Dyninst patching:

Basic Block - based:

- ~~[illegible]~~ flagged as modified
  ⇒ + content moved to newly created section ~~[illegible]~~
  + old BB points to new BB (*)
  + new BB fallthrough is essentially a backjump @ start of fallthrough (end of new chain)

using snippets that can be place before ~~[illegible]~~ after or instead of any instr within a BB

(*) Dyninst 9.2 uses the following options:
(generic Branching!)

a) "distance" $(BB_{old}, BB_{new}) \leq 2^8$ ~~[illegible]~~
   0x EB $disp (2 byte)
   8 bit displacement (RIP based)
   ⇒ bias usually too big

b) "distance" $(BB_{old}, BB_{new}) - 5 \leq 2^{32}$
   0x E9 $disp (5 byte)
   32 bit displacement (RIP based)
   bias usually below 2^{32} byte = 4GB
   ⇒ should be the normal case

c) ~~[illegible]~~
   32 bit address ?
   [push imm] 0x 68 $address
   [jump] ret
   } 6 byte
   32bit fn call ⇒ not applied, as we have 64bit bias (unnecessarily)
   weight interf ⇒ will break CFI

d) 64 bit address
   [push imm] 0x68 $low^{32} address
   [fixup stack + push high] 0x67 0x44 0x24 0x04 $high^{32} address
   [jump] ret
   } 14 byte

~~Callsites~~

Calltargets: goal: store data for access through any callsite

idea: store data within the entryblock of a calltarget using the Dyninst padding method:

a) mark the entryBlock of a to tagfix as modified
b) annotate entryBlock with our data (5 bytes)
↳ Codegen will apply patches (usually 5 byte for 32 bit disp jump + 5 bytes payload).

⇒ access for the callsite via a const offset from target entry addr:

($target +5) is data start

Callsites: goal: ~~access~~ of data and check against own signature

idea: ~~mask~~ mask the data with our own signature ~~and~~ then or each byte should the byte not be 0x FF, we can assume non matching