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

Shweta Shinde[*]
Aquinas Hobor[†]

Shengyi Wang
Abhik Roychoudhury

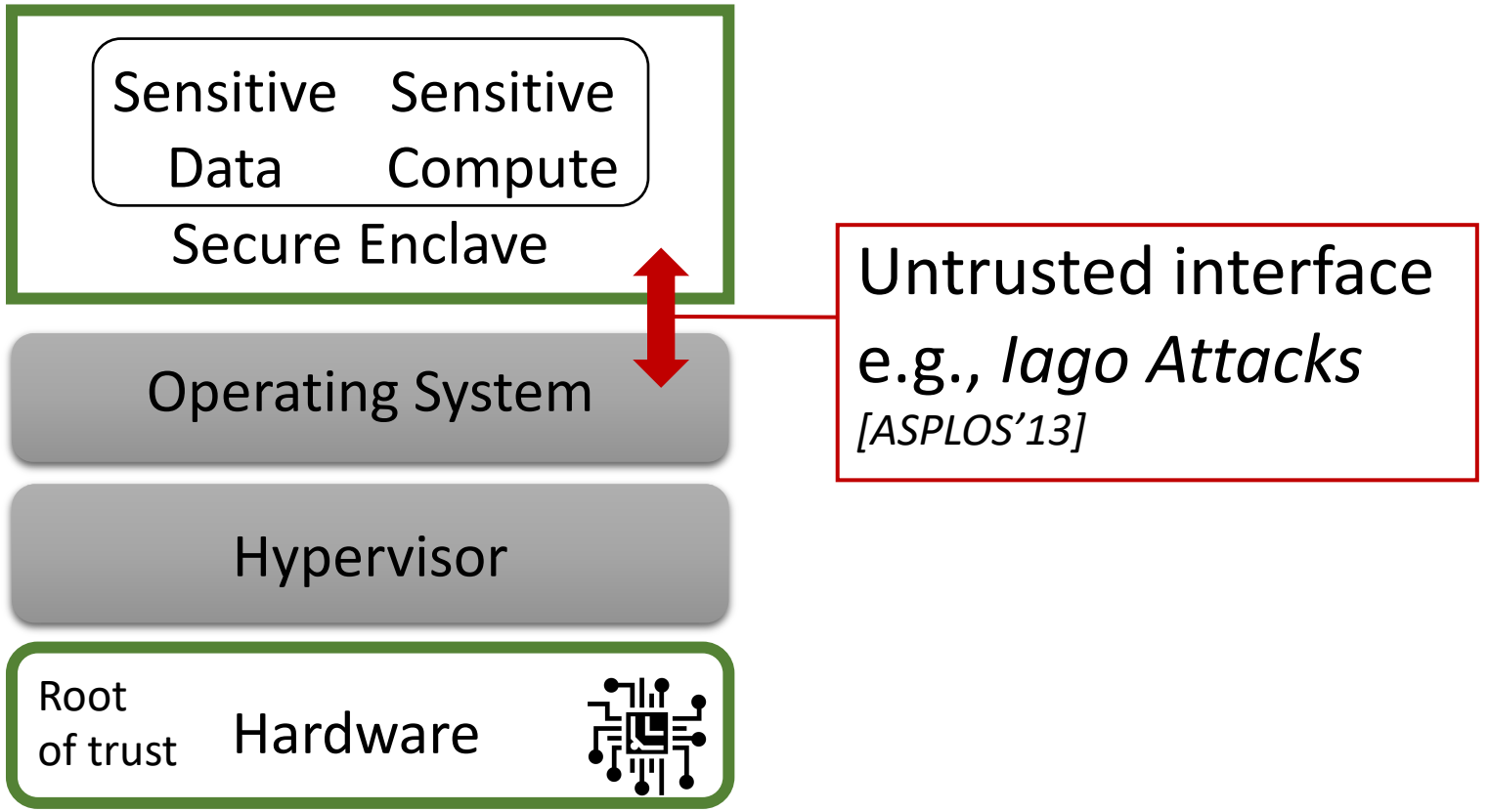
Pinghai Yuan
Prateek Saxena

[*]*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

```
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

```
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

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

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

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

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

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

[ASPLOS'20]

COIN Attacks: On Insecurity of Enclave Untrusted Interfaces in SGX

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

Yueqiang Cheng
chengyueqiang@baidu.com
Baidu Security

Zhi Wang
zwang@cs.fsu.edu

Tao Wei
lenx@baidu.com

Potential Defenses:

- Narrow & limited interface
- Input-output sanitization
- Compiler-based checks

Necessary but incomplete---
No guarantees

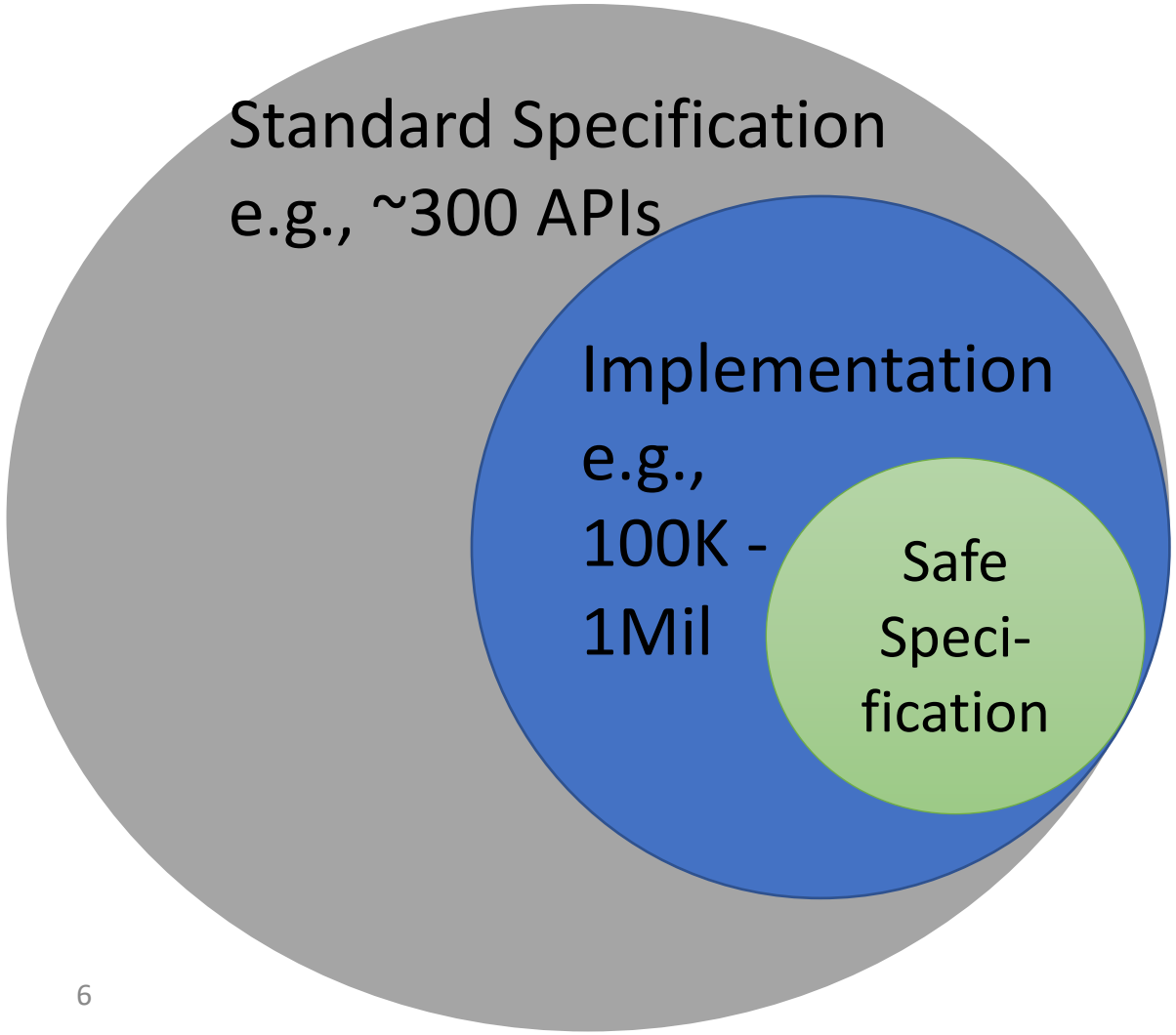
A Formal Verification Approach

- 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

$\texttt{fs\_close}\ (h{:}\,Id) \rightarrow (e{:}\,ERROR)$

Pre-condition $Pre_i(S)$	Transition Relation $\tau_i(S, S')$
$\exists o, o_{Id} = h\ \wedge o \in O$	$S' = S[O \mid O\ - \{o\}]\ \wedge e = ESucc$

True before and after a call

Scaling the Specification Safety Proof of BesFS

State Transition Safety

Given a good state S satisfying pre-conditions 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 τ_i

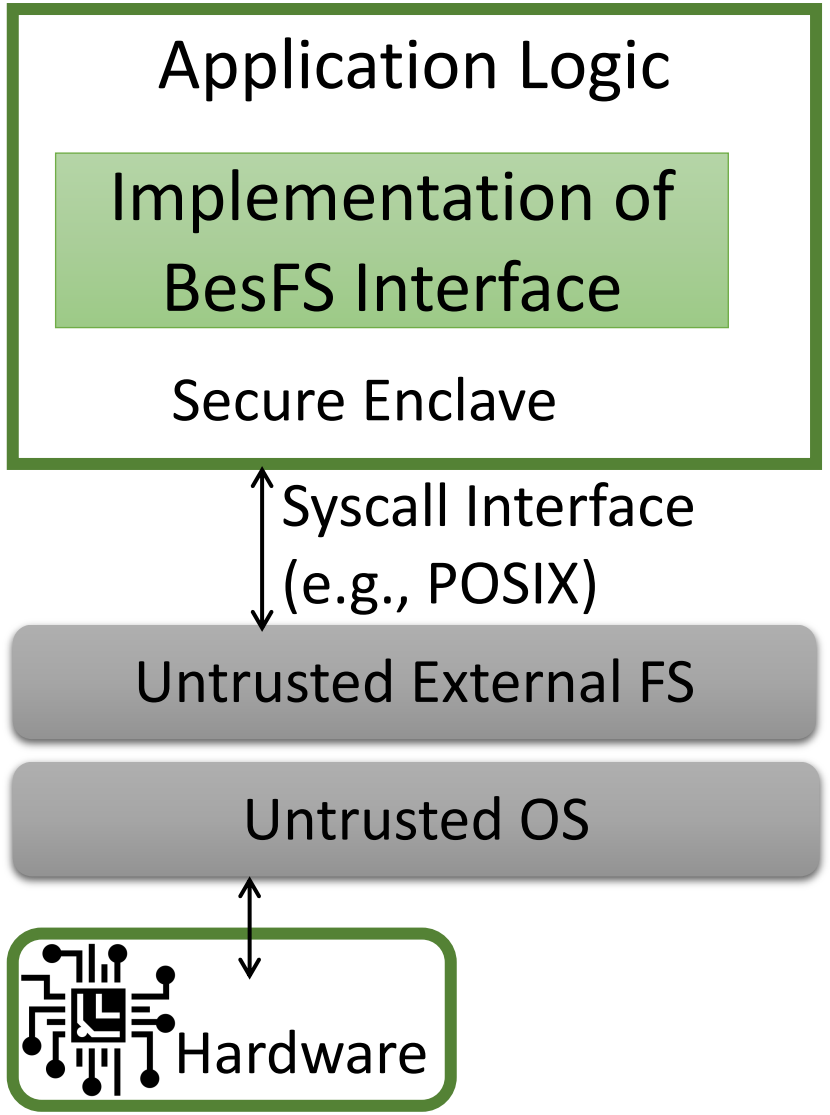
Sequential Composition Safety

Given a good initial state S_0 subject to a sequence of transitions τ_{m1} , . . . , τ_{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

Component	Language	LOC
Specification & Machine-proved Implementation		
Coq definitions & Proofs	Gallina	4625
Hand-coded C Implementation		
Implementation	C	863
External Calls	C	469
SGX Utils	C	117
Total		1449

167 lemmas
2 main theorems

Evaluation II: Expressiveness

Libc API	LOC	BESFS Core API used for composition of LibC API												
		fstat	read	open	close	seek	create	mkdir	rmdir	remove	chmod	readdir	truncate	write
read	7		✓											
fread	25		✓											
fscanf	34		✓											
fwrite	12	✓												✓
write	20	✓												✓
fprintf	15	✓												✓
fopen	78	✓		✓		✓	✓						✓	
open	60	✓		✓		✓	✓						✓	
fclose	9				✓									
close	17				✓									
fseek	31	✓				✓								
lseek	39	✓				✓								
rewind	5					✓								
creat	30			✓			✓							
mkdir	25							✓						
unlink	21									✓				
chmod	23										✓			
ftruncate	5												✓	
ftell	12	✓												
fgetc	9		✓											
fgets	25		✓											
readdir	10											✓		

`fwrite` composed from `write` and `fstat`

37 Total APIs:

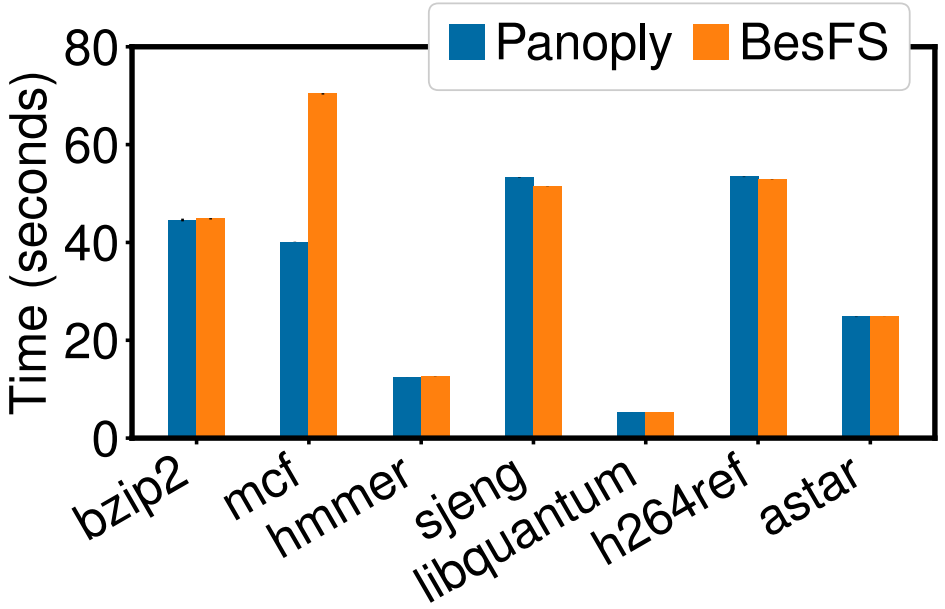
22 additional composed from 15 core

Evaluation III: Compatibility

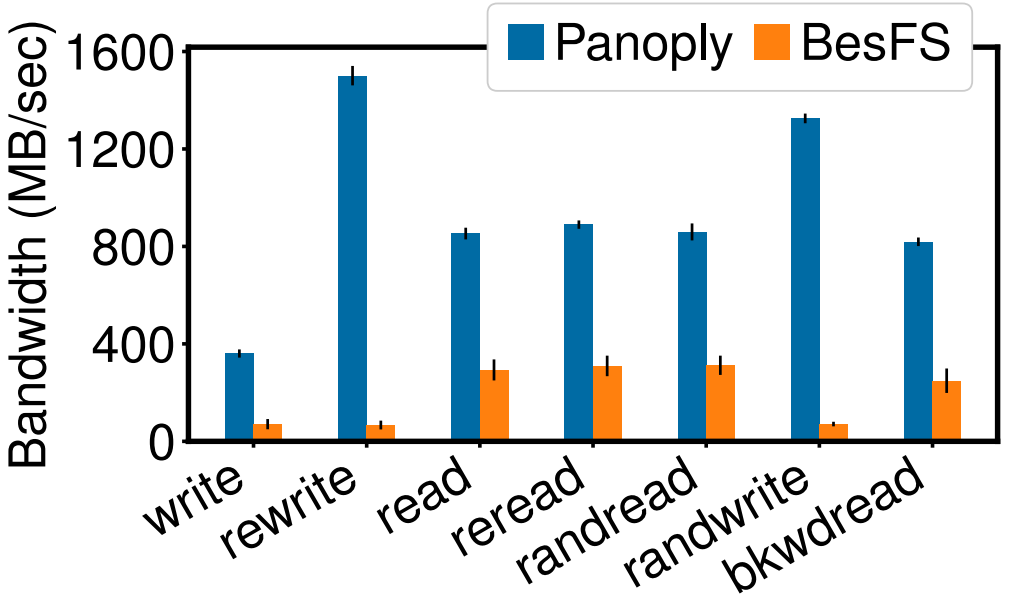
LibC Calls	SPEC CINT 2006							FSCQ		Total
	astar	mcf	bzip2	hmmer	libqu	h264	sjeng	small	large	
BESFS Core Calls										
open	3	0	1	0	0	7	0	2	1	14
read	27	0	4	0	0	129	0	1	3072	3233
write	0	0	0	0	0	0	0	1	66560	66561
lseek	0	0	0	0	0	75	0	0	66563	66638
remove	0	0	0	0	0	0	0	2	1	3
close	3	0	1	0	0	7	0	2	1	14
mkdir	0	0	0	0	0	0	0	100	0	100
BESFS Auxiliary Calls										
fopen	1	2	0	5	0	6	1	0	0	15
fread	1	0	0	1	0	1	0	0	0	3
fwrite	0	1035	0	6	0	13	2	0	0	1056
fgets	0	90435	0	108	0	0	5	0	0	90548
fscanf	12	0	0	0	0	24	0	0	0	36
fprintf	0	5985	0	605	0	17	162	0	0	6769
fseek	0	0	0	0	0	2	0	0	0	2
ftell	0	0	0	4	0	1	0	0	0	5
rewind	0	0	0	3	0	0	0	0	0	3
Unsafe Calls										
fsync	0	0	0	0	0	0	0	0	2	2
rename	0	0	0	0	0	0	6	0	0	6
Total	47	97457	6	732	0	282	176	108	136200	**235008**

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