# Comparison between EH-Digger and existing tools

EH-Digger starts from an error handling, and then learns why the error occurs and when it has to be handled using inter-procedural analysis. Existing approaches start from an API, and learn if the API requires error handling nearby. According to our research (Sec. 2.2 Finding 1), 43.3% of the error-handling code snippets are located in different functions with their corresponding fault statements. These cases may lead to three limitations.

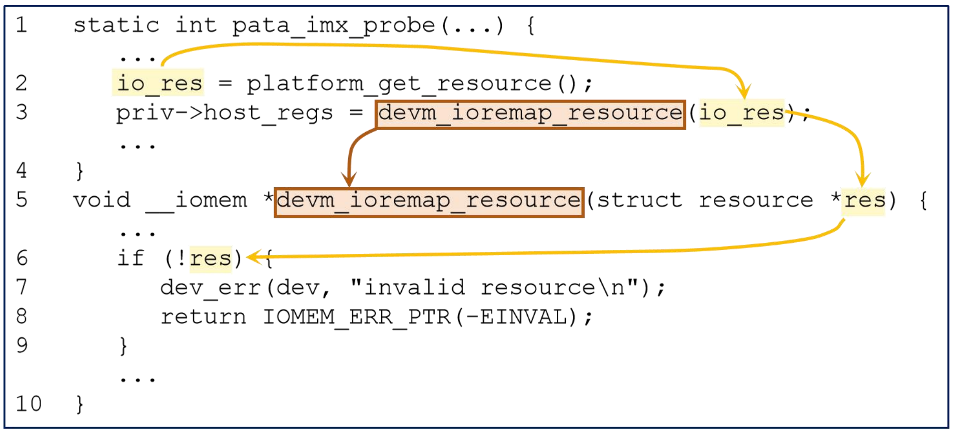
## 1. False Negative

During the learning phase, existing approaches could miss error-handling patterns. **38.3%** of the bugs found by EH-Digger could not be found by any existing approach (Sec. 4.2). Fig.1 shows a historical bug successfully identified by EH-Digger.

```
    static int unimac_mdio_probe(...) {
        ...
        struct resource *r;
        r = platform_get_resource(...);
  +     if (!r)
  +         return -EINVAL;
        ...
    }
```

Fig.1 Historical bug in Linux kernel (Linux 4.14 drivers/net/phy/mdio-bcm-unimac.c)

This bug is missed by all existing approaches. As shown in Fig.2, the return value of platform_get_resource is examined and handled in devm_ioremap_resource. Existing approaches are failed to learn this pattern, and thus miss the bug.



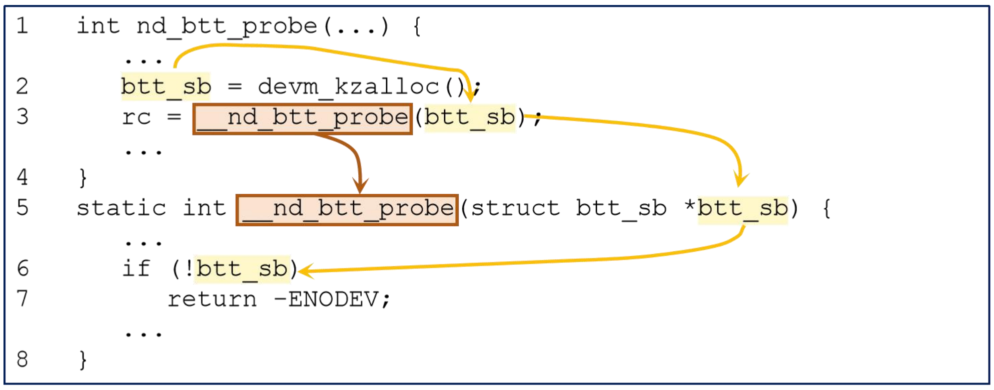
```
static int pata_imx_probe(...) {
    ...
    io_res = platform_get_resource();
    priv->host_regs = devm_ioremap_resource(io_res);
    ...
}
void __iomem *devm_ioremap_resource(struct resource *res) {
    ...
    if (!res) {
        dev_err(dev, "invalid resource\n");
        return IOMEM_ERR_PTR(-EINVAL);
    }
    ...
}
```

Fig.2 Frequent usage of platform_get_resource

## 2. False Positive

During the detection phase, an error may already have an inter-procedural handling. Existing tools may report false positives if they cannot link the handling to the corresponding error. The precision of EH-Digger is **19.4%** higher than general tools like EH-Miner (Sec. 4.2.2), and is similar to Err-hunter which is specifically designed for Linux kernel.

We give an example of a false positive reported by EH-Miner in Fig. 3. In the Linux kernel, the return value of devm_kzalloc usually requires error handling, and thus existing learning-based approaches can learn its error-handling pattern. However, sometimes devm_kzalloc is not handled at the place where it is invoked. For example, In Fig.3, the return value of devm_kzalloc is handled in __nd_btt_probe (line 6-7). Since there is no error handling in nd_btt_probe (line 1-4), existing approaches will assume that the return value of devm_kzalloc is not handled, and report it as a bug. Such cases will lead to false positives of existing approaches.

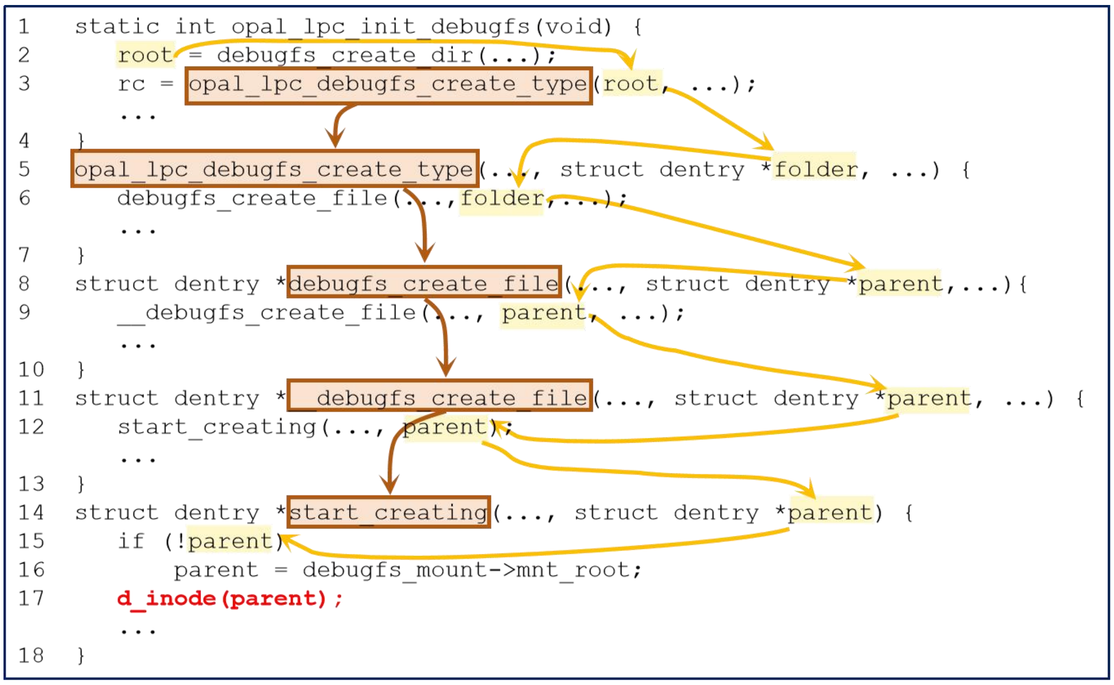


```
1    int nd_btt_probe(...) {
        ...
2       btt_sb = devm_kzalloc();
3       rc = __nd_btt_probe(btt_sb);
        ...
4    }
5    static int __nd_btt_probe(struct btt_sb *btt_sb) {
        ...
6       if (!btt_sb)
7          return -ENODEV;
        ...
8    }
```

Fig.3 False positive caused by inter-procedural handling

## 3. Deadline of error handling

The differences between EH-Digger and existing tools are not just inter-procedural and intra-procedural. Existing approaches may learn that there should be an error handling after someplace, but are hard to know the deadline of the handling (i.e., the handling should be performed before someplace).



```
1    static int opal_lpc_init_debugfs(void) {
2       root = debugfs_create_dir(...);
3       rc = opal_lpc_debugfs_create_type(root, ...);
        ...
4    }
5    opal_lpc_debugfs_create_type(..., struct dentry *folder, ...) {
6       debugfs_create_file(...,folder,...);
        ...
7    }
8    struct dentry *debugfs_create_file(..., struct dentry *parent,...){
9       __debugfs_create_file(..., parent, ...);
        ...
10   }
11   struct dentry *__debugfs_create_file(..., struct dentry *parent, ...) {
12      start_creating(..., parent);
        ...
13   }
14   struct dentry *start_creating(..., struct dentry *parent) {
15      if (!parent)
16         parent = debugfs_mount->mnt_root;
17      d_inode(parent);
        ...
18   }
```

Fig.4 Deadline of debugfs_create_dir

For example, in Fig.4, the return value of debugfs_create_dir propagates through 4 functions (line 3, 6, 9, 12). It does not need to be handled until d_inode is called. In such a complex situation, existing approaches cannot determine the deadline of error handling, so they usually assume that the error needs to be handled as soon as it occurs, thus often producing false positives.