used — Low

### Package: SharedFunctionality

### SharedFunctionality/json.hpp, line 8407 (Poor Style: Variable Never Used) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** Variable: i
**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8407
**Taint Flags:**

```
8404 }
8405
8406 // first n-1 elements
8407 for (auto i = val.m_value.array->cbegin();
8408 i != val.m_value.array->cend() - 1; ++i)
8409 {
8410 o->write_characters(indent_string.c_str(), new_indent);
```

### SharedFunctionality/json.hpp, line 8429 (Poor Style: Variable Never Used) — Low

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** Variable: i
**Enclosing Method:** dump()
**File:** SharedFunctionality/json.hpp:8429
**Taint Flags:**

```
8426 o->write_character('[');
8427
8428 // first n-1 elements
8429 for (auto i = val.m_value.array->cbegin();
8430 i != val.m_value.array->cend() - 1; ++i)
8431 {
8432 dump(*i, false, ensure_ascii, indent_step, current_indent);
```

## Privacy Violation: Heap Inspection (1 issue)

### Abstract

Storing sensitive data in an insecure manner makes it possible to extract the data via inspecting the heap.

### Explanation

Sensitive data (such as passwords, social security numbers, credit card numbers, encryption keys etc.) stored in an unmanaged memory buffer can be leaked if it is not explicitly zeroed out, even if it is freed. The unmanaged buffers are often not encrypted by default, so anyone that can read the process' memory will be able to see the contents. Furthermore, if the process' memory gets swapped out to disk, the unencrypted contents of the string will be written to a swap file. In the event of an application crash, a memory dump of the application might reveal sensitive data. **Example 1:** The following example creates a symmetric key before using it.

```
public static void CreateAndUseEncryptor()
{
  SymmetricAlgorithm aesAlgorithm = SymmetricAlgorithm.Create("AES");
  aesAlgorithm.GenerateKey();
  aesAlgorithm.GenerateIV();
  Encrypt(aesAlgorithm);
}
```

Since neither `CreateAndUseEncryptor()` nor `Encrypt()` run `Clear()` or `Dispose(true)` on the `SymmetricAlgorithm` object, the key and initialization vector (IV) will not be zeroed out in memory.

### Recommendation

After creating an initialization vector (IV) or encryption key, it is absolutely necessary to make sure they are cleared from memory by either running `Clear()` or `Dispose(true)` on the object. **Example 1:** The following method generates a key and an IV, then uses a finally block to make sure the key and IV are zeroed out in memory.

```
public static void CreateAndUseEncryptor()
{
  SymmetricAlgorithm aesAlgorithm = null;
  try
  {
    aesAlgorithm = SymmetricAlgorithm.Create("AES");
    aesAlgorithm.GenerateKey();
    aesAlgorithm.GenerateIV();
    Encrypt(aesAlgorithm);
  }
  finally
  {
    if (aesAlgorithm != null)
    {
      aesAlgorithm.Clear();
    }
  }
}
```

**Example 2:** The following example uses a using-block that automatically calls `Dispose()`, which zeroes out the key and IV in memory.

```
public static void CreateAndUseEncryptor()
{
  using (SymmetricAlgorithm aesAlgorithm = SymmetricAlgorithm.Create("AES"))
  {
    aesAlgorithm.GenerateKey();
```

```
        aesAlgorithm.GenerateIV();
        Encrypt(aesAlgorithm);
    }
}
```

In the various symmetric, asymmetric and hash algorithm implementations, `Dispose()` is overridden by calling `Clear()` then `Dispose(true)`

## Issue Summary

## Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Privacy Violation: Heap Inspection | 1 | 0 | 0 | 1 |
| **Total** | **1** | **0** | **0** | **1** |

## Privacy Violation: Heap Inspection — High

## Package: CageConfigurator

### CageConfigurator/CageConfiguratorForm.cs, line 478 (Privacy Violation: Heap Inspection) — High

#### Issue Details

**Kingdom:** Security Features
**Scan Engine:** SCA (Control Flow)

#### Sink Details

**Sink:** sha = new SHA512Managed() : Key algorithm initialized
**Enclosing Method:** GetSha512Hash()
**File:** CageConfigurator/CageConfiguratorForm.cs:478
**Taint Flags:**

```
475 {
476 using (var bs = new BufferedStream(File.OpenRead(file_path), 1048576))
477 {
478 var sha = new SHA512Managed();
479 byte[] hash = sha.ComputeHash(bs);
480 return BitConverter.ToString(hash).Replace("-", String.Empty);
481 }
```

## Type Mismatch: Signed to Unsigned (1 issue)

### Abstract

The function is declared to return an unsigned number but returns a signed value.

### Explanation

It is dangerous to rely on implicit casts between signed and unsigned numbers because the result can take on an unexpected value and violate weak assumptions made elsewhere in the program. **Example:** In this example, depending on the return value of `accecssmainframe()`, the variable `amount` can hold a negative value when it is returned. Because the function is declared to return an unsigned value, `amount` will be implicitly cast to an unsigned number.

```
unsigned int readdata () {
    int amount = 0;
...
amount = accessmainframe();
...
    return amount;
}
```

If the return value of `accessmainframe()` is -1, then the return value of `readdata()` will be 4,294,967,295 on a system that uses 32-bit integers. Conversion between signed and unsigned values can lead to a variety of errors, but from a security standpoint is most commonly associated with integer overflow and buffer overflow vulnerabilities.

### Recommendation

Although unexpected conversion between signed and unsigned quantities typically creates general quality problems, depending on the assumptions that a conversion violates, it can lead to serious security risks. Pay attention to compiler warnings related to signed/unsigned conversions. Some programmers may believe that these warnings are innocuous, but in some cases they point out potential integer overflow problems.

### Issue Summary

### Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Type Mismatch: Signed to Unsigned | 1 | 0 | 0 | 1 |
| **Total** | **1** | **0** | **0** | **1** |

## Type Mismatch: Signed to Unsigned — High

### Package: <none>

### CageManager/base64.cpp, line 29 (Type Mismatch: Signed to Unsigned) — High

#### Issue Details

**Kingdom:** Code Quality
**Scan Engine:** SCA (Structural)

#### Sink Details

**Sink:** AssignmentStatement
**Enclosing Method:** base64_encode()
**File:** CageManager/base64.cpp:29
**Taint Flags:**

```
26
27 while (in_len--)
28 {
29 char_array_3[i++] = *(to_encode++);
30 if (i == 3)
31 {
32 char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
```

## Unsafe Native Invoke (1 issue)

### Abstract

Improper use of the Platform Invocation Services can render managed applications vulnerable to security flaws in other languages.

### Explanation

Unsafe Native Invoke errors occur when a managed application uses P/Invoke to call native (unmanaged) code written in another programming language. **Example:** The following C# code defines a class named `Echo`. The class declares one native method (defined below), which uses C to echo commands entered on the console back to the user.

```
class Echo
{
  [DllImport("mylib.dll")]
  internal static extern void RunEcho();

  static void main(String[] args)
  {
    RunEcho();
  }
}
```

The following C code defines the native method implemented in the `Echo` class:

```
#include <stdio.h>

void __stdcall RunEcho()
{
  char* buf = (char*) malloc(64 * sizeof(char));
  gets(buf);
  printf(buf);
}
```

Because the Echo is implemented in managed code, it may appear that it is immune to memory issues like buffer overflow vulnerabilities. Although the managed environment does do a good job of making memory operations safe, this protection does not extend to vulnerabilities occurring in native code accessed using P/Invoke. Despite the memory protections offered in the managed runtime environment, the native code in this example is vulnerable to a buffer overflow because it makes use of `gets()`, which does not perform any bounds checking on its input. As well, `buf` is allocated but not freed and therefore is a memory leak. The vulnerability in the example above could easily be detected through a source code audit of the native method implementation. This may not be practical or possible depending on the availability of source code and the way the project is built, but in many cases it may suffice. However, the ability to share objects between the managed and native environments expands the potential risk to much more insidious cases where improper data handling in managed code may lead to unexpected vulnerabilities in native code or to unsafe operations in native code corrupting data structures in managed code. Vulnerabilities in native code accessed through a managed application are typically exploited in the same manner as they are in applications written in the native language. The only challenge to such an attack is for the attacker to identify that the managed application uses native code to perform certain operations. This can be accomplished in a variety of ways, including identifying specific behaviors that are often implemented with native code or by exploiting a system information leak in the managed application that exposes its use of P/Invoke.

### Recommendation

Audit all source code comprising a given application, including native methods implemented in native code. During audits, ensure that differences in bounds checking and other behavior between managed and native code are accounted for and handled correctly. In particular, verify that shared objects are handled correctly at all stages: before they are passed to native code, while they are manipulated by native code, and after they are returned to the managed application.

## Issue Summary

## Engine Breakdown

| | SCA | WebInspect | SecurityScope | Total |
|---|---|---|---|---|
| Unsafe Native Invoke | 1 | 0 | 0 | 1 |
| **Total** | **1** | **0** | **0** | **1** |

| Unsafe Native Invoke | High |
|---|---|
| **Package: CageChooser** | |
| **CageChooser/CageChooserForm.cs, line 149 (Unsafe Native Invoke)** | **High** |

### Issue Details

**Kingdom:** Input Validation and Representation
**Scan Engine:** SCA (Data Flow)

### Source Details

**Source:** System.Windows.Forms.TextBox.get_Text()
**From:** CageChooser.CageChooserForm.openButton_Click
**File:** CageChooser/CageChooserForm.cs:149

```
146 {
147 if (configPath.Text != String.Empty)
148 {
149 NativeMethods.SendConfigAndExternalProgram(configPath.Text);
150
151 // bring the form back in focus
152 Activate();
```

### Sink Details

**Sink:** CageChooser.CageChooserForm.NativeMethods.SendConfigAndExternalProgram()
**Enclosing Method:** openButton_Click()
**File:** CageChooser/CageChooserForm.cs:149
**Taint Flags:** GUI_FORM

## Unsafe Native Invoke — High

### Package: CageChooser

### CageChooser/CageChooserForm.cs, line 149 (Unsafe Native Invoke) — High

```
{
if (configPath.Text != String.Empty)
{
NativeMethods.SendConfigAndExternalProgram(configPath.Text);

// bring the form back in focus
Activate();
```

# Description of Key Terminology

## Likelihood and Impact

**Likelihood**
Likelihood is the probability that a vulnerability will be accurately identified and successfully exploited.

**Impact**
Impact is the potential damage an attacker could do to assets by successfully exploiting a vulnerability. This damage can be in the form of, but not limited to, financial loss, compliance violation, loss of brand reputation, and negative publicity.

## Fortify Priority Order

**Critical**
Critical-priority issues have high impact and high likelihood. Critical-priority issues are easy to detect and exploit and result in large asset damage. These issues represent the highest security risk to the application. As such, they should be remediated immediately.

SQL Injection is an example of a critical issue.

**High**
High-priority issues have high impact and low likelihood. High-priority issues are often difficult to detect and exploit, but can result in large asset damage. These issues represent a high security risk to the application. High-priority issues should be remediated in the next scheduled patch release.

Password Management: Hardcoded Password is an example of a high issue.

**Medium**
Medium-priority issues have low impact and high likelihood. Medium-priority issues are easy to detect and exploit, but typically result in small asset damage. These issues represent a moderate security risk to the application. Medium-priority issues should be remediated in the next scheduled product update.

Path Manipulation is an example of a medium issue.

**Low**
Low-priority issues have low impact and low likelihood. Low-priority issues can be difficult to detect and exploit and typically result in small asset damage. These issues represent a minor security risk to the application. Low-priority issues should be remediated as time allows.

Dead Code is an example of a low issue.

## About Fortify Solutions

Fortify is the leader in end-to-end application security solutions with the flexibility of testing on-premise and on-demand to cover the entire software development lifecycle. Learn more at software.microfocus.com/en-us/solutions/application-security.