BesFS: A POSIX Filesystem for Enclaves with a Mechanized Safety Proof

Shweta Shinde*  Shengyi Wang  Pinghai Yuan
Aquinas Hobor†  Abhik Roychoudhury  Prateek Saxena

*UC Berkeley  National University of Singapore  †Yale-NUS College
Trusted Execution Environments

- E.g., Intel SGX [HASP’13], Keystone [EuroSys’20]
Interface Attacks on Existing Frameworks

```c
int enc_untrusted_open(const char *path_name, int flags) {
    uint32_t mode = 0;
    int result;
    sgx_status_t status = ocall_enc_untrusted_open(&result,
            path_name, flags, mode);
    if (status != SGX_SUCCESS) {
        errno = EINTR;
        return -1;
    }
    return result;
}
```

`fopen`: Google Asylo

```c
static SGX_FILE* sgx_fopen_internal
    (const char* filename, const char* mode) {
    protected_fs_file* file = NULL;
    if (filename == NULL || mode == NULL) {
        errno = EINVAL;
        return NULL;
    }
    ...
}
```

`fopen`: Intel SDK
Attack Potency and Existing Solutions

**[CCS’20]**

**A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes**

Jo Van Bulck
imec-DistriNet, KU Leuven
jo.vanbulck@cs.kuleuven.be

Abdulla Aldosery
The University of Birmingham, UK
axa1170@student.bham.ac.uk

David Oswald
The University of Birmingham, UK
d.f.oswald@cs.bham.ac.uk

Flavio D. Garcia
The University of Birmingham, UK
f.garcia@cs.bham.ac.uk

Eduard Marin
The University of Birmingham, UK
e.marin@cs.bham.ac.uk

Frank Piessens
imec-DistriNet, KU Leuven
frank.piessens@cs.kuleuven.be

**Potential Defenses:**
- Narrow & limited interface
- Input-output sanitization
- Compiler-based checks

---

**[ASPLOS’20]**

**COIN Attacks: On Insecurity of Enclave Untrusted Interfaces in SGX**

Mustakimur Rahman Khandaker
mrk15e@my.fsu.edu
Florida State University

Zhi Wang
zwang@cs.fsu.edu

Yueqiang Cheng
chengyueqiang@baidu.com
Baidu Security

Tao Wei
lenx@baidu.com

**Necessary but incomplete---No guarantees**
A Formal Verification Approach

- Safe specification
  - Encapsulates accepted behavior of the interface (e.g., open)

- A machine checked interface
  - Guarantees detection of specification violation (e.g., malicious return value)

Complicated verification problem: Adversary OS can deviate arbitrarily
A Formal Verification Approach: How to scale to a large interface (e.g., POSIX)?

The scalability challenge:
- Specification for safe behavior for the entire POSIX API
- Proving safe implementation
  - entire libc (glibc, musl)
  - filesystem (ext4)
BesFS Interface: Designing Scalable Specification

• Our Approach
  • 15 core APIs: e.g., open, close, read, write
  • Allow to execute any sequence of these while maintaining safety property

• Can be composed to express higher-level interfaces
  • e.g., fwrite can be composed with write and fstat
  • Created 22 auxiliary APIs witnessed in applications
Designing Specification for BesFS Interface

State Safety Properties

• All the file and directory paths are unique
• All open file IDs have to be registered
• All open file IDs have unique entries
• No overlaps between virtual addresses
• Current cursor position can only take values between 0 and EOF

True for all states

Transition Safety Properties

\[
fs\_close\ (h:Id) \rightarrow (e:ERROR)
\]

<table>
<thead>
<tr>
<th>Pre-condition ( Pre_i(S) )</th>
<th>Transition Relation ( \tau_i(S,S') )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \exists o, o_{Id} = h \land o \in O )</td>
<td>( S' = S[O\mid O - {o}] \land e = ESucc )</td>
</tr>
</tbody>
</table>

True before and after a call
Scaling the Specification Safety Proof of BesFS

State Transition Safety

Given a good state $S$ satisfying pre-conditions $\text{pre}_i$, then if we execute $f_i$ to reach state $S'$, then $S'$ is always a good state and relation between $S$ and $S'$ is valid according to the transition relation $\tau_i$

Sequential Composition Safety

Given a good initial state $S_0$ subject to a sequence of transitions $\tau_{m1}, \ldots, \tau_{mn}$ always produces a good final state $S_n$
Proving Implementation Safety

- Employ the state and transition safety checks
- Encryption
- Data structures to keep state
  - File and directory layout
  - Memory map
  - File handles
  - Permissions and sizes
  - Page hashes

• Implementation is proof checked
  • In higher level language (e.g., Gallina+Coq)
Evaluation Goals

• **TCB**: Do the checks increase the enclave code size?

• **Expressiveness**: Is the subset enough to run interesting applications?

• **Compatibility**: Do existing systems adhere to BesFS specifications?

• **Performance**: What is the cost of machine-checked security guarantees?
## Evaluation I: Small TCB

<table>
<thead>
<tr>
<th>Component</th>
<th>Language</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Specification &amp; Machine-proved Implementation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coq definitions &amp; Proofs</td>
<td>Gallina</td>
<td>4625</td>
</tr>
<tr>
<td><strong>Hand-coded C Implementation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>C</td>
<td>863</td>
</tr>
<tr>
<td>External Calls</td>
<td>C</td>
<td>469</td>
</tr>
<tr>
<td>SGX Utils</td>
<td>C</td>
<td>117</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>1449</td>
</tr>
</tbody>
</table>

167 lemmas
2 main theorems
### Evaluation II: Expressiveness

<table>
<thead>
<tr>
<th>Libc API</th>
<th>LOC</th>
<th>BextFS Core API used for composition of LibC API</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>7</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fread</td>
<td>25</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fscanf</td>
<td>34</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fwrite</td>
<td>12</td>
<td><img src="#" alt="Checkmark" /> <img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>write</td>
<td>20</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fprintf</td>
<td>15</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fopen</td>
<td>78</td>
<td><img src="#" alt="Checkmark" /> <img src="#" alt="Checkmark" /> <img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>open</td>
<td>60</td>
<td><img src="#" alt="Checkmark" /> <img src="#" alt="Checkmark" /> <img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fclose</td>
<td>9</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>close</td>
<td>17</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fseek</td>
<td>31</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fseek</td>
<td>39</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>rewind</td>
<td>5</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>creat</td>
<td>30</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>mkdir</td>
<td>25</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>unlink</td>
<td>21</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>chmod</td>
<td>23</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>truncate</td>
<td>5</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>ftell</td>
<td>12</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fgetc</td>
<td>9</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>fgets</td>
<td>25</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>readdir</td>
<td>10</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
</tbody>
</table>

- **fwrite** composed from `write` and `fstat`

- **37 Total APIs:**
  - **22 additional composed from 15 core APIs**
## Evaluation III: Compatibility

<table>
<thead>
<tr>
<th>LibC Calls</th>
<th>SPEC CINT 2006</th>
<th>FSCQ</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>astar</td>
<td>mcf</td>
<td>bzip2</td>
</tr>
<tr>
<td>BESFS Core Calls</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>open</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>read</td>
<td>27</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>write</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lseek</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>remove</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>close</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mkdir</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BESFS Auxiliary Calls</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fopen</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>fread</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fwrite</td>
<td>0</td>
<td>1035</td>
<td>0</td>
</tr>
<tr>
<td>fgets</td>
<td>0</td>
<td>90435</td>
<td>0</td>
</tr>
<tr>
<td>fscanf</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fprintf</td>
<td>0</td>
<td>5985</td>
<td>0</td>
</tr>
<tr>
<td>fseek</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ftell</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rewind</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Unsafe Calls</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fsync</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rename</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>47</td>
<td>97457</td>
<td>6</td>
</tr>
</tbody>
</table>

Protects 235000/235008 APIs in our benchmarks
Evaluation IV: Performance

CPU-intensive: ~12.22%

IO-intensive: ~480%
Do Proofs Help in Eliminating Bugs?

• Example 1: seek Specification Bug
  • if pos < size

• Example 2: write Implementation Bug
  • Variable scope overlaps

• Example 3: Panoply & Intel SGX SDK Bugs
  • enclave stack is corrupted for large sizes

• Example 4: Panoply Error Code Bugs
  • 7 distinct functions where PANOPLY’s error codes were incorrect
Project Page & Contact

BesFS Webpage
(Coq Spec, Implementation, and Proofs)

https://shwetasshinde24.github.io/BesFS

Shweta Shinde
shweta.shivajishinde@inf.eth.ch

@shw3ta_shinde