PDG Edge Classification

New Requests

- Use TYPE_SUBTYPE for node and edges.
- SUBTYPEs are disjointed; union of SUBTYPES is TYPE.
- create nodes for annotation instruction (must be a sink node).
  - INST_ANNO_LOCAL
  - INST_ANNO_GLOBAL
- Create edges for ANNO_FUNC and ANNO_VAR.
  - ANNO_FUNC: from function entry node to annotation node.
  - ANNO_VAR: from variable node to annotation node.
- CONTROLDEP Edges
  - CONTROLDEP_CALLINV
  - CONTROLDEP_CALLRET
  - CONTROLDEP_ENTRY
  - CONTROLDEP_BR
  - CONTROLDEP_IND_BR ?
  - Every instruction node should be reachable from the FUNC_ENTRY node through CONTROLDEP_* edges.
  - **Assertion**: every instruction in the function body that are not in control blocks is directly reachable from the FUNC_ENTRY node through CONTROLDEP_* edges. Control dependence edge preserves the control structure inside a function when there is no goto statement in the program.
- Maintain version for the PDG specification and separated PDG implementation.
- PDG capture C at the moment. Need more changes on handling C++ in future.

- For unhandled nodes and edges, label them with TYPE_OTHERNODE, TYPE_OTHEREDGE.
    - if an attributes can apply to more than one subtypes, make it an attribute of the type (a field).
- GLOBALVAR_GLOBAL / GLOBALVAR_LOCAL
- Remove "sensitive" info from the annotation node.
- In C program, add function pointer (invoke) and indirect branch examples.
- In C program, add two level pointer example.

Edges Types

The followings are the edges types defined in our current PDG implementation.

```
enum class EdgeType
{
  IND_CALL,
  CONTROLDEP_CALLINV,
  CONTROLDEP_CALLRET,
  CONTROLDEP_ENTRY,
  CONTROLDEP_BR,
  CONTROLDEP_IND_BR,
  DATA_DEF_USE,
  DATA_RAW,
  DATA_READ,
  DATA_ALIAS,
  DATA_RET,
  PARAMETER_IN,
  PARAMETER_OUT,
  PARAMETER_FIELD,
  GLOBAL_DEP,
  VAL_DEP,
  ANNO_FUNC,
  ANNO_VAR,
  TYPE_OTHEREDGE
};
```

> add annotation edges

Node Types

```
INST, (INST)
FORMAL_IN,
FORMAL_OUT,
```

```
ACTUAL_IN,
ACTUAL_OUT,
RETURN, (INST_RET)
FUNC_ENTRY,
GLOBAL_VAR,
CALL (INST_CALL)
```

> add Annotation node type

Example Program

We use the following example to demonstrate all edges in PDG.

```
#include <string.h>

 char __attribute__((annotate("sensitive"))) *key ;
 char *ciphertext;
 unsigned int i;

void greeter (char *str, int* s) {
     char* p = str;
     printf("%s\n", p);
     printf(", welcome!\n");
     *s = 15;
 }

 void initkey (int sz) {
     key = (char *) (malloc (sz));
     // init the key randomly; code omitted
     for (i=0; i<sz; i++) key[i]= 1;
 }

int encrypt (char *plaintext, int sz) {
     ciphertext = (char *) (malloc (sz));
     for (i=0; i<sz; i++)
         ciphertext[i]=plaintext[i] ^ key[i];
     return sz;
 }

 int main (){
     int age = 10;
     char __attribute__((annotate("sensitive"))) username[20], text[1024];
     printf("Enter username: ");
     scanf("%19s",username);
     greeter(username, &age);
     printf("Enter plaintext: ");
     scanf("%1023s",text);

     initkey(strlen(text));
     int sz = encrypt(text, strlen(text));
     printf("Cipher text: ");
```

```
    for (i=0; i<strlen(text); i++)
        printf("%x ",ciphertext[i]);
    printf("encryption length: %d", sz);
    return 0;
}
```

CONTROLDEP

Description:

CONTROLDEP edges are consists of five subtypes:

1. CONTROLDEP_ENTRY
2. CONTROLDEP_BR
3. CONTROLDEP_IND_BR
4. CONTROLDEP_CALLINV
5. CONTROLDEP_CALLRET

Line: Black solid line with label {CONTROLDEP_[TYPE]}.

Case 1: CONTROLDEP_ENTRY

Description:

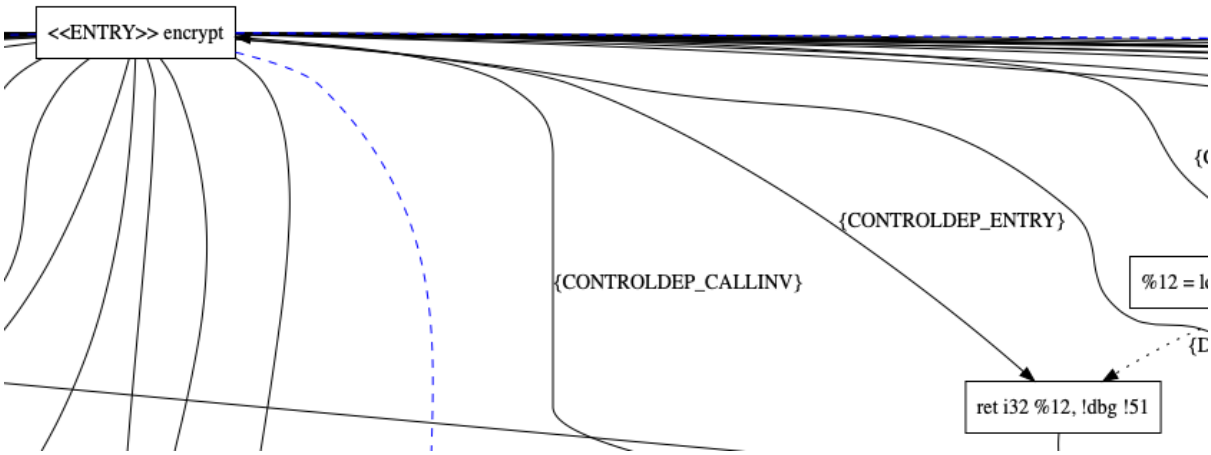
Connects the function entry node with all the instructions in the function body.

Edge type: CONTROLDEP_ENTRY

From node: ENTRY, **IR Instruction:** N/A, **IR line number**: N/A, **Source line number**: N/A

To node: INST, **IR Instruction**: ret i32 %12, !dbg !92, **IR line number**: 140, **Source line number**: 26

Example



Case 2: CONTROLDEP_BR

Description:

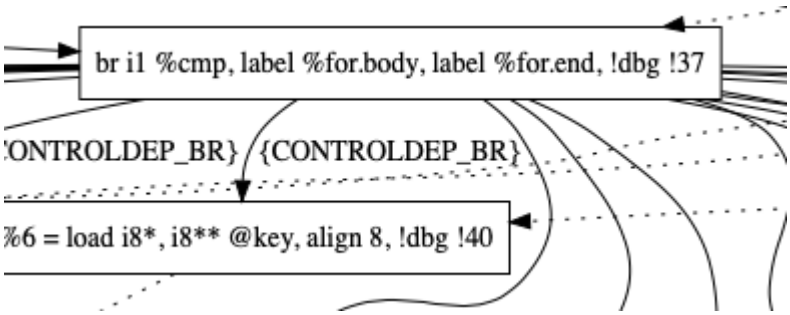
Connects the control dependence block's terminator to the instructions in the control dependent block.

Edge type: CONTROLDEP_BR

From node: INST, **IR Instruction:** br i1 %cmp, label %for.body, label %for.end, !dbg !78, **IR line number**: 108, **Source line number**: 24

To node: INST, **IR Instruction**: %6 = load i8*, i8** @key, align 8, !dbg !81, **IR line number**: 117, **Source line number**: 25

Example



Case 3: CONTROLDEP_IND_BR

Need an example from Rob.

Case 4: CONTROLDEP_CALLINV

Description:

CONTROLDEP_CALLINV edge connects a call site with the function entry point of callee. It indicates the control flow transition from the caller to callee.

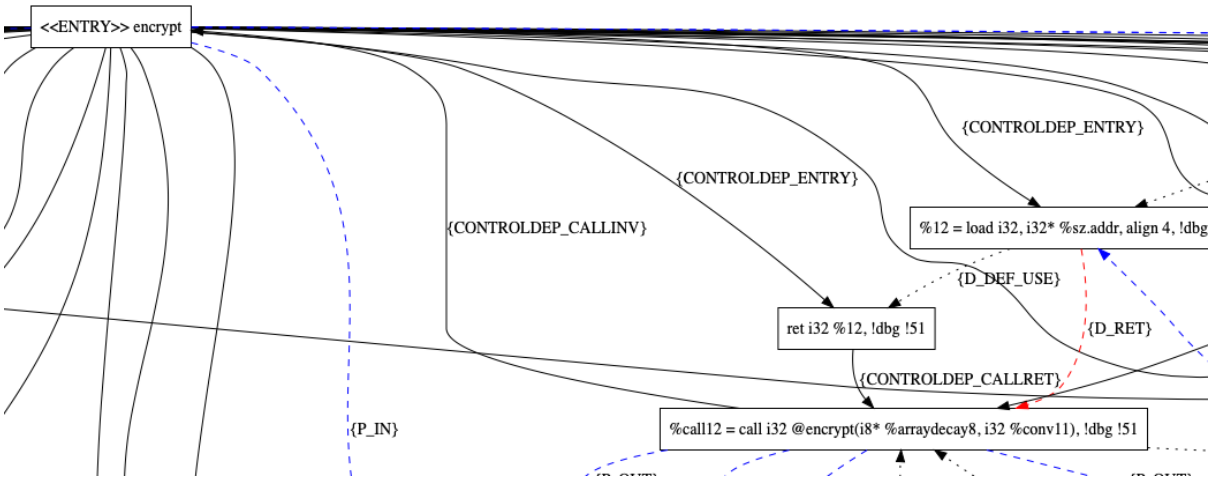
Edge type: D_RET

From node: CALL, **IR Instruction:** %call12 = call i32 @encrypt(i8* %arraydecay8, i32 %conv11), !dbg !124, **IR line number**: 177, **Source line number**: 39

To node: FUNC_ENTRY, **IR Instruction**: N/A, **IR line number**: N/A, **Source line number**: N/A

Example

In the example program, the main function calls the encrypt function.



Case 5: CONTROLDEP_CALLRET

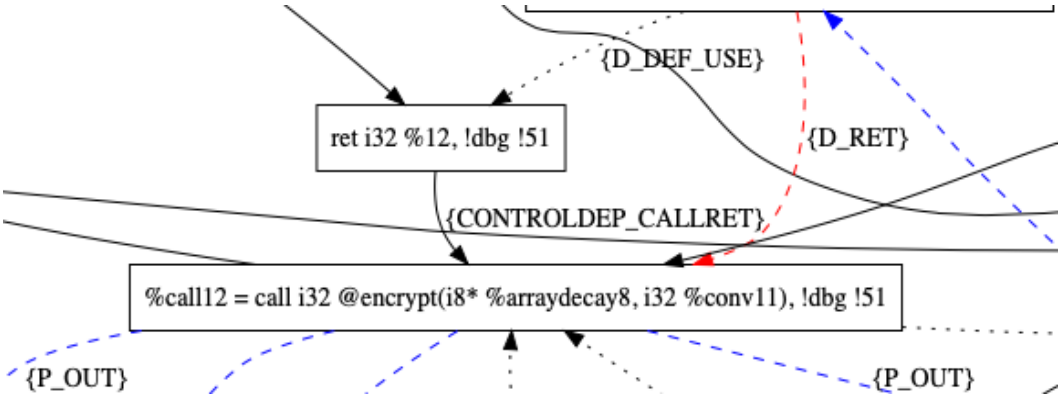
Description:

The return instruction in the **initkey** function is connected with the call site of **initkey** in the **main** function.

Edge type: CONTROLDEP_CALLRET

From node: INST, **IR Instruction:** ret i32 %12, !dbg !92, **IR line number**: 140, **Source line number**: 26

To node: CALL, **IR Instruction**: %call12 = call i32 @encrypt(i8* %arraydecay8, i32 %conv11), !dbg !124, **IR line number**: 177, **Source line number**: 39



DATA_RET

Description:

DATA_RET edge connects the return value to the call instruction in the caller function. It indicates the data flow from the return instruction to the call instruction.

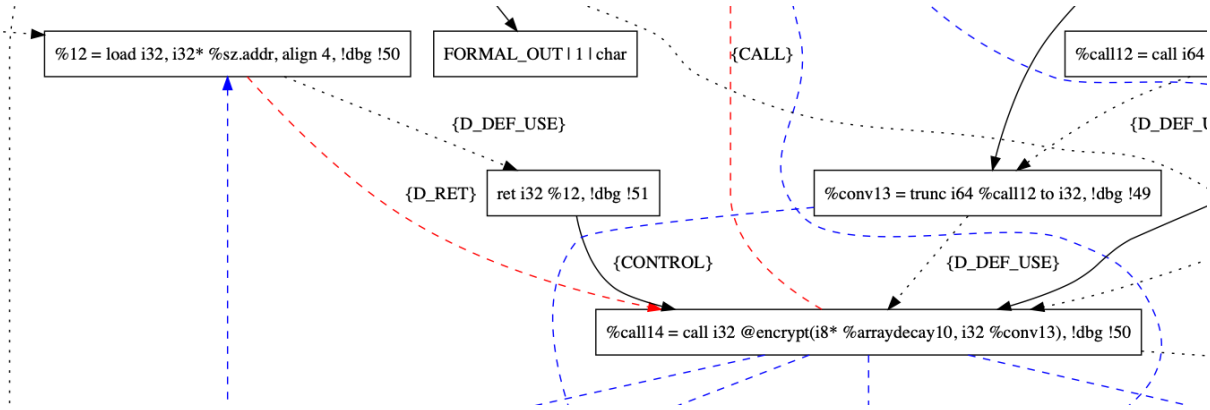
Line: Red dash line with label {D_RET}.

Edge type: D_RET

From node: INST, **IR Instruction:** %12 = load i32, i32* %sz.addr, align 4 , **IR line number**: 139, **Source line number**: 26

To node: CALL, **IR Instruction**: %call14 = call i32 @encrypt(i8* %arraydecay10, i32 %conv13), **IR line number**: 177, **Source line number**: 39

Example



DATA_DEF_USE

Description:

DATA_DEF_USE edge connects two nodes with def and use. It is directly computed from the LLVM def-use chain.

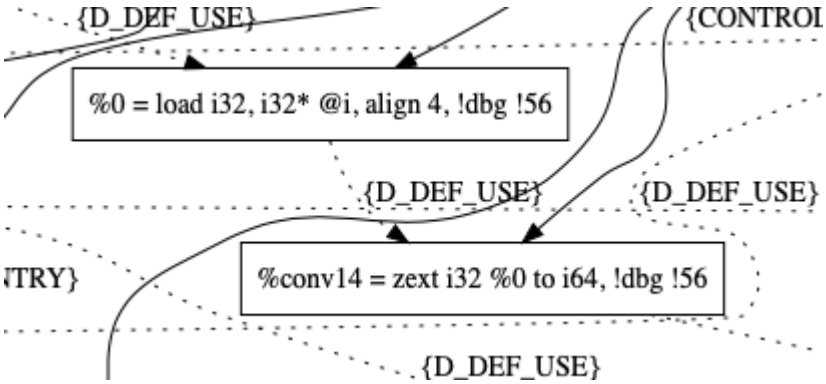
Line: Black dotted line with label {D_DEF_USE}.

Edge type: D_DEF_USE

From node: INST, **IR Instruction:** %0 = load i32, i32* @i, align 4, !dbg !130, **IR line number**: 184, **Source line number**: 41

To node: INST, **IR Instruction**: %conv16 = zext i32 %0 to i64, !dbg !130, **IR line number**: 185, **Source line number**: 41

Example



DATA_RAW

Description:

DATA_RAW connects two nodes with read after write dependence. This is flow sensitive. We use memory dependency LLVM pass to compute this information.

Line: Black dotted line with label {D_RAW}.

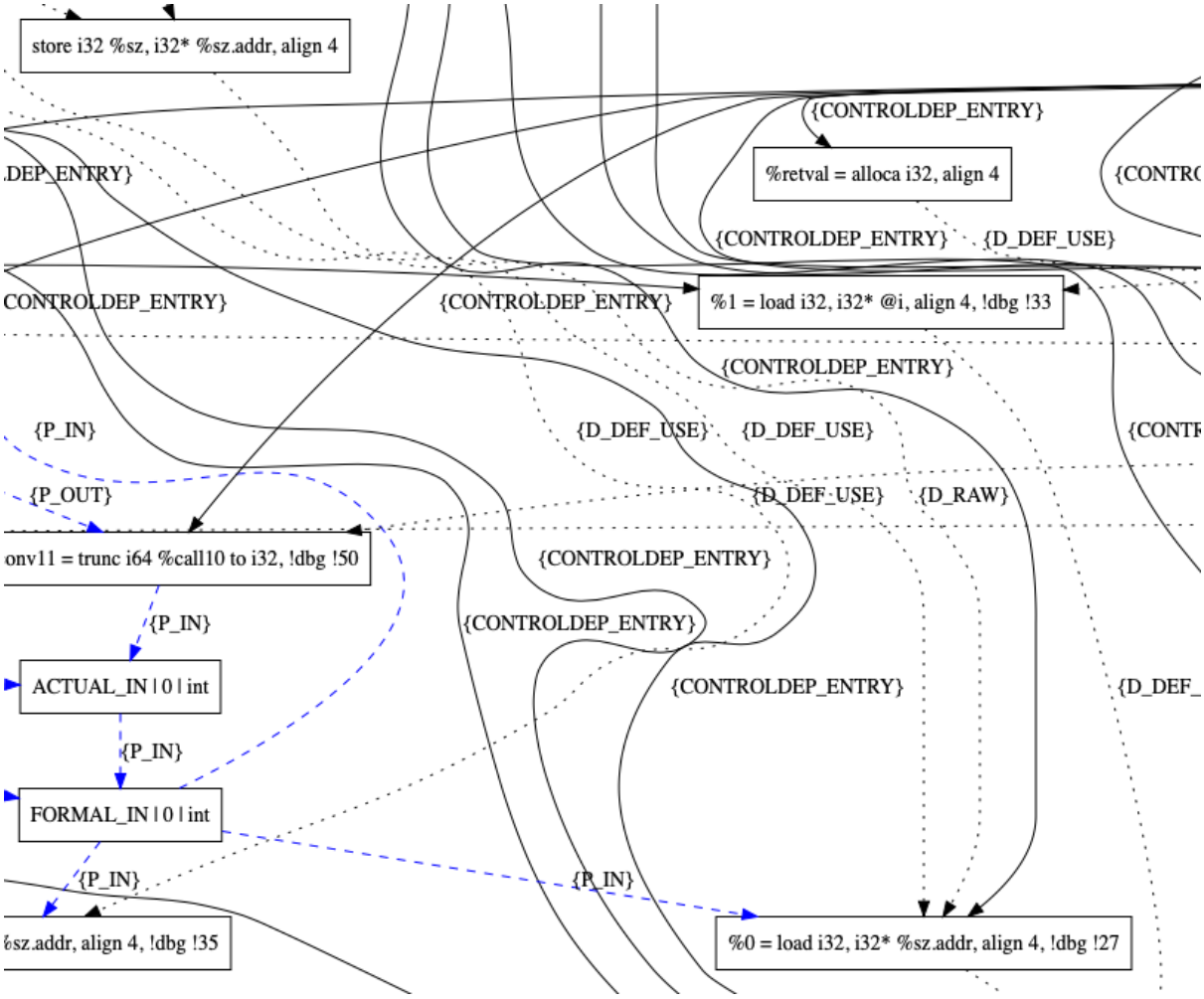
Edge type: D_RAW

From node: INST, **IR Instruction:** store i32 %sz, i32* %sz.addr, align 4, **IR line number**: 95, **Source line number**: 23

To node: INST, **IR Instruction**: %0 = load i32, i32* %sz.addr, align 4, !dbg !68, **IR line number**: 97, **Source line number**: 23

Example

The load instruction reads from the address which is written by the store instruction.



DATA_ALIAS

Description:

DATA_ALIAS edge connects two nodes that have may_alias relations. Note that if two nodes n1 and n2 have may_alias relation, then there are two DATA_ALIAS edges exist between them, one from n1 to n2 and one from n2 to n1. This is because the alias relation is undirectional.

Line: Black dotted line with label {D_ALIAS}.

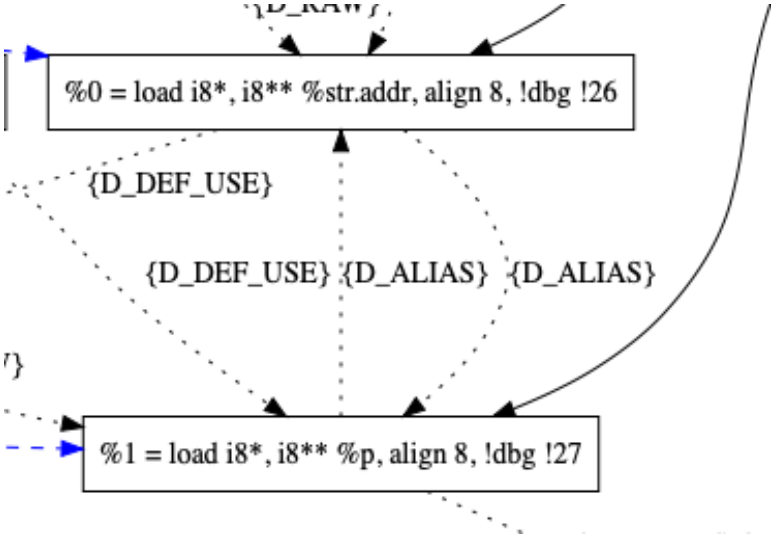
Edge type: D_ALIAS

From node: INST, **IR Instruction:** %0 = load i8*, i8** %str.addr, align 8, !dbg !30, **IR line number**: 33, **Source line number**: 10

To node: INST, **IR Instruction**: %1 = load i8*, i8** %p, align 8, !dbg !31 , **IR line number**: 35, **Source line number**: 10

Example

In the greeter function, we a local variable **char* p** is defined and it's an alias to the **str** argument. The value of p is stored in %1 register and the argument str is stored in the register %0. Thus, there is an alias edge from %0 and %1, and an alias edge from %1 to %0.



Parameter Trees

In the **main** function, there is a call to the **encrypt** function, which takes takes a **char*** as the first argument. The parameter trees for the char* argument are consists of two nodes: the root node represents the char* parameter and the char child node represent the pointed char.

PARAMETER_IN

Description:

PARAMETER_IN edge represents interprocedural data flow from caller to callee. It connects

1. actual_parameter_in_tree node and formal_in tree nodes

2. formal_in_tree node and the IR variables that corresponds to this formal_in_tree nodes in the callee.

Line: Blue dash line with label {P_IN}.

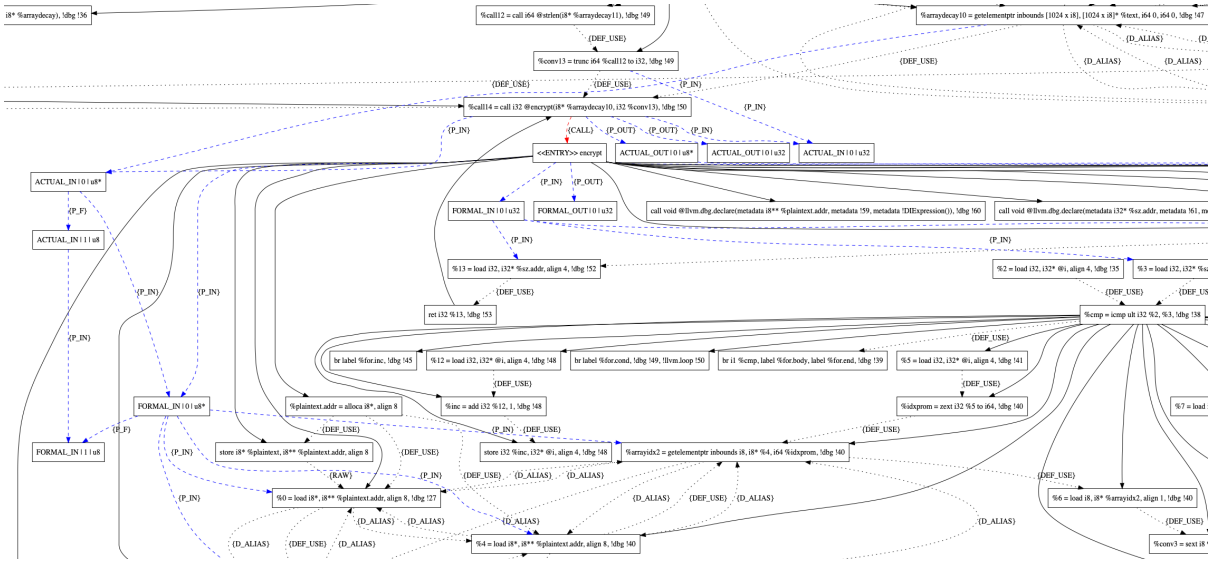
Node Definition:

Src: Actual in tree node / Formal in tree node

Dst: Formal in tree node / INST

Example

Start with the call to **encrypt** function, the %arraydecay10 is passed as an argument to the call. Thus, there is a parameter in edge, from the %arraydecay10 to the actual_in node **ACTUAL_IN | 0 | u8***. Then, the actual in node is linked with the formal in tree node which is defined for the first formal parameter **text**. Next, the value of **text** parameter is stored to a the stack address %plaintext.addr. So, any load from the stack address will return the value of text parameter (in this case, it's the char pointer). In the example, a load instruction **%4 = load i8*, i8** %plaintext.addr, align 8** loads from the stack address. Thus, there is a parameter_in edge from the formal tree node of text to the load instruction.



PARAMETER_OUT

Description:

PARAMETER_OUT edge represents data flow from callee to caller. It connects

1. arguments modified in callee to formal_out_tree node.

2. formal_out_tree node to actual_out_tree node.

3. actual_out_tree node to the modified variable in caller.

Line: Blue dash line with label {P_OUT}.

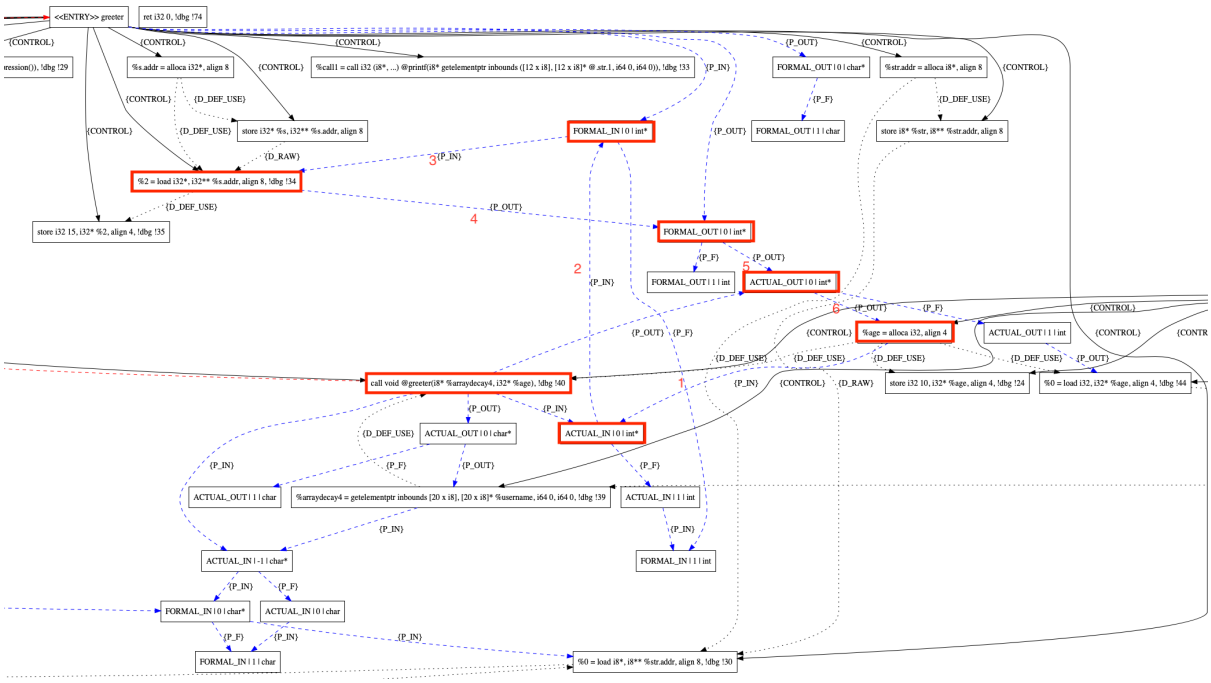
Node Definition:

Src: INST / FORMAL_OUT / ACTUAL_OUT

Dst: INST / FORMAL_OUT / ACTUAL_OUT

Example

The relevant nodes are highlighted by red rectangle box. And the flow order is shown on the edges. The %age variable in main is passed to function **greeter** and get modified in the function. Thus, there is an actual out edge connects the second actual parameter with the %age variable.



PARAMETER_FIELD

Description:

PARAMETER_FIELD edge connects a parent parameter tree node to a child parameter tree node.

Line: Blue dash line with label {P_F}.

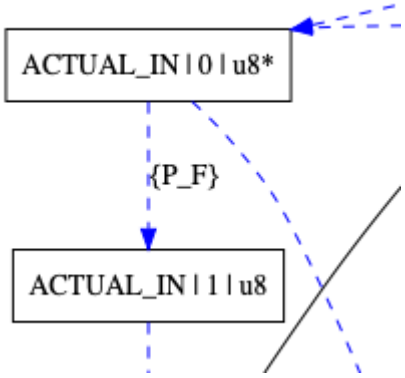
Node Definition:

Src: ACTUAL_IN / ACTUAL_OUT / FORMAL_IN / FORMAL_OUT

Dst: ACTUAL_IN / ACTUAL_OUT / FORMAL_IN / FORMAL_OUT

Example

The actual out tree for the first parameter in the encrypt function. The type of the parameter is char*, thus, there are two nodes represent in the parameter trees: the root node represents the pointer and the child node represent the pointed char.



Annotations

Annotation Nodes are connected to annotated nodes with ANNO_VAR edge type.

Line: green dash line with label {ANNO_VAR}.

Local annotation

Description:

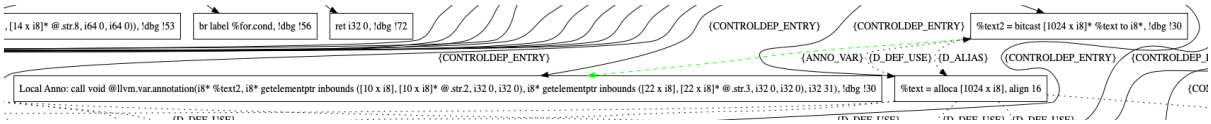
The annotation node for local variable is stored in the function body.

Edge type: ANNO_VAR

From node: INST, **IR Instruction:** %text2 = bitcast [1024 x i8]* %text to i8*, !dbg !103, **IR line number**: 158, **Source line number**: 31

To node: INST_ANNO_LOCAL, **IR Instruction**: call void @llvm.var.annotation(i8* %text2, i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.2, i32 0, i32 0), i8* getelementptr inbounds ([22 x i8], [22 x i8]* @.str. 3, i32 0, i32 0), i32 31), !dbg !103, **IR line number**: 159, **Source line number**: 31

Example:



Global annotation

Description:

The annotation node for global variable is stored in the global scope.

Edge type: ANNO_VAR

From node: GLOBAL_VAL_GLOBAL/GLOBAL_VAR_LOCAL, **IR Instruction:** @key = common global i8* null, align 8, !dbg !0, **IR line number**: 8, **Source line number**: 5

To node: INST_ANNO_GLOBAL, **IR Instruction**: @llvm.global.annotations = appending global [1 x { i8*, i8*, i8*, i32 }] [{ i8*, i8*, i8*, i32 } { i8* bitcast (i8** @key to i8*), i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.2, i32 0, i32 0), i8* getelementptr inbounds ([22 x i8], [22 x i8]* @.str. 3, i32 0, i32 0), i32 5 }], section "llvm.metadata", **IR line number**: 20, **Source line number**: 5

Example: