Problem

- GCC makes different optimizations during compilation
- Problem? Could be…
  - Depending on how live patches are constructed and generated
  - Some are harmless, some could be fatal if one is not careful
- As of GCC 6.2
- Only a subset of optimizations. There is more.

GCC optimizations

- Inlining
- `-fpartial-inlining`
  - Inline parts of the function
- `-fipa-sra`
  - Removal of unused parameters, passing by reference → passing by value etc.
- `-fipa-cp`
  - Interprocedural constant propagation. Optimizes functions if values passed to them are constants.
  - Even several clones of a function are possible if a set of values is limited.

GCC optimizations

- `-fipa-pure-const`
  - Discover which functions are pure or constant.
  - GCC can eliminate calls to such functions, memory accesses can be removed etc.
  - What if a function is no longer pure const with a fix applied?
- `-fipa-icf`
  - Identical code folding for functions and read-only variables. Replaces a function with an equivalent one.
  - Problem for stack unwinder too

GCC optimizations

- Code eliminations

```
int global;
int foo(void) { return 22; }
int main(void)
{
  int i;
  global = 1;
  i = foo();
  if (global == 2) return 11;
  return 33;
}
```

```
int foo(void)
{
  global = 2;
  return 22;
}
```

- But generally always when a function somehow affects another one.

GCC optimizations

- `-fipa-ra`
  - Use caller save registers for allocation if those registers are not used by any called function. In that case it is not necessary to save and restore them around calls.
  - Currently disabled thanks to -pg option (gcc bug #64287), but this can change easily
  - Reportedly dubious gain for the kernel
  - Only a small fraction of functions affected (~0.5 %)