Version 9.6.4 - Queries Release Notes

New Queries:

Language	Group	Name	CWE
Rust	Rust_Low_Visibility	Missing_Password_Field_Masking	549
Rust	Rust_Medium_Threat	Empty_Password_In_Connection_String	521
Rust	Rust_Medium_Threat	Hardcoded_Password_in_Connection_String	547
Rust	Rust_Medium_Threat	Password_In_Comment	615
Rust	Rust_Medium_Threat	SSRF	918
Rust	Rust_Medium_Threat	Unrestricted_Delete_S3	639
Rust	Rust_Medium_Threat	Unrestricted_Read_S3	639
Rust	Rust_Medium_Threat	Unrestricted_Write_S3	639
Rust	Rust_Medium_Threat	Use_Of_Hardcoded_Password	259

Changed Queries:

Language	Group	Name	CWE	Changed Fields
CPP	CPP_Best_Coding_Practice	Buffer_Size_Literal_Condition	118	**Source** has changed
CPP	CPP_Best_Coding_Practice	Empty_Methods	398	**Source** has changed
CPP	CPP_Buffer_Overflow	Buffer_Overflow_AddressOfLocalVarReturned	562	**Source** has changed
CPP	CPP_Buffer_Overflow	Buffer_Overflow_LongString	120	**Source** has changed
CPP	CPP_Buffer_Overflow	Buffer_Overflow_Unbounded_Buffer	120	**Source** has changed
CPP	CPP_Buffer_Overflow	Buffer_Overflow_Unbounded_Format	120	**Source** has changed
CPP	CPP_Buffer_Overflow	Off_by_One_Error	193	**Source** has changed
CPP	CPP_Low_Visibility	NULL_Pointer_Dereference	476	**Source** has changed
CPP	CPP_Low_Visibility	Privacy_Violation	359	**Source** has changed
CPP	CPP_Medium_Threat	Divide_By_Zero	369	**Source** has changed
CPP	CPP_Medium_Threat	Double_Free	415	**Source** has changed
CPP	CPP_Medium_Threat	MemoryFree_on_StackVariable	590	**Source** has changed
CPP	CPP_Medium_Threat	Memory_Leak	401	**Source** has changed
CPP	CPP_Medium_Threat	Pointer_Subtraction_Determines_Size	469	**Source** has changed
CPP	CPP_MISRA_C	R08_05_Object_Function_In_Header_File	0	**Source** has changed
CPP	CPP_MISRA_C	R16_01_Function_With_Variable_Number_Of_Arguments	0	**Source** has changed
CPP	CPP_MISRA_C	R19_01_Non_Prepocessor_Command_Before_Include_In_File	0	**Source** has changed
CPP	CPP_MISRA_C	R19_02_Non_Standard_Chars_In_Header_File_Name	0	**Source** has changed
CPP	CPP_MISRA_C	R19_03_Include_Directive_In_Wrong_Format	0	**Source** has changed
CPP	CPP_MISRA_C	R20_05_Using_Errno_Indicator_From_Errno_H	0	**Source** has changed
CPP	CPP_MISRA_C	R20_06_Using_Offsetof_Macro_From_Stddef_H	0	**Source** has changed
CPP	CPP_MISRA_C	R20_07_Using_Setjmp_Longjmp_Macros_From_Setjmp_H	0	**Source** has changed
CPP	CPP_MISRA_C	R20_08_Using_Signal_Handling_From_Signal_H	0	**Source** has changed
CPP	CPP_MISRA_C	R20_09_Using_Input_Output_From_Stdio_H	0	**Source** has changed
CPP	CPP_MISRA_C	R20_10_Using_Atof_Atoi_Atol_Functions_From_Stdlib_H	0	**Source** has changed
CPP	CPP_MISRA_C	R20_11_Using_Abort_Exit_Getenv_System_Functions_From_Stdlib_H	0	**Source** has changed
CPP	CPP_MISRA_C	R20_12_Using_Time_Handling_From_Time_H	0	**Source** has changed
CPP	CPP_MISRA_CPP	R00_01_05_Find_Unused_Typedefs	10776	**Source** has changed
CPP	CPP_MISRA_CPP	R00_01_10_Find_Unused_Defined_Functions	10777	**Source** has changed
CPP	CPP_MISRA_CPP	R07_03_01_Definitions_in_Global_Namespace	10786	**Source** has changed
CPP	CPP_MISRA_CPP	R16_00_03_Use_Of_Undef_Directive	0	**Source** has changed

Language	Group	Name	CWE	Changed Fields
CPP	CPP_MISRA_CPP	R16_00_07_Undefined_Macro_Identifiers	10799	**Source** has changed
CPP	CPP_MISRA_CPP	R16_01_01_Defined_Standart_Forms	10774	**Source** has changed
CPP	CPP_MISRA_CPP	R16_02_06_Include_Directive_In_Wrong_Format	0	**Source** has changed
CPP	CPP_MISRA_CPP	R18_00_04_Ctime	0	**Source** has changed
CPP	CPP_MISRA_CPP	R18_07_01_Csignal	0	**Source** has changed
CPP	CPP_MISRA_C_2012	R02_X_Unused_Code	0	**Source** has changed
CPP	CPP_MISRA_C_2012	R05_X_Identifiers	0	**Source** has changed
CPP	CPP_MISRA_C_2012	R08_02_Function_Prototype_With_Named_Parameters	0	**Source** has changed
CPP	CPP_MISRA_C_2012	R17_07_Value_Returned_By_Non_Void_Function_Shall_Be_Used	0	**Source** has changed
CPP	CPP_MISRA_C_2012	R17_08_Function_Parameter_Should_Not_Be_Modified	0	**Source** has changed
CPP	CPP_MISRA_C_2012	R18_05_Pointer_Nesting	0	**Source** has changed
CPP	CPP_MISRA_C_2012	R18_07_to_08_Variable_Length_And_Flexible_Arrays	0	**Source** has changed
CPP	CPP_MISRA_C_2012	R20_01_Include_Directive_Precedence	0	**Source** has changed
CSharp	CSharp_APISecurity	CSharp_WebApi_GetApiList	0	**Source** has changed
CSharp	CSharp_Exploitable_Path	CSharp_Find_UnresolvedMethods	0	**Source** has changed
CSharp	CSharp_Heuristic	Heuristic_CSRF	352	**Source** has changed
CSharp	CSharp_High_Risk	Reflected_XSS_All_Clients	79	**Source** has changed
CSharp	CSharp_High_Risk	Second_Order_SQL_Injection	89	**Source** has changed
CSharp	CSharp_High_Risk	Stored_XSS	79	**Source** has changed
CSharp	CSharp_Low_Visibility	Log_Forging	117	**Source** has changed
Go	Go_High_Risk	Stored_Command_Injection	77	**Source** has changed
Go	Go_High_Risk	Stored_XSS_All_Clients	79	**Source** has changed
Go	Go_Low_Visibility	Stored_Command_Argument_Injection	88	**Source** has changed
Go	Go_Medium_Threat	Stored_Absolute_Path_Traversal	36	**Source** has changed
Go	Go_Medium_Threat	Stored_Relative_Path_Traversal	23	**Source** has changed
Java	Java_APISecurity	Java_WebApi_GetApiList	0	**Source** has changed
Java	Java_Exploitable_Path	Java_Find_UnresolvedMethods	0	**Source** has changed
Java	Java_High_Risk	Reflected_XSS_All_Clients	79	**Source** has changed
Java	Java_High_Risk	Stored_XSS	79	**Source** has changed
Java	Java_Low_Visibility	Log_Forging	117	**Source** has changed
Java	Java_Low_Visibility	Trust_Boundary_Violation_in_Session_Variables	501	**Source** has changed
Java	Java_Low_Visibility	Use_Of_Hardcoded_Password	259	**Source** has changed
Java	Java_Medium_Threat	Cleartext_Submission_of_Sensitive_Information	319	**Source** has changed
Java	Java_Medium_Threat	Excessive_Data_Exposure	201	**Source** has changed
Java	Java_Medium_Threat	Improper_Restriction_of_Stored_XXE_Ref	611	**Source** has changed
Java	Java_Medium_Threat	Improper_Restriction_of_XXE_Ref	611	**Source** has changed
Java	Java_Medium_Threat	JWT_Lack_Of_Expiration_Time	613	**Source** has changed
Java	Java_Medium_Threat	JWT_No_Signature_Verification	287	**Source** has changed
Java	Java_Medium_Threat	JWT_Sensitive_Information_Exposure	201	**Source** has changed
Java	Java_Medium_Threat	JWT_Use_Of_Hardcoded_Secret	798	**Source** has changed
Java	Java_Medium_Threat	Privacy_Violation	359	**Source** has changed
Java	Java_Spring	Spring_Use_Of_Hardcoded_Password	259	**Source** has changed
JavaScript	JavaScript_Exploitable_Path	JavaScript_Find_UnresolvedMethods	0	**Source** has changed
JavaScript	JavaScript_High_Risk	Client_DOM_XSS	79	**Source** has changed
JavaScript	JavaScript_Medium_Threat	Unchecked_Input_For_Loop_Condition	606	**Source** has changed
JavaScript	JavaScript_Server_Side_Vulnerabilities	Comparing_instead_of_Assigning	482	**Source** has changed
JavaScript	JavaScript_Server_Side_Vulnerabilities	Missing_Encryption_of_Sensitive_Data	311	**Source** has changed

Language	Group	Name	CWE	Changed Fields
JavaScript	JavaScript_Server_Side_Vulnerabilities	MongoDB_NoSQL_Injection	89	**Source** has changed
JavaScript	JavaScript_Server_Side_Vulnerabilities	Reflected_XSS	79	**Source** has changed
JavaScript	JavaScript_Server_Side_Vulnerabilities	Use_Of_Hardcoded_Password	259	**Source** has changed
JavaScript	JavaScript_Vue	Use_of_Single_Word_Named_Vue_Components	710	**Source** has changed
Kotlin	Kotlin_Android	Use_of_WebView_AddJavascriptInterface	749	**Source** has changed
Lua	Lua_Low_Visibility	Password_In_Comment	615	**Source** has changed
PHP	PHP_High_Risk	Code_Injection	94	**Source** has changed
PHP	PHP_High_Risk	Deserialization_of_Untrusted_Data	502	**Source** has changed
PHP	PHP_High_Risk	Reflected_XSS	79	**Source** has changed
PHP	PHP_Low_Visibility	Cookie_Overly_Broad_Path_In_Config	539	**Source** has changed
PHP	PHP_Low_Visibility	Trust_Boundary_Violation_in_Session_Variables	501	**Source** has changed
PHP	PHP_Medium_Threat	HttpOnly_Cookie_Flag_Not_Set_In_Config	1004	**Source** has changed
PHP	PHP_Medium_Threat	Insecure_Value_of_the_SameSite_Cookie_Attribute_In_Config	1275	**Source** has changed
PHP	PHP_Medium_Threat	Stored_Code_Injection	94	**Source** has changed
Python	Python_Exploitable_Path	Python_Find_UnresolvedMethods	0	**Source** has changed
Rust	Rust_High_Risk	Connection_String_Injection	99	**Source** has changed
Rust	Rust_Medium_Threat	Privacy_Violation	359	**Source** has changed
Rust	Rust_High_Risk	Absolute_Path_Traversal	36	Name changed from Interactive_Absolute_Path_Traversal to **Absolute_Path_Traversal**
Rust	Rust_High_Risk	Relative_Path_Traversal	23	Name changed from Interactive_Relative_Path_Traversal to **Relative_Path_Traversal**
VbNet	VbNet_Low_Visibility	Heap_Inspection	244	**Source** has changed

Changed Source:

CPP / CPP_Best_Coding_Practice / Buffer_Size_Literal_Condition

Code changes

```
---
+++
@@ -3,76 +3,112 @@
 //
 // This query find arrays that their size is a literal,
 // there is a use with that literal in a condition
-// and inside the condition there is a use with that array.
+// and inside the condition there is a use with that array.


 CxList integerLiteral = Find_Integer_Literals();
 CxList integersAsParameters = integerLiteral.GetByAncs(Find_Methods());
 CxList ArrayDefinition = Find_ArrayCreateExpr();
 CxList unkRefs = Find_Unknown_References();
-CxList foundSizeInt = All.NewCxList();
 CxList customAttribute = Find_CustomAttribute().FindByShortName("CxNotLiteralBufferSize").GetFathers();
 customAttribute.Add(customAttribute.FindByType<CustomAttributeCollection>().GetFathers());
+ArrayDefinition -= ArrayDefinition.GetByAncs(customAttribute);
+CxList conditions = Find_Conditions();
+CxList relevantDeclarators = ArrayDefinition.GetAncOfType<Declarator>();
+CxList refsOfDefs = All.FindAllReferences(relevantDeclarators);
+CxList allReferencesOfDeclUnderCond = refsOfDefs.GetByAncs(conditions);
+allReferencesOfDeclUnderCond.Add(refsOfDefs.GetByAncs(conditions.GetFathers()));
```

```
-ArrayDefinition -= ArrayDefinition.GetByAncs(customAttribute);
+Func<CxList, string, CxList, CxList, CxList> matchingConditions = (arrayDef, sizeString, sizeIntegerLiteral, foundSizeInt) =>
+   {
+   CxList matches = allReferencesOfDeclUnderCond.NewCxList();
+   CxList minimizingIntegerGroup = All.NewCxList();
+
+   if (sizeString != null)
+       minimizingIntegerGroup = integerLiteral.FindByShortName(sizeString).GetByAncs(conditions);
+   else
+       minimizingIntegerGroup = integerLiteral.FindByShortName(foundSizeInt).GetByAncs(conditions);
+
+   minimizingIntegerGroup -= integersAsParameters;
+   CxList scope = arrayDef.GetAncOfType<MethodDecl>();
+
+   if (scope.Count == 0) {
+       scope = arrayDef.GetAncOfType<ClassDecl>();
+   }
+
+   CxList allUnderScope = All.GetByAncs(scope);
+   CxList sameNumberFound = All.NewCxList();
+
+   if (sizeString != null)
+       sameNumberFound = minimizingIntegerGroup.FindByShortName(sizeString) * allUnderScope;
+   else
+       sameNumberFound = minimizingIntegerGroup.FindByShortName(sizeIntegerLiteral) * allUnderScope;
+
+
+   CxList refsOfArrayDef = allReferencesOfDeclUnderCond.FindAllReferences(arrayDef.GetAncOfType<Declarator>());
+
+   //look inside conditions, that we find both the array and the number under the same condition
+   CxList relevantConditions = conditions * allUnderScope;
+
+   foreach (CxList condition in relevantConditions)
+   {
+       CxList allUnderCondition = All.GetByAncs(condition);
+       CxList number = sameNumberFound * allUnderCondition;
+       CxList arrayReference = refsOfArrayDef * allUnderCondition;
+       arrayReference.Add(refsOfArrayDef.GetByAncs(condition.GetFathers()));
+       arrayReference -= unkRefs;
+
+       if (number.Count > 0 && arrayReference.Count > 0) {
+           matches.Add((sizeString != null ? arrayDef : sizeIntegerLiteral).ConcatenateAllTargets(arrayReference));
+       }
+   }
+
+   return matches;
+   };
```

```
 //find arrays that are initialized with an int as a size
-foreach(CxList arrayDef in ArrayDefinition)
+foreach (CxList arrayDef in ArrayDefinition)
 {
    try
    {
        ArrayCreateExpr g = arrayDef.TryGetCSharpGraph<ArrayCreateExpr>();
-       if(g == null || g.Sizes == null)
-       {
-           continue;
-       }
-       ExpressionCollection arraySizes = g.Sizes;
-       foreach(Expression size in arraySizes)
-       {
-           if (size != null)
-           {
-               foundSizeInt.Add(integerLiteral.GetByAncs(All.FindById(size.NodeId)));
+       string arraySize = null;
+
+       if (g != null) {
+           IAbstractValue absValue = g.AbsValue;
+
+           if (absValue is ObjectAbstractValue objectAbsValue){
+               arraySize = (objectAbsValue.AllocatedSize)?.UpperIntervalBound?.ToString();
+           }
+           else if (absValue is IntegerIntervalAbstractValue intAbsValue){
+               arraySize = intAbsValue.UpperIntervalBound.ToString();
+           }
+
+           if (arraySize != null){
+               result.Add(matchingConditions(arrayDef, arraySize, null, null));
+           }
+           else {
+               CxList foundSizeInt = All.NewCxList();
+
+               ExpressionCollection arraySizes = g.Sizes;
+               if (arraySizes != null)
+               {
+                   foreach (Expression size in arraySizes)
+                   {
+                       if (size != null) {
+                           foundSizeInt.Add(integerLiteral.GetByAncs(All.FindById(size.NodeId)));
+                       }
+                   }
+               }
+               if (foundSizeInt.Count > 0) {
+                   //find scope of the declaration of the array
```

```
+                    foreach (CxList size in foundSizeInt) {
+                        result.Add(matchingConditions(arrayDef, null, size, foundSizeInt));
+                    }
+                }
            }
        }
    }
-   catch(Exception e)
-   {
+   catch (Exception e) {
        cxLog.WriteDebugMessage(e);
    }
}
-
-if(foundSizeInt.Count > 0)
-{
-   CxList conditions = Find_Conditions();
-   CxList minimizingIntegerGroup = integerLiteral.FindByShortName(foundSizeInt).GetByAncs(conditions);
-   minimizingIntegerGroup -= integersAsParameters;
-   CxList relevantDeclarators = ArrayDefinition.GetAncOfType(typeof(Declarator));
-   CxList refsOfDefs = All.FindAllReferences(relevantDeclarators);
-   CxList allReferencesOfDeclUnderCond = refsOfDefs.GetByAncs(conditions);
-   allReferencesOfDeclUnderCond.Add(refsOfDefs.GetByAncs(conditions.GetFathers()));
-   //find scope of the declaration of the array
-   foreach(CxList size in foundSizeInt)
-   {
-       CxList scope = size.GetAncOfType(typeof(MethodDecl));
-       if(scope.Count == 0)
-       {
-           scope = size.GetAncOfType(typeof(ClassDecl));
-       }
-       CxList allUnderScope = All.GetByAncs(scope);
-       CxList sameNumberFound = minimizingIntegerGroup.FindByShortName(size) * allUnderScope;
-       CxList refsOfArrayDef = allReferencesOfDeclUnderCond.FindAllReferences(size.GetAncOfType(typeof(Declarator)));
-       //look inside conditions, that we find both the array and the number under the same condition
-       CxList relevantConditions = conditions * allUnderScope;
-       foreach(CxList condition in relevantConditions)
-       {
-           CxList allUnderCondition = All.GetByAncs(condition);
-           CxList number = sameNumberFound * allUnderCondition;
-           CxList arrayReference = refsOfArrayDef * allUnderCondition;
-           arrayReference.Add(refsOfArrayDef.GetByAncs(condition.GetFathers()));
-           arrayReference -= unkRefs;
-           if(number.Count > 0 && arrayReference.Count > 0)
-           {
-               result.Add(size.ConcatenateAllTargets(arrayReference));
-           }
-       }
```

```
-   }
-}
```

CPP / CPP_Best_Coding_Practice / Empty_Methods

Code changes

```
---
+++
@@ -1,2 +1 @@
 result = Common_General.Find_Empty_Methods();
-result -= Find_Cx_Method_Declarations();
```

CPP / CPP_Buffer_Overflow / Buffer_Overflow_AddressOfLocalVarReturned

Code changes

```
---
+++
@@ -28,40 +28,48 @@
 CxList objectCreateExpr = Find_ObjectCreations();
 CxList returnStmt = Find_ReturnStmt();
 CxList arrays = Find_Array_Declaration();
-CxList arrayCreate = arrays.FindByType(typeof(ArrayCreateExpr));
-CxList arrayDeclare =  arrays.FindByType(typeof(VariableDeclStmt));
+CxList arrayCreate = arrays.FindByType<ArrayCreateExpr>();
+CxList arrayDeclare =  arrays.FindByType<VariableDeclStmt>();
 CxList declarator = Find_Declarators();
 CxList unary = Find_Unarys();
-CxList pointers = Find_Pointer_Decl();
 CxList alloc = Find_Memory_Allocation();
+CxList methods = Find_Methods();
 CxList address = unary.FindByShortName("Address");
+
+CxList pointerUnary = unary.FindByShortName("Pointer");
+address -= address.GetByAncs(pointerUnary); // remove things where we do *(... &x)
+
 CxList addressVars = unknownRef.GetByAncs(address);

 //remove all non local variables
-CxList unkrUnderMethods = unknownRef.GetAncOfType(typeof(MethodDecl));
+CxList unkrUnderMethods = unknownRef.GetAncOfType<MethodDecl>();
 CxList localVariables = declarator.GetByAncs(unkrUnderMethods);
 CxList notLocalDeclarators = declarator - localVariables;

-unknownRef -= unknownRef.FindAllReferences((paramDecl + fieldDecl + notLocalDeclarators).FindDefinition(unknownRef));
+CxList refList = All.NewCxList(paramDecl, fieldDecl, notLocalDeclarators);
+unknownRef -= unknownRef.FindAllReferences(refList.FindDefinition(unknownRef));

 //remove all parameters
-unknownRef -= unknownRef.GetByAncs(unknownRef.GetAncOfType(typeof(MethodInvokeExpr)));
+unknownRef -= unknownRef.GetByAncs(unknownRef.GetAncOfType<MethodInvokeExpr>());
```

```
 CxList unknownInReturn = unknownRef.GetByAncs(returnStmt);

 //remove all heap variables
-unknownInReturn -= unknownInReturn.DataInfluencedBy(alloc + objectCreateExpr);
+CxList inputLst = All.NewCxList(alloc, objectCreateExpr);
+unknownInReturn -= unknownInReturn.DataInfluencedBy(inputLst );



 //only gets address vars that are in the same scope as declarator
 CxList addressVarsReturned = All.NewCxList();

+CxList allDefs = All.FindDefinition(addressVars);
+allDefs -= allDefs.FindByType<ParamDecl>();
 foreach(CxList addressVar in addressVars)
 {
-   CxList definition = All.FindDefinition(addressVar);
-   CxList definitionMethods = All.GetMethod(definition - definition.FindByType(typeof(ParamDecl)));
+   CxList definition = allDefs.FindDefinition(addressVar);
+   CxList definitionMethods = methods.GetMethod(definition);

-   if( ( All.GetMethod(addressVar) * definitionMethods ).Count > 0)
+   if( ( methods.GetMethod(addressVar) * definitionMethods ).Count > 0)
    {
        addressVarsReturned.Add(addressVar);
    }
@@ -73,8 +81,6 @@

 result.Add(addressesReturned);

-
-
 //handle the  : return &x statement
 result.Add(unknownInReturn.FindByFathers(unknownInReturn.GetFathers() * address));

@@ -82,22 +88,20 @@
 result -= result.GetMembersOfTarget().GetTargetOfMembers();

 CxList definitionOfVariable = declarator.FindDefinition(unknownInReturn);
-//handle local variables that are declared as pointers
-//pointers are not local variables
-//result.Add(unknownInReturn.FindAllReferences(definitionOfVariable * pointers));

 //handle local arrays definitions that are not defined on the heap
 CxList arrayDef = arrayCreate.FindByFathers(definitionOfVariable);
 CxList arrayFathers = arrayDef.GetFathers();
 arrayFathers.Add(declarator.GetByAncs(arrayDeclare));
 result.Add(unknownInReturn.FindAllReferences(arrayFathers));
```

```
+result -= result.FindByFathers(Find_IndexerRefs());


 //remove all results that are influenced by parameter declarations
-CxList parameters = paramDecl.GetByAncs(result.GetAncOfType(typeof(MethodDecl)));
-CxList resultInsideMethodWithParam = result.GetByAncs(parameters.GetAncOfType(typeof(MethodDecl)));
+CxList parameters = paramDecl.GetByAncs(result.GetAncOfType<MethodDecl>());
+CxList resultInsideMethodWithParam = result.GetByAncs(parameters.GetAncOfType<MethodDecl>());
 result -= resultInsideMethodWithParam.DataInfluencedBy(parameters);


 //remove all the static returned variables
 CxList allStatics = Find_VariableDeclStmt().FindByFieldAttributes(Modifiers.Static);
-allStatics = All.FindAllReferences(All.GetByAncs(allStatics).FindByType(typeof (Declarator)));
+allStatics = result.FindAllReferences(declarator.FindDescendantsOfType<Declarator>(allStatics));
 result -= allStatics;
```

CPP / CPP_Buffer_Overflow / Buffer_Overflow_LongString

Code changes

```
---
+++
@@ -35,6 +35,7 @@
 CxList flow = relevant.InfluencedByAndNotSanitized(stringLiteral, sanitizers);


 CxList arrayCreateExpr = Find_ArrayCreateExpr();
+CxList pointerArrayCreateExpr = Find_Pointers().GetFathers() * arrayCreateExpr;
 CxList customAttribute = Find_CustomAttribute().FindByShortName("CxNotLiteralBufferSize").GetAncOfType<VariableDeclStmt>();


 CxList valueAccess = varsOfTypeCharPointer.FindByFathers(varsOfTypeCharPointer.GetFathers().FindByShortName("Pointer"));
@@ -51,8 +52,9 @@
     {
         try{
             ArrayCreateExpr r = arr.TryGetCSharpGraph<ArrayCreateExpr>();
-            //scenario #1 declarator of type char x[size]
-            if (r.Sizes == null)
+            //scenario #1 declarator of type char x[size]
+            CxList pointerArr = pointerArrayCreateExpr * arr;
+            if (r.Sizes == null || pointerArr.Count == 1)
             {
                 continue;
             }
```

CPP / CPP_Buffer_Overflow / Buffer_Overflow_Unbounded_Buffer

Code changes

```
---
+++
@@ -43,9 +43,7 @@
     return false;
     };
```

```
-CxList inputs = Find_Unbounded_Inputs();
-inputs.Add(Find_Read());
-inputs.Add(Find_DB());
+CxList inputs = All.NewCxList(Find_Unbounded_Inputs(), Find_Read(), Find_DB());


 CxList strlenParams = All.GetParameters(Find_All_Strlen());
 strlenParams.Add(unknownRefs.GetByAncs(strlenParams));
@@ -67,7 +65,6 @@
    CxList fmtParam = fmtMethodParams.FindByAbstractValue(val => {
        if (val is StringAbstractValue) {
            Match match = Regex.Match((val as StringAbstractValue).Content, @"%(\d*)(\'|\*|\.)*(\d*)(\w)");
-           cxLog.WriteDebugMessage(match.Success);
            if (!match.Success) {
                return true;
            }
```

CPP / CPP_Buffer_Overflow / Buffer_Overflow_Unbounded_Format

Code changes

```
---
+++
@@ -3,49 +3,8 @@
 CxList stringLiterals = Find_Strings();
 CxList memAllocs = Find_Memory_Allocation();

-// Helper delegate to compare two parameters as for their AbsValue (param1 < param2)
-Func <CxList, CxList, bool> CompareExprsByAbsValLT = delegate(CxList fstParam, CxList sndParam){
-
-   Func<Expression, IntegerIntervalAbstractValue> GetExprAbsValue = delegate(Expression paramExpr) {
-       IntegerIntervalAbstractValue paramAbsValue = null;
-       if(paramExpr != null)
-       {
-           IAbstractValue absValue = paramExpr.AbsValue;
-           if (absValue is ObjectAbstractValue)
-           {
-               paramAbsValue = (absValue as ObjectAbstractValue).AllocatedSize;
-           }
-           else if (absValue is IntegerIntervalAbstractValue)
-           {
-               paramAbsValue = absValue as IntegerIntervalAbstractValue;
-           }
-           else if (absValue is StringAbstractValue)
-           {
-               StringAbstractValue stringAbsValue = absValue as StringAbstractValue;
-               paramAbsValue = new IntegerIntervalAbstractValue(stringAbsValue.Content.Length);
-           }
-       }
-       return paramAbsValue;
-       };
```

```
-
-   Expression firstParamExpr = fstParam.TryGetCSharpGraph<Expression>();
-   Expression secondParamExpr = sndParam.TryGetCSharpGraph<Expression>();
-
-   IntegerIntervalAbstractValue firstParamAbsValue = GetExprAbsValue(firstParamExpr);
-   IntegerIntervalAbstractValue secondParamAbsValue = GetExprAbsValue(secondParamExpr);
-
-   if (firstParamAbsValue != null && secondParamAbsValue != null)
-   {
-       IAbstractValue absValueResult = firstParamAbsValue.LessThan(secondParamAbsValue);
-       return (absValueResult is TrueAbstractValue);
-   }
-
-   return false;
-};
-
 CxList inputs = Find_Unbounded_Inputs();
-inputs.Add(Find_Read());
-inputs.Add(Find_DB());
+inputs.Add(Find_Read(), Find_DB());

 // Sanitizer 1
 CxList strlenParams = All.GetParameters(Find_All_Strlen());
@@ -58,6 +17,10 @@
 // Methods that use format specifiers and could lead to Buffer Overflow
 CxList methodsWithFormat = Find_BufferOverflow_ScanPrint_Funcs();

+CxList fopens = methodInvokes.FindByShortName("fopen", false).FindByParameters(stringLiterals
+   .FindByShortNames(new string []{"r", "r+", "w+", "a+"}));
+CxList relevInfluence = All.NewCxList(inputs, fopens);
+
 foreach(CxList fmtMethod in methodsWithFormat)
 {
    CxList fmtMethodParams = All.GetParameters(fmtMethod);
@@ -69,16 +32,16 @@
        return false;
        }); // StringLiterals or UnknownReferences

-   CxList fmtStringLiteralParam = fmtParam.FindByType(typeof(StringLiteral));
+   CxList fmtStringLiteralParam = fmtParam.FindByType<StringLiteral>();

    //If the parameter is not a StringLiteral, check if it's a reference of one (and get it)
    if (fmtStringLiteralParam.Count == 0)
    {
-       CxList fmtUnknownRefParam = fmtParam.FindByType(typeof(UnknownReference));
+       CxList fmtUnknownRefParam = fmtParam.FindByType<UnknownReference>();
        CxList stringRefs = stringLiterals.FindByAbstractValues(fmtUnknownRefParam);
        CxList correctStringRef = stringRefs.InfluencingOn(fmtUnknownRefParam)
```

```
-           .GetStartAndEndNodes(CxList.GetStartEndNodesType.StartNodesOnly);
-       fmtStringLiteralParam = correctStringRef.FindByType(typeof(StringLiteral));
+           .GetFirstNodesInPath();
+       fmtStringLiteralParam = correctStringRef.FindByType<StringLiteral>();
    }

    MethodInvokeExpr methodGraph = fmtMethod.TryGetCSharpGraph<MethodInvokeExpr>();
@@ -114,19 +77,23 @@
            switch(methodName)
            {
                case "scanf":
-                   sourceParam = fmtMethod;
+                   sourceParam = All.NewCxList(fmtMethod);
+                   inputToSourceParam = sourceParam.InfluencedBy(inputs);
                    destinationParam = All.GetByAncs(All.GetParameters(fmtMethod, matchNo + 1));
                    break;
                case "vscanf":
-                   sourceParam = fmtMethod;
+                   sourceParam = All.NewCxList(fmtMethod);
+                   inputToSourceParam = sourceParam.InfluencedBy(inputs);
                    destinationParam = All.GetByAncs(All.GetParameters(fmtMethod, 1));
                    break;
                case "fscanf":
                    sourceParam = All.GetByAncs(All.GetParameters(fmtMethod, 0));
+                   inputToSourceParam = sourceParam.InfluencedBy(relevInfluence);
                    destinationParam = All.GetByAncs(All.GetParameters(fmtMethod, matchNo + 2));
                    break;
                case "vfscanf":
                    sourceParam = All.GetByAncs(All.GetParameters(fmtMethod, 0));
+                   inputToSourceParam = sourceParam.InfluencedBy(relevInfluence);
                    destinationParam = All.GetByAncs(All.GetParameters(fmtMethod, 2));
                    break;
                case "sscanf":
@@ -147,12 +114,12 @@
            }
            matchNo++;

-           CxList sourceActualParams = sourceParam.FindByType(typeof(Param));
+           CxList sourceActualParams = sourceParam.FindByType<Param>();
            if (sourceActualParams.Count > 0)
            {
                sourceParam = All.FindByFathers(sourceActualParams);
            }
-           destinationParam = destinationParam.FindByType(typeof(UnknownReference));
+           destinationParam = destinationParam.FindByType<UnknownReference>();

            CxList sanitized = All.NewCxList();
```

```
@@ -202,33 +169,7 @@
                            CxList fromInputBufferOverflow = inputToSourceParam.ConcatenatePath(destinationParam, false);
                            result.Add(fromInputBufferOverflow);
                        }
-                       else
-                       {
-                           // Keep only nodes that can have size
-                           CxList destWithSize = destinationParam * unknownRefs;
-                           destWithSize.Add(destinationParam * stringLiterals);
-                           CxList srcWithSize = sourceParam * unknownRefs;
-                           srcWithSize.Add(sourceParam * stringLiterals);

-                           bool srcSmallerThanDest = CompareExprsByAbsValLT(srcWithSize, destWithSize);
-                           // Source parameter is not guaranteed to be smaller than destination parameter
-                           if (!srcSmallerThanDest)
-                           {
-                               srcWithSize.Add(sourceParam * methodInvokes); //To show better flow in scanf
-                               CxList boWithDestination = srcWithSize.ConcatenatePath(destWithSize, false);
-                               result.Add(boWithDestination);
-                           }
-                           else
-                           {
-                               bool destSmallerThanSrc = CompareExprsByAbsValLT(destWithSize, srcWithSize);
-                               // Destination parameter is smaller than source parameter
-                               if (destSmallerThanSrc)
-                               {
-                                   CxList sourceToDestination = srcWithSize.ConcatenatePath(destWithSize, false);
-                                   result.Add(sourceToDestination);
-                               }
-                           }
-                       }
                    }
                }
            }
```

CPP / CPP_Buffer_Overflow / Off_by_One_Error

Code changes

```
---
+++
@@ -1,25 +1,24 @@
 // This query complements the results of the Improper_Buffer_Access query.
 CxList methods = Find_Methods();
-CxList binaryExprs = Find_BinaryExpressions();
 CxList unknRefs = Find_Unknown_References();

 //Arrays
 result.Add(Find_Improper_Index_Access(true));
```

```
 //Methods that do not check boundaries
-CxList copyMethodsWithSize = methods.FindByShortName("readlink");
-copyMethodsWithSize.Add(Find_All_strncpy());
-copyMethodsWithSize.Add(Find_All_strncat());
+CxList copyMethodsWithSize = All.NewCxList(methods.FindByShortName("readlink"), Find_All_strncpy(), Find_All_strncat());
 CxList sizeOf = methods.FindByShortName("sizeof");
 result.Add(sizeOf.GetParameters(copyMethodsWithSize));

+
 //Methods that copy without checking \0
-CxList copyMethods = methods.FindByShortNames(new List<string>{"strcpy_s", "strcat", "strcpy"});
+CxList copyMethods = methods.FindByShortNames("strcpy_s", "strcat", "strcpy");

 //Helper delegate to obtain expression's abstract value
-Func<Expression, IntegerIntervalAbstractValue> GetExprAbsValue = delegate(Expression paramExpr) {
+Func<Expression, IntegerIntervalAbstractValue> GetExprAbsValue = delegate (Expression paramExpr)
+   {
    IntegerIntervalAbstractValue paramAbsValue = null;
-   if(paramExpr != null)
+   if (paramExpr != null)
    {
        IAbstractValue absValue = paramExpr.AbsValue;
        if (absValue is ObjectAbstractValue)
@@ -37,23 +36,23 @@
    }
    return paramAbsValue;
-};
+   };

-foreach(CxList method in copyMethods)
+foreach (CxList method in copyMethods)
 {
    CxList firstParam = unknRefs.GetParameters(method, 0);
    CxList secondParam = unknRefs.GetParameters(method, 1);
-
+
    Expression firstParamExpr = firstParam.TryGetCSharpGraph<Expression>();
    Expression secondParamExpr = secondParam.TryGetCSharpGraph<Expression>();
-
-   IntegerIntervalAbstractValue firstParamAbsValue = GetExprAbsValue(firstParamExpr);
+
+   IntegerIntervalAbstractValue firstParamAbsValue = GetExprAbsValue(firstParamExpr);
    IntegerIntervalAbstractValue secondParamAbsValue = GetExprAbsValue(secondParamExpr);
-
+
    if (firstParamAbsValue != null && secondParamAbsValue != null)
    {
```

```
        IAbstractValue absValueResult = firstParamAbsValue.LessThanOrEqual(secondParamAbsValue);
-       if(absValueResult is TrueAbstractValue)
+       if (absValueResult is TrueAbstractValue)
        {
            result.Add(method);
        }
```

CPP / CPP_Low_Visibility / NULL_Pointer_Dereference

Code changes

```
---
+++
@@ -1,6 +1,7 @@
 CxList pointers = Find_Pointers();
 CxList unary = Find_Unarys();
-CxList pointerTotarget = pointers.GetByAncs(unary.FindByShortName("Pointer"));
+CxList pointerTotarget = pointers.GetByAncs(All.NewCxList(unary.FindByShortName("Pointer"),
+                                                          pointers.FindByType<IndexerRef>()));
 CxList unknownReferences = Find_Unknown_References();

 // Remove pointers that check pointer condition
@@ -9,27 +10,25 @@
 CxList ifs = pointerCondition.GetFathers().FindByType<IfStmt>();
 pointerTotarget -= pointerTotarget.GetByAncs(ifs);

-// Remove pointers used as indexerRefs
-pointerTotarget -= pointerTotarget.GetByAncs(Find_IndexerRefs());

 ////////////////////// Null Values ////////////////////////
-CxList nullValues = All.NewCxList(Find_NullLiteral(), Find_CharLiteral().FindByName("\0"));
-CxList zero = Find_IntegerLiterals().FindByShortName("0");
-
-//Remove 0 that are in IterationStmt
-CxList zeroIter = zero.GetAncOfType<IterationStmt>();
-zero -= zero.GetByAncs(zeroIter);
-
-//Remove 0 assigned to declarators/IndexerRefs that are not pointers
-CxList zeroAssignee = zero.GetAssignee();
-CxList zeroSafeAssignees = zeroAssignee.FindByType<Declarator>();
-zeroSafeAssignees.Add(zeroAssignee.FindByType<IndexerRef>());
-CxList pointersAssignedToZero = zeroSafeAssignees  * pointers;
-CxList declaratorsAssignedToZeroNotPointer = zeroSafeAssignees  - pointersAssignedToZero;
-zero -= declaratorsAssignedToZeroNotPointer.GetAssigner();
-
-
-nullValues.Add(zero);
+CxList nullValues = All.NewCxList(Find_NullLiteral(),
+   Find_CharLiteral().FindByShortName("\0"),
+   Find_IntegerLiterals().FindByShortName("0"));
```

```
+
+//Remove Null Values Assigned to IndexerRefs that the base type is not a pointer.
+CxList nullAssignee = nullValues.GetAssignee();
+CxList nullSafeAssignees = nullAssignee.FindByType<IndexerRef>();
+nullValues -= nullSafeAssignees.GetAssigner();
+
+//Remove Null Values Assigned to IndexerRefs as an Initializer
+nullValues -= nullValues.GetByAncs(Find_ArrayInitializer());
+
+//Remove 0 assigned to Declarators/UnknownReference that are not pointers
+CxList nullAssigneesToTest = nullAssignee.FindByTypes(typeof(Declarator), typeof(UnknownReference));
+CxList pointersAssignedToZero = nullAssigneesToTest * pointers;
+CxList declaratorsAssignedToZeroNotPointer = nullAssigneesToTest - pointersAssignedToZero;
+nullValues -= declaratorsAssignedToZeroNotPointer.GetAssigner();

 //Remove 0 that are in MethodInvokeExpr
 CxList zeroMethodInvokeExpr = nullValues.GetAncOfType<Param>();
@@ -44,6 +43,7 @@

 // Remove all assignees of null values from pointers
 pointerTotarget -= pointerTotarget.GetByAncs(nullValuesAssigned);
+

 ////////////////////////// Influencing ////////////////////////
 // Remove the 0 or 1  ( return 0  or return 1 )
@@ -66,6 +66,7 @@
 // Remove the 0 that are in an IndexerRef
 CxList zeroIndexerRef = nullValues.GetAncOfType<IndexerRef>();
 influencing -= nullValues.GetByAncs(zeroIndexerRef);
+

 ////////////////////////// Sanitizers ////////////////////////
 CxList sanitizers = All.NewCxList();
@@ -113,9 +114,44 @@
     .DataInfluencedBy(nullValueAssigners);
 sanitizers.Add(nullValuesInfluencingAssignees.GetLastNodesInPath());

+
 ////////////////////////// Results ////////////////////////
 CxList pointerToInfluencing = pointerTotarget.InfluencedByAndNotSanitized(influencing, sanitizers);
 pointerToInfluencing = pointerToInfluencing.ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
+
+// For Flows inside IterationStmt, remove Flows that cross scopes.
+foreach (CxList pti in pointerToInfluencing.GetCxListByPath()){
+   if(pti.GetAncOfType<IterationStmt>().Count == 1){
+       CxList nullAssignement = pti.GetFirstNodesInPath();
+       CxList nullDereference = pti.GetLastNodesInPath();
+       CxList codeBlocksInBetween = All.FindInScope(nullAssignement, nullDereference)
```

```
+           .FindByType<StatementCollection>();
+       if (codeBlocksInBetween.Count > 0){
+           pointerToInfluencing -= pti;
+       }
+   }
+}
+

+//Sanitize dereferences that are previously initialized inside a (x==NULL) condition.
+CxList refsOfTarget = unknownReferences.FindAllReferences(pointerTotarget);
+CxList assignNotNull = refsOfTarget.GetAncOfType<AssignExpr>() - nullValues.GetAncOfType<AssignExpr>();
+CxList ifNullWithSafeAssign = equalsNull.GetAncOfType<IfStmt>() * assignNotNull.GetAncOfType<IfStmt>();
+CxList safeIfBinaryExpr = binaryExpr.GetByAncs(ifNullWithSafeAssign);
+CxList safeRefs = refsOfTarget.GetByAncs(safeIfBinaryExpr);
+foreach (CxList pti in pointerToInfluencing.GetCxListByPath()){
+   CSharpGraph nullAssignement = pti.GetFirstNodesInPath().GetFirstGraph();
+   CSharpGraph nullDereference = pti.GetLastNodesInPath().GetFirstGraph();
+   CxList ifInBetween = safeRefs.FilterByDomProperty<UnknownReference>(x =>
+       x.VariableName == nullDereference.ShortName &&
+       x.LinePragma.FileName == nullAssignement.LinePragma.FileName &&
+       x.LinePragma.FileName == nullDereference.LinePragma.FileName &&
+       x.LinePragma.Line > nullAssignement.LinePragma.Line &&
+       x.LinePragma.Line < nullDereference.LinePragma.Line);
+   if (ifInBetween.Count > 0)
+   {
+       pointerToInfluencing -= pti;
+   }
+}


 ////////////////////// Pointer member access without memory allocation ////////////////////////
 CxList pointerMemberAccess = pointers.GetTargetsWithMembers();
@@ -190,4 +226,4 @@
 }


 result.Add(Find_Null_Array_Access(), // Include access like buff[x]=x after buff has been set to null
-           pointerToInfluencing);
+   pointerToInfluencing);
```

CPP / CPP_Low_Visibility / Privacy_Violation

Code changes

```
---
+++
@@ -118,5 +118,4 @@
 result = variableRefPath;
 result.Add(ConstInfuelncedByInput);
 result = result.ReduceFlow(CxList.ReduceFlowType.ReduceSmallFlow);
-result.Add(Find_Personal_Details());
 result.Add(Find_Token_Strings());
```

CPP / CPP_Medium_Threat / Divide_By_Zero

Code changes

```
---
+++
@@ -6,6 +6,7 @@
 CxList methodsInvokes = Find_Methods();
 CxList methodDecls = Find_MethodDecls();
 CxList conditions = Get_Conditions();
+CxList unaries = Find_Unarys();

 CxList methods = All.NewCxList();
 methods.Add(methodsInvokes);
@@ -80,6 +81,52 @@
     }
 }

+
+// Remove divs preceded by an if with return statement and
+// that checks if the divisor is null
+
+CxList ifs = Find_Ifs();
+CxList returnStmts = Find_ReturnStmt();
+
+//Only ifs with return stmts considered as they end the execution
+CxList relevIfs = returnStmts.GetByAncs(ifs).GetAncOfType<IfStmt>();
+
+//Get conditons like if (!x)
+CxList relevNots = unaries.FindByShortName("Not").FindByFathers(relevIfs);
+
+//Get conditons like if (x == 0)
+CxList relevZeros = zero.GetByAncs(
+   binaryExprs.GetByBinaryOperator(BinaryOperator.IdentityEquality).FindByFathers(relevIfs).GetFathers());
+
+//Get the variables iniside those conditions
+CxList ifVars = All.NewCxList(
+   relevZeros.GetFathers().CxSelectDomProperty<BinaryExpr>(x => x.Left),
+   relevNots.CxSelectDomProperty<UnaryExpr>(x => x.Right));
+
+//Check if the items in rightToDiv are preceded by those ifs, if so they're safe
+foreach (CxList varRef in rightToDiv) {
+   CxList currIfVar = unknown.FindAllReferences(varRef) * ifVars;
+
+   if (currIfVar.Count > 0) {
+
+       CSharpGraph ifVarGraph = currIfVar.GetFirstGraph();
+       CSharpGraph divVarGraph = varRef.GetFirstGraph();
+
+       if (ifVarGraph.ShortName == divVarGraph.ShortName &&
```

```
+           ifVarGraph != divVarGraph &&
+           ifVarGraph.LinePragma.FileName == divVarGraph.LinePragma.FileName &&
+           ifVarGraph.LinePragma.Line < divVarGraph.LinePragma.Line)
+       {
+
+           CxList stmts = currIfVar.GetFathers().GetFathers().CxSelectDomProperty<IfStmt>(x => x.TrueStatements);
+
+           if (varRef.GetByAncs(stmts).Count == 0){
+               rightToDiv -= varRef;
+           }
+       }
+   }
+}
+
 // When abstract interpretation is able to calculate a value for the right side of assign expressions
 // such as a/=0 and a%=0 as well as right side of divide and modulus operations, add it to the results.
 CxList zeroAbsVal = rightToDiv.FindByAbstractValue(
@@ -100,7 +147,7 @@
 //Find possible zeros that are in the context of an IF/ELSE/Loop where it appears also in the condition
 CxList possibleZerosInConditions = conditions * possibleZeros;
 CxList conditionsIfsLoops = All.NewCxList();
-conditionsIfsLoops.Add(Find_Ifs(), Find_IterationStmt());
+conditionsIfsLoops.Add(ifs, Find_IterationStmt());
 CxList possibleZerosInConditionalStmts = possibleZeros.GetByAncs(conditionsIfsLoops);
 foreach(CxList possibleZero in possibleZerosInConditions){
     CxList condStmt = possibleZero.GetAncOfType<IfStmt>();
```

CPP / CPP_Medium_Threat / Double_Free

Code changes

```
---
+++
@@ -43,6 +43,7 @@
 allDeclarators.Add(Find_ConstantDecl(), Find_Declarators(), Find_Enum_Declarations(), Find_FieldDecls(),
     Find_ParamDecl(), Find_StructDecl(), Find_VariableDeclStmt());

+
 foreach(CxList flow in flows.GetCxListByPath())
 {
     CxList startNode = flow.GetFirstNodesInPath();
@@ -74,6 +75,23 @@
     else if(startNodeTypeRef == null || endNodeTypeRef == null)
     {
         toRemove.Add(flow);
+    }
+        // If Types are Pointers, check if their Types are different
+    else if (startNodeTypeRef.TypeName == "pointer")
+    {
+        try
```

```
+       {
+           PointerTypeRef startNodePointerTypeRef = startNodeType.TryGetCSharpGraph<PointerTypeRef>();
+           PointerTypeRef endNodePointerTypeRef = endNodeType.TryGetCSharpGraph<PointerTypeRef>();
+           if (startNodePointerTypeRef.PointerType.TypeName != endNodePointerTypeRef.PointerType.TypeName)
+           {
+               toRemove.Add(flow);
+           }
+       }
+       catch
+       {
+           cxLog.WriteDebugMessage("Error converting to TypeRef");
+       }
    }
        // Check for flow breakers
    else
```

CPP / CPP_Medium_Threat / MemoryFree_on_StackVariable

Code changes

```
---
+++
@@ -7,6 +7,9 @@
 CxList pointers = Find_Pointer_Decl();
 CxList builtInTypes = Find_Builtin_Types();
 CxList integers = Find_Integers();
+CxList unary = Find_Unarys();
+CxList pointerOps = unary.FindByShortName("Pointer");
+CxList addressOps = unary.FindByShortName("Address");

 // Declarators beeing allocated
 CxList declaratorsAllocated = allocs.GetAncOfType<Declarator>();
@@ -58,7 +61,7 @@
 freedVars -= freedVars.FindAllReferences(allocatedFreedVars);
 freedVars -= Find_MemberAccesses();
 freedVars -= Find_ThisRef();
-freedVars -= Find_Unarys().FindByShortName("Pointer");
+freedVars -= pointerOps;


 //Stack variables that are assigned to pointers but not sanitized
@@ -71,7 +74,13 @@
 freedVars.Add(pointers.FindAllReferences(reallocs.GetAssignee()));
 freedVars.Add(pointVars);

+// Flow
+CxList resultFlows = freedVars.InfluencedByAndNotSanitized(start, sanitizers)
+    .ReduceFlow(CxList.ReduceFlowType.ReduceSmallFlow);

-// Flow
```

```
-result = freedVars.InfluencedByAndNotSanitized(start, sanitizers)
-   .ReduceFlow(CxList.ReduceFlowType.ReduceSmallFlow);
+// Remove Flows Sanitized by Unary Pointers, where no Unary Address is used.
+CxList flowsWithPointers = resultFlows.IntersectWithNodes(pointerOps);
+flowsWithPointers -= flowsWithPointers.IntersectWithNodes(addressOps);
+resultFlows -= flowsWithPointers;
+
+result = resultFlows;
```

CPP / CPP_Medium_Threat / Memory_Leak

Code changes

```
---
+++
@@ -1,8 +1,9 @@
 CxList unknownRefs = Find_Unknown_References();
 CxList memoryAllocation = Find_Memory_Allocation();
 CxList memoryDeallocation = Find_Memory_Deallocation();
-CxList throwStmts = Find_ThrowStmt();
-throwStmts.Add(Find_ThrowExpr());
+CxList catchMemoryDeallocation = memoryDeallocation.GetByAncs(Find_Catch());
+CxList nonCatchMemoryDeallocation = memoryDeallocation - catchMemoryDeallocation;
+CxList throwStmts = All.NewCxList(Find_ThrowStmt(), Find_ThrowExpr());
 CxList destructorDecls = Find_DestructorDecl();
 CxList decls = Find_VariableDeclStmt();
 CxList fieldDecls = Find_FieldDecls();
@@ -11,8 +12,7 @@
 CxList methods = Find_Methods();
 CxList parms = Find_Parameters().CxSelectDomProperty<Param>(x => x.Value);
 CxList fieldDeclaration = declarators.GetByAncs(fieldDecls);
-CxList references = All.NewCxList();
-references.Add(unknownRefs, Find_IndexerRefs());
+CxList references = All.NewCxList(unknownRefs, Find_IndexerRefs());

 // 0. Arrays that use "delete" (deletes single object) and not "delete[]" (deletes group of objects)
 CxList arrayCreateExpr = memoryAllocation.FindByType<ArrayCreateExpr>();
@@ -20,9 +20,11 @@
 CxList singleDeletes = Find_String_Literal().GetParameters(methods).FindByName("singleObjDel").
    GetAncOfType<MethodInvokeExpr>();
 // These are array deletes
-CxList arrayDeletes = memoryDeallocation.FindByShortName("delete").FindByNumberOfParameters(1);
+CxList arrayDeletes = nonCatchMemoryDeallocation.FindByShortName("delete").FindByNumberOfParameters(1);
+
 // Remove proper deallocated arrays
 arrayCreateExpr -= arrayDeletes.DataInfluencedBy(arrayCreateExpr).GetFirstNodesInPath();
+

 // 1. Find deletes inside destructors for news where there is no flow
 // Find assignments involving array creation expressions
```

```
@@ -45,19 +47,24 @@
        delDeallocation.Add(arrayExpr.GetAssignee());
    }
 }
+
 // 1.1 Throw statement and expressions between memory allocations and memory deallocation
 CxList throwAfterAllocation = All.NewCxList();
 foreach(CxList allocation in memoryAllocation)
 {
-   CxList foundThrow = throwStmts.FindInScope(allocation, memoryDeallocation);
-   if(foundThrow.Count == 1)
-   {
-       throwAfterAllocation.Add(allocation.ConcatenatePath(foundThrow, false));
+   foreach(CxList deallocation in memoryDeallocation){
+       CxList deallocationInScope = memoryDeallocation.FindInScope(allocation, deallocation);
+       CxList foundThrow = throwStmts.FindInScope(allocation, deallocation);
+       if(foundThrow.Count == 1 && deallocationInScope.Count == 0)
+       {
+           throwAfterAllocation.Add(allocation.ConcatenatePath(foundThrow, false));
+       }
    }
 }

 // 2. Remove allocations that influence on a memory deallocation
-memoryAllocation -= memoryAllocation.DataInfluencingOn(memoryDeallocation).GetFirstNodesInPath();
+memoryAllocation -= memoryAllocation.DataInfluencingOn(nonCatchMemoryDeallocation).GetFirstNodesInPath();
+
 CxList memmoryAllocVar = memoryAllocation.GetAncOfType<AssignExpr>().CxSelectDomProperty<AssignExpr>(assign => assign.Left);
 // remove if they are used inside a function call
 memmoryAllocVar -= memmoryAllocVar.GetByAncs(methods);
@@ -65,13 +72,15 @@
 // 3. Find allocated declaration and its references
 CxList allocatedReferences = memoryAllocation.GetAssignee();
 CxList unaryReferences = allocatedReferences.FindByType<UnaryExpr>();
-allocatedReferences.Add(unknownRefs.GetByAncs(unaryReferences));
-allocatedReferences.Add(memoryAllocation.GetFathers().FindByType<CastExpr>().GetAssignee());
-allocatedReferences.Add(memoryAllocation * unknownRefs);
+allocatedReferences.Add(
+   unknownRefs.GetByAncs(unaryReferences),
+   memoryAllocation.GetFathers().FindByType<CastExpr>().GetAssignee(),
+   memoryAllocation * unknownRefs);

 CxList assignedAllocation = allocatedReferences.GetAssigner() * memoryAllocation;
 assignedAllocation.Add(memoryAllocation.FindByFathers(
    allocatedReferences.GetAssigner().FindByType<CastExpr>()));
+
 memoryAllocation -= assignedAllocation;
```

```
 // 4. Free inside class destructor
@@ -92,7 +101,7 @@

 // 5. Allocated references that influence on a memory deallocation (whose allocation don't produces flow...)
 allocatedReferences -= unaryReferences;
-allocatedReferences -= allocatedReferences.InfluencingOn(memoryDeallocation);
+allocatedReferences -= allocatedReferences.InfluencingOn(nonCatchMemoryDeallocation);

 // 6. Disregard allocation to static variables (not static pointers)
 declarators -= Find_Pointers();
@@ -105,13 +114,13 @@
 CxList allRefs = unknownRefs.FindAllReferences(memmoryAllocVar);
 allocatedReferences -= allRefs.DataInfluencingOn(methods.FindByShortName("free"));

-// 8. Disregard allocation of auto_ptr variables (they perform automatic cleanup
+// 8. Disregard allocation of auto_ptr and unique_ptr variables (they perform automatic cleanup
 // when the object is no longer needed)
-CxList autoPtrDecls = Find_Smart_Pointer_Declarators(new[]{"auto_ptr"});
-allocatedReferences -= allocatedReferences.FindAllReferences(autoPtrDecls);
+CxList smartPtrDecls = Find_Smart_Pointer_Declarators(new[]{"auto_ptr", "unique_ptr"});
+allocatedReferences -= allocatedReferences.FindAllReferences(smartPtrDecls);
+memoryAllocation -= memoryAllocation.FindByType<ObjectCreateExpr>().GetByAncs(smartPtrDecls);

 // Remove variables that have destructor assigned to array creation expressions to be deleted
 delDeallocation -= varToDelete;

-result = allocatedReferences;
-result.Add(delDeallocation, memoryAllocation, throwAfterAllocation);
+result.Add(allocatedReferences, delDeallocation, memoryAllocation, throwAfterAllocation);
```

CPP / CPP_Medium_Threat / Pointer_Subtraction_Determines_Size

Code changes

```
---
+++
@@ -130,6 +130,7 @@
 // foreach pointer in binary subtractions
 List<string> basicTypes = new List<string> { "int", "char", "float", "bool", "double" };
 HashSet<int> cacheBinaries = new HashSet<int>();
+
 foreach(CxList pointer in pointers)
 {
    // search the other side of the subtraction
@@ -163,5 +164,10 @@
        {
            result.Add(binaryExpr);
        }
+       // check if the pointer is one more level deep
+       else if(pointerType.Contains("**"))
```

```
+       {
+           result.Add(binaryExpr);
+       }
    }
 }
```

CPP / CPP_MISRA_C / R08_05_Object_Function_In_Header_File

Code changes

```
---
+++
@@ -12,34 +12,31 @@


 // header files are defined to be anything used in #include (not just h files)
-CxList headerFiles = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList headerFiles = Find_Includes();

 // add object definitions (assignments)
-CxList problemIfInHeader = Find_All_Declarators() -
-   All.FindByType(typeof(ObjectCreateExpr)).GetFathers() -
-   All.FindByType(typeof(ArrayCreateExpr)).GetFathers();
-problemIfInHeader.Add(All.FindByType(typeof(AssignExpr)));
+CxList toIgnore = All.NewCxList(Find_ObjectCreations(), Find_ArrayCreateExpr()).GetFathers();
+CxList problemIfInHeader = Find_All_Declarators() - toIgnore;
+problemIfInHeader.Add(Find_AssignExpr());

 // Add method definitions
-problemIfInHeader.Add(All.FindByType(typeof(StatementCollection)).GetAncOfType(typeof(MethodDecl)));
+problemIfInHeader.Add(Find_StatementCollection().GetAncOfType<MethodDecl>());

 // remove redundent results
 problemIfInHeader -= (problemIfInHeader.GetByAncs(problemIfInHeader) - problemIfInHeader);
-problemIfInHeader -= problemIfInHeader.FindByShortName("INCLUDEREPLACE*");
-problemIfInHeader -= All.FindByType(typeof(StringLiteral)).FindByName("CX_TYPEDEF").GetFathers();
+problemIfInHeader -= Find_String_Literal().FindByName("CX_TYPEDEF").GetFathers();


 // intersect what is not allowed with the contents of the header files
 foreach (CxList cur in headerFiles){
-   StringLiteral inclusion = cur.TryGetCSharpGraph<StringLiteral>();
-   if (inclusion.Value == null)
+   CSharpGraph inclusion = cur.TryGetCSharpGraph<CSharpGraph>();
+   if (inclusion.ShortName == null)
        continue;

-   string headerFileName = "*" + inclusion.Value;
+   string headerFileName = "*" + inclusion.ShortName;
```

```
    result.Add(problemIfInHeader.FindByFileName(headerFileName));
 }


 foreach (CxList cur in result){
-       result -= (result.GetByAncs(cur) - cur);
+   result -= (result.GetByAncs(cur) - cur);
 }
```

CPP / CPP_MISRA_C / R16_01_Function_With_Variable_Number_Of_Arguments

Code changes

```
---
+++
@@ -12,18 +12,16 @@
 // find instances of the "..." operation
 CxList varArgs = All.FindByRegex(@"\.\.\.", false, false, false, All.NewCxList());

-result.Add(varArgs.GetAncOfType(typeof(ParamDeclCollection)).GetAncOfType(typeof(MethodDecl)));
-result.Add(varArgs.FindByType(typeof(StatementCollection)).GetFathers().FindByType(typeof(MethodDecl)));
+result.Add(
+   varArgs.GetAncOfType<ParamDeclCollection>().GetAncOfType<MethodDecl>(),
+   varArgs.FindByType<StatementCollection>().GetFathers().FindByType<MethodDecl>());

 // Safety check for the violating h file
 // (it is also a violation in itself)
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();
 CxList hFile = includes.FindByShortName("stdarg.h");
 if (hFile.Count > 0){
+   CxList methods = Find_Methods();
    // Non compliant functions;
-   result.Add(hFile +
-       All.FindByType(typeof(MethodInvokeExpr)).FindByShortName("va_arg") +
-       All.FindByType(typeof(MethodInvokeExpr)).FindByShortName("va_start") +
-       All.FindByType(typeof(MethodInvokeExpr)).FindByShortName("va_end"));
+   result.Add(hFile, methods.FindByShortNames("va_arg", "va_start", "va_end"));
 }
```

CPP / CPP_MISRA_C / R19_01_Non_Prepocessor_Command_Before_Include_In_File

Code changes

```
---
+++
@@ -15,22 +15,22 @@

 */

-CxList includes = All.FindByType(typeof(MethodDecl)).FindByShortName("INCLUDEREPLACE");
+CxList includes = Find_Includes();
```

```
 if (includes.Count > 0){
    // Removes includes and all associated dom objects except the class decl
    // (we'll remove that later);
-   CxList nonPermitted = All - All.GetByAncs(includes);
+   CxList nonPermitted = All - includes;

    // initialize first file stats
-   MethodDecl firstInc = includes.TryGetCSharpGraph<MethodDecl>();
+   CSharpGraph firstInc = includes.TryGetCSharpGraph<CSharpGraph>();
    string lastFileName = firstInc.LinePragma.FileName;
    int furthestIncludeLineInFile = firstInc.LinePragma.Line;
    int furthestIncludeColumnInFile = firstInc.LinePragma.Column;

    // go over all includes, finding out the furthest includue in each file
    foreach (CxList inc in includes){
-       MethodDecl curInc = inc.TryGetCSharpGraph<MethodDecl>();
+       CSharpGraph curInc = inc.TryGetCSharpGraph<CSharpGraph>();
        string curFileName = curInc.LinePragma.FileName;
        int curLine = curInc.LinePragma.Line;
        int curColumn = curInc.LinePragma.Column;
@@ -42,7 +42,6 @@
            foreach (CxList curNonPermitted in curFileNonPermitted){
                CSharpGraph nonPermittedCommand = curNonPermitted.GetFirstGraph();
                int nonPermLine = nonPermittedCommand.LinePragma.Line;
-               int nonPermColumn = curInc.LinePragma.Column;
                if (nonPermLine < furthestIncludeLineInFile){
                    result.Add(curNonPermitted);
                }
@@ -67,7 +66,7 @@

    // remove all class decls resulting from includes
    foreach (CxList inc in includes){
-       MethodDecl curInc = inc.TryGetCSharpGraph<MethodDecl>();
+       CSharpGraph curInc = inc.TryGetCSharpGraph<CSharpGraph>();
        string curFileName = curInc.LinePragma.FileName;
        int curLine = curInc.LinePragma.Line;
        result -= result.FindByPosition(curFileName, curLine);
```

CPP / CPP_MISRA_C / R19_02_Non_Standard_Chars_In_Header_File_Name

Code changes

```
---
+++
@@ -13,8 +13,7 @@

 */

-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
```

```
-   .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();

 // check for the illegal name flag
 foreach (CxList cur in includes){
```

CPP / CPP_MISRA_C / R19_03_Include_Directive_In_Wrong_Format

Code changes

```
---
+++
@@ -9,16 +9,15 @@

 */

-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();

 // check for the wrong format flag
 foreach (CxList cur in includes){
-   StringLiteral g = cur.TryGetCSharpGraph<StringLiteral>();
-   if (g == null || g.Value == null) {
+   CSharpGraph g = cur.TryGetCSharpGraph<CSharpGraph>();
+   if (g == null || g.ShortName == null) {
        continue;
    }
-   string curFileName = g.Value;
+   string curFileName = g.ShortName;
    if (String.Compare(curFileName, "INVALID_INCLUDE_FILE_NAME_") == 0){
        result.Add(cur);
    }
```

CPP / CPP_MISRA_C / R20_05_Using_Errno_Indicator_From_Errno_H

Code changes

```
---
+++
@@ -12,8 +12,7 @@
 */

 // Safety check for the violating h file
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();
 CxList hFile = includes.FindByShortName("errno.h");
 if (hFile.Count > 0){
    // Start with all instances of errno
```

CPP / CPP_MISRA_C / R20_06_Using_Offsetof_Macro_From_Stddef_H

Code changes

```
---
+++
@@ -16,8 +16,7 @@
 */


 // Safety check for the violating h file
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-    .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();
 CxList hFile = includes.FindByShortName("stddef.h");
 if (hFile.Count > 0){
    // Start with all instances of offsetof
```

CPP / CPP_MISRA_C / R20_07_Using_Setjmp_Longjmp_Macros_From_Setjmp_H

Code changes

```
---
+++
@@ -24,13 +24,11 @@
 */


 // Safety check for the violating h file
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-    .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();
 CxList hFile = includes.FindByShortName("setjmp.h");
 if (hFile.Count == 0){
    // Start with all instances of jumps
-   CxList jmps = All.FindByType(typeof(MethodInvokeExpr)).FindByShortName("setjmp") +
-       All.FindByType(typeof(MethodInvokeExpr)).FindByShortName("longjmp");
+   CxList jmps = Find_Methods().FindByShortNames("setjmp", "longjmp");


    // Remove all locally defined instances
    CxList defs = All.FindDefinition(jmps);
```

CPP / CPP_MISRA_C / R20_08_Using_Signal_Handling_From_Signal_H

Code changes

```
---
+++
@@ -17,17 +17,15 @@


 // Safety check for the violating h file
 // (it is also a violation in itself)
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-    .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();
+CxList methods = Find_Methods();
```

```
 CxList hFile = includes.FindByShortName("signal.h");
 if (hFile.Count > 0){
    // The functions defined by signal.h
-   result.Add(All.FindByType(typeof(MethodInvokeExpr)).FindByShortName("signal"));
-   result.Add(All.FindByType(typeof(MethodInvokeExpr)).FindByShortName("raise"));
+   result.Add(methods.FindByShortNames("signal", "raise"));

    // The macros defined by signal.h
-   result.Add(All.FindByShortNames(new List<string>()
-       {"SIG_DFL","SIG_ERR","SIG_IGN","SIGABRT","SIGFPE","SIGILL","SIGINT","SIGSEGV","SIGTERM"}));
+   result.Add(All.FindByShortNames("SIG_DFL","SIG_ERR","SIG_IGN","SIGABRT","SIGFPE","SIGILL","SIGINT","SIGSEGV","SIGTERM"));

    // Remove all locally defined instances
    CxList defs = All.FindDefinition(result);
@@ -37,5 +35,5 @@
    result.Add(hFile);

    // The types
-   result.Add(All.FindByType(typeof(TypeRef)).FindByShortName("sig_atomic_t"));
+   result.Add(Find_TypeRef().FindByShortName("sig_atomic_t"));
 }
```

CPP / CPP_MISRA_C / R20_09_Using_Input_Output_From_Stdio_H

Code changes

```
---
+++
@@ -15,16 +15,15 @@
 */

 // Safety check for the violating h file
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();
 CxList hFile = includes.FindByShortName("stdio.h");
 if (hFile.Count > 0){

    // The functions defined by stdio.h
-   CxList methodInvokes = All.FindByType(typeof(MethodInvokeExpr));
+   CxList methodInvokes = Find_Methods();



-   result.Add(methodInvokes.FindByShortNames(new List<string>(){
+   result.Add(methodInvokes.FindByShortNames(
            //Operations on files:
            "signal","remove","rename","tmpfile","tmpnam",
            //File access:
@@ -39,12 +38,12 @@
            "fgetpos","fseek","fsetpos","ftell","rewind",
```

```
            //Error-handling:
            "clearerr","feof","ferror","perror"
-           }));
+           ));

    // The macros defined by stdio.h
-   result.Add(All.FindByShortNames(new List <string>(){
+   result.Add(All.FindByShortNames(
            "_IOFBF","_IOLBF","_IONBF","BUFSIZ","EOF","FOPEN_MAX","FILENAME_MAX",
-           "SEEK_CUR","SEEK_END","SEEK_SET","TMP_MAX","stderr","stdin","stdout"}));
+           "SEEK_CUR","SEEK_END","SEEK_SET","TMP_MAX","stderr","stdin","stdout"));

    // Remove all locally defined instances
    CxList defs = All.FindDefinition(result);
@@ -53,8 +52,6 @@
    // the include
    result.Add(hFile);

-   // The types
-   result.Add(All.FindByType(typeof(TypeRef)).FindByShortName("size_t") +
-       All.FindByType(typeof(TypeRef)).FindByShortName("FILE") +
-       All.FindByType(typeof(TypeRef)).FindByShortName("fpos_t"));
+   // The types
+   result.Add(Find_TypeRef().FindByShortNames("size_t", "FILE", "fpos_t"));
 }
```

CPP / CPP_MISRA_C / R20_10_Using_Atof_Atoi_Atol_Functions_From_Stdlib_H

Code changes

```
---
+++
@@ -20,15 +20,12 @@
 */

 // Safety check for the violating h file
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();
 CxList hFile = includes.FindByShortName("stdlib.h");
 if (hFile.Count > 0){
    // Start with all instances of the functions
-   CxList methodInvokes = All.FindByType(typeof(MethodInvokeExpr));
-   CxList funcs = methodInvokes.FindByShortName("atof") +
-       methodInvokes.FindByShortName("atoi") +
-       methodInvokes.FindByShortName("atol");
+   CxList methodInvokes = Find_Methods();
+   CxList funcs = methodInvokes.FindByShortNames("atof", "atoi", "atol");

    // Remove all locally defined instances
```

```
    CxList defs = All.FindDefinition(funcs);
```

CPP / CPP_MISRA_C / R20_11_Using_Abort_Exit_Getenv_System_Functions_From_Stdlib_H

Code changes

```
---
+++
@@ -18,13 +18,12 @@
 */


 // Safety check for the violating h file
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();
 CxList hFile = includes.FindByShortName("stdlib.h");
 if (hFile.Count > 0){
    // Start with all instances of the functions
-   CxList methodInvokes = All.FindByType(typeof(MethodInvokeExpr));
-   CxList funcs = methodInvokes.FindByShortNames(new List<string>(){"abort","exit","getenv","system"});
+   CxList methodInvokes = Find_Methods();
+   CxList funcs = methodInvokes.FindByShortNames("abort","exit","getenv","system");


    // Remove all locally defined instances
    CxList defs = All.FindDefinition(funcs);
```

CPP / CPP_MISRA_C / R20_12_Using_Time_Handling_From_Time_H

Code changes

```
---
+++
@@ -14,22 +14,20 @@


 // Safety check for the violating h file
 // (it is also a violation in itself)
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();
 CxList hFile = includes.FindByShortName("time.h");
 if (hFile.Count > 0){
    // The functions defined by time.h
-   CxList methodInvokes = All.FindByType(typeof(MethodInvokeExpr));
+   CxList methodInvokes = Find_Methods();


    //Time manipulation:
-   result.Add(methodInvokes.FindByShortNames(new List<string>(){
+   result.Add(methodInvokes.FindByShortNames(
            "clock","difftime","mktime","time",
            //Conversion:
-           "asctime","ctime","gmtime","localtime","strftime"}));
+           "asctime","ctime","gmtime","localtime","strftime"));
```

```
    // The macros defined by time.h
-   result.Add(All.FindByShortName("null") +
-       All.FindByShortName("CLOCKS_PER_SEC"));
+   result.Add(All.FindByShortNames("null", "CLOCKS_PER_SEC"));

    // Remove all locally defined instances
    CxList defs = All.FindDefinition(result);
@@ -39,20 +37,10 @@
    result.Add(hFile);

    // The types
-   result.Add(All.FindByType(typeof(TypeRef)).FindByShortName("clock_t") +
-       All.FindByType(typeof(TypeRef)).FindByShortName("size_t") +
-       All.FindByType(typeof(TypeRef)).FindByShortName("time_t") +
-       All.FindByType(typeof(TypeRef)).FindByRegex("struct tm"));
+   CxList types = Find_TypeRef();
+   result.Add(types.FindByShortNames("clock_t", "size_t", "time_t"),
+              types.FindByRegex("struct tm"));
    // struct tm members
-   CxList memberAccess = All.FindByType(typeof(MemberAccess));
-   result.Add(memberAccess.FindByRegex("tm_sec;") +
-       memberAccess.FindByRegex("tm_min") +
-       memberAccess.FindByRegex("tm_hour") +
-       memberAccess.FindByRegex("tm_mday") +
-       memberAccess.FindByRegex("tm_mon") +
-       memberAccess.FindByRegex("tm_year") +
-       memberAccess.FindByRegex("tm_wday") +
-       memberAccess.FindByRegex("tm_yday") +
-       memberAccess.FindByRegex("tm_yday") +
-       memberAccess.FindByRegex("tm"));
+   CxList memberAccess = Find_MemberAccesses();
+   result.Add(memberAccess.FindByRegex("tm(_(sec;|min|hour|mday|mon|year|wday|yday))?"));
 }
```

CPP / CPP_MISRA_CPP / R00_01_05_Find_Unused_Typedefs

Code changes

```
---
+++
@@ -14,19 +14,17 @@
 */

 //Find typedefs
-CxList typedefs = All.FindByType(typeof(StringLiteral)).FindByName("CX_TYPEDEF");
-typedefs = typedefs.GetAncOfType(typeof(VariableDeclStmt))
-   + typedefs.GetAncOfType(typeof(FieldDecl));
+CxList typedefs = Find_Strings().FindByName("CX_TYPEDEF");
+typedefs = All.NewCxList(typedefs.GetAncOfType(new[]{typeof(VariableDeclStmt), typeof(FieldDecl)}));
```

```
 typedefs = Find_All_Declarators().GetByAncs(typedefs);


 CxList unused = All.FindDefinition(All.FindAllReferences(typedefs) - typedefs);
 unused = typedefs - unused;


 //Start finding typedefs used with includes.
-CxList headerFiles = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList headerFiles = Find_Includes();


-CxList typeRefs = All.FindByType(typeof(TypeRef));
+CxList typeRefs = Find_TypeRef();
 ArrayList files = new ArrayList();
 foreach(CxList cur in unused){
    CSharpGraph curGraph = cur.GetFirstGraph();
@@ -39,7 +37,7 @@
    // build list of all files that include current file
    CxList curFileTypedefs = typedefs.FindByFileName(curFile);
    string[] splitFile = curFile.Split(cxEnv.Path.DirectorySeparatorChar);
-   CxList includesOfCurFile = headerFiles.FindByShortName(splitFile[splitFile.Length-1], false);
+   CxList includesOfCurFile = headerFiles.FindByShortName(splitFile[splitFile.Length - 1], false);
    SortedList filesThatIncludeCurFile = new SortedList(new Checkmarx.DataCollections.PragmaComparer());
    foreach(CxList cur in includesOfCurFile){//Add filename of each include
        CSharpGraph curGraph = cur.GetFirstGraph();
```

CPP / CPP_MISRA_CPP / R00_01_10_Find_Unused_Defined_Functions

Code changes

```
---
+++
@@ -15,19 +15,19 @@
        }
 */


-CxList protoMethods = All.FindByType(typeof(MethodDecl));
+CxList protoMethods = Find_MethodDecls();
 //Remove main and includes.
-protoMethods -= protoMethods.FindByShortName("main") + protoMethods.FindByShortName("INCLUDEREPLACE");
+protoMethods -= protoMethods.FindByShortName("main");
 //Split prototype methods and regular ones.
-CxList methods = All.FindByFathers(protoMethods).FindByType(typeof(StatementCollection)).GetFathers();
+CxList methods = All.FindByFathers(protoMethods).FindByType<StatementCollection>().GetFathers();
 protoMethods -= methods;
-CxList methodInvokes = All.FindByType(typeof(MethodInvokeExpr));
+CxList methodInvokes = Find_Methods();
 //Remove regular methods with invokes under their definition.
 CxList unused = methods - methods.FindDefinition(methodInvokes.FindByShortName(methods));
 //Remove methods that were called by using-directive
-CxList imports = All.FindByType(typeof(Import));
```

```
-CxList classes = All.FindByType(typeof(ClassDecl));
-CxList namespaces = All.FindByType(typeof(NamespaceDecl));
+CxList imports = Find_Import();
+CxList classes = Find_ClassDecl();
+CxList namespaces = Find_NamespaceDecl();
 foreach(CxList cur in imports) {
    string[] use = cur.TryGetCSharpGraph<Import>().Namespace.Split('.');
    if (use.Length != 2) {
@@ -40,8 +40,9 @@


 //Keep prototype methods with invokes under their definition.
 CxList protoUsed = protoMethods.FindDefinition(methodInvokes.FindByShortName(protoMethods));
-CxList allParams = All.FindByType(typeof(ParamDecl)).GetParameters(protoUsed + unused);
-CxList typerefs = All.FindByType(typeof(TypeRef)).GetByAncs(allParams.GetParameters(protoUsed + unused));
+CxList relevantParams = All.NewCxList(protoUsed, unused);
+CxList allParams = Find_ParamDecl().GetParameters(relevantParams);
+CxList typerefs = Find_TypeRef().GetByAncs(allParams.GetParameters(relevantParams));
 string oldFile = "";
 CxList others = All.NewCxList();


@@ -63,8 +64,8 @@
        //Check if otherMethod overrides currMethod.
        if(currParams.Count == otherParams.Count) {
            for(int i = 0; i < currParams.Count; i++) {
-               string cName = ((TypeRef) currParams.data.GetByIndex(i)).TypeName;
-               string oName = ((TypeRef) otherParams.data.GetByIndex(i)).TypeName;
+               string cName = (currParams.ElementAt(i).TryGetCSharpGraph<TypeRef>()).TypeName;
+               string oName = (otherParams.ElementAt(i).TryGetCSharpGraph<TypeRef>()).TypeName;
                if(!cName.Equals(oName)) {
                    isOverride = false;
                    break;
```

CPP / CPP_MISRA_CPP / R07_03_01_Definitions_in_Global_Namespace

Code changes

```
---
+++
@@ -15,21 +15,16 @@


 //Find typedefs that create compliance with Rule 3-9-2.
 // start with all type objects
-CxList basicTypes = All.FindByType(typeof(TypeRef));
+CxList basicTypes = Find_TypeRef();


 // we only care about basic types
-basicTypes = basicTypes.FindByName("char") +
-   basicTypes.FindByName("short") +
-   basicTypes.FindByName("int") +
-   basicTypes.FindByName("long") +
```

```
-   basicTypes.FindByName("float") +
-   basicTypes.FindByName("double");
+basicTypes = basicTypes.FindByNames("char", "short", "int", "long", "float", "double");

 // remove redundent instances
-basicTypes -= basicTypes.FindByFathers(All.FindByType(typeof(ObjectCreateExpr)));
+basicTypes -= basicTypes.FindByFathers(Find_ObjectCreations());

 // Find typedef'd types, save relevant ones in typedefs.
-CxList typedefDecl = All.FindByType(typeof(StringLiteral)).FindByName("CX_TYPEDEF").GetAncOfType(typeof(FieldDecl));
+CxList typedefDecl = Find_String_Literal().FindByName("CX_TYPEDEF").GetAncOfType<FieldDecl>();
 CxList typedefs = All.NewCxList();
 foreach (CxList typedefD in typedefDecl) {
    CxList cut = All.FindByFathers(typedefD) * basicTypes;
@@ -39,14 +34,12 @@
 }

 //Build potential objects that may be under global namespace.
-CxList methods = All.FindByType(typeof(MethodDecl));
-CxList classes = All.FindByType(typeof(ClassDecl));
+CxList methods = Find_Methods();
+CxList classes = Find_ClassDecl();
 classes -= classes.FindByShortName("checkmarx_default_classname*");
-CxList potent = All.FindByType(typeof(FieldDecl)) +
-   All.FindByType(typeof(StructDecl)) +
-   All.FindByType(typeof(EnumDecl)) +
-   methods - methods.FindByShortName("main") - methods.FindByShortName("INCLUDEREPLACE")
-   - typedefs;
+CxList potent = All.NewCxList(Find_FieldDecls(), Find_StructDecl(), Find_Enum_Declarations(), methods);
+potent -= methods.FindByShortName("main");
+potent -= typedefs;
 potent -= potent.FindByShortName("checkmarx_default_*");

 foreach(CxList obj in potent) {
@@ -54,13 +47,13 @@
        continue;
    }
    // if obj is not in a class, still relevant.
-   CxList scope = obj.GetAncOfType(typeof(ClassDecl));
+   CxList scope = obj.GetAncOfType<ClassDecl>();
    if (scope.FindByShortName("checkmarx_default_classname*").Count == 0) {
        continue;
    }

    //Check if obj is in a defined namespace.
-   scope = obj.GetAncOfType(typeof(NamespaceDecl));
+   scope = obj.GetAncOfType<NamespaceDecl>();
```

```
    if (scope.FindByShortName("Namespace*").Count >= 1) {
        result.Add(obj);
@@ -70,12 +63,12 @@
 //Take care of classes.
 foreach(CxList obj in classes) {
    //Check if in defined namespace
-   CxList scope = obj.GetAncOfType(typeof(NamespaceDecl));
+   CxList scope = obj.GetAncOfType<NamespaceDecl>();
    if (scope.FindByShortName("Namespace*").Count == 0){
        continue;
    }
    //Check if in a method
-   scope = obj.GetAncOfType(typeof(MethodDecl));
+   scope = obj.GetAncOfType<MethodDecl>();
    if (scope.Count > 0) {
        continue;
    }
```

CPP / CPP_MISRA_CPP / R16_00_03_Use_Of_Undef_Directive

Code changes

```
---
+++
@@ -15,4 +15,6 @@
 */

-result = All.FindByRegex(@"#undef\W", false, false, false, All.NewCxList());
+Regex undef = new Regex(@"\s*#\s*undef\W", RegexOptions.Compiled);
+
+result = Find_CPPDirectives().FilterByDomProperty<Comment>(cmt => undef.IsMatch(cmt.ShortName));
```

CPP / CPP_MISRA_CPP / R16_00_07_Undefined_Macro_Identifiers

Code changes

```
---
+++
@@ -13,23 +13,15 @@

 */

-System.Collections.Generic.List<string> preDefined = new System.Collections.Generic.List<string>();
-preDefined.Add("__LINE__");
-preDefined.Add("__FILE__");
-preDefined.Add("__DATE__");
-preDefined.Add("__TIME__");
-preDefined.Add("__STDC__");
-preDefined.Add("__cplusplus");
+string[] preDefined = {"__LINE__", "__FILE__", "__DATE__", "__TIME__", "__STDC__", "__cplusplus"};
```

```
 //Find all #define and #if/#elif
 CxList regexDefine = All.NewCxList();
-CxList definedConsts = All.NewCxList();
 CxList regexIfs = All.NewCxList();
 All.FindByRegex(@"(?<=^\s*#\s*define\s+)\w+[^(]", regexDefine);
 All.FindByRegex(@"^\s*#\s*(if|elif)\s", regexIfs);

-CxList headerFiles = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-    .FindByShortName("CX_INCL"));
+CxList headerFiles = Find_Includes();

 ArrayList files = new ArrayList();
 foreach(CxList cur in regexIfs){
@@ -115,7 +107,7 @@
                 continue;
             }
             if(preDefined.Contains(match) ||
-            defined.ContainsKey(match) ||
+                defined.ContainsKey(match) ||
             match.Equals("if") || match.Equals("elif")) {
                 continue;
             }
```

CPP / CPP_MISRA_CPP / R16_01_01_Defined_Standart_Forms

Code changes

```
---
+++
@@ -20,12 +20,18 @@
 */



+CxList cppDirectives = Find_CPPDirectives();
+
+Regex cComment = new Regex(@"//.*|/\*.*", RegexOptions.Compiled);
+
+Func<string, string> removeComment = (str) => cComment.Replace(str, "");
+
 // makes sure that the defined is in one of the two standard forms:
-CxList nonStandart = All.FindByRegex(@"\s*#\s*(if|elif)\s*[^\n\r]*defined((?=\s+\w+[^\w\n\|\|&&]+\w)|(?=(\s*[(]\s*\w+[^\w\n\s]+\w*\s*[)])))",
-    All.NewCxList());
+Regex re1 = new Regex(@"\s*#\s*(if|elif)\s*[^\n\r]*defined((?=\s+[_\w]+[^_\w\n\|\|&&]+[_\w])|(?=(\s*[(]\s*[_\w]+[^[_\w]\n\s]+[_\w]*\s*[)])))", RegexOptions.Compiled);
+//deals with the special case:
+Regex re2 = new Regex(@"#\s*define\s+(\w+)\s+defined\s+.*\n*.*\s*#(?=\s*if|elif\s+\1+\s+\w+|[(]\w+[)]\s+)", RegexOptions.Compiled);

-//deals with the special case:
-CxList specialCase = All.FindByRegex(@"#\s*define\s+(\w+)\s+defined\s+.*\n*.*\s*#(?=\s*if|elif\s+\1+\s+\w+|[(]\w+[)]\s+)",
-    All.NewCxList());
-
```

```
-result = nonStandart + specialCase;
+result = cppDirectives.FilterByDomProperty<Comment>(c => {
+   string shortName = removeComment(c.ShortName);
+   return re1.IsMatch(shortName) || re2.IsMatch(shortName);
+   });
```

CPP / CPP_MISRA_CPP / R16_02_06_Include_Directive_In_Wrong_Format

Code changes

```
---
+++
@@ -9,8 +9,7 @@

 */

-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
+CxList includes = Find_Includes();

 // check for the wrong format flag
 foreach (CxList cur in includes){
```

CPP / CPP_MISRA_CPP / R18_00_04_Ctime

Code changes

```
---
+++
@@ -14,24 +14,15 @@

 // Safety check for the violating h file
 // (it is also a violation in itself)
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
-CxList hFile = includes.FindByShortName("time.h") + includes.FindByShortName("ctime");
+CxList includes = Find_Includes();
+CxList hFile = includes.FindByShortNames("time.h", "ctime");
 if (hFile.Count > 0){
    // The functions defined by time.h
-   CxList methodInvokes = All.FindByType(typeof(MethodInvokeExpr));
+   CxList methodInvokes = Find_Methods();

    //Time manipulation:
-   result.Add(methodInvokes.FindByShortName("clock") +
-       methodInvokes.FindByShortName("difftime") +
-       methodInvokes.FindByShortName("mktime") +
-       methodInvokes.FindByShortName("time") +
-       //Conversion:
-       methodInvokes.FindByShortName("asctime") +
-       methodInvokes.FindByShortName("ctime") +
-       methodInvokes.FindByShortName("gmtime") +
```

```
-       methodInvokes.FindByShortName("localtime") +
-       methodInvokes.FindByShortName("strftime"));
+   result.Add(methodInvokes.FindByShortNames("clock","difftime", "mktime",
+       "time", "asctime", "ctime", "gmtime", "localtime", "strftime"));

    // The macros defined by time.h
    result.Add(All.FindByShortName("CLOCKS_PER_SEC"));
@@ -45,23 +36,13 @@

    // The types

-
-   result.Add(All.FindByType(typeof(TypeRef)).FindByShortName("clock_t") +
-       All.FindByType(typeof(TypeRef)).FindByShortName("size_t") +
-       All.FindByType(typeof(TypeRef)).FindByShortName("time_t") +
-       All.FindByType(typeof(TypeRef)).FindByRegex("struct tm", false, false, false));
+   CxList typeRefs = Find_TypeRef();
+   result.Add(typeRefs.FindByShortNames("clock_t", "size_t", "time_t"),
+              typeRefs.FindByRegex("struct tm", false, false, false));
    // struct tm members
-   CxList memberAccess = All.FindByType(typeof(MemberAccess));
+   CxList memberAccess = Find_MemberAccesses();
-   result.Add(memberAccess.FindByRegex("tm_sec;", All.NewCxList()) +
-       memberAccess.FindByRegex("tm_min", false, false, false) +
-       memberAccess.FindByRegex("tm_hour", false, false, false) +
-       memberAccess.FindByRegex("tm_mday", false, false, false) +
-       memberAccess.FindByRegex("tm_mon", false, false, false) +
-       memberAccess.FindByRegex("tm_year", false, false, false) +
-       memberAccess.FindByRegex("tm_wday", false, false, false) +
-       memberAccess.FindByRegex("tm_yday", false, false, false) +
-       memberAccess.FindByRegex("tm_yday", All.NewCxList()) +
-       memberAccess.FindByRegex("tm", All.NewCxList()));
+   result.Add(memberAccess.FindByRegex("tm(_(yday|sec;))?", All.NewCxList()),
+       memberAccess.FindByRegex("tm_(min|hour|mday|mon|year|wday|yday)", false, false, false));
 }
```

CPP / CPP_MISRA_CPP / R18_07_01_Csignal

Code changes

```
---
+++
@@ -18,16 +18,15 @@

 // Safety check for the violating h file
 // (it is also a violation in itself)
-CxList includes = All.FindByType(typeof(StringLiteral)).GetParameters(Find_Methods()
-   .FindByShortName("CX_INCL"));
```

```
-CxList hFile = includes.FindByShortName("signal.h") + includes.FindByShortName("csignal");
+CxList includes = Find_Includes();
+CxList methods = Find_Methods();
+CxList hFile = includes.FindByShortNames("signal.h", "csignal");
 if (hFile.Count > 0){
    // The functions defined by signal.h
-   result.Add(All.FindByType(typeof(MethodInvokeExpr)).FindByShortName("signal") +
-       All.FindByType(typeof(MethodInvokeExpr)).FindByShortName("raise"));
+   result.Add(methods.FindByShortNames("signal", "raise"));

    // The macros defined by signal.h
-   result.Add(All.FindByShortNames(new List<string>(){"SIG_DFL","SIG_ERR","SIG_IGN","SIGABRT","SIGFPE","SIGILL","SIGINT","SIGSEGV","SIGTERM"}));
+   result.Add(All.FindByShortNames("SIG_DFL","SIG_ERR","SIG_IGN","SIGABRT","SIGFPE","SIGILL","SIGINT","SIGSEGV","SIGTERM"));

    // Remove all locally defined instances
    CxList defs = All.FindDefinition(result);
@@ -37,5 +36,5 @@
    result.Add(hFile);

    // The types
-   result.Add(All.FindByType(typeof(TypeRef)).FindByShortName("sig_atomic_t"));
+   result.Add(Find_TypeRef().FindByShortName("sig_atomic_t"));
 }
```

CPP / CPP_MISRA_C_2012 / R02_X_Unused_Code

Code changes

```
---
+++
@@ -69,11 +69,10 @@
    return invalidOperand.GetAncOfType<BinaryExpr>();
 };

-//Add call of empty functions to result
-result.Add(Find_Methods().FindAllReferences(Find_Empty_Methods()));
-
-//Add to result +, -, * and / dead operators
 result.Add(
+   // Call of empty functions are hard to track, so add empty methods directly
+   Find_Empty_Methods(),
+   //Add to result +, -, * and / dead operators
    getDeadOp(Checkmarx.Dom.BinaryOperator.Add, 0, false),
    getDeadOp(Checkmarx.Dom.BinaryOperator.Subtract, 0, false),
    getDeadOp(Checkmarx.Dom.BinaryOperator.Multiply, 1, false),
@@ -97,7 +96,6 @@

 // Find for all methods definitions
 CxList methodDefinitions = stmtCollections.GetByAncs(methodDecls).GetAncOfType<MethodDecl>();
-methodDefinitions -= Find_Cx_Method_Declarations();
```

```
 // Get all statements from method declarations
 CxList stmtsFromMethodDecls = stmtCollections.GetByAncs(methodDefinitions);
@@ -109,20 +107,21 @@
 result.Add(unassignedExprs.FindByFathers(exprsMethodDeclFathers));

 // R02.04
-// Create a list of CxAnony refs
-CxList toRemove = All.NewCxList();
-toRemove.Add(typeRefs.FindByShortName("Cx*"));
+// Find references to anonymous types
+CxList anonymousTypes = typeRefs.FindByShortName("Cx*");
+CxList typedefs = Find_TypeAliasDecl();

-// Find for all typeRefs that have tags and remove invalid refs
-CxList typeRefsFromTags = typeRefs.FindByFathers(Find_TypeAliasDecl());
-typeRefsFromTags -= toRemove;
+// Find all aliased types that are not anonymous
+CxList aliasedTypes = typeRefs.FindByFathers(typedefs);
+aliasedTypes-= anonymousTypes;

 // Find for used tags from refs
-CxList usedTags = typeRefs.FindAllReferences(typeRefsFromTags);
-usedTags -= typeRefsFromTags;
+CxList usedTags = typeRefs.FindAllReferences(aliasedTypes);
+usedTags -= aliasedTypes;

 // Get struct decl from tags used in the code and add to result
 result.Add(structDecls.FindDefinition(usedTags));
+result -= result.FindDefinition(usedTags.GetByAncs(structDecls));

 // R02.06
 // Find for all used labeled statements in goto
```

CPP / CPP_MISRA_C_2012 / R05_X_Identifiers

Code changes

```
---
+++
@@ -12,11 +12,12 @@
 CxList defaultClasses = classes.FindByShortName("DefaultClass");
 CxList namespaces = Find_NamespaceDecl();
 CxList methods = Find_MethodDecls();
+CxList declarators = Find_Declarators();

 CxList identifiers = All.NewCxList();
 identifiers.Add(
    Find_MemberDecl(),
-   Find_Declarators().GetByAncs(Find_VariableDeclStmt())
```

```
+   declarators.GetByAncs(Find_VariableDeclStmt())
    );


 identifiers -= identifiers.FindByFieldAttributes(Dom.Modifiers.Extern);
@@ -32,17 +33,15 @@
 identifiers -= Find_BinaryExpressions();
 // removing operators (operators overloads)
 // (Note: the "*" was removed from the List because it is the 'FindByShortNames' wildcard)
-identifiers -= identifiers.FindByShortNames(new List<string>
-   {"+","-","/","~","*=","-=","+=","++","--","==","!=",">",">=","<","<=",
-       "&&","||","<<",">>",">>>","&","^","|",")","(","]","[" });
+identifiers -= identifiers.FindByShortNames("+","-","/","~","*=",
+       "-=", "+=", "++", "--", "==", "!=", ">", ">=", "<", "<=",
+       "&&","||","<<",">>",">>>","&","^","|",")","(","]","[");

 // removing class names
 identifiers -= classes;

 // remove more artificially added identifiers
-identifiers -= identifiers.FindByShortNames(new List<string>
-   { "CxAnony*","CxLeftShiftVar*", "Lambda" });
-identifiers -= identifiers.FindByShortName(Find_Cx_Method_Declarations());
+identifiers -= identifiers.FindByShortNames("CxAnony*","CxLeftShiftVar*", "Lambda");
```

```
 // get all identifiers whose fathers are default classes
 CxList defaultClassesSons = All.NewCxList();
@@ -220,7 +219,7 @@
 Dictionary < string, CxList> parCounts = new Dictionary<string, CxList>();
 // all declarators
 CxList allDeclarations = All.NewCxList();
-allDeclarations.Add(Find_Declarators(), methods, Find_ParamDecl());
+allDeclarations.Add(declarators, methods, Find_ParamDecl());

 foreach (CxList define in defines)
 {
```

CPP / CPP_MISRA_C_2012 / R08_02_Function_Prototype_With_Named_Parameters

Code changes

```
---
+++
@@ -13,23 +13,19 @@
    its prototype should use the keyword "void".
 */

-CxList noPrototypeFunctions = All.NewCxList();
 CxList noncompliantFunctions = All.NewCxList();

 // get method declarations list and clean it up
```

```
 CxList methodDeclarations = Find_Method_Declarations();
-methodDeclarations -= Find_Cx_Method_Declarations();
-methodDeclarations -= methodDeclarations.FindByShortNames(new string[] {"main", "Lambda",
+methodDeclarations -= methodDeclarations.FindByShortNames("main", "Lambda",
     // removing operators overloads
     // (Note: the "*" was removed from the List because it is the 'FindByShortNames' wildcard)
     "+","-","/","~","*=","-=","+=","++","//","//","==","!=",">",">=","<","<=",
-    "&&","||","<<",">>",">>>","&","^","|",")","(","]","[" });
+    "&&","||","<<",">>",">>>","&","^","|",")","(","]","[" );

 // remove what 'FindByShortNames' couldn't
 methodDeclarations -= methodDeclarations
     .FilterByDomProperty<MethodDecl>(method => method.Name.Equals("*") || method.Name.EndsWith("/"));
-
-

 // finding all methods with statements
 CxList methodDefinitions = Find_StatementCollection()
```

CPP / CPP_MISRA_C_2012 / R17_07_Value_Returned_By_Non_Void_Function_Shall_Be_Used

Code changes

```
---
+++
@@ -36,7 +36,6 @@
 // Remove all invalid return types and cx methods from methods declarations list
 CxList methodDeclsWithReturnType = typeRefsReturnType.GetAncOfType<MethodDecl>();
-methodDeclsWithReturnType -= Find_Cx_Method_Declarations();

 // Get all method call from method declaration that have non-void return type
 CxList methodInvokesFromDecls = methodInvokes.FindAllReferences(methodDeclsWithReturnType);
```

CPP / CPP_MISRA_C_2012 / R17_08_Function_Parameter_Should_Not_Be_Modified

Code changes

```
---
+++
@@ -22,8 +22,9 @@
 methodDecls = Find_StatementCollection().FindByFathers(methodDecls).GetFathers();

 // Create a list of references
-CxList refs = All.NewCxList();
-refs.Add(Find_UnknownReference(), Find_IndexerRefs());
+CxList indexerRefs = Find_IndexerRefs();
+CxList refs = All.NewCxList(indexerRefs, Find_UnknownReference());
+refs -= refs.FindByFathers(indexerRefs);

 // Find for all methods declarations parameters and get all references
 CxList methodParams = Find_ParamDecl().GetParameters(methodDecls);
```

CPP / CPP_MISRA_C_2012 / R18_05_Pointer_Nesting

Code changes

```
---
+++
@@ -13,7 +13,10 @@
 CxList ptrTypeRefs = Find_PointerTypeRef();


 // Get Function Type Parents
-CxList funcParents = All.FindByType<FunctionTypeRef>().FindByFathers(ptrTypeRefs).GetFathers();
+CxList funcParents = Find_FunctionTypeRef().FindByFathers(ptrTypeRefs).GetFathers();
+
+// Don't consider pointers with associated ArrayRanks
+ptrTypeRefs -= ptrTypeRefs.FilterByDomProperty<PointerTypeRef>(x => x.ArrayRanks != null && x.ArrayRanks.Count != 0);


 // pointers of pointers
 CxList nestedPtrTypeRefs = ptrTypeRefs.FindByFathers(ptrTypeRefs).GetFathers() - funcParents;
```

CPP / CPP_MISRA_C_2012 / R18_07_to_08_Variable_Length_And_Flexible_Arrays

Code changes

```
---
+++
@@ -24,9 +24,9 @@
 */

 // Generals
+CxList arrayInitializers = Find_ArrayInitializer();
 CxList arraySizesReferences = Find_Array_Size_References_Without_Initializer();
-CxList arraySizes = Find_Array_Size_Literals();
-arraySizes.Add(arraySizesReferences);
+CxList arraySizes = All.NewCxList(Find_Array_Size_Literals(), arraySizesReferences);


 // Find for all array rank specifiers that have dimension == 1
 CxList filteredRankSpecifiers = Find_RankSpecifier().FilterByDomProperty<RankSpecifier>(
@@ -41,3 +41,6 @@
     declsFromRankSpecifiers,
     // Add to result references used as an array size - R18.8
     arraySizesReferences.GetAncOfType<Declarator>());
+
+// Cases where there is a initialization are not flexible (compiler declares correct size)
+result -= arrayInitializers.GetByAncs(result).GetAncOfType<Declarator>();
```

CPP / CPP_MISRA_C_2012 / R20_01_Include_Directive_Precedence

Code changes

```
---
+++
@@ -15,26 +15,19 @@


 */
```

```
-CxList methodDecls = Find_MethodDecls();
-CxList includes = methodDecls.FindByShortName("INCLUDEREPLACE");
+CxList includes = Find_Includes();


 if (includes.Count > 0)
 {
    // Defines the DOM scope
-   CxList targetElements = All.NewCxList();
-   targetElements.Add(
-       methodDecls,
+   CxList targetElements = All.NewCxList(
+       Find_MethodDecls(),
        Find_MethodRef(),
        Find_UnknownReference(),
        Find_TypeAliasDecl(),
        Find_Declarators(),
        Find_Enum_Declarations()
        );
-
-   // Removes includes and all associated dom objects
-   targetElements -= targetElements.GetByAncs(includes);
-   // Removes undefs and all associated dom objects
-   targetElements -= targetElements.GetByAncs(Find_Cx_Method_Declarations());
    // group all includes by FileId
    Dictionary<int, CxList> dicIncludes = new Dictionary<int, CxList>();
@@ -53,6 +46,7 @@
            dicIncludes.Add(fileId, include.Clone());
        }
    }
+


    // group all targetElements by FileId
    Dictionary<int, CxList> dicTargetElements = new Dictionary<int, CxList>();
@@ -71,6 +65,7 @@
            dicTargetElements.Add(fileId, targetElement.Clone());
        }
    }
+


    List<int> fileIds = dicIncludes.Keys.ToList();


@@ -81,7 +76,7 @@
            CxList fileIncludes = dicIncludes[fileId];
            CxList fileTargetElements = dicTargetElements[fileId];


-           int maxIncludeLine = fileIncludes.CxSelectElementValue<MethodDecl, LinePragma>(p => p.LinePragma)
```

```
+           int maxIncludeLine = fileIncludes.CxSelectElementValue<CSharpGraph, LinePragma>(p => p.LinePragma)
                .Select(s => s.Line)
                .Max();
            int minTargetElementLine = fileTargetElements.CxSelectElementValue<CSharpGraph, LinePragma>(p => p.LinePragma)
@@ -95,4 +90,5 @@
            }
        }
    }
+
 }
```

CSharp / CSharp_APISecurity / CSharp_WebApi_GetApiList

Code changes

```
---
+++
@@ -1,76 +1,46 @@
-/*
-CSharp_WebApi_GetApiList
-Given the following API:
+//Java_WebApi_GetApiList
+//Find more info about this query here:
+//https://checkmarx.atlassian.net/wiki/spaces/AS/pages/7119143142/C+Queries+Performance+Research

-> [HttpPost]
-> public IActionResult Post([FromBody] CreateTextModule model)
+CxList customAttributes = Find_CustomAttribute();
+CxList classDecls = Find_ClassDecl();
+CxList methodDecls = Find_MethodDecls();
+CxList methodInvokes = Find_Methods();

-This query will return
-{
-"url":"api/apps/text/[controller]/Post",
-"method" : {"name" :"Post", "row": 37, "column": 30},
-"httpMethod": "POST",
-"responseContentType": "" ,
-"responseInfo":[{
-   httpCode:200,
-   type: "IActionResult",
-   typeStructure: {
-   (...)
-   }
-}]
-"requestInfo":[{
-"ParamName": "model",
-"fileName": "/Weapsy-master/src/Weapsy.Apps.Text/Api/TextController.cs",
-"ParamRow": 37,
-"ParamColumn": 63,
```

```
-"ParamType": "CreateTextModule",
-"typeStructure":{
-"ModuleId": "Guid",
-"Id": "Guid",
-"Content": "string"
-},
-"ParamLocation": "bodyParam",
-"FrameworkArgs": "False"
-}]
-}
-*/
-// Find costum attributes
-CxList customAttributes = Find_CustomAttribute();
-// Find controllers
-CxList classDecl = Find_ClassDecl();
-CxList methodInvokes = Find_Methods();
-// Find controllers
-CxList controllerInstances = Find_TypeRef().FindByShortNames(new List<string>{
-       "ControllerBase", "ApiController", "Controller", "PageModel", "ODataController"});
-// Find controllers in controllers
+CxList allAnnotations = All.NewCxList();
+
+string[] classControllerAnnotations = new string[5]{"ControllerBase", "ApiController",
+   "Controller", "PageModel", "ODataController"};
+string[] mapAnnotations = new string[4]{"MapRoute", "MapHttpRoute", "MapControllerRoute",
+   "MapDefaultControllerRoute"};
+string[] specialAnnotations = new string[4]{"MapGet","MapPost", "MapPut", "MapDelete"};
+string[] annotationsList = new string[8]{"HttpGet", "HttpPost", "HttpPut", "HttpDelete",
+   "HttpPatch", "HttpHead", "HttpOptions", "ODataRoute"};
+
+//Find controllers and its methods
+CxList controllerInstances = Find_TypeRef().FindByShortNames(classControllerAnnotations);
 CxList controllers = controllerInstances.GetFathers().GetFathers().FindByType<ClassDecl>();
-controllers.Add(classDecl.InheritsFrom(controllers));
-// Find all methods of controllers
-CxList methods = Find_MethodDecls();
-CxList methodDecls = methods.GetByClass(controllers);
-List < string > annotationsToIgnore = new List<string>(){"NonAction", "ODataIgnored"};
-methodDecls -= customAttributes.FindByShortNames(annotationsToIgnore).GetAncOfType<MethodDecl>();
-// Keep only public methods
-methodDecls = methodDecls.FindByFieldAttributes(Modifiers.Public);
-List <string> annotations = new List<string>() {
-       "HttpGet", "HttpPost", "HttpPut", "HttpDelete", "HttpPatch", "HttpHead", "HttpOptions", "ODataRoute"};
-List <string> contentResponseTypeAnnotations = new List <string>() {"Produces", "Consumes"};
-CxList allMapRoute = methodInvokes.FindByShortNames(new List<string>{"MapRoute","MapHttpRoute", "MapControllerRoute",
-       "MapDefaultControllerRoute"});
-CxList specialMapRoute = methodInvokes.FindByShortNames(new List<string>{"MapGet","MapPost", "MapPut", "MapDelete"});
-// Find HttpGet, HttpPost, HttpPut and HttpDelete annotations
```

```
-CxList allRequestAnnotations = customAttributes.FindByShortNames(annotations);
-CxList allResponseContentTypeAnnotations = customAttributes.FindByShortNames(contentResponseTypeAnnotations);
-CxList allAnnotations = All.NewCxList();
-allAnnotations.Add(allRequestAnnotations, allResponseContentTypeAnnotations);
-//Find Route annotations
+controllers.Add(classDecls.InheritsFrom(controllers));
+
+CxList methodsList = methodDecls.GetByClass(controllers).FindByFieldAttributes(Modifiers.Public);
+methodsList -= customAttributes.FindByShortNames(new string[2]{"NonAction", "ODataIgnored"}).GetAncOfType<MethodDecl>();
+
+//Find minimal APIs(basic support) and convention-based routing
+CxList specialMapRoute = methodInvokes.FindByShortNames(specialAnnotations);
+CxList allMapRoute = methodInvokes.FindByShortNames(mapAnnotations);
+
+//Find Http annotations, responseContentType annotations, Route annotations and AcceptVerbs Annotations
+CxList allRequestAnnotations = customAttributes.FindByShortNames(annotationsList);
+CxList allResponseContentTypeAnnotations = customAttributes.FindByShortNames(new string[2]{"Produces", "Consumes"});
 CxList allRouteAnnotations = customAttributes.FindByShortName("Route");
-allAnnotations.Add(allRouteAnnotations);
-// Find AcceptVerbs Annotations
 CxList allAcceptVerbsAnnotations = customAttributes.FindByShortNames("AcceptVerbs");
+allAnnotations.Add(allRequestAnnotations, allResponseContentTypeAnnotations, allRouteAnnotations);
 allAnnotations.Add(allAcceptVerbsAnnotations);
+
+//Methods inventory (annotated and unannotated)
 CxList methodAnnotations = allAnnotations.GetAncOfType<MethodDecl>();
-CxList methodNoAnnotations = methodDecls - methodAnnotations;
+CxList methodNoAnnotations = methodsList - methodAnnotations;
+
 result.Add(CSharp_WebApi_MethodAnnotation(methodAnnotations, customAttributes, allMapRoute,
-   allRequestAnnotations, allRouteAnnotations, allAcceptVerbsAnnotations, allAnnotations),
+   allRequestAnnotations, allRouteAnnotations, allAcceptVerbsAnnotations, allAnnotations, classDecls),
    CSharp_WebApi_MethodNoAnnotation(methodNoAnnotations, customAttributes, allMapRoute, specialMapRoute, allRouteAnnotations));
```

CSharp / CSharp_Exploitable_Path / CSharp_Find_UnresolvedMethods

Code changes

```
---
+++
@@ -24,14 +24,6 @@
    typeof(StructDecl),
    typeof(VariableDecl)
    );
-
-Func<string, string, string[], LinePragma, int> AddToResults = (objectName, methodName, methodParams,LP) => {
-   string signature = objectName + methodName + "||(" + string.Join("||,", methodParams) + "||)";
-   Comment comment = new Comment(methodName, methodName, rootComment, LP);
-   comment.ResolveShortName(signature);
-   result.Add(comment.NodeId, comment);
```

```
-   return 1;
-   };

 // Imports

@@ -63,48 +55,85 @@
    return "object";
    };

+//Get full projectPath to remove it from filename
+string projectPath = cxScan.GetScanProperty("projectPath");
+int lastIndexOfBackSlash = projectPath.LastIndexOf(cxEnv.Path.DirectorySeparatorChar);
+Match guid = Regex.Match(projectPath, @"(?im)[{(]?[0-9A-F]{8}[-]?(?:[0-9A-F]{4}[-]?){3}[0-9A-F]{12}[)}]?$");
+if(guid.Success) lastIndexOfBackSlash = projectPath.IndexOf(guid.Value) + guid.Value.Length;

-foreach(CxList method in allMethods){
-   if (definitions.FindDefinition(method).Count != 0){
-       continue;
+List<string> names = allMethods.CxSelectElementValues<Expression, string>(x => x.ShortName);
+
+//For each name
+foreach(String str in names.Distinct()){
+
+   int i = 0;
+   CxList methodsByName = allMethods.FindByShortName(str);
+   string methodInfoFinal = "{ \n \"method\": {\"name\":" + "\"" + str + "\", \n \"calls\": [";
+
+   foreach(CxList method in methodsByName){
+       if (definitions.FindDefinition(method).Count != 0){
+           continue;
+       }
+
+       MethodInvokeExpr graph = method.TryGetCSharpGraph<MethodInvokeExpr>();
+       LinePragma lp = graph.LinePragma;
+       List<string> methodParams = new List<string>();
+       foreach(Param parameter in graph.Parameters){
+           methodParams.Add(GetParamType(parameter));
+       }
+       string paramsStr = string.Join(",", methodParams);
+
+       CxList target = method.GetTargetOfMembers();
+       CxList def = definitions.FindDefinition(target);
+       CxList objectCreate = ctors.GetByAncs(def);
+
+       Expression methObject = method.TryGetCSharpGraph<Expression>();
+       string methodName = method.GetName();
+
+       bool foundSignature = false;
```

```
+       string name = "";
+       string objectName = "";
+       if (methObject is ObjectCreateExpr) {
+           foundSignature = true;
+           name = methodName;
+           // In case of ctor, method and class names match
+           objectName = methodName;
+       }
+       else if (objectCreate.Count > 0)
+       {
+           // When refering to Object Create Expression
+           foundSignature = true;
+           name = objectCreate.GetName();
+           objectName = objectCreate.GetName();
+       }
+
+       if (objectName.StartsWith("CxOrphanClass_")) {
+           foundSignature = true;
+           objectName = "object";
+       }
+
+       string fullFilename = lp.FileName.Remove(0, lastIndexOfBackSlash);
+       char pathSeparator = cxEnv.Path.DirectorySeparatorChar;
+       string fileName = pathSeparator.Equals('\\') ? Regex.Replace(fullFilename, @"\\", "/") : fullFilename;
+
+       methodInfoFinal += " \n { "
+           + "\n \"fullname\": " + "\"" + objectName + "||." + methodName + "\""
+           + ", \n \"shortname\": " + "\"" + methodName + "\""
+           + ", \n \"parameters\": " + "\"" + paramsStr + "\""
+           + ", \n \"line\": " + lp.Line
+           + ", \n \"column\": " + lp.Column
+           + ", \n \"sourcefile\": " + "\"" + fileName + "\""
+           + "\n }, ";
+
+       i++;
    }
-
-   MethodInvokeExpr graph = method.TryGetCSharpGraph<MethodInvokeExpr>();
-   List<string> methodParams = new List<string>();
-   foreach(Param parameter in graph.Parameters){
-       methodParams.Add(GetParamType(parameter));
+   if(i > 0){
+       methodInfoFinal += "\n ] \n } \n }";
+       methodInfoFinal = new System.Text.StringBuilder(methodInfoFinal).ToString();
+
+       Comment comment = new Comment(methodInfoFinal, methodInfoFinal, rootComment, methodsByName.GetFirstGraph().LinePragma);
+       comment.ResolveShortName(methodInfoFinal);
+
```

```
+       result.Add(comment.NodeId, comment);
    }
-
-   CxList target = method.GetTargetOfMembers();
-   CxList def = definitions.FindDefinition(target);
-   CxList objectCreate = ctors.GetByAncs(def);
-
-   Expression methObject = method.TryGetCSharpGraph<Expression>();
-   string methodName = method.GetName();
-
-   bool foundSignature = false;
-   string name = "";
-   string objectName = "";
-   if (methObject is ObjectCreateExpr) {
-       foundSignature = true;
-       name = methodName;
-       // In case of ctor, method and class names match
-       objectName = methodName;
-   }
-   else if (objectCreate.Count > 0)
-   {
-       // When refering to Object Create Expression
-       foundSignature = true;
-       name = objectCreate.GetName();
-       objectName = objectCreate.GetName();
-   }
-
-   if (objectName.StartsWith("CxOrphanClass_")) {
-       foundSignature = true;
-       objectName = "object";
-   }
-
-   string signature = foundSignature ? objectName + "|.|." : "";
-
-   AddToResults(signature, methodName, methodParams.ToArray(), graph.LinePragma);
 }
```

CSharp / CSharp_Heuristic / Heuristic_CSRF

Code changes

```
---
+++
@@ -12,9 +12,8 @@
        CxList requests = Find_Interactive_Inputs();
        requests.Add(All.FindByName("*Request.QueryString*"));
        CxList strings = Find_Strings();
-       CxList write = strings.FindByNames(new string [] {"*update*", "*delete*", "*insert*"}
-           , StringComparison.OrdinalIgnoreCase);
-
```

```
+       CxList write = strings.FindByNames(new string [] {"update*", "delete*", "insert*"}
+           , StringComparison.OrdinalIgnoreCase);
        result = possible_db.DataInfluencedBy(write).DataInfluencedBy(requests);
    }
 }
```

CSharp / CSharp_High_Risk / Reflected_XSS_All_Clients

Code changes

```
---
+++
@@ -1,4 +1,4 @@
-if((All.isWebApplication || Check_Web_Application().Any()) && Find_ASP_XSS_Safe().Count == 0)
+if(All.isWebApplication || Check_Web_Application().Any())
 {
    CxList methods = Find_Methods();
    CxList parameters = Find_Parameters();
```

CSharp / CSharp_High_Risk / Second_Order_SQL_Injection

Code changes

CSharp / CSharp_High_Risk / Stored_XSS

Code changes

```
---
+++
@@ -1,4 +1,4 @@
-if((All.isWebApplication || Check_Web_Application().Count() > 0) && Find_ASP_XSS_Safe().Count == 0)
+if(All.isWebApplication || Check_Web_Application().Count() > 0)
 {
    CxList inputs = Find_Stored_Inputs();
    CxList outputs = Find_XSS_Outputs();
```

CSharp / CSharp_Low_Visibility / Log_Forging

Code changes

```
---
+++
@@ -1,9 +1,41 @@
+CxList collections = All.NewCxList(Find_Methods(), Find_MemberAccesses(), Find_IndexerRefs(), Find_Param());
+CxList unkRefs = Find_UnknownReference();
+CxList decoders = All.NewCxList(Find_Decode_base64(), collections.FindByMemberAccess("HttpUtility.UrlDecode"));
+
 CxList Inputs = Find_Interactive_Inputs();
+
+//Inputs that cannot contain linebreaks
+string[] noLinebreakInputProperties = new string[]{"Cookies", "Headers", "HttpMethod", "Path", "PathInfo",
+   "QueryString", "RawUrl", "RequestType", "Url", "UrlReferrer", "MapPath"};
+string[] classNames = new string[]{"HttpRequest", "Request"};
+CxList noLinebreakInputs = All.NewCxList(
```

```
+   collections.FindByMemberAccesses(classNames, noLinebreakInputProperties, false),
+   collections.FindByName("HttpContext.Current.Request")
+       .GetMembersOfTarget().FindByShortNames(noLinebreakInputProperties),
+   unkRefs.GetByAncs(
+       Inputs.FindByShortNames(noLinebreakInputProperties, false).FindByTypes(typeof(Param), typeof(IndexerRef))).
+       FindByShortNames(classNames).GetMembersOfTarget().FindByShortNames(noLinebreakInputProperties).GetFathers(),
+   Inputs.FindByShortNames(noLinebreakInputProperties, false).FindByType<Param>()
+       .CxSelectDomProperty<Param>(p => p.Value).GetTargetOfMembers()
+       .FindByShortNames(classNames).GetMembersOfTarget().GetAncOfType<Param>(),
+   unkRefs.FindDescendantsOfType<UnknownReference>(Inputs.FindByShortNames(noLinebreakInputProperties, false)
+       .FindByType<Param>().CxSelectDomProperty<Param>(p => p.Value))
+       .FindByShortNames(classNames).GetAncOfType<Param>().FindByShortNames(noLinebreakInputProperties));
+
+CxList toRemove = All.NewCxList(
+   noLinebreakInputs,
+   noLinebreakInputs.GetMembersOfTarget());
+Inputs = Inputs - toRemove;
+
 CxList Log = Find_Log_Outputs();

 CxList sanitize = Find_Log_Sanitizers();

-result = Log.InfluencedByAndNotSanitized(Inputs, sanitize);
+result = Log.InfluencedByAndNotSanitized(Inputs, sanitize).ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
+
+CxList noLinebreakInputResults = Log.InfluencedByAndNotSanitized(noLinebreakInputs, sanitize);
+CxList validResults = noLinebreakInputResults.IntersectWithNodes(decoders);
+result.Add(validResults.ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow));

 // When the flow goes through an "Remote input" (a request to a remote server) it is not vulnerable
 result -= result.IntersectWithNodes(Find_Remote_Requests());
```

Go / Go_High_Risk / Stored_Command_Injection

Code changes

```
---
+++
@@ -1,6 +1,6 @@
 /*** Finding inputs and sanitizers ***/
 CxList inputs = Find_Read();
-inputs.Add(Find_DB_Out(), S3_Find_DB_Out(), DynamoDB_Find_Inputs());
+inputs.Add(Find_DB_Out());
 CxList sanitizers = Find_Command_Injection_Sanitize();
 sanitizers.Add(Find_Test_Code());
```

Go / Go_High_Risk / Stored_XSS_All_Clients

Code changes

```
---
+++
@@ -9,11 +9,11 @@
 CxList outputs = All.NewCxList();
 CxList sanitize = All.NewCxList();

-fromDB.Add(Find_DB(), S3_Find_DB_Out(), DynamoDB_Find_Inputs());
+fromDB.Add(Find_DB());

 fromFiles = Find_Read();

-stored.Add(fromDB,fromFiles);
+stored.Add(fromDB, fromFiles);
 outputs = Find_XSS_Outputs();
 sanitize = Find_XSS_Sanitize();

@@ -27,7 +27,7 @@
 CxList fileRead = fromFiles.FindByShortName("Read");
 CxList fileReadAt = fromFiles.FindByShortName("ReadAt");
 open.Add(All.FindByMemberAccess("os.*").FindByShortNames(new List<string>{"Open", "OpenFile"}));
-read.Add(fileRead,fileReadAt);
+read.Add(fileRead, fileReadAt);

 CxList firstPath = read.InfluencedBy(open); // no need to check for sanitizers here
 CxList arguments = All.GetParameters(read, 0);
```

Go / Go_Low_Visibility / Stored_Command_Argument_Injection

Code changes

```
---
+++
@@ -1,5 +1,5 @@
 /*** Finding inputs and sanitizers ***/
-CxList inputs = All.NewCxList(Find_Read(), Find_DB_Out(), S3_Find_DB_Out(), DynamoDB_Find_Inputs());
+CxList inputs = All.NewCxList(Find_Read(), Find_DB_Out());

 CxList sanitizers = Find_Command_Injection_Sanitize();
 sanitizers.Add(Find_Test_Code());
```

Go / Go_Medium_Threat / Stored_Absolute_Path_Traversal

Code changes

```
---
+++
@@ -1,13 +1,8 @@
-CxList inputs = All.NewCxList();
-inputs.Add(
-   Find_Read(),
-   Find_DB_Out(),
-   S3_Find_DB_Out()
```

```
-   , DynamoDB_Find_Inputs());
+CxList inputs = All.NewCxList(Find_Read(), Find_DB_Out());


 CxList outputs = Find_Absolute_Path_Sinks();


 CxList sanitizers = Find_AbsolutePathTraversal_Sanitizers();


 result = inputs.InfluencingOnAndNotSanitized(outputs, sanitizers)
-           .ReduceFlowByPragma().ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
+   .ReduceFlowByPragma().ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
```

Go / Go_Medium_Threat / Stored_Relative_Path_Traversal

Code changes

```
---
+++
@@ -1,13 +1,8 @@
-CxList inputs = All.NewCxList();
-inputs.Add(
-   Find_Read(),
-   Find_DB_Out(),
-   S3_Find_DB_Out(),
-   DynamoDB_Find_Inputs());
+CxList inputs = All.NewCxList(Find_Read(), Find_DB_Out());


 CxList outputs = Find_Relative_Path_Sinks();


 CxList sanitizers = Find_RelativePathTraversal_Sanitizers();


 result = inputs.InfluencingOnAndNotSanitized(outputs, sanitizers)
-           .ReduceFlowByPragma().ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
+   .ReduceFlowByPragma().ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
```

Java / Java_APISecurity / Java_WebApi_GetApiList

Code changes

```
---
+++
@@ -1,54 +1,6 @@
-/*Java_WebApi_GetApiList
-
-Given the following API:
-
-> @RequestMapping(value="/editUser/{id}", method = RequestMethod.GET)
-> public ModelAndView editPost(@PathVariable("id") int userId){
->  ModelAndView modelAndView = new ModelAndView();
->  Authentication auth = SecurityContextHolder.getContext().getAuthentication();
->  User loggedUser = userService.findUserByEmail(auth.getName());
->  User user = userService.findUserById(userId);
->  modelAndView.addObject("user", user);
```

```
->  return modelAndView;
-> }
-
-This query will return
-
-{
-    "url":"/editUser/{id}",
-   "method" : {"name" :"editPost", "row": 155, "column": 22},
-   "httpMethod": "GET",
-   "responseContentType": "",
-   "responseInfo:"[
-    {
-   "httpCode": 200,
-   "type": "ModelAndView",
-   "typeStructure" :
-   {
-   "user":
-   {
-   "id": "int",
-   "email": "String",
-   "password": "String",
-   "name": "String",
-   "lastName": "String",
-   "bio": "String",
-   "active": "int",
-   "roles": "Set<Role>",
-   "posts
-   }],
-   "requestInfo":[
-       {
-       "ParamName": "userId",
-       "ParamType": "int",
-       "ParamLocation": "pathParam",
-       "fileName": "spring_boot_goat/src/main/java/com/example/controller/UserController.java",
-       "ParamRow": 155,
-       "ParamColumn": 55,
-       "FrameworkArgs": "False"
-   }]
-}
-*/
+//Java_WebApi_GetApiList
+//Find more info about this query here:
+//https://checkmarx.atlassian.net/wiki/spaces/AS/pages/7119634481/Java+Queries+Performance+Research


 // Finds the necessary general queries
 CxList customAttributes = Find_CustomAttribute();
@@ -62,7 +14,7 @@
 allClasses = allClasses.FindByFieldAttributes(Modifiers.Public);
```

```
 allClasses -= allClasses.FindByFieldAttributes(Modifiers.Static);
 // List of API annotations
-List <string> annotationsNames = new List<string>() {
+string[] annotationsNames = new string[] {
        "RequestMapping", "GetMapping", "PostMapping", "PutMapping", "DeleteMapping", "PatchMapping"};
 // Find all methods
 CxList allMethodDecls = methodDecls.GetByClass(allClasses);
@@ -72,14 +24,6 @@
 // Kepp only methods with annotations
 CxList methodsWithAnnotations = allMethodDecls * customAttributes.FindByShortNames(annotationsNames).
    GetAncOfType<MethodDecl>();
-
-// Retrieve a list of methods without annotations
-// This section is responsible for filtering the 'methodsNoAnnotation' list based on the following criteria:
-// > @PostConstruct, @ExceptionHandler, @Bean, @Scheduled, @InitBinder and @ModelAttribute annotations
-// > Static methods
-// > @Override annotations
-// > @Value annotations
-// > Method Invokes
 CxList methodsNoAnnotation = allMethodDecls - methodsWithAnnotations;

 // Step 1: Filtering by @PostConstruct, @ExceptionHandler, @Bean, @Scheduled, @InitBinder and @ModelAttribute annotations
@@ -105,7 +49,7 @@
 // Step 3: Remove methods that have @Override annotation but no routing annotations
 CxList classOfMethodNoAnnotation = methodsNoAnnotation.GetAncOfType<ClassDecl>();
 CxList overriddenMethods = methodsNoAnnotation.FindByFieldAttributes(Modifiers.Override);
-CxList allExtendedInterfaces = All.FindDefinition(Find_TypeRef()
+CxList allExtendedInterfaces = Find_Class_Decl().FindDefinition(Find_TypeRef()
    .FindByFathers(typeRefCollections.FindByFathers(classOfMethodNoAnnotation)));
 CxList allOverriddenMethodsInInterface = methodDecls.GetByClass(allExtendedInterfaces)
    .FilterByDomProperty<MethodDecl>(method => method.Name == overriddenMethods.FindByShortName(method.Name).GetName());
```

Java / Java_Exploitable_Path / Java_Find_UnresolvedMethods

Code changes

```
---
+++
@@ -1,3 +1,4 @@
+Comment rootComment = new Comment();
 Func < CSharpGraph,string > GetTypeName = graph => {
    if(graph is BooleanLiteral) return "boolean";
    if(graph is CharLiteral) return "char";
@@ -22,19 +23,18 @@
 // TODOs:
 // - Box.java - super.IsRunning() is missed as it has no object reference. That's a SAST bug
 // - Method params that are method invoke expressions are not handled, and marked as UNKNOWN
-CxList allParams = Find_Params();
-CxList methods = Find_Methods();
+CxList allParams = All.FindByType<Param>();
```

```
+CxList methods = All.FindByType<MethodInvokeExpr>();
 CxList methodToFor = methods.Clone();
-CxList ctors = Find_Object_Create();
-CxList typeRefs = Find_TypeRef();
-CxList unknownReferences = Find_UnknownReference();
-CxList genericTypes = Find_GenericTypeRefs();
-CxList declarators = Find_Declarators();
-CxList methodDecls = Find_MethodDecls();
-CxList viewDecls = Find_ViewDecls();
-CxList memberAccesses = Find_MemberAccesses();
-CxList classDecls = Find_ClassDecl();
-Comment rootComment = new Comment();
+CxList ctors = All.FindByType<ObjectCreateExpr>();
+CxList typeRefs = All.FindByType<TypeRef>();
+CxList unknownReferences = All.FindByType<UnknownReference>();
+CxList genericTypes = All.FindByType<GenericTypeRef>();
+CxList declarators = All.FindByType<Declarator>();
+CxList methodDecls = All.FindByType<MethodDecl>();
+CxList viewDecls = All.FindByType<ViewDecl>();
+CxList memberAccesses = All.FindByType<MemberAccess>();
+CxList classDecls = All.FindByType<ClassDecl>();

 CxList javaCollections = typeRefs.FindByShortNames(new List<string>
    {
@@ -85,7 +85,8 @@
 //Remove Checkmarx generated code for frameworks:
 //View Calls, View Output Statement, View Escaped Output Statement and View Input Statements
 methods -= methods.FindByType<ViewCall>();
-methods -= methods.FindByFathers(All.FindByTypes(typeof(ViewOutputStmt), typeof(ViewEscapedOutputStmt), typeof(ViewInputStmt)));
+methods -= methods.FindByFathers(All.FindByTypes(
+   typeof(ViewOutputStmt), typeof(ViewEscapedOutputStmt), typeof(ViewInputStmt)));
 methods -= methods.FindByShortName("_ViewInput");

 //Remove Checkmarx generated methods
@@ -154,152 +155,188 @@
    }
 }

-// For declarators cases
-CxList relevantTypes = genericTypes;
-relevantTypes.Add(typeRefs);
-relevantTypes.Add(ctors);
+//Get full projectPath to remove it from filename
+string projectPath = cxScan.GetScanProperty("projectPath");
+int lastIndexOfBackSlash = projectPath.LastIndexOf(cxEnv.Path.DirectorySeparatorChar);
+Match guid = Regex.Match(projectPath, @"(?im)[{(]?[0-9A-F]{8}[-]?(?:[0-9A-F]{4}[-]?){3}[0-9A-F]{12}[)}]?$");
+if(guid.Success) lastIndexOfBackSlash = projectPath.IndexOf(guid.Value) + guid.Value.Length;
+
```

```
+//For declarators cases
+CxList relevantTypes = genericTypes.Clone();
+relevantTypes.Add(typeRefs, ctors);
 relevantTypes -= javaCollections;
 relevantTypes -= relevantTypes.FindByShortName("?");


-foreach(CxList method in methods){
+
+//For each name
+List<string> names = methods.CxSelectElementValues<Expression, string>(x => x.ShortName);
+
+foreach(String str in names.Distinct()){
+
+   int i = 0;
+   CxList methodsByName = methods.FindByShortName(str);
+   string methodInfoFinal = "{ \n \"method\": {\"name\":" + "\"" + str + "\", \n \"calls\": [";
+
+   foreach(CxList method in  methodsByName){
+       if(methodDecls.FindDefinition(method).Count == 0){
+           CxList target = method.GetLeftmostTarget();
+           //Cases when target is already the leftmost object
+           if(target.Count == 0)
+           {
+               target = method.Clone();
+           }
+           CxList def = All.FindDefinition(target);
+
+           try{
+               Expression methObject = method.TryGetCSharpGraph<Expression>();
+               LinePragma lp = methObject.LinePragma;
+
+               if (lp.FileName.ToLower().Contains("\\plugins\\java\\"))
+               {
+                   continue;
+               }
+
+               string methodName = method.GetName();
+
+               CxList methodParams = allParams.GetParameters(method);
+               var elements = methodParams.CxSelectDomProperty<Param>(_ => _.Value)
+                   .CxSelectElementValue<CSharpGraph,string>(GetTypeName);
+
+               string paramsStr = string.Join(",", elements);
+
+               CxList objectCreate = ctors.GetByAncs(def) - javaCollections;
+
+               if((typeRefs.GetByAncs(def) * javaCollections).Count > 0){
+                   continue;
```

```
+               }
+
+               CxList typeRef = typeRefs.FindByFathers(def) - javaCollections;
+
+               CxList declar = declarators.GetByAncs(def);
+
+               //Cases where def is empty,
+               //so we look for a declarator among the references of target
+               if(declar.Count == 0)
+               {
+                   declar = declarators.FindAllReferences(target);
+               }
+
+               CxList memberAccess = target * unknownReferences;
+
+               bool foundSignature = false;
+               string name = "";
+               string objectName = "";
+
+               if (methObject is ObjectCreateExpr) {
+                   foundSignature = true;
+                   name = methodName;
+                   // In case of ctor, method and class names match
+                   objectName = methodName;
+               }
+               else if (objectCreate.Count > 0)
+               {
+                   // When refering to Object Create Expression
+                   foundSignature = true;
+                   name = objectCreate.GetName();
+                   objectName = objectCreate.GetName();
+               }
+               else if (typeRef.Count > 0)
+               {
+                   // When refering to Type Ref
+                   foundSignature = true;
+                   name = typeRef.GetName();
+                   objectName = typeRef.GetName();
+
+               }
+               else if (declar.Count > 0)
+               {
+                   // When refering to Declarators
+                   Declarator declarator = declar.TryGetCSharpGraph<Declarator>();
+                   LinePragma lpDecl = declarator.LinePragma;
+
+                   CxList typeName = relevantTypes.FindByPosition(lpDecl.FileName, lpDecl.Line);
```

```
-   if(methodDecls.FindDefinition(method).Count == 0){
-       CxList target = method.GetLeftmostTarget();
-       //Cases when target is already the leftmost object
-       if(target.Count == 0)
-       {
-           target = method;
-       }
-       CxList def = All.FindDefinition(target);
-
-       try{
-           Expression methObject = method.TryGetCSharpGraph<Expression>();
-           LinePragma lp = methObject.LinePragma;
-
-           if (lp.FileName.Contains("\\plugins\\java\\"))
-           {
-               continue;
-           }
-
-           string methodName = method.GetName();
-
-           CxList methodParams = allParams.GetParameters(method);
-           var elements = methodParams.CxSelectDomProperty<Param>(_ => _.Value)
-               .CxSelectElementValue<CSharpGraph,string>(GetTypeName);
-
-           string paramsStr = "||(" + string.Join("||,", elements) + "||)";
-
-           CxList objectCreate = ctors.GetByAncs(def) - javaCollections;
-
-           if((typeRefs.GetByAncs(def) * javaCollections).Count > 0){
-               continue;
-           }
-
-           CxList typeRef = typeRefs.FindByFathers(def) - javaCollections;
-
-           CxList declar = declarators.GetByAncs(def);
+                   if (typeName.Count > 0)
+                   {
+                       foundSignature = true;
+                       name = typeName.GetName();
+                       objectName = typeName.GetName();
+                       typeRef = typeName.Clone();
+                   }
+               }
+               else if (memberAccess.Count > 0)
+               {
+                   // When refering to member accesses
+                   def = declarators.FindDefinition(memberAccess);
+                   if (def.Count == 0)
```

```
+                    {
+                        objectName = memberAccess.GetName();
+                    }
+                }
+                else if (target.FindByType<CastExpr>().Count > 0)
+                {
+                    // When refering to cast expression
+                    CxList relevantCast = All.NewCxList();
+                    CastExpr castExpr = target.TryGetCSharpGraph<CastExpr>();
+                    relevantCast.Add(castExpr.TargetType.NodeId, castExpr.TargetType);
+                    objectName = relevantCast.GetName();
+                }
+                else if ((target * methodToFor).Count > 0)
+                {
+                    //When refering to method invoke expression
+                    CxList classMethod = classDecls.GetClass(target);
+                    objectName = classMethod.GetName();
+                }
+                else if(target.FindByTypes(typeof(BaseRef), typeof(ThisRef)).Count > 0 )
+                {
+                    //When refering to this or super keywords
+                    objectName = target.GetName();
+                }
+
+                if (objectName.StartsWith("CxOrphanClass_"))
+                {
+                    objectName = "object";
+                }
+
+                if (objectName == "String" && stringMethods.Contains(method.GetName()))
+                {
+                    continue;
+                }
+                    //Remove type parameters of Java Collections like List, HashMap, etc
+                else if ((typeRef.GetFathers() * javaCollections).Count == 1)
+                {
+                    continue;
+                }
+                else if (javaCollections.FindByShortName(objectName).Count > 0)
+                {
+                    continue;
+                }

-            //Cases where def is empty,
-            //so we look for a declarator among the references of target
-            if(declar.Count == 0)
-            {
-                declar = declarators.FindAllReferences(target);
```

```
-           }
-
-           CxList memberAccess = target * unknownReferences;
-
-           bool foundSignature = false;
-           string name = "";
-           string objectName = "";
-
-           if (methObject is ObjectCreateExpr) {
-               foundSignature = true;
-               name = methodName;
-               // In case of ctor, method and class names match
-               objectName = methodName;
-           }
-           else if (objectCreate.Count > 0)
-           {
-               // When refering to Object Create Expression
-               foundSignature = true;
-               name = objectCreate.GetName();
-               objectName = objectCreate.GetName();
-           }
-           else if (typeRef.Count > 0)
-           {
-               // When refering to Type Ref
-               foundSignature = true;
-               name = typeRef.GetName();
-               objectName = typeRef.GetName();
-
-           }
-           else if (declar.Count > 0)
-           {
-               // When refering to Declarators
-               Declarator declarator = declar.TryGetCSharpGraph<Declarator>();
-               LinePragma lpDecl = declarator.LinePragma;
-
-               CxList typeName = relevantTypes.FindByPosition(lpDecl.FileName, lpDecl.Line);
-
-               if (typeName.Count > 0)
-               {
-                   foundSignature = true;
-                   name = typeName.GetName();
-                   objectName = typeName.GetName();
-                   typeRef = typeName;
-               }
-           }
-           else if (memberAccess.Count > 0)
-           {
-               // When refering to member accesses
```

```
-               def = declarators.FindDefinition(memberAccess);
-               if (def.Count == 0)
-               {
-                   objectName = memberAccess.GetName();
-               }
-           }
-           else if (target.FindByType<CastExpr>().Count > 0)
-           {
-               // When refering to cast expression
-               CxList relevantCast = All.NewCxList();
-               CastExpr castExpr = target.TryGetCSharpGraph<CastExpr>();
-               relevantCast.Add(castExpr.TargetType.NodeId, castExpr.TargetType);
-               objectName = relevantCast.GetName();
-           }
-           else if ((target * methodToFor).Count > 0)
-           {
-               //When refering to method invoke expression
-               CxList classMethod = classDecls.GetClass(target);
-               objectName = classMethod.GetName();
-           }
-           else if(target.FindByTypes(typeof(BaseRef), typeof(ThisRef)).Count > 0 )
-           {
-               //When refering to this or super keywords
-               objectName = target.GetName();
-           }
-
-           if (objectName.StartsWith("CxOrphanClass_"))
-           {
-               objectName = "object";
-           }
-
-           if (objectName == "String" && stringMethods.Contains(method.GetName()))
-           {
-               continue;
-           }
-               //Remove type parameters of Java Collections like List, HashMap, etc
-           else if ((typeRef.GetFathers() * javaCollections).Count == 1)
-           {
-               continue;
-           }
-           else if (javaCollections.FindByShortName(objectName).Count > 0)
-           {
-               continue;
-           }
-
-           Comment comment = new Comment(methodName, methodName, rootComment, lp);
-           comment.ResolveShortName(objectName + "||." + methodName + paramsStr);
-           result.Add(comment.NodeId, comment);
```

```
-       }catch(Exception ex){}
+               string fullFilename = lp.FileName.Remove(0, lastIndexOfBackSlash);
+               char pathSeparator = cxEnv.Path.DirectorySeparatorChar;
+               string fileName = pathSeparator.Equals('\\') ? Regex.Replace(fullFilename, @"\\", "/") : fullFilename;
+
+               methodInfoFinal += " \n { "
+                   + "\n \"fullname\": " + "\"" + objectName + "||." + methodName + "\""
+                   + ", \n \"shortname\": " + "\"" + methodName + "\""
+                   + ", \n \"parameters\": " + "\"" + paramsStr + "\""
+                   + ", \n \"line\": " + lp.Line
+                   + ", \n \"column\": " + lp.Column
+                   + ", \n \"sourcefile\": " + "\"" + fileName + "\""
+                   + "\n }, ";
+
+               i++;
+
+           }catch(Exception ex){}
+       }
+   }
+   if(i > 0){
+       methodInfoFinal += "\n ] \n } \n }";
+       methodInfoFinal = new System.Text.StringBuilder(methodInfoFinal).ToString();
+
+       Comment comment = new Comment(methodInfoFinal, methodInfoFinal, rootComment, methodsByName.GetFirstGraph().LinePragma);
+       comment.ResolveShortName(methodInfoFinal);
+
+       result.Add(comment.NodeId, comment);
     }
  }
```

Java / Java_High_Risk / Reflected_XSS_All_Clients

Code changes

```
---
+++
@@ -5,11 +5,12 @@
 CxList inputs = Find_Interactive_Inputs();
 inputs -= Find_Properties_Input();

-CxList findAttrMembers = methods.FindByMemberAccess("pagecontext.findAttribute").GetMembersOfTarget();
+CxList findAttr = methods.FindByMemberAccess("pagecontext.findAttribute");
+CxList findAttrMembers = findAttr.GetMembersOfTarget();
 CxList relevantMethods = methods.FindByMemberAccess("request.getHeader");
 relevantMethods.Add(findAttrMembers);
 inputs.Add(All.GetParameters(relevantMethods, 0),
-           findAttrMembers);
+   findAttrMembers);

 // Remove argc/argv from inputs
```

```
 inputs -= All.GetParameters(methodDecls.FindByName("*main*"));
@@ -20,6 +21,17 @@
 inputs -= All.InfluencedBy(openStreamStoredInput).GetLastNodesInPath();


 CxList outputs = Find_XSS_Outputs();
+
+CxList outputsInForEachCondition = outputs.FindByFathers(Find_ForEachStmt());
+outputs -= outputsInForEachCondition;
+outputsInForEachCondition = outputsInForEachCondition.GetFathers();
+CxList varDeclsInForEach = Find_Declarators().GetByAncs(Find_VariableDeclStmt().FindByFathers(outputsInForEachCondition));
+
+CxList conditions = Find_Ifs().CxSelectDomProperty<IfStmt>(x => x.Condition);
+CxList varDeclsRefs = unkRefs.FindAllReferences(varDeclsInForEach);
+varDeclsRefs-= varDeclsRefs.GetByAncs(conditions);
+
+outputs.Add(varDeclsRefs.GetMembersOfTarget());


 // Add outputs from EL expressions (e.g. "${param.field}" -> "request.getParameter("field")") in JSP files
 CxList jspOutputs = Find_Jsp_Code().FindByMemberAccesses(new string[]{"request.getParameter", "request.getHeader"});
```

Java / Java_High_Risk / Stored_XSS

Code changes

```
---
+++
@@ -9,14 +9,11 @@
 // Remove Properties as they are considered potential inputs and are handled by the Potential_Stored_XSS query
 read -= read.FindByMemberAccess("Properties.getProperty");


-CxList getRequestSessionMethods = Find_GET_Request_Session_Methods();
-
 CxList inputs = All.NewCxList(
    Find_DB_Out(),
    read,
    Find_Vulnerable_Nio_Files_Methods(),
    Find_Vulnerable_Io_File_Methods(),
-   getRequestSessionMethods,
    Find_Cloud_Storage_In());


 CxList outputs = Find_XSS_Outputs();
```

Java / Java_Low_Visibility / Log_Forging

Code changes

```
---
+++
@@ -9,7 +9,7 @@
 inputs -= Find_Readline_From_Stored();


 /* Log Outputs */
```

```
-CxList log = Find_Log_Outputs();
+CxList log = Find_Log_Outputs() - Find_Secure_SLF4J_Loggers();
 CxList logParams = All.GetParameters(log);

 /* Sanitize */
```

Java / Java_Low_Visibility / Trust_Boundary_Violation_in_Session_Variables

Code changes

```
---
+++
@@ -1,24 +1,28 @@
+CxList methods = Find_Methods();
+
 CxList input = Find_Interactive_Inputs();

-CxList setAttr = Find_Implicit_Object_Members().FindByMemberAccesses(new string[]{"Session", "HttpSession"}, new string[]{"setAttribute", "putValue"});
+CxList setAttr = Find_Implicit_Object_Members().FindByMemberAccesses(
+   new string[]{"Session", "HttpSession"}, new string[]{"setAttribute", "putValue"});

-CxList invocations = Find_Methods();
-string[] relevantNames = new string[] {
-   "*session.setAttribute",
-   "*session.putValue"
-   };
-
-setAttr.Add(invocations.FindByMemberAccesses(new string[] { "session.putValue" }),
-   invocations.FindByNames(relevantNames, false));
+setAttr.Add(methods.FindByMemberAccesses(new string[] { "session.putValue" }),
+   methods.FindByNames(new string[] {"*session.setAttribute", "*session.putValue"}, false));

 CxList setAttrParams = All.GetParameters(setAttr);

-CxList sanitizers = Find_General_Sanitize();
-sanitizers.Add(Trust_Boundary_Violation_Session_Sanitize(), Trust_Boundary_Violation_Context_Sanitize());
+CxList sanitizers = All.NewCxList(
+   Find_General_Sanitize(),
+   Trust_Boundary_Violation_Session_Sanitize(),
+   Trust_Boundary_Violation_Context_Sanitize());

-result = setAttrParams.InfluencedByAndNotSanitized(input, sanitizers);
-result = result.ReduceFlow(CxList.ReduceFlowType.ReduceSmallFlow);
+result = setAttrParams.InfluencedByAndNotSanitized(input, sanitizers).ReduceFlow(CxList.ReduceFlowType.ReduceSmallFlow);

 // When the flow goes through a "Remote input" (a request to a remote server) it is not vulnerable
 CxList remoteInputs = All.NewCxList(Find_Remote_ReadMethods(), Find_Remote_Connections());
-result -= result.IntersectWithNodes(remoteInputs);
+
+// Stored inputs in the middle of flow break said flow, no longer being a good result
```

```
+CxList storedInputs = All.NewCxList(Find_Read());
+
+CxList flowVulnerabilitySanitizers = All.NewCxList(remoteInputs, storedInputs);
+
+result -= result.IntersectWithNodes(flowVulnerabilitySanitizers);
```

Java / Java_Low_Visibility / Use_Of_Hardcoded_Password

Code changes

```
---
+++
@@ -5,6 +5,7 @@
 CxList passwordString = Find_Password_Strings();
 CxList methods = Find_Methods();

+CxList paramValue = Find_Param().CxSelectDomProperty<Param>(p => p.Value);
 CxList passAndStrings = All.NewCxList(passwordString, psw);

 // Find password in an initialization operation
@@ -60,9 +61,9 @@
 assignPassword = litInRSide.GetByAncs(assignPassword);

 CxList connection = methods.FindByShortName("getConnection");
-CxList connetionParam2 = All.GetParameters(connection, 2);
+CxList connetionParam2 = paramValue.GetParameters(connection, 2);
-CxList connetionParam0 = All.GetParameters(connection, 0);
+CxList connetionParam0 = paramValue.GetParameters(connection, 0);
 CxList pwdInFirstConnectionParam = connetionParam0.FindByType<StringLiteral>().FindByName("*PWD=*");
 pwdInFirstConnectionParam.Add(connetionParam0.FindByType<StringLiteral>().FindByName("*PWD =*"));

@@ -72,14 +73,14 @@
 CxList kerberosKey = kerberosKeyType.FindByTypes(typeof(ObjectCreateExpr), typeof(Declarator));

 // Get second parameter
-CxList KerberosKeyParam1 = All.GetParameters(kerberosKey, 1);
+CxList KerberosKeyParam1 = paramValue.GetParameters(kerberosKey, 1);

 // Add also KerberosPrincipal's second parameter as a potentially vulnerable hardcoded parameter
 CxList passwordAuthenticationType = All.FindByType("PasswordAuthentication");
 CxList passwordAuthentication = passwordAuthenticationType.FindByTypes(typeof(ObjectCreateExpr), typeof(Declarator));

 // Get second parameter
-CxList PasswordAuthenticationParam1 = All.GetParameters(passwordAuthentication, 1);
+CxList PasswordAuthenticationParam1 = paramValue.GetParameters(passwordAuthentication, 1);

 CxList relevantParams = All.NewCxList(KerberosKeyParam1, connetionParam2, PasswordAuthenticationParam1);

@@ -100,28 +101,9 @@
```

```
 CxList passwordParams = stringLiterals.GetParameters(setPasswordMethod);
 CxList hardcodedPasswordInMethod = setPasswordMethod.DataInfluencedBy(passwordParams);


-//Enum members with hardcoded password
-CxList enumMembers = Find_EnumMemberDecl();
-CxList literalsInEnum = strLiterals.FindByFathers(enumMembers);
-
-//Case 1:
-//public enum Credentials {
-//USER("admin:password"), WHITE_LIST("user:password");
-//}
-CxList passwordInEnum = literalsInEnum * passwordString;
-passwordInEnum = passwordInEnum.GetFathers();
-
-//Case 2:
-//public enum Credentials {
-//PASSWORD("123456"), PWD("user:password");
-//}
-CxList enumWithLiterals = strLiterals.GetFathers() * enumMembers;
-passwordInEnum.Add(enumWithLiterals * psw);
-
 // Android password in SharedPreferences
 CxList putString = methods.FindByMemberAccess("Editor.putString");
-CxList putStringParam0 = All.GetParameters(putString, 0);
-
+CxList putStringParam0 = paramValue.GetParameters(putString, 0);

 CxList StringParam0AndPass = (putStringParam0 * passwordString);

@@ -129,7 +111,7 @@
 putStringParam0Paswd.Add(StringParam0AndPass);

 CxList relevantPutString = putStringParam0Paswd.GetAncOfType<MethodInvokeExpr>() * putString;
-CxList passwordValueInPreference = All.GetParameters(relevantPutString, 1);
+CxList passwordValueInPreference = paramValue.GetParameters(relevantPutString, 1);

 CxList hardcodedPasswordString = passwordValueInPreference.FindByType<StringLiteral>();

@@ -149,7 +131,6 @@
             paramsAffectedByString,
             hardcodedPasswordInMethod,
             pwdInFirstConnectionParam,
-            passwordInEnum,
             Password_In_Credentails(),
             passwordInPreferences);
```

Java / Java_Medium_Threat / Cleartext_Submission_of_Sensitive_Information

Code changes

```
---
+++
@@ -6,11 +6,12 @@
 // and not really sensitive information.
 string[] integersToNotExclude = new string[]{"*salary*", "*ccnlimit*"};
 CxList strings = Find_Strings();
+CxList methods = Find_Methods();

 //A ResourceBundle can access its stored data by passing a key in the ResourceBundle.getString("key") method
 //Since the key is a string literal, we are removing these cases from all the strings
 CxList personalInfoStrings = personalInfo.FindByType<StringLiteral>();
-CxList getStringMethods = Find_Methods().FindByShortName("getString");
+CxList getStringMethods = methods.FindByShortName("getString");
 CxList resourceBundleKeys = personalInfoStrings.GetByAncs(getStringMethods);
 strings -= resourceBundleKeys;

@@ -27,16 +28,14 @@

 personalInfo -= assignedNull;

-bool isUsingSSL = Framework_Is_Using_SSL().Count > 0;
-
-if (!isUsingSSL)
-{
-  result = Find_Web_Cleartext_Submission_of_Sensitive_Information(personalInfo);
-}
-
 result.Add(Find_Server_to_Server_Cleartext_Submission_of_Sensitive_Information(personalInfo));

 // Add hash functions as sanitizer
 CxList sanitize = Find_Hash_Strings();
 sanitize.Add(Find_HashSanitize());
+
+//Add signWith() params as sanitizer
+CxList signWithMethods = methods.FindByShortName("signWith");
+sanitize.Add(personalInfo.GetParameters(signWithMethods));
+
 result -= result.IntersectWithNodes(sanitize);
```

Java / Java_Medium_Threat / Excessive_Data_Exposure

Code changes

```
---
+++
@@ -22,6 +22,11 @@
 sensitive -= sensitive.GetMembersOfTarget().GetTargetOfMembers();
 sensitive -= sensitive.GetByAncs(Find_Conditions());

+//Remove signWith() params
```

```
+CxList methods = Find_Methods();
+CxList signWithMethods = methods.FindByShortName("signWith");
+sensitive -= sensitive.GetParameters(signWithMethods);
+
 //Get every method output with an "mapping" annotation
 CxList methodsWithAnnotation = Find_Spring_Inputs_Annotations().GetAncOfType<MethodDecl>();
 CxList outputs = All.FindByFathers(Find_ReturnStmt().GetByAncs(methodsWithAnnotation));
@@ -30,7 +35,7 @@
 outputs.Add(Find_API_Response_Outputs(), Find_Write());

 CxList outputStmts = outputs.GetAncOfType<ExprStmt>();
-outputStmts.Add(outputs.GetAncOfType<ReturnStmt>(), Find_Methods().FindByShortName("writeValueAsString"));
+outputStmts.Add(outputs.GetAncOfType<ReturnStmt>(), methods.FindByShortName("writeValueAsString"));

 //Get methods that are sanitized by rules\filters\authorizations annotations
 CxList sanitizers = Find_Spring_Security_Annotations();
```

Java / Java_Medium_Threat / Improper_Restriction_of_Stored_XXE_Ref

Code changes

```
---
+++
@@ -1,4 +1,5 @@
 // Find Stored_XXE (XML External Entity vulnerability) in Java
+CxList ifs = Find_Ifs();
 CxList inputs = Find_Read_NonDB();
 inputs.Add(Find_DB_Out());
@@ -7,10 +8,34 @@
 CxList xxe = Find_XXE_Requests();

 CxList sanitizers = Find_XXE_Sanitize();
-// To remove factories influenced by sanitizers
-sanitizers.Add(All.FindAllReferences(sanitizers).GetMembersOfTarget());
 // remove targets influenced by sanitizers
-xxe -= xxe.GetTargetOfMembers().InfluencedBy(sanitizers).GetMembersOfTarget();
+CxList sanitizersNotIfs = sanitizers - sanitizers.GetByAncs(ifs);
+sanitizers.Add(All.FindAllReferences(sanitizersNotIfs).GetMembersOfTarget());
+xxe -= xxe.GetTargetOfMembers().InfluencedBy(sanitizersNotIfs).GetMembersOfTarget();
+
+//Handle sanitization inside if statements
+foreach(CxList ifStmt in ifs){
+   CxList trueStmts = ifStmt.CxSelectDomProperty<IfStmt>(x => x.TrueStatements);
+
+   CxList trueStmtSanitizers = sanitizers.GetByAncs(trueStmts);
+   CxList trueStmtXxe = xxe.GetByAncs(trueStmts);
+   xxe -= trueStmtXxe.GetTargetOfMembers().InfluencedBy(trueStmtSanitizers).GetMembersOfTarget();
+   sanitizers.Add(All.FindAllReferences(trueStmtSanitizers).GetMembersOfTarget().GetByAncs(trueStmts));
+
```

```
+   /* Skip else if statements when it comes being false statements
+      The logic for else ifs will eventually be done in the loop using the true statements
+   */
+   if(ifStmt.FilterByDomProperty<IfStmt>(x => x.FalseStatements.Count == 1)
+       .CxSelectDomProperty<IfStmt>(x => x.FalseStatements[0]).Count == 1)
+       continue;
+
+   //The following logic should only be done for else statements
+   CxList falseStmts = ifStmt.CxSelectDomProperty<IfStmt>(x => x.FalseStatements);
+   CxList falseStmtSanitizers = sanitizers.GetByAncs(falseStmts);
+   CxList falseStmtXxe = xxe.GetByAncs(falseStmts);
+   xxe -= falseStmtXxe.GetTargetOfMembers().InfluencedBy(falseStmtSanitizers).GetMembersOfTarget();
+   sanitizers.Add(All.FindAllReferences(falseStmtSanitizers).GetMembersOfTarget().GetByAncs(falseStmts));
+}


 // All integers are sanitizers with the exception of bytes because they are used to read XML
 sanitizers.Add(Find_Integers());
```

Java / Java_Medium_Threat / Improper_Restriction_of_XXE_Ref

Code changes

```
---
+++
@@ -1,13 +1,12 @@
 // Find XXE (XML External Entity vulnerability) in Java
 CxList methods = Find_Methods();
+CxList ifs = Find_Ifs();
 CxList inputs = All.NewCxList(Find_Interactive_Inputs(), Find_Cloud_Storage_In());


 CxList xxe = Find_XXE_Requests();
 xxe.Add(Find_XXE_Spring());


 CxList sanitizers = Find_XXE_Sanitize();
-// To remove factories influenced by sanitizers
-sanitizers.Add(All.FindAllReferences(sanitizers).GetMembersOfTarget());


 CxList transformMethods = methods.FindByMemberAccess("Transformer.transform");
 sanitizers.Add(All.GetParameters(transformMethods, 1));
@@ -16,11 +15,37 @@
 sanitizers.Add(
     Find_Read() - methods.FindByMemberAccess("SAXReader.read"),
     Find_ObjectCreations().FindByShortName("File*"),
-    // To remove all DB Outs
+    // To remove all DB Outs
     Find_DB_Out());


 // remove targets influenced by sanitizers
-xxe -= xxe.GetTargetOfMembers().InfluencedBy(sanitizers).GetMembersOfTarget();
+CxList sanitizersNotIfs = sanitizers - sanitizers.GetByAncs(ifs);
```

```
+sanitizers.Add(All.FindAllReferences(sanitizersNotIfs).GetMembersOfTarget());
+xxe -= xxe.GetTargetOfMembers().InfluencedBy(sanitizersNotIfs).GetMembersOfTarget();
+
+//Handle sanitization inside if statements
+foreach(CxList ifStmt in ifs){
+   CxList trueStmts = ifStmt.CxSelectDomProperty<IfStmt>(x => x.TrueStatements);
+
+   CxList trueStmtSanitizers = sanitizers.GetByAncs(trueStmts);
+   CxList trueStmtXxe = xxe.GetByAncs(trueStmts);
+   xxe -= trueStmtXxe.GetTargetOfMembers().InfluencedBy(trueStmtSanitizers).GetMembersOfTarget();
+   sanitizers.Add(All.FindAllReferences(trueStmtSanitizers).GetMembersOfTarget().GetByAncs(trueStmts));
+
+   /* Skip else if statements when it comes being false statements
+      The logic for else ifs will eventually be done in the loop using the true statements
+   */
+   if(ifStmt.FilterByDomProperty<IfStmt>(x => x.FalseStatements.Count == 1)
+       .CxSelectDomProperty<IfStmt>(x => x.FalseStatements[0]).Count == 1)
+       continue;
+
+   //The following logic should only be done for else statements
+   CxList falseStmts = ifStmt.CxSelectDomProperty<IfStmt>(x => x.FalseStatements);
+   CxList falseStmtSanitizers = sanitizers.GetByAncs(falseStmts);
+   CxList falseStmtXxe = xxe.GetByAncs(falseStmts);
+   xxe -= falseStmtXxe.GetTargetOfMembers().InfluencedBy(falseStmtSanitizers).GetMembersOfTarget();
+   sanitizers.Add(All.FindAllReferences(falseStmtSanitizers).GetMembersOfTarget().GetByAncs(falseStmts));
+}


 // All integers are sanitizers with the exception of bytes because they are used to read XML
 sanitizers.Add(Find_Integers());
```

Java / Java_Medium_Threat / JWT_Lack_Of_Expiration_Time

Code changes

```
---
+++
@@ -1,7 +1,12 @@
 CxList methods = Find_Methods();
 CxList jwtsBuildRef = methods.FindByMemberAccess("Jwts.builder");
+
 CxList sanitizedMethod = methods.FindByMemberAccess("JwtBuilder.setExpiration").GetTargetOfMembers();
 sanitizedMethod.Add(methods.FindByShortName("setExpiration"));

+//Add setClaims() calls influenced by a Claim with setExpiration() to the sanitizers
+CxList setClaimsMethods = methods.FindByShortName("setClaims");
+sanitizedMethod.Add(setClaimsMethods.InfluencedBy(methods.FindByShortName("setExpiration").GetTargetOfMembers()));
+
 CxList lastNodeFlowJwts = methods.FindByShortName("compact");
 result = lastNodeFlowJwts.InfluencedByAndNotSanitized(jwtsBuildRef, sanitizedMethod);
```

Java / Java_Medium_Threat / JWT_No_Signature_Verification

Code changes

```
---
+++
@@ -1,15 +1,5 @@
 CxList methods = Find_Methods();
-CxList methodDecls = Find_MethodDecls();


 CxList jwtsParserRef = methods.FindByMemberAccess("Jwts.parser").GetTargetOfMembers();
 CxList parseMethod = methods.FindByShortName("parse");
 result = parseMethod.InfluencedBy(jwtsParserRef);
-
-// Check returns influenced by inputs in resolveSigningKeyBytes
-CxList resolveMethod = methodDecls.FindByShortName("resolveSigningKeyBytes");
-CxList inputs = methods.FindByMemberAccess("JwsHeader.get");
-CxList returns = Find_ReturnStmt().GetByAncs(resolveMethod);
-
-CxList taintedReturns = returns.CxSelectDomProperty<ReturnStmt>(x => x.Expression).DataInfluencedBy(inputs);
-// Connect returned item to the method declaration
-result.Add(taintedReturns.ConcatenatePath(resolveMethod.GetMethod(taintedReturns)));
```

Java / Java_Medium_Threat / JWT_Sensitive_Information_Exposure

Code changes

```
---
+++
@@ -1,8 +1,17 @@
-string[] sinkNames = new string[] { "setClaims", "claim", "addClaims", "setPayload" };
-
+// This query should find PII, passwords, secrets being included in JWTs claims.
 CxList methods = Find_Methods();
 CxList jwtsBuildRef = methods.FindByMemberAccess("Jwts.builder");
-CxList passwords = Find_All_Passwords();
+CxList sensitiveInfo = All.NewCxList(Find_Personal_Info(), Find_Passwords_Unsafe());
+
+// Sinks for jjwt
+string[] sinkNames = new string[] { "setClaims", "claim", "addClaims", "setPayload" };
 CxList sinks = methods.FindByShortNames(sinkNames).InfluencedBy(jwtsBuildRef).GetLastNodesInPath();


-result = sinks.InfluencedBy(passwords).ReduceFlow(CxList.ReduceFlowType.ReduceSmallFlow);
+result = sinks.InfluencedBy(sensitiveInfo).ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
+
+// If the input is from the second argument of "signWith" (method that tells the JWT builder
+// how to sign the token) it is not vulnerable
+CxList invalid = jwtsBuildRef.GetMembersOfTarget().FindByShortName("signWith");
+invalid = Find_By_Parameter_Position(invalid, 1, sensitiveInfo);
+
+result -= result.IntersectWithNodes(invalid);
```

Java / Java_Medium_Threat / JWT_Use_Of_Hardcoded_Secret

Code changes

```
---
+++
@@ -1,25 +1,26 @@
 CxList methods = Find_Methods();
+CxList paramValue = Find_Param().CxSelectDomProperty<Param>(p => p.Value);

 CxList jwtsRef = methods.FindByMemberAccess("Jwts.builder").GetTargetOfMembers();
 CxList signWithMethod = methods.FindByShortName("signWith");
 CxList refSignWithMethod = signWithMethod.InfluencedBy(jwtsRef);

 CxList signWithMet = refSignWithMethod.FindByShortName("signWith");
-CxList signWithParam = All.GetParameters(signWithMet);
+CxList signWithParam = paramValue.GetParameters(signWithMet);
 CxList parameterInMethod = signWithParam.FindByType("Key");

-CxList flows = parameterInMethod.DataInfluencedBy(Find_Strings())
-    .ReduceFlow(CxList.ReduceFlowType.ReduceSmallFlow);
+CxList flows = parameterInMethod.DataInfluencedBy(Find_Strings());

 flows.Add(signWithParam.FindByType<StringLiteral>());

 CxList attributesValue = Create_Flow_Spring_CustomAttribute();

 // Sanitize flows were JWT is not being signed through signWith method
-attributesValue -= attributesValue.SanitizeCxList(All.GetParameters(signWithMet, 1).Contained(attributesValue, CxList.GetStartEndNodesType.AllNodes));
+attributesValue -= attributesValue.SanitizeCxList(paramValue.GetParameters(signWithMet, 1)
+    .Contained(attributesValue, CxList.GetStartEndNodesType.AllNodes));

 CxList sanitizedMethods = methods.FindByMemberAccess("Cipher.getInstance");
 attributesValue -= attributesValue.IntersectWithNodes(sanitizedMethods);

 flows.Add(attributesValue);
-result = flows;
+result = flows.ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
```

Java / Java_Medium_Threat / Privacy_Violation

Code changes

```
---
+++
@@ -89,7 +89,7 @@
     exceptions,
     exceptionsCtorsWithSuper,
     Find_Cloud_Outputs()
-);
+    );
```

```
 // Define sanitize
 CxList sanitize = All.NewCxList(Find_DB(), Find_Encrypt(), Find_UnitTest_Code(), Find_HashSanitize());
@@ -97,7 +97,7 @@
 // Add additional "integer" sanitizers
 sanitize.Add(All.FindByShortNames(new string[] {"size", "length", "Index*", "indexOf"}, false),
    All.FindByName("*boolean.class.cast", StringComparison.OrdinalIgnoreCase),
-   methods.FindByMemberAccess("Boolean.parse*"));
+   methods.FindByMemberAccess("Boolean.parse*"), methods.FindByReturnType("bool"), Find_BooleanLiteral());

 CxList javaSqlMethods = methods.FindByMemberAccess("DriverManager.getConnection");
 CxList connections = unknRefs.FindAllReferences(javaSqlMethods.GetAssignee());
```

Java / Java_Spring / Spring_Use_Of_Hardcoded_Password

Code changes

```
---
+++
@@ -5,6 +5,8 @@

 // Passwords in @Value
 CxList defaultValuePasswords = Find_CustomAttribute().FindByShortName("Value")
-   .FindByFathers(Find_All_Passwords()).GetFathers();
-
+   .FindByFathers(Find_All_Passwords()).GetFathers();
+// Excluding cases where the password is a variable in a configuration property.
+defaultValuePasswords -= defaultValuePasswords.GetByAncs(Find_TypeRef().FindByFathers(Find_Field_Decl()).GetFathers());
+
 result.Add(webSecurityCfgPasswords, defaultValuePasswords);
```

JavaScript / JavaScript_Exploitable_Path / JavaScript_Find_UnresolvedMethods

Code changes

```
---
+++
@@ -1,3 +1,4 @@
+Comment rootComment = new Comment();
 Func < CSharpGraph, string > GetTypeName = ((CSharpGraph graph) => {
    if(graph is BooleanLiteral) return "boolean";
    if(graph is CharLiteral) return "char";
@@ -98,7 +99,7 @@
    if (require.Count == 1)
    {
        Import requirePkg = require.TryGetCSharpGraph<Import>();
-       return $"package:{requirePkg.ImportedFilename}||";
+       return $"{requirePkg.ImportedFilename}";
    }

    return null;
@@ -114,65 +115,103 @@
 CxList thirdPartyMethods = GetThirdPartyMethods();
```

```
 methods -= thirdPartyMethods;


-foreach(CxList method in methods){
-   if(All.FindDefinition(method).Count == 0){
-       Comment rootComment = new Comment();
-       CxList target = method.GetTargetOfMembers();
-       CxList def = All.FindDefinition(target);
-
-       Expression methObject = method.TryGetCSharpGraph<Expression>();
-       LinePragma lp = methObject.LinePragma;
-       string methodName = method.GetName();
-
-       CxList methodParams = allParams.GetParameters(method);
-       var elements = methodParams.CxSelectDomProperty<Param>(_ => _.Value)
-           .CxSelectElementValue<CSharpGraph,string>(GetTypeName);
-
-       string paramsStr = "||(" + string.Join("||,", elements) + "||)";
-
-       CxList typeRef = typeRefs.GetByAncs(def);
-       CxList declar = declarators.GetByAncs(def);
-
-       bool foundSignature = false;
-       string name = "";
-       string objectName = "";
-       if (methObject is ObjectCreateExpr) {
-           foundSignature = true;
-           name = methodName;
-           // In case of ctor, method and class names match
-           objectName = methodName;
+//Get full projectPath to remove it from filename
+string projectPath = cxScan.GetScanProperty("projectPath");
+int lastIndexOfBackSlash = projectPath.LastIndexOf(cxEnv.Path.DirectorySeparatorChar);
+Match guid = Regex.Match(projectPath, @"(?im)[{(]?[0-9A-F]{8}[-]?(?:[0-9A-F]{4}[-]?){3}[0-9A-F]{12}[)}]?$");
+if(guid.Success) lastIndexOfBackSlash = projectPath.IndexOf(guid.Value) + guid.Value.Length;
+
+List<string> names = methods.CxSelectElementValues<Expression, string>(x => x.ShortName);
+
+//For each name
+foreach(String str in names.Distinct()){
+
+   int i = 0;
+   CxList methodsByName = methods.FindByShortName(str);
+   string auxStr = str.Replace("\"", "");
+   string methodInfoFinal = "{ \n \"method\": {\"name\":" + "\"" + auxStr + "\", \n \"calls\": [";
+
+   foreach(CxList method in methodsByName){
+       if(All.FindDefinition(method).Count == 0){
+           CxList target = method.GetTargetOfMembers();
```

```
+           CxList def = All.FindDefinition(target);
+
+           Expression methObject = method.TryGetCSharpGraph<Expression>();
+           LinePragma lp = methObject.LinePragma;
+           string methodName = method.GetName();
+
+           CxList methodParams = allParams.GetParameters(method);
+           var elements = methodParams.CxSelectDomProperty<Param>(_ => _.Value)
+               .CxSelectElementValue<CSharpGraph,string>(GetTypeName);
+
+           string paramsStr = string.Join(", ", elements);
+
+           CxList typeRef = typeRefs.GetByAncs(def);
+           CxList declar = declarators.GetByAncs(def);
+
+           bool foundSignature = false;
+           string name = "";
+           string objectName = "";
+           if (methObject is ObjectCreateExpr) {
+               foundSignature = true;
+               name = methodName;
+               // In case of ctor, method and class names match
+               objectName = methodName;
+           }
+           else if (typeRef.Count > 0)
+           {
+               // When refering to Type Ref
+               foundSignature = true;
+               name = typeRef.GetName();
+               objectName = typeRef.GetName();
+           }
+           else if (declar.Count > 0)
+           {
+               Declarator declarator = declar.TryGetCSharpGraph<Declarator>();
+               LinePragma lpDecl = declarator.LinePragma;
+
+               CxList typeName = typeRefs.FindByPosition(lpDecl.FileName, lpDecl.Line);
+
+               typeName -= typeName.FindByShortName("?");
+               if (typeName.Count > 0)
+               {
+                   foundSignature = true;
+                   name = typeName.GetName();
+                   objectName = typeName.GetName();
+               }
+           }
+
+           if (objectName.StartsWith("CxOrphanClass_")) {
```

```
+            objectName = "object";
+        }
+
+        string namePrefix = foundSignature ? ((GetPackageName(target) ?? "")) : "";
+
+        string fullFilename = lp.FileName.Remove(0, lastIndexOfBackSlash);
+        char pathSeparator = cxEnv.Path.DirectorySeparatorChar;
+        string fileName = pathSeparator.Equals('\\') ? Regex.Replace(fullFilename, @"\\", "/") : fullFilename;
+
+        methodInfoFinal += " \n { "
+            + " \n \"package\": " + "\"" + namePrefix + "\""
+            + ", \n \"fullname\": " + "\"" + objectName + "||." + methodName + "\""
+            + ", \n \"shortname\": " + "\"" + methodName + "\""
+            + ", \n \"parameters\": " + "\"" + paramsStr + "\""
+            + ", \n \"line\": " + lp.Line
+            + ", \n \"column\": " + lp.Column
+            + ", \n \"sourcefile\": " + "\"" + fileName + "\""
+            + "\n }, ";
+
+        i++;
    }
-    else if (typeRef.Count > 0)
-    {
-        // When refering to Type Ref
-        foundSignature = true;
-        name = typeRef.GetName();
-        objectName = typeRef.GetName();
-    }
-    else if (declar.Count > 0)
-    {
-        Declarator declarator = declar.TryGetCSharpGraph<Declarator>();
-        LinePragma lpDecl = declarator.LinePragma;
-
-        CxList typeName = typeRefs.FindByPosition(lpDecl.FileName, lpDecl.Line);
-
-        typeName -= typeName.FindByShortName("?");
-        if (typeName.Count > 0)
-        {
-            foundSignature = true;
-            name = typeName.GetName();
-            objectName = typeName.GetName();
-        }
-    }
-
-    if (objectName.StartsWith("CxOrphanClass_")) {
-        objectName = "object";
-    }
-
```

```
-       string namePrefix = foundSignature ? ((GetPackageName(target) ?? "") + objectName + "||.") : "";
-
-       Comment comment = new Comment(methodName, methodName, rootComment, lp);
-       comment.ResolveShortName(namePrefix + methodName + paramsStr);
+   }
+   if(i > 0){
+       methodInfoFinal += "\n ] \n } \n }";
+       methodInfoFinal = methodInfoFinal.Replace(@"$", @"\\$");
+       methodInfoFinal = new System.Text.StringBuilder(methodInfoFinal).ToString();
+
+       Comment comment = new Comment(methodInfoFinal, methodInfoFinal, rootComment, methodsByName.GetFirstGraph().LinePragma);
+       comment.ResolveShortName(methodInfoFinal);
+
        result.Add(comment.NodeId, comment);
    }
 }
```

JavaScript / JavaScript_High_Risk / Client_DOM_XSS

Code changes

```
---
+++
@@ -2,14 +2,15 @@
 CxList thisRef = Find_ThisRef();
 CxList binExprs = Find_BinaryExpr();
 CxList fieldDecls = Find_FieldDecls();
+CxList memberAccesses = Find_MemberAccesses();

-CxList rClass = Find_MemberAccesses().GetByAncs(Find_ClassDecl().InheritsFrom("React.Component"));
+CxList rClass = memberAccesses.GetByAncs(Find_ClassDecl().InheritsFrom("React.Component"));
 CxList propsOutputs = rClass.FindByShortName("props").GetMembersOfTarget();

 CxList outputs = Find_Outputs_XSS();

 // Remove location outputs where a hardcoded prefix is used
-CxList urlOutputs = Find_Outputs_Redirection().FindByShortNames(new List<string>{ "location", "href" });
+CxList urlOutputs = Find_Outputs_Redirection().FindByShortNames(new string[]{ "location", "href" });
 CxList location = urlOutputs.FindByShortName("location");
 CxList locationPrefix = location.GetAssigner().CxSelectDomProperty<BinaryExpr>(x => x.Left).FindByType<StringLiteral>();
 urlOutputs -= locationPrefix.GetAncOfType<BinaryExpr>().GetAssignee();
@@ -19,13 +20,15 @@
 CxList inputs = Find_Inputs_NoWindowLocation();
 inputs.Add(propsOutputs);

-// Exclude inputs that are suffixes in a string only the leftmost input in a string is considered valid to control the scheme.
+// Exclude inputs that are suffixes in a string only the leftmost input
+// in a string is considered valid to control the scheme.
 CxList validLeftMostPrefix = inputs.GetFathers().CxSelectDomProperty<BinaryExpr>(x => x.Left).FindByType<MemberAccess>();
 CxList invalidInputSufix = inputs.FindDescendantsOfType<MemberAccess>(binExprs) - validLeftMostPrefix;
```

```
 // An exception when there is an empty string preceding the vulnerable input, still making it valid to control the scheme.
 CxList emptyStringBinExpr = binExprs.InfluencedBy(Find_Empty_Strings());
-CxList validInputAfterEmptyString = emptyStringBinExpr.CxSelectDomProperty<BinaryExpr>(x => x.Right).FindByType<MemberAccess>();
+CxList validInputAfterEmptyString = emptyStringBinExpr.CxSelectDomProperty<BinaryExpr>(x => x.Right)
+    .FindByType<MemberAccess>();
 validInputAfterEmptyString = validInputAfterEmptyString * inputs;

 // Another exception, when there is a html heading like '<h1>' + input + '</h1>', it is still vulnerable
@@ -38,6 +41,19 @@

 inputs -= invalidInputSufix;
 inputs.Add(validInputAfterEmptyString, rightInputBetweenHtmlTag);
+
+inputs -= Find_Potential_Inputs();
+
+// The input in viewInputStmt is susceptible to DOM XSS only when the source code includes a desanitizer.
+// In other words, the input becomes a concern only if there are Angular Sanitizer Bypasses present in the code.
+if(cxScan.IsFrameworkActive("Angular") || Find_Import().FindByName("@angular/*").Count > 0) {
+    CxList viewInputStmt = Find_ViewInputStmt();
+    CxList viewInputs = memberAccesses.FindByFathers(viewInputStmt);
+    inputs -= viewInputs;
+    // If Angular Bypasses affect viewInputs, we consider the input valid and add it back.
+    CxList desanitizedInput = viewInputs.DataInfluencedBy(Find_Angular_Sanitizers_Bypass()).GetLastNodesInPath();
+    inputs.Add(desanitizedInput);
+}

 CxList sanitize = basic_Sanitize();
 sanitize.Add(Find_XSS_Sanitize(),
@@ -53,4 +69,5 @@
 result.Add(Find_Source_Equals_Sink(inputs, outputs));

 result = result.ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
+
 result.Add(AngularJS_Find_DOM_XSS());
```

JavaScript / JavaScript_Medium_Threat / Unchecked_Input_For_Loop_Condition

Code changes

```
---
+++
@@ -62,8 +62,8 @@

 //add to sanitizers when input is checked in an if stmt
 CxList ifStmtConditions = Find_Ifs().CxSelectDomProperty<IfStmt>(_ => _.Condition).FindByType<BinaryExpr>()
-    .FindByShortNames("<", "<=", "==", ">", ">=");
-CxList equalConditions = ifStmtConditions.FindByShortName("==");
+    .FindByShortNames("<", "<=", "==", "===", ">", ">=");
+CxList equalConditions = ifStmtConditions.FindByShortNames("==", "===");
```

```
 CxList conditionsValidValues = All.NewCxList(
    ifStmtConditions.FindByShortNames("<", "<=").CxSelectDomProperty<BinaryExpr>(_ => _.Left),
    ifStmtConditions.FindByShortNames(">", ">=").CxSelectDomProperty<BinaryExpr>(_ => _.Right),
```

JavaScript / JavaScript_Server_Side_Vulnerabilities / Comparing_instead_of_Assigning

Code changes

```
---
+++
@@ -3,7 +3,7 @@
    find all StatementCollection, CatchCollection and ExprStmt that wraps comparisons statements
    return the comparisons that are in those collections
 */
-CxList compare = All.FindByShortName("==");
+CxList compare = All.FindByShortNames("==", "===");

 CxList _toRemove = NodeJS_Find_Conditions_Parameters();
 _toRemove.Add(compare.GetByAncs(Find_Param()));
```

JavaScript / JavaScript_Server_Side_Vulnerabilities / Missing_Encryption_of_Sensitive_Data

Code changes

```
---
+++
@@ -1,36 +1,37 @@
-CxList personalInfo = Find_Personal_Info() - Find_String_Literal();
-CxList sanitizers = NodeJS_Find_Encrypt();
-sanitizers.Add(All.GetParameters(Find_ObjectCreations().FindByShortName("Sequelize")),
-           Find_Integers());
-
-//Remove cases like: (password : hash), where hash was generated by encryption method.
-CxList rightSideValue = All.FindByAssignmentSide(CxList.AssignmentSide.Right).GetByAncs(personalInfo.GetFathers());
-CxList toRemove = sanitizers.DataInfluencingOn(rightSideValue).GetLastNodesInPath();
-toRemove = All.FindByAssignmentSide(CxList.AssignmentSide.Left).GetByAncs(toRemove.GetFathers());
-toRemove.Add(All.FindDefinition(toRemove),
-            All_Passwords());
-personalInfo -= toRemove;
+CxList vulnSeqQueryMethods = Find_SensitiveData_Vuln_Sequelize_Methods();

 CxList nodeJsDbIn = NodeJS_Find_DB_IN();
 CxList storage = All.NewCxList(
    nodeJsDbIn,
    NodeJS_Find_Write(),
    Find_Cloud_Outputs());
+
+// When dealing with Sequelize.query methods,
+// only vulnerable if flow passes trough 1st parameter and query options (2nd param) are set to either INSERT or UPDATE
+CxList sequelizeMethods = nodeJsDbIn.FindByMemberAccess("Sequelize.query");
+storage -= All.NewCxList(sequelizeMethods, storage.GetByAncs(sequelizeMethods));
```

```
-//When dealing with Sequelize.query methods,
-//only vulnerable if flow passes trough 1st parameter and query options (2nd param) are set to either INSERT or UPDATE
-CxList sequelizeMethods = nodeJsDbIn.FindByMemberAccess("Sequelize.query");
-CxList vulnSeqQueryMethods = Find_SensitiveData_Vuln_Sequelize_Methods();
+// If there are no sinks, there is no need to calculate personal info (sources)
+if (vulnSeqQueryMethods.Count == 0 && storage.Count == 0)
+    return All.NewCxList();
+
+CxList personalInfo = Find_Personal_Info() - Find_String_Literal();
+CxList sanitizers = NodeJS_Find_Encrypt();
+sanitizers.Add(All.GetParameters(Find_ObjectCreations().FindByShortName("Sequelize")),
+    Find_Integers());
+
+// Remove cases like: (password : hash), where hash was generated by encryption method.
+CxList rightSideValue = All.FindByAssignmentSide(CxList.AssignmentSide.Right).GetByAncs(personalInfo.GetFathers());
+CxList toRemove = sanitizers.DataInfluencingOn(rightSideValue).GetLastNodesInPath();
+toRemove = All.FindByAssignmentSide(CxList.AssignmentSide.Left).GetByAncs(toRemove.GetFathers());
+toRemove.Add(All.FindDefinition(toRemove), All_Passwords());
+personalInfo -= toRemove;
+
 CxList parameters = Find_Parameters().CxSelectDomProperty<Param>(_ => _.Value);
 CxList seqFirstParam = parameters.GetParameters(vulnSeqQueryMethods, 0);

 result = personalInfo.InfluencingOnAndNotSanitized(vulnSeqQueryMethods, sanitizers).IntersectWithNodes(seqFirstParam);
-CxList toRemoveSequelize = All.NewCxList(
-    sequelizeMethods,
-    All.GetByAncs(sequelizeMethods));
-storage -= toRemoveSequelize;
-
 result.Add(personalInfo.InfluencingOnAndNotSanitized(storage, sanitizers)
     .ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow)
     .ReduceFlowByPragma());
```

JavaScript / JavaScript_Server_Side_Vulnerabilities / MongoDB_NoSQL_Injection

Code changes

```
---
+++
@@ -2,6 +2,7 @@
 CxList methods = Find_Methods();
 CxList outputs = All.NewCxList();
 CxList fields = Find_FieldDecls();
+CxList paramValue = Find_Param().CxSelectDomProperty<Param>(p => p.Value);

 //Add MongoDB
 outputs.Add(NodeJS_MongoDB_Input_Methods(),
@@ -39,7 +40,7 @@
```

```
 // Sanitize regex usages (if a global regex contains "$", we consider it sanitized)
 CxList regExprItems = methods.FindByShortName("RegExp");
-CxList regexParams = All.GetParameters(regExprItems);
+CxList regexParams = paramValue.GetParameters(regExprItems);
 CxList sanitizedRegexes = All.NewCxList();

 foreach (CxList item in regexParams) {
```

JavaScript / JavaScript_Server_Side_Vulnerabilities / Reflected_XSS

Code changes

```
---
+++
@@ -3,6 +3,7 @@
 CxList unknRefs = Find_UnknownReference();
 CxList fieldDecls = Find_FieldDecls();
 CxList declarators = Find_Declarators();
+CxList paramDecls = Find_ParamDecl();
 CxList outputs = NodeJS_Find_Interactive_Outputs();
 outputs.Add(Find_Outputs_XSS());

@@ -38,27 +39,6 @@
 sanitizedFields -= sanitizedFields.FindByShortName("children");
 outputs -= sanitizedFields;

-CxList lambdas = Find_LambdaExpr();
-CxList paramDecls = Find_ParamDecl();
-CxList memberAccess = Find_MemberAccesses();
-CxList activatedRoutes = All.FindByType("ActivatedRoute");
-CxList potencialInputs = memberAccess.GetByAncs(activatedRoutes)
-   .FindByShortNames(new List<string>(){"queryParams", "params"}).GetMembersOfTarget();
-
-CxList potencialServiceSubscriptions = (methods.DataInfluencedBy(potencialInputs))
-   .GetLastNodesInPath();
-CxList callbackParams = paramDecls.GetParameters(lambdas.GetParameters(potencialServiceSubscriptions));
-
-CxList callbackParamsRefs = All.FindAllReferences(callbackParams).GetRightmostMember();
-// In case of IndexerRef
-callbackParamsRefs.Add(callbackParamsRefs.GetFathers().GetMembersOfTarget());
-
-CxList influencedResults = All.DataInfluencedBy(callbackParamsRefs)
-   .ReduceFlow(Checkmarx.DataCollections.CxQueryProvidersInterface.CxList.ReduceFlowType.ReduceSmallFlow)
-   .GetFirstNodesInPath();
-
-inputs.Add(influencedResults);
-
 // If SWIG is not prone to XSS (auto escaped) then remove SWIG outputs.
 if(NodeJS_Find_Swig_Autoescape_False().Count == 0)
 {
```

JavaScript / JavaScript_Server_Side_Vulnerabilities / Use_Of_Hardcoded_Password

Code changes

```
---
+++
@@ -35,10 +35,10 @@
 stringLiterals -= literalsToRemove;

   // Get the crypto parameters to use as sanitizer
-List<string> cryptoParamsToDiscard = new List<string>()
+string[] cryptoParamsToDiscard = new string[]
     { "ascii", "utf8", "utf16le", "ucs2", "base64", "latin1", "binary", "hex" , "md5", "sha1", "SHA1", "sha256" };

-List<string> cryptoFunctions = new List<string>()
+string[] cryptoFunctions = new string[]
     { "digest", "createHash" };

 CxList cryptoParamLiterals = stringLiterals.FindByShortNames(cryptoParamsToDiscard);
@@ -51,7 +51,7 @@
 stringLiterals -= influencedPasswords;

 // Get the delimiter parameters to use as sanitizer
-List<string> stringDelimiterMethods = new List<string>()
+string[] stringDelimiterMethods = new string[]
     { "split","replace", "endsWith", "startsWith", "match", "matchAll", "indexOf" , "lastIndexOf", "localeCompare", "search" };

 CxList delimiterMethods = methods.FindByShortNames(stringDelimiterMethods, false);
@@ -61,13 +61,19 @@
 CxList thenParam = All.GetParameters(methods.FindByShortName("then"));
 thenParam.Add(All.GetParameters(thenParam.FindByType<LambdaExpr>()));

+// add binary expr to sanitizers when password is concatenated to non hardcoded value
+CxList binaryExprs = Find_BinaryExpr().FilterByDomProperty<BinaryExpr>(_ => _.Operator == BinaryOperator.Add);
+CxList safeBinaryExprs = binaryExprs - binaryExprs.FindByAbstractValue(_ => _ is StringAbstractValue);
+safeBinaryExprs.Add(binaryExprs.FindDescendantsOfType<BinaryExpr>(safeBinaryExprs));
+
 // Sanitizers
 CxList sanitizers = All.NewCxList();
 sanitizers.Add(
     integers,
     thenParam,
     cryptoParams,
-    delimiterParams);
+    delimiterParams,
+    safeBinaryExprs);

 // Since 'ParamDecl' has no flow, we search by fathers with the 'StringLiteral'
 CxList passParamDecl = password.FindByType<ParamDecl>();
@@ -79,18 +85,20 @@
```

```
 CxList hardcodedPasswordInBinaryExpr = All.NewCxList();
 hardcodedPasswordInBinaryExpr.Add(
    stringsInBinaryExpr.GetByBinaryOperator(BinaryOperator.IdentityEquality),
-   stringsInBinaryExpr.GetByBinaryOperator(BinaryOperator.IdentityInequality));
+   stringsInBinaryExpr.GetByBinaryOperator(BinaryOperator.IdentityInequality),
+   stringsInBinaryExpr.GetByBinaryOperator(BinaryOperator.StrictEquality),
+   stringsInBinaryExpr.GetByBinaryOperator(BinaryOperator.StrictInequality));
 foreach(CxList binaryExpr in hardcodedPasswordInBinaryExpr)
 {
-   BinaryExpr binary = binaryExpr.TryGetCSharpGraph<BinaryExpr>();
-   if(binary.Left != null && binary.Right != null)
+   //BinaryExpr binary = binaryExpr.TryGetCSharpGraph<BinaryExpr>();
+   CxList binaryLeft = binaryExpr.CxSelectDomProperty<BinaryExpr>(_ => _.Left);
+   CxList binaryRight = binaryExpr.CxSelectDomProperty<BinaryExpr>(_ => _.Right);
+   if(binaryLeft != null && binaryRight != null)
    {
        // Get all the suspected passwords related with the hardcorded strings
        CxList suspectedPassword = All.NewCxList();
-       suspectedPassword.Add(
-           password.FindById(binary.Left.NodeId),
-           password.FindById(binary.Right.NodeId));
-
+       suspectedPassword.Add(All.NewCxList(binaryLeft, binaryRight) * password);
+
        if(suspectedPassword.Count > 0)
        {
            // Add to result the hardcoded passwords in binary expressions
```

JavaScript / JavaScript_Vue / Use_of_Single_Word_Named_Vue_Components

Code changes

```
---
+++
@@ -1,13 +1,10 @@
-Func<string, bool> isComponentNameInvalid = (s) =>
-   {
-   int pascalCaseWordCount = s.Count(c => char.IsUpper(c));
-   int kebabCaseWordCount = s.Split('-').Count();
-   int kebabTransformedWordCount = s.Split('_').Count();
-
-   return(pascalCaseWordCount < 2 && kebabCaseWordCount < 2 && kebabTransformedWordCount < 2);
-   };
+Func<string, bool> isComponentNameInvalid = (name) =>
+{
+   return !name.Substring(1).Any(c => char.IsUpper(c) || c == '-' || c == '_');
+};

-if(cxScan.IsFrameworkActive("VueJS")){
+if(cxScan.IsFrameworkActive("VueJS"))
```

```
+{
     CxList vueComponents = All.NewCxList();
     CxList vueSingleComponents = All.NewCxList();
     CxList vueGlobalComponents = All.NewCxList();
@@ -31,34 +28,40 @@
     CxList componentsWithoutName = vueSingleComponents - nameDeclarations.GetFathers();
     CxList nameValues = Find_String_Literal().GetByAncs(nameDeclarations);

-    foreach(CxList component in nameValues){
-        if(isComponentNameInvalid(component.GetName())){
+    foreach(CxList component in nameValues)
+    {
+        if(isComponentNameInvalid(component.GetName()))
             oneWordComponents.Add(component);
-        }
     }

-    foreach(CxList component in componentsWithoutName){
+    foreach(CxList component in componentsWithoutName)
+    {
         try
         {
             CSharpGraph componentGraph = component.GetFirstGraph();
-            string [] fileNamePath = componentGraph.LinePragma.FileName.Split(cxEnv.Path.DirectorySeparatorChar);
-            string fileName = fileNamePath[fileNamePath.Length - 1];
-
-            if(isComponentNameInvalid(fileName)){
-                oneWordComponents.Add(component);
+            string fileName = cxEnv.Path.GetFileNameWithoutExtension(componentGraph.LinePragma.FileName);
+            if(isComponentNameInvalid(fileName))
+            {
+                Comment root = new Comment();
+                Comment componentName = new Comment(fileName, fileName, root, componentGraph.LinePragma);
+                componentName.ResolveShortName(fileName);
+
+                // if the component does not have a name, use the file name as the component name
+                oneWordComponents.Add(componentName.NodeId, componentName);
             }
         }
         catch
         {
+            continue;
         }
     }

     // ---------------- global components name check ----------------------------
-    foreach(CxList component in vueModelComponents){
```

```
-       if(isComponentNameInvalid(component.GetName())){
+   foreach(CxList component in vueModelComponents)
+   {
+       if(isComponentNameInvalid(component.GetName()))
            oneWordComponents.Add(component);
-       }
    }

    result = oneWordComponents;
```

Kotlin / Kotlin_Android / Use_of_WebView_AddJavascriptInterface

Code changes

```
---
+++
@@ -1,16 +1,30 @@
-CxList methods = Find_Methods();
-CxList members = Find_MemberAccesses();
+List<int> sdkVersionList = new List<int>();
+string pattern = @"minSdkVersion\s*=?\s*([0-9]+)";

-CxList webviews = methods.FindByMemberAccess("WebView.addJavascriptInterface");
+CxList targets = All.FindByRegexExt(pattern, "build.gradle", false);

-CxList setEnable = methods.FindByShortName("setJavaScriptEnabled");
-CxList enable = All.GetParameters(setEnable);
-enable.Add(members.FindByMemberAccess("WebSettings.javaScriptEnabled").GetAssigner());
+if(targets.Count > 0)
+{
+   // Find sdk version in gradle build files
+   foreach(CxList comment in targets)
+   {
+       Match target = Regex.Match(comment.GetName(), pattern);
+       bool isInt = int.TryParse(target.Groups[1].Value, out int sdkVersion);
+       if(isInt)
+       {
+           sdkVersionList.Add(sdkVersion);
+       }
+   }
+}

-CxList enableTrue = enable.FindByAbstractValue(abstractValue => abstractValue is TrueAbstractValue);
+// addJavascriptInterface vulnerability is fixed in API 17 and above
+// If minSdkVersion is not declared, the system assumes a default value of 1
+if(!sdkVersionList.Any() || sdkVersionList.Min() < 17)
+{
+   CxList webviews = Find_Methods().FindByMemberAccess("WebView.addJavascriptInterface");
+   CxList attribute = Find_CustomAttribute().FindByCustomAttribute("JavascriptInterface");
```

```
-CxList attribute = All.FindByCustomAttribute("JavascriptInterface");
-
-result = webviews;
-result.Add(enableTrue);
-result.Add(attribute);
+   result = All.NewCxList(
+       webviews,
+       attribute);
+}
```

Lua / Lua_Low_Visibility / Password_In_Comment

Code changes

```
---
+++
@@ -5,7 +5,7 @@
    @"|salasana|schluessel|schluesselwort|senha|sifre|wachtwoord|wagwoord|watchword|zugangswort|PAROLACHIAVE|PAROLA CHIAVE" +
    @"|PAROLECHIAVI|PAROLE CHIAVI|paroladordine|verschluesselt|sisma)[^\s]*)";

-CxList singleLineComment = All.FindByRegexExt(@"(?<=--|#)(?!.*\[\[)(?:(?!--|#).)*" + passwords
+CxList singleLineComment = All.FindByRegexExt(@"(?<=--|#)(?!.*\[\[)(?:(?!//|#).)*" + passwords
    + @"(?=[^\n\w]*[:={[-]|[^\n\w]+[""']\w).*", new List<string>{"*.lua", "*.conf"}, true,
    CxList.CxRegexOptions.None, RegexOptions.Multiline | RegexOptions.IgnoreCase);
```

PHP / PHP_High_Risk / Code_Injection

Code changes

```
---
+++
@@ -1,2 +1,2 @@
 CxList inputs = Find_Interactive_Inputs();
-result = Code_Injection_Sinks(inputs);
+result = Find_Code_Injection(inputs);
```

PHP / PHP_High_Risk / Deserialization_of_Untrusted_Data

Code changes

```
---
+++
@@ -1,17 +1,9 @@
+CxList unkRefs = Find_UnknownReference();
 CxList inputs = Find_Deserialization_Inputs();
 CxList deserializers = Find_Unsafe_Deserializers();
-CxList sanitizers = Find_Integers();

-//remove all the deserializers in coditioned blocks where the condition is influenced by hash
-CxList allCond = Find_Conditions();
-
-CxList relevantComparison = Find_BinaryExpr().GetByAncs(allCond).FindByShortNames(new []{"==", "!="});
-relevantComparison = sanitizers.GetByAncs(relevantComparison);
```

```
-
-CxList relevantIfs = relevantComparison.GetAncOfType<IfStmt>();
-relevantIfs.Add(relevantComparison.GetAncOfType<IterationStmt>());
-
-deserializers -= deserializers.GetByAncs(relevantIfs);
+CxList sanitizers = unkRefs.GetParameters(Find_Hashing_Functions());
+inputs -= unkRefs.FindAllReferences(sanitizers);

 result.Add(inputs.InfluencingOnAndNotSanitized(deserializers, sanitizers)
     .ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow));
```

PHP / PHP_High_Risk / Reflected_XSS

Code changes

```
---
+++
@@ -1,8 +1,10 @@
+CxList methods = Find_Methods();
+
 CxList inputs = Find_Interactive_Inputs();
 CxList outputs = Find_Interactive_Outputs();

 // `readfile` automatically outputs the contents: it's not a sanitizer
-CxList readfile = Find_Methods().FindByShortName("readfile", false);
+CxList readfile = methods.FindByShortName("readfile", false);
 readfile.Add(
     Find_Params().CxSelectDomProperty<Param>(p => p.Value).GetParameters(readfile, 0)
     );
@@ -24,6 +26,5 @@
     // Duplicated nodes
     (inputs * outputs) - sanitized,
     // Flow
-    inputs.InfluencingOnAndNotSanitized(outputs, sanitized)
-        .ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow)
+    inputs.InfluencingOnAndNotSanitized(outputs, sanitized).ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow)
 );
```

PHP / PHP_Low_Visibility / Cookie_Overly_Broad_Path_In_Config

Code changes

PHP / PHP_Low_Visibility / Trust_Boundary_Violation_in_Session_Variables

Code changes

```
---
+++
@@ -8,10 +8,6 @@
 // general sanitizers
 CxList sanitizers = Find_General_Sanitize();
```

```
-// add 'preg_match' method in condition as sanitizers
-CxList pregMatch = Find_Methods().FindByShortName("preg_match", false);
-sanitizers.Add(Find_UnknownReference().GetSanitizerByMethodInCondition(pregMatch));
-
 // add inputs in conditions as sanitizers
 CxList sessionInIfs = session.GetAncOfType<IfStmt>();
 CxList sessionConditions = sessionInIfs.CxSelectDomProperty<IfStmt>(i => i.Condition);
```

PHP / PHP_Medium_Threat / HttpOnly_Cookie_Flag_Not_Set_In_Config

Code changes

PHP / PHP_Medium_Threat / Insecure_Value_of_the_SameSite_Cookie_Attribute_In_Config

Code changes

PHP / PHP_Medium_Threat / Stored_Code_Injection

Code changes

```
---
+++
@@ -1,4 +1,4 @@
 CxList inputs = Find_DB_Out();
 inputs.Add(Find_Read());
-result = Code_Injection_Sinks(inputs);
+result = Find_Code_Injection(inputs);
```

Python / Python_Exploitable_Path / Python_Find_UnresolvedMethods

Code changes

```
---
+++
@@ -1,3 +1,4 @@
+Comment rootComment = new Comment();
 Func < CSharpGraph, string > GetTypeName = ((CSharpGraph graph) => {
     if(graph is BooleanLiteral) return "boolean";
     if(graph is CharLiteral) return "char";
@@ -26,39 +27,94 @@
     }
 }

-Comment rootComment = new Comment();
-foreach(CxList method in methods){
-   try{
-       if(unknownReferences.FindDefinition(method).Count > 0)
-           continue;
-
-       Expression methObject = method.TryGetCSharpGraph<Expression>();
-       LinePragma lp = methObject.LinePragma;
```

```
-       string methodName = method.GetName();
+Func<CxList, string> GetPackageName = (method) => {
+   CxList target = method.GetLeftmostTarget().Count > 0 ? method.GetLeftmostTarget() : method;
+   CxList relevantNamespace = target.GetAncOfType(typeof(NamespaceDecl));
+   CxList nsImports = relevantNamespace.CxSelectElements<NamespaceDecl>(n => n.Imports);
+
+   CxList foundImport = nsImports.FilterByDomProperty<Import>(i => i.ImportedFilename == target.GetName());
+   if (foundImport.Count == 1) return $"{target.GetName()}";
+
+   foundImport = nsImports.FilterByDomProperty<Import>(i =>
+       i.Symbols.Contains(target.GetName()) || i.SymbolAliases.ContainsKey(target.GetName())
+       );
+   if (foundImport.Count == 1){
+       Import requirePkg = foundImport.TryGetCSharpGraph<Import>();
+       return $"{requirePkg.ImportedFilename}";
+   }
+
+   return null;
+   };


-       CxList methodParams = allParams.GetParameters(method);
-
-       var elements = methodParams.CxSelectDomProperty<Param>(_ => _.Value)
-           .CxSelectElementValue<CSharpGraph,string>(GetTypeName);
-
-       string paramsStr = "||(" + string.Join("||,", elements) + "||)";
-
-       bool foundSignature = false;
-       string objectName = "";
-       if (methObject is ObjectCreateExpr) {
-           foundSignature = true;
-           // In case of ctor, method and class names match
-           objectName = methodName;
-       }
+//Get full projectPath to remove it from filename
+string projectPath = cxScan.GetScanProperty("projectPath");
+int lastIndexOfBackSlash = projectPath.LastIndexOf(cxEnv.Path.DirectorySeparatorChar);
+Match guid = Regex.Match(projectPath, @"(?im)[{(]?[0-9A-F]{8}[-]?(?:[0-9A-F]{4}[-]?){3}[0-9A-F]{12}[)}]?$");
+if(guid.Success) lastIndexOfBackSlash = projectPath.IndexOf(guid.Value) + guid.Value.Length;
+
+List<string> names = methods.CxSelectElementValues<Expression, string>(x => x.ShortName);
+
+//For each name
+foreach(String str in names.Distinct()){
+
+   int i = 0;
+   CxList methodsByName = methods.FindByShortName(str);
+   string methodInfoFinal = "{ \n \"method\": {\"name\":" + "\"" + str + "\", \n \"calls\": [";
```

```
+
+   foreach(CxList method in methodsByName){
+       try{
+           if(unknownReferences.FindDefinition(method).Count > 0)
+               continue;
+
+           Expression methObject = method.TryGetCSharpGraph<Expression>();
+           LinePragma lp = methObject.LinePragma;
+
+           string methodName = method.GetName();
+           CxList methodParams = allParams.GetParameters(method);

-       if (objectName.StartsWith("CxOrphanClass_")) {
-           objectName = "object";
-       }
+           var elements = methodParams.CxSelectDomProperty<Param>(_ => _.Value)
+               .CxSelectElementValue<CSharpGraph,string>(GetTypeName);

-       string resolvedName = (foundSignature ? objectName + "||." : "") + methodName + paramsStr;
-
-       Comment comment = new Comment(methodName, methodName, rootComment, lp);
-       comment.ResolveShortName(resolvedName);
+           string paramsStr = string.Join(", ", elements);
+
+           bool foundSignature = false;
+           string objectName = "";
+           if (methObject is ObjectCreateExpr) {
+               foundSignature = true;
+               // In case of ctor, method and class names match
+               objectName = methodName;
+           }
+
+           if (objectName.StartsWith("CxOrphanClass_")) {
+               objectName = "object";
+           }
+
+           string namePrefix = GetPackageName(method) ?? "";
+           string fullFilename = lp.FileName.Remove(0, lastIndexOfBackSlash);
+           char pathSeparator = cxEnv.Path.DirectorySeparatorChar;
+           string fileName = pathSeparator.Equals('\\') ? Regex.Replace(fullFilename, @"\\", "/") : fullFilename;
+
+           methodInfoFinal += " \n { "
+               + " \n \"package\": " + "\"" + namePrefix + "\""
+               + ", \n \"fullname\": " + "\"" + objectName + "||." + methodName + "\""
+               + ", \n \"shortname\": " + "\"" + methodName + "\""
+               + ", \n \"parameters\": " + "\"" + paramsStr + "\""
+               + ", \n \"line\": " + lp.Line
+               + ", \n \"column\": " + lp.Column
```

```
+               + ", \n \"sourcefile\": " + "\"" + fileName + "\""
+               + "\n }, ";
+
+           i++;
+
+       }catch(Exception e){}
+   }
+   if(i > 0){
+       methodInfoFinal += "\n ] \n } \n }";
+       methodInfoFinal = new System.Text.StringBuilder(methodInfoFinal).ToString();
+
+       Comment comment = new Comment(methodInfoFinal, methodInfoFinal, rootComment, methodsByName.GetFirstGraph().LinePragma);
+       comment.ResolveShortName(methodInfoFinal);
+
        result.Add(comment.NodeId, comment);
-   }catch(Exception e){}
+   }
 }
```

Rust / Rust_High_Risk / Connection_String_Injection

Code changes

```
---
+++
@@ -1,15 +1,16 @@
 CxList methods = Find_Methods();
-CxList unknownReferences = Find_UnknownReference();
+CxList importRefs = Find_UnknownReference();
+importRefs.Add(Find_MemberAccesses());

 // Mongo
-CxList mongoConnectionMembers = unknownReferences.FindByShortNames(
+CxList mongoConnectionMembers = importRefs.FindByShortNames(
        "Client",
        "ClientOptions",
        "ConnectionString").GetMembersOfTarget();
 CxList mongoConnectionMethods = mongoConnectionMembers.FindByShortNames("with_uri_str*", "parse", "from_str");

 // Sqlx
-CxList sqlxConnectionMembers = unknownReferences.FindByShortNames(
+CxList sqlxConnectionMembers = importRefs.FindByShortNames(
        "PgPool",
        "PgConnection",
        "MySqlPool",
@@ -17,10 +18,11 @@
        "AnyPool",
        "AnyConnection",
        "SqlitePool",
-       "SqliteConnection").GetMembersOfTarget();
```

```
+   "SqliteConnection").GetMembersOfTarget();
+
 CxList sqlxConnectMethods = sqlxConnectionMembers.FindByShortNames("connect*", "from_url");


-CxList sqlxConnectOptionsMembers = unknownReferences.FindByShortNames(
+CxList sqlxConnectOptionsMembers = importRefs.FindByShortNames(
        "PgConnectOptions",
        "MySqlConnectOptions",
        "AnyConnectOptions",
```

Rust / Rust_Medium_Threat / Privacy_Violation

Code changes

```
---
+++
@@ -1,109 +1,15 @@
-// This query searches for variables and constants that could contain personal sensitive data which is streamed to an output
-CxList strings = Find_Strings();
-CxList integerLiteral = Find_IntegerLiterals();
-
-CxList inputs = All.NewCxList();
-inputs.Add(
-   Find_Inputs(),
-   Find_Stored_Inputs());
-
-CxList literals = All.NewCxList();
-literals.Add(
-   strings,
-   integerLiteral);
+//Find_Personal_Info does not include e-mail
+CxList inputs = Find_Personal_Info();



-// Define the personal info ////////////////////////////////////////////////////////
+CxList outputs = Find_Interactive_Outputs();

-// Find names that are suspected to be personal info, e.g., String PASSWORD, Integer SSN...
-// 1) And remove string literals, such as x = "password"
-CxList personalInfo = Find_Personal_Info() - strings;
+CxList sanitizers = All.NewCxList(
+   Find_Encrypt(),
+   Find_KDF(),
+   Find_Weak_or_Broken_KDF(),
+   Find_Hash(),
+   Find_Weak_or_Broken_Hash());

-// 2) Exclude variables that are all uppercase - usually describes the pattern of the data, such as PASSWORDPATTERN
-personalInfo -= personalInfo.Filter(_ => string.Equals(_.ShortName, _.ShortName.ToUpper()));
-
```

```
-// 3) Exclude constants that are assigned a literal
-CxList constantDeclStmts = Find_ConstantDeclStmt().FindByShortName(personalInfo);
-CxList constants = personalInfo.FindDescendantsOfType<Declarator>(constantDeclStmts);
-
-CxList allConstRef = personalInfo.FindAllReferences(constants);
-CxList allConstRefOrigin = allConstRef.Clone();
-
-// Find all assignments of string or integer literals
-CxList intStringLiteral = strings.Clone();
-intStringLiteral.Add(integerLiteral);
-CxList constAssignedL = intStringLiteral.FindByFathers(allConstRef.FindByType<Declarator>());
-
-// Remove assignments of constants to string or integer literals
-allConstRef -= personalInfo.FindAllReferences(constAssignedL.GetFathers());
-
-// Find constants assigned as null, even if the assign is not in the declaration line
-// e.g. val userPassword:String?; userPassword=null;
-CxList nullConsts = allConstRef.FindByAbstractValue(_ => _ is NullAbstractValue);
-
-// Find all assignments to constants
-CxList constAssignments = allConstRef.FindByAssignmentSide(CxList.AssignmentSide.Left).GetFathers();
-
-// Find assignments of literals to constant personal_info (Pi) and remove
-CxList piLiterals = literals.GetFathers() * constAssignments;
-CxList allConstRefAssignedWithPiLiterals = allConstRef.FindByFathers(piLiterals);
-
-CxList toRemove = All.NewCxList();
-toRemove.Add(
-   nullConsts,
-   allConstRefAssignedWithPiLiterals);
-
-// Remove previous findings
-allConstRef -= allConstRef.FindAllReferences(toRemove);
-
-// Remove from personal_info all references that were removed above
-personalInfo -= (allConstRefOrigin - allConstRef);
-
-// Define the outputs ////////////////////////////////////////////////////////
-
-// Add exceptions (that could be thrown) to outputs
-CxList exceptions = Find_ObjectCreations().FindByName("*Exception");
-CxList exceptionsCtors = Find_ConstructorDecl().FindByName("*Exception");
-
-// Handle the case where the super (base) constructor of the exception is used to create a new throwable exception
-CxList exceptionsCtorsWithSuper = exceptionsCtors.FilterByDomProperty<ConstructorDecl>(_ => _.BaseParameters.Count > 0);
-
-CxList outputs = All.NewCxList();
-outputs.Add(
```

```
-   Find_Outputs(),
-   exceptions,
-   exceptionsCtorsWithSuper);
-
-
-// Define the sanitizers ////////////////////////////////////////////////////////
-
-CxList sanitize = All.NewCxList();
-sanitize.Add(
-   Find_DB_In(), // In some languages is called Find_DB, Find_DB_In, Find_DB_Input
-   Find_Hash());
-
-// Encrypted strings are considered safe
-CxList encrypt = Find_Encrypt();
-CxList encoded = Find_KDF();
-encoded.Add(Find_Methods().FindByShortName("*Encode*", false)); // All variables labelled encode(ed) are considered safe
-
-sanitize.Add(
-   encrypt,
-   encoded);
-
-// Calculate the results ////////////////////////////////////////////////////////
-
-// Split personalInfo into variables and constants
-CxList variableRef = personalInfo - allConstRef;
-
-// Find all constants that are assigned from an input (directly or indirectly) and are influencing an output
-CxList constInfluencedByInput = outputs.InfluencedByAndNotSanitized(inputs, sanitize).IntersectWithNodes(allConstRef);
-
-// Find all variables and that are influencing an output
-CxList variableRefPath = outputs.InfluencedByAndNotSanitized(variableRef, sanitize);
-
-result.Add(
-   variableRefPath,
-   constInfluencedByInput);
-
-result = result.ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
+result = inputs.InfluencingOnAndNotSanitized(outputs, sanitizers)
+   .ReduceFlow(CxList.ReduceFlowType.ReduceSmallFlow);
```

VbNet / VbNet_Low_Visibility / Heap_Inspection

Code changes

```
---
+++
@@ -6,46 +6,35 @@


 CxList passwords = Find_All_Passwords();
```

```
-// 1)exclude variables that are all uppercase - usually describes the pattern of the data, such as PASSWORDPATTERN, PASSORDTYPE...
-CxList upperCase = All.NewCxList();
-foreach (CxList res in passwords)
-{
-   string name = res.GetName();
-   if (name.ToUpper().Equals(name))
-   {
-       upperCase.Add(res);
-   }
-}
-passwords -= upperCase;
+// 1) Exclude variables that are all uppercase - usually describes the pattern of the data,
+// such as PASSWORDPATTERN, PASSORDTYPE...
+passwords -= passwords.Filter(p => !p.ShortName.Any(c => char.IsLower(c)));

-//2) define sanitizers = encryption
-CxList sanitizeMethods = All.FindByType(typeof(ObjectCreateExpr)).FindByShortName("SecureString", false);
+//2) Define sanitizers = encryption
+CxList sanitizeMethods = Find_ObjectCreations().FindByShortName("SecureString", false);
 sanitizeMethods.Add(All.FindByMemberAccess("CryptoStream.Write*", false));
 CxList sanitize = All.FindByFathers(sanitizeMethods);
 sanitize.Add(Find_Encrypt());

-//3) define safe types
+//3) Define safe types
 string[] safeTypes = {"*.SecureString", "SecureString", "*.CryptoStream", "CryptoStream"};
 passwords -= passwords.FindByTypes(safeTypes, false);
-passwords -= passwords.FindByType(typeof(ConstantDecl));
+passwords -= passwords.FindByType<ConstantDecl>();

-//4) find assignments of literals
-CxList literals = Find_Strings();
-literals.Add(All.FindByType(typeof(IntegerLiteral)));
+//4) Find assignments of literals
+CxList literals = All.NewCxList(Find_Strings(), Find_IntegerLiterals());
 CxList assignLiteral = literals.GetFathers();

-//find the declaration statements of passwords
-CxList passDecl = passwords.FindByType(typeof(Declarator));
-passDecl.Add(passwords.FindByType(typeof(FieldDecl))); // property of classes are of FieldDecl type
-
-// find password that is assigned a literal
+// Find and remove passwords assigned a literal
 CxList passAssigned = passwords.FindByAssignmentSide(CxList.AssignmentSide.Left);
 CxList passAssignLiteral = passAssigned.FindByFathers(assignLiteral);
 passAssignLiteral.Add(assignLiteral);
+passAssigned -= passAssignLiteral;
```

```
-passAssigned -= passAssignLiteral; //remove password that is assigned a literal
+// Remove references from aspx files
+passAssigned -= passAssigned.GetByAncs(Find_MethodDecls().FindByShortName("checkmarx_class_init"));


-//remove password that is assigned a safe object or an encrypted object
+// Remove password that is assigned a safe object or an encrypted object
 passAssigned -= passAssigned.InfluencedBy(sanitize);
-
+
 result = All.FindDefinition(passAssigned);
```