Open Source software copyright issues with special focus on redistributing Docker images

Basic lecture: Copyright basics, adapter's copyright

Carsten Emde
Open Source Automation Development Lab (OSADL) eG

Copyright (1)

- International copyright law rules that exclusive rights of use (*i.e.* to copy and to distribute) are granted to an author of a work. These rights may be licensed to third parties.
- A work is any piece of art or literature that
  - was created by a human being,
  - is perceivable by a human being,
  - is the result of an individual creativity.
- Software is considered a literary work and as such is protected by copyright law.
- Copyright is granted automatically and immediately when a work is created; it does not require any formality such as registration.

Copyright (2)

- To copy and distribute a work without permission constitutes an infringement of law that may entail serious consequences, since the right holder may assert claims against the offender, for example, that:
  - the offender must inform all unlawful recipients of the work to stop using it,
  - the offender must provide lists of all unlawful recipients to the author,
  - all unauthorized copies are physically destroyed,
  - penalties are imposed for infringement of copyright law.

Copyright (2)

- To copy and distribute a work without permission constitutes an infringement of law that may entail serious consequences, since the right holder may assert claims against the offender, for example, that:
  - the offender must inform all unlawful recipients of the work to stop using it,
  - the offender must provide lists of all unlawful recipients to the author,
  - all unauthorized copies are physically destroyed,
  - penalties are imposed for infringement of copyright law.

Copyright law is a sharp sword that grants considerable rights to an author of a work.

Copyright and adaptor's copyright

- When an author adapts a work of another author with permission a so-called adapter's copyright is granted. The adapter's copyright includes the same rights as the rights of the primary author:
  - Free selection of type of license
  - Free selection of license conditions
- A licensee of an adapted work must fulfill all license conditions of each author. The adapted work is called a **derivative** of the original work.
- The license conditions of the two authors must not contradict each other. If they do, the licenses are "incompatible" and the work may not be licensed at all.

Adapter's copyright in software

- There is probably no software that is not sooner or later adapted (fixing bugs, implementing new features, considering altered conditions).
- Why is it important to know whether a software adaptation is creating a derivative work?
  - If derivative work: Licenses must be compatible, and obligations of both licenses must be fulfilled.
  - If no derivative work: Obligations of every license must be fulfilled, but this may be done independently. Licenses may be incompatible.

How to find out whether a software adaptation creates a derivative work?

1. Find out what constitutes a derivative work in non-software works.
2. Transpose the findings to software.
3. Check out the various ways two software components may interact and decide whether they create a derivative work according to
   - Mainstream interpretation of copyright law
   - View of the Free Software Foundation (FSF)
   - Recommendation of OSADL

How to find out whether a software adaptation creates a derivative work?

1. Find out what constitutes a derivative work in non-software works.
2. Transpose the findings to software.
3. Check out the various ways these software components may interact and decide whether they create a derivative work according to
   - Mainstream interpretation of copyright law
     - View of the Free Software Foundation (FSF)
     - Recommendation of OSADL

THIS IS FURTHER DESCRIBED IN THE SUPPLEMENT ON "DERIVATIVE WORK" OF THE OSADL OPEN SOURCE POLICY TEMPLATE

Painted ...

Mona Lisa
Leonardo da Vinci
1503 to 1506, until 1517?

… and painted on top

Mona Lisa
Adapted by Marcel Duchamp
1919

… and painted on top

Painting on top of an existing painting is known to create a derivative work. The two works merge in such a way that they can no longer be separated without destroying the new work.

What would be the software equivalence of painting on top of an existing painting?

… and painted on top

Painting on top of an existing painting is known to create a derivative work. The two works merge in such a way that they **can no longer be separated** without destroying the new work.

What would be the software equivalence of painting on top of an existing painting?

Let's have a look at various **scenarios**.

Scenario 1: Modify existing source code

Before applying the modification:

```
int a;
```
Declare the integer variable named "a"

```
a = 1;
```
Assign a value of 1 to the integer variable "a"

Scenario 1: Modify existing source code

After applying the modification:

```
int a;                          Declare the integer variable named "a"
int condition;                  Declare the integer variable named "condition"

if (condition)                  Test whether the variable "condition" is true or not
  a = 2;                        If "condition" is true, assign a value of 2 to the variable "a"
else
  a = 1;                        If "condition" is false, assign a value of 1 to the variable "a"
```

Scenario 1: Modify existing source code

The related patch file to apply the above modification would look like:

```
 int a;
+int condition;

-a = 1;
+if (condition)
+  a = 2;
+else
+  a = 1;
```

Scenario 1: Modify existing source code

Will this result in a derivative work?

Mainstream interpretation of copyright law	View of FSF	Recommendation of OSADL
Yes	Yes	Yes

Scenario 2: Add a function to a single source code file and call it from code of this file

Before applying the modification:

```
int a;
```
Declare the integer variable named "a"

```
a = 1;
```
Assign a value of 1 to the integer variable "a"

Scenario 2: Add a function to a single source code file and call it from code of this file

After applying the modification:

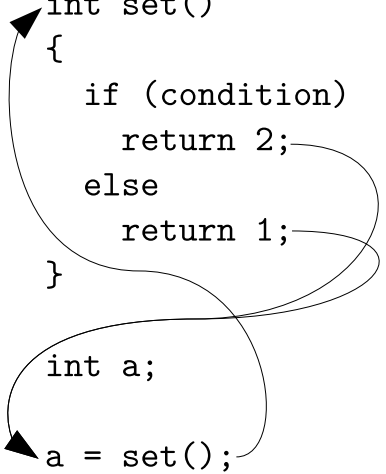
```
int condition;              Declare the integer variable named "condition"
int set()                   Define a function named "set" that returns an integer value
{
  if (condition)            Test whether the variable "condition" is true or not
    return 2;               If "condition" is true, return the value 2
  else
    return 1;               If "condition" is false, return the value 1
}

int a;                      Declare the integer variable named "a"

a = set();                  Call the function "set" and assign the return value to the variable "a"
```

Scenario 2: Add a function to a single source code file and call it from code of this file

After applying the modification:



```
int condition;
int set()
{
  if (condition)
    return 2;
  else
    return 1;
}

int a;

a = set();
```

Declare the integer variable named "condition"

Define a function named "set" that returns an integer value

Test whether the variable "condition" is true or not

If "condition" is true, return the value 2

If "condition" is false, return the value 1

Declare the integer variable named "a"

Call the function "set" and assign the return value to the variable "a"

Scenario 2: Add a function to a single source code file and call it from code of this file

The related patch would look like:

```
+int condition;
+int set()
+{
+  if (condition)
+    return 2;
+  else
+    return 1;
+}

 int a;

-a = 1;
+a = set();
```

Scenario 2: Add a function to a single source code file and call it from code of this file

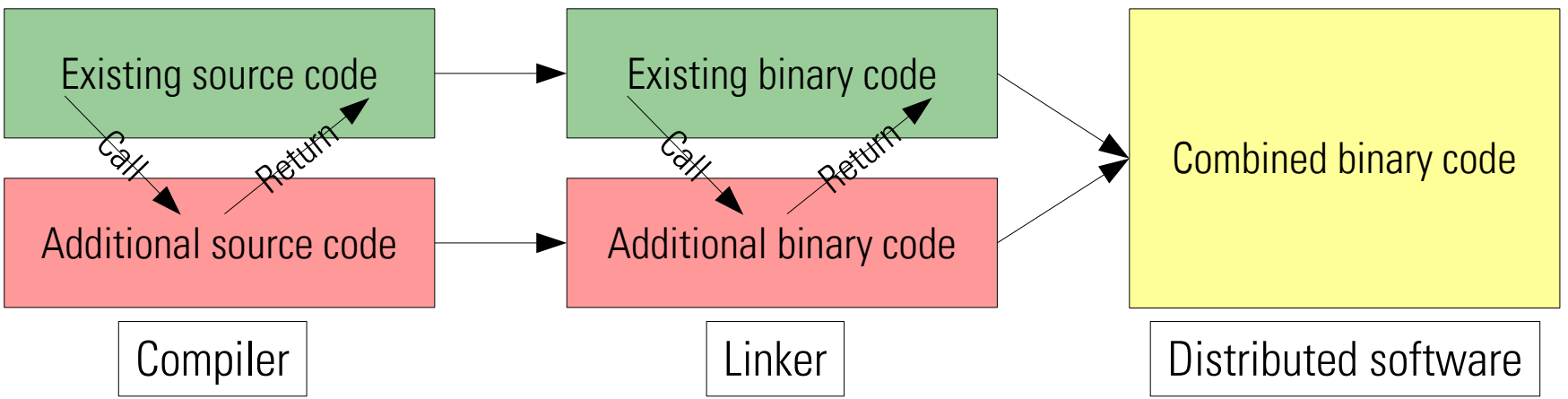
Will this result in a derivative work?

Mainstream interpretation of copyright law	View of FSF	Recommendation of OSADL
Yes	Yes	Yes

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

This scenario can be realized in two different ways:

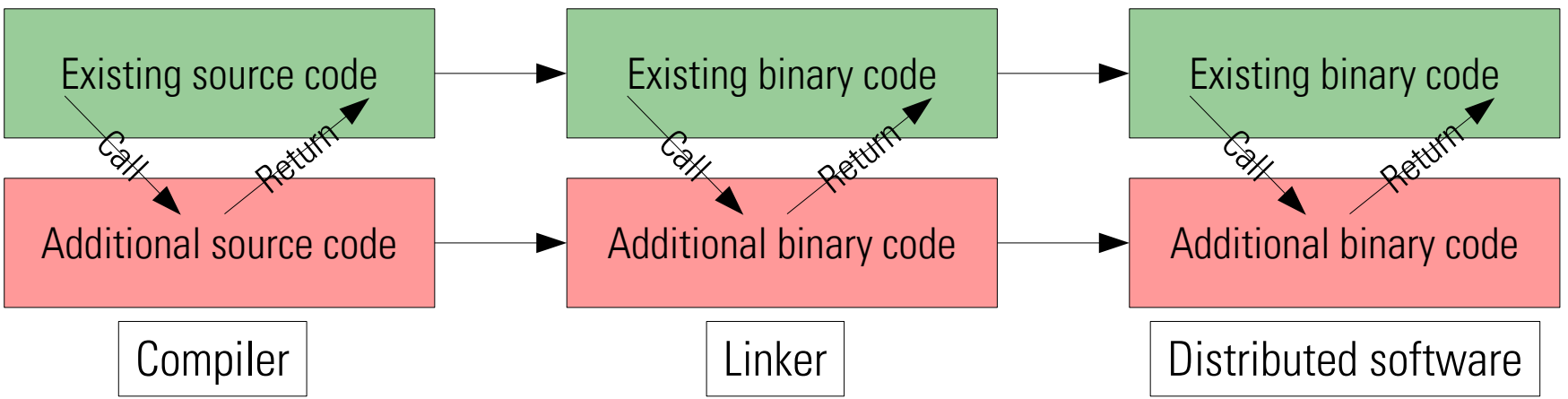
1. The additional and the existing source code file are statically linked together and cannot be separated at run time.



Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

This scenario can be realized in two different ways:

2. The additional source code file is organized as a software library and will dynamically be linked only at run time so the two files will stay separate.



Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

The way of providing the function, i.e. either inseparably in the same source code file or separately in another file, has an important implication, as the various Open Source licenses differ from each other. For example, a **particular proprietary software** may

- be combined with another software irrespective of whether the components may be separated later on or not, if the other software is under a **"permissive license"** such as a BSD-type license

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

The way of providing the function, i.e. either inseparably in the same source code file or separately in another file, has an important implication, as the various Open Source licenses differ from each other. For example, a **particular proprietary software** may

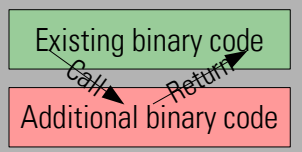
- be combined with another software irrespective of whether the components may be separated later on or not, if the other software is under a **"permissive license"** such as a BSD-type license,
- be combined with another software, if the components always can be separated and exchanged individually, if the other software is under a **"license with restricted copyleft"** such as the Lesser GNU Public License (LGPL)

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

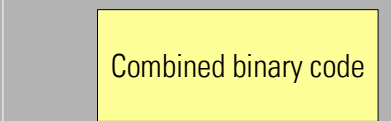
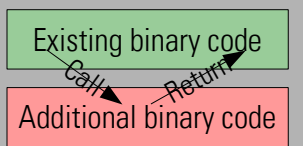
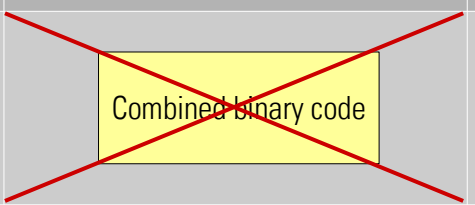
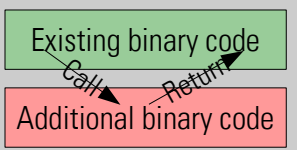
The way of providing the function, i.e. either inseparably in the same source code file or separately in another file, has an important implication, as the various Open Source licenses differ from each other. For example, a **particular proprietary software** may

- be combined with another software irrespective of whether the components may be separated later on or not, if the other software is under a **"permissive license"** such as a BSD-type license,
- be combined with another software, if the components always can be separated and exchanged individually, if the other software is under a **"license with restricted copyleft"** such as the Lesser GNU Public License (LGPL),
- never be combined with another software, if the other software is under a **"license with strong copyleft"** such as the GNU General Public License (GPL).

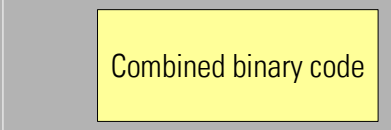
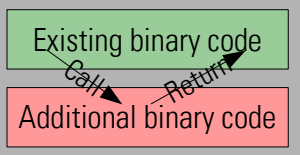
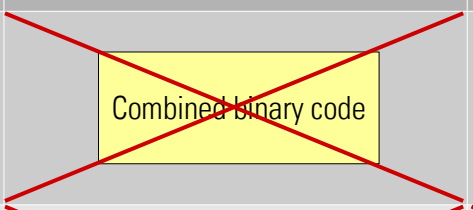
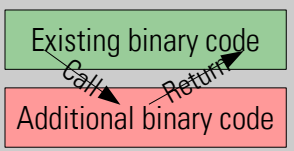
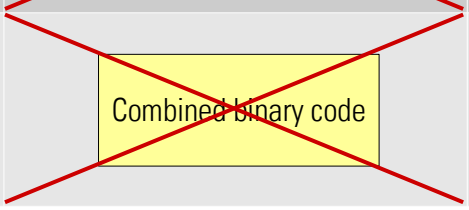
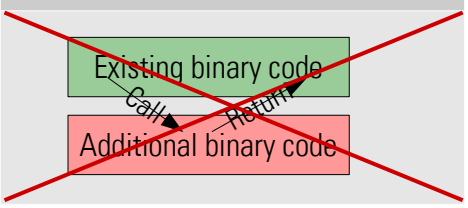
Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

Proprietary software plus software under a permissive license, e.g. BSD	Combined binary code	
Proprietary software plus software under a license with restricted copyleft, e.g. LGPL		
Proprietary software plus software under a license with strong copyleft, e.g. GPL		

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

Proprietary software plus software under a permissive license, e.g. BSD		
Proprietary software plus software under a license with restricted copyleft, e.g. LGPL		
Proprietary software plus software under a license with strong copyleft, e.g. GPL		

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

Proprietary software plus software under a permissive license, e.g. BSD		
Proprietary software plus software under a license with restricted copyleft, e.g. LGPL		
Proprietary software plus software under a license with strong copyleft, e.g. GPL		

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

Before applying the modification:

```
int a;
```
Declare the integer variable named "a"

```
a = 1;
```
Assign a value of 1 to the integer variable "a"

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

After applying the modification:
Code in a file named *code.c*:

```
int a;                  Declare the integer variable named "a"

a = set();              Call the function "set" and assign the return value to the variable "a"
```

Code in a separate file named *set.c*:

```
int condition;          Declare the integer variable named "condition"
int set()               Define a function named "set" that returns an integer value
{
  if (condition)        Test whether the variable "condition" is true or not
    return 2;           If "condition" is true, return the value 2
  else
    return 1;           If "condition" is false, return the value 1
}
```

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

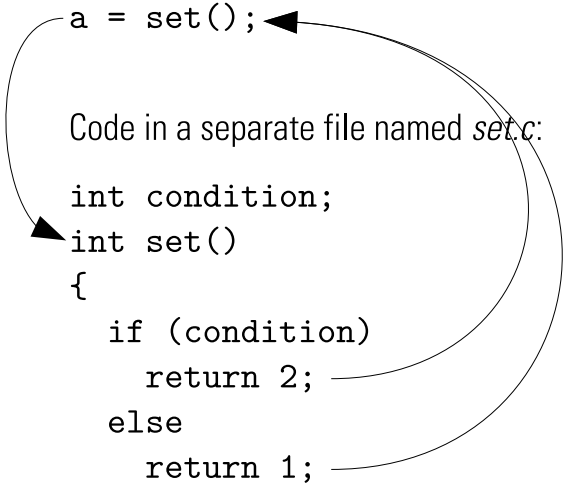
After applying the modification:
Code in a file named *code.c*:

```
int a;                          Declare the integer variable named "a"

a = set();                      Call the function "set" and assign the return value to the variable "a"
```

Code in a separate file named *set.c*:

```
int condition;                  Declare the integer variable named "condition"
int set()                       Define a function named "set" that returns an integer value
{
  if (condition)                Test whether the variable "condition" is true or not
    return 2;                   If "condition" is true, return the value 2
  else
    return 1;                   If "condition" is false, return the value 1
}
```



Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

The related patch would look like:

```
Index: code.c
```
This indicates that the following patch instructions relate to the file "code.c"

```
===================================================================
```

```
--- code.c.orig
```
This indicates that there was an existing original file "code.c" to be modified

```
+++ code.c
```
This indicates that the modified file will be "code.c"

```
@@ -1,3 +1,3 @@
```
This indicates that the patch deals with lines 1 to 3

```
 int a;


-a = 1;
+a = set();
```
Here come the actual patch instructions

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

```
Index: set.c
```
This indicates that the following patch instructions relate to the file "set.c"

```
===================================================================
--- /dev/null
```
This indicates that there was no existing file

```
+++ set.c
```
This indicates that the new file will be created as "set.c"

```
@@ -0,0 +1,9 @@
```
This indicates that the patch found 0 lines and added lines 1 to 9

```
+int condition;
+int set()
+{
+  if (condition)
+    return 2;
+  else
+    return 1;
+}
```
Here come the actual patch instructions

Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

Will this result in a derivative work, if **combined in a single file**?

Mainstream interpretation of copyright law	View of FSF	Recommendation of OSADL
Yes	Yes	Yes

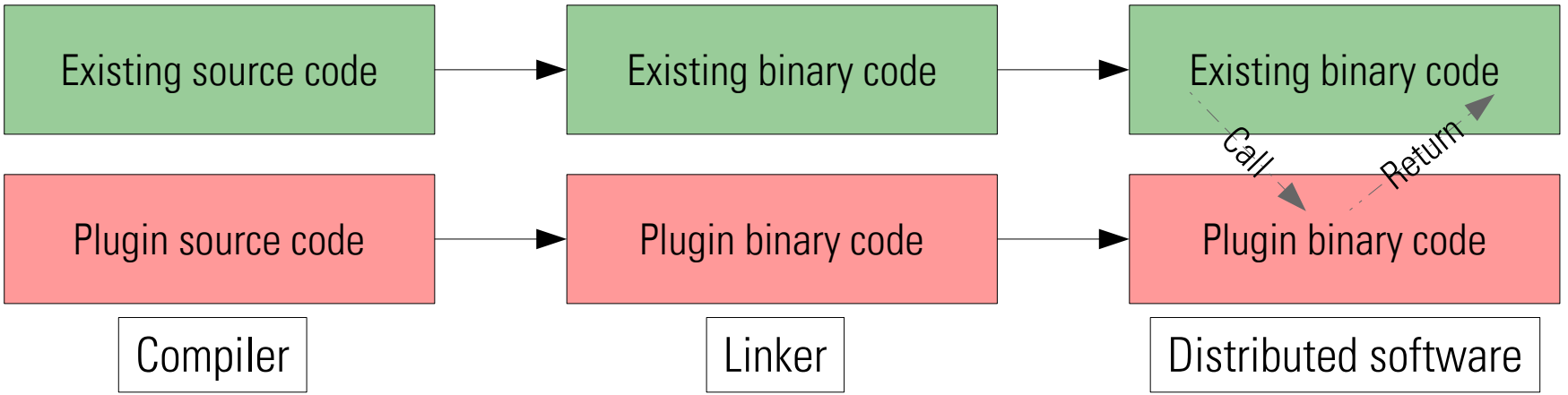
Scenario 3: Provide an additional source code file with a new function and add a call to that function to existing source code

Will this result in a derivative work, if **distributed in separate files**?

Mainstream interpretation of copyright law	View of FSF	Recommendation of OSADL
Unknown	Yes	Yes

Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

This scenario is similar to scenario 3 except that the existing code may run even if the additional code is not available. It may not have the entire functionality, though. The plugin may optionally be connected at run-time via function pointer.



Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

C language:

```
int set()                 Define a function named "set" that returns an integer value
{
  return 1;               Return the value 1
}

int a = set();            Call the function "set" and use the return value
```

Normal call to a static function

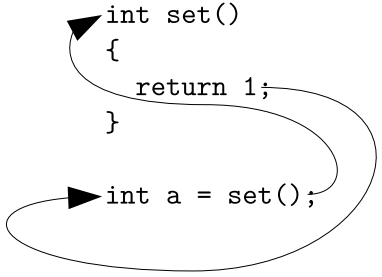
The above source code in Intel assembly language:

```
<set>:                    Define the label "set", this is the start of the subroutine
  mov $0x1,%eax           Load the value 1 to the machine register "eax"
  ret                     Return from subroutine, this is the end of the subroutine

  call <set>              Call the subroutine "set" and return from it
                           (cannot be another address, since "set" is a static  address)
  move %eax, ...          Use the return value for future computation
```

Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

C language:



```
int set()                    Define a function named "set" that returns an integer value
{
  return 1;                  Return the value 1
}

int a = set();               Call the function "set" and use the return value
```

Normal call to a static function

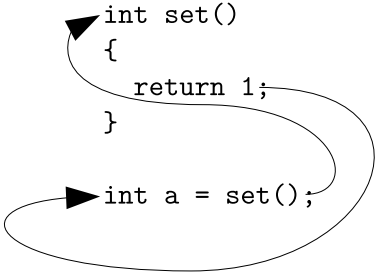
The above source code in Intel assembly language:

```
<set>:                       Define the label "set", this is the start of the subroutine
  mov $0x1,%eax              Load the value 1 to the machine register "eax"
  ret                        Return from subroutine, this is the end of the subroutine

  call <set>                 Call the subroutine "set" and return from it
                              (cannot be another address, since "set" is a static  address)
  move %eax, ...             Use the return value for future computation
```

Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

C language:



```
int set()
{
  return 1;
}

int a = set();
```

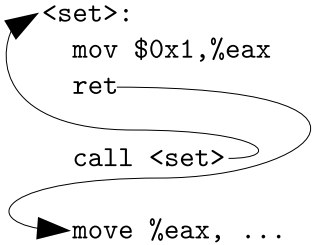
Define a function named "set" that returns an integer value

Return the value 1

Call the function "set" and use the return value

Normal call to a static function

The above source code in Intel assembly language:



```
<set>:
  mov $0x1,%eax
  ret

  call <set>

move %eax, ...
```

Define the label "set", this is the start of the subroutine
Load the value 1 to the machine register "eax"
Return from subroutine, this is the end of the subroutine

Call the subroutine "set" and return from it
(cannot be another address, since "set" is a static address)
Use the return value for future computation

Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

C language:

```
int set()                    Define a function named "set" that returns an integer value
{                              This function may be exchanged by adifferent plugin
  return 1;                  Return the value 1
}

int (*func)() = set;         Declare the variable function pointer "func" and assign
                               the start address of the function "set" to it
int a = func();              Call the function and use the return value
```

Call to a function via variable pointer

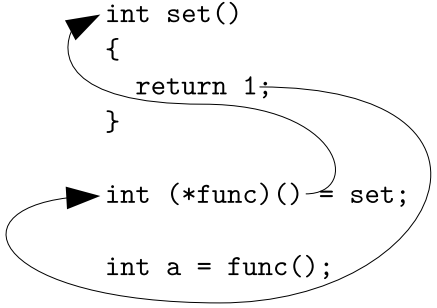
The above source code in Intel assembly language:

```
<set>:                       Define the label "set", this is the start of the subroutine
  mov $0x1,%eax              Load the value 1 to the machine register "eax"
  ret                        Return from subroutine, this is the end of the subroutine

  mov <set>,%rdx             Store the address of the function "set" to the machine register "rdx"
                               (could be another address, if desired)
  call *%rdx                 Evaluate the address of the machine register "rdx", branch to and
                               return from it
  move %eax, ...             Use the return value for future computation
```

Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

C language:



```
int set()
{
  return 1;
}

int (*func)() = set;

int a = func();
```

Define a function named "set" that returns an integer value
This function may be exchanged by adifferent plugin
Return the value 1

Declare the variable function pointer "func" and assign the start address of the function "set" to it
Call the function and use the return value

Call to a function via variable pointer

The above source code in Intel assembly language:

```
<set>:
  mov $0x1,%eax
  ret

  mov <set>,%rdx

  call *%rdx

  move %eax, ...
```

Define the label "set", this is the start of the subroutine
Load the value 1 to the machine register "eax"
Return from subroutine, this is the end of the subroutine

Store the address of the function "set" to the machine register "rdx" (could be another address, if desired)

Evaluate the address of the machine register "rdx", branch to and return from it

Use the return value for future computation

Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

C language:

```
int set()
{
   return 1;
}

int (*func)() = set;

int a = func();
```

Define a function named "set" that returns an integer value
This function may be exchanged by adifferent plugin
Return the value 1

Declare the variable function pointer "func" and assign the start address of the function "set" to it
Call the function and use the return value

Call to a function via variable pointer

The above source code in Intel assembly language:

```
<set>:
   mov $0x1,%eax
   ret

   mov <set>,%rdx

   call *%rdx

move %eax, ...
```

Define the label "set", this is the start of the subroutine
Load the value 1 to the machine register "eax"
Return from subroutine, this is the end of the subroutine

Store the address of the function "set" to the machine register "rdx" (could be another address, if desired)
Evaluate the address of the machine register "rdx", branch to and return from it
Use the return value for future computation

Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

C language:

```
int set()                    Define a function named "set" that returns an integer value
{
  return 1;                  Return the value 1
}

int a = set();               Call the function "set" and use the return value
```

Normal call to a static function

The above source code in Intel assembly language:

```
<set>:                       Define the label "set", this is the start of the subroutine
  mov $0x1,%eax              Load the value 1 to the machine register "eax"
  ret                        Return from subroutine, this is the end of the subroutine

  call <set>                 Call the subroutine "set" and return from it
                              (cannot be another address, since "set" is a static  address)
  move %eax, ...             Use the return value for future computation
```

Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

C language:

```
int set()                       Define a function named "set" that returns an integer value
{                                 This function may be exchanged by adifferent plugin
  return 1;                     Return the value 1
}

int (*func)() = set;            Declare the variable function pointer "func" and assign
                                  the start address of the function "set" to it
int a = func();                 Call the function and use the return value
```

Call to a function via variable pointer

The mechanism is identical

The above source code in Intel assembly language:

```
<set>:                          Define the label "set", this is the start of the subroutine
  mov $0x1,%eax                 Load the value 1 to the machine register "eax"
  ret                           Return from subroutine, this is the end of the subroutine

  mov <set>,%rdx                Store the address of the function "set" to the machine register "rdx"
                                  (could be another address, if desired)
  call *%rdx                    Evaluate the address of the machine register "rdx", branch to and
                                  return from it
  move %eax, ...                Use the return value for future computation
```

Scenario 4: Provide a so-called plugin with a new function and call that function from existing source code via function pointer at run-time

Will this result in a derivative work, if the call uses a function pointer?

Mainstream interpretation of copyright law	View of FSF	Recommendation of OSADL
Unknown	Yes	Yes

Scenario 5: Any connection between software components other than by function call

a) User-space program and operating system kernel

b) Compiler and source code

c) Interpreter and script

d) Programs connected via local Unix socket

e) Programs connected via local or remote network connection

f) Software connected via local bus (e.g. PCIe) interface

g) Programs connected via remote bus (e.g. USB) interface

Components interact only temporarily, can be separated at any time and may work independently elsewhere.

Scenario 5: Any connection between software components other than by function call

Will this result in a derivative work?

Mainstream interpretation of copyright law	Assumed view of FSF	Recommendation of OSADL
Unknown	No	No

Scenario 5: Any connection between software components other than by function call

Will this result in a derivative work?

Mainstream interpretation of copyright law	Assumed view of FSF	Recommendation of OSADL
Unknown	No[a]	No

[a] However, if two independent programs establish intimate communication by sharing complex data structures, or shipping them back and forth they might be considered a derivative work.

Rule of thumb (exceptions may apply)

- A derivative work of a software that may trigger licensing issues is created when
  - existing code is modified inline *or*
  - a call to a newly provided function is added locally or in a separate file
- A mere aggregate of software components that does not trigger licensing issues is created, if
  - existing code is not modified *and*
  - the software components do not call each other's functions