LibSEAL: Detecting Service Integrity Violations Using Trusted Execution

Pierre-Louis Aublin[†1], Florian Kelbert[†], Dan O'Keeffe[†], Divya Muthukumaran[†], Christian Priebe[†2], Joshua Lind[†2], Robert Krahn[‡2], Christof Fetzer[‡], David Eyers[¶], Peter Pietzuch[†]
[†]Imperial College London [‡]TU Dresden [¶]University of Otago

Introduction. Internet users have become reliant on a swathe of online services for everyday tasks and expect them to uphold service integrity. However, data loss or corruption do happen despite service providers' best efforts. In such cases, users often have little recourse. Our goal is to strengthen the position of users by helping them to discover and prove integrity violations by Internet services.

LibSEAL is a SEcure Audit Library for Internet services that (i) transparently creates a non-repudiable audit log of service operations and (ii) checks invariants over that log to discover service integrity violations. LibSEAL protects the confidentiality of code and data by executing inside an Intel SGX trusted execution environment (called *enclave*). LibSEAL securely and effectively discovers service integrity violations, while reducing throughput by at most 32%.

Objectives. LibSEAL meets the following objectives.

O1: Ease-of-deployment. LibSEAL is easy to deploy with existing Internet services, requiring minimal to no changes to existing service and client implementations.

O2: Generality and flexibility. LibSEAL is general and widely applicable, supporting a multitude of Internet services with different specifications of integrity.

O3: Security and privacy. LibSEAL neither affects the confidentiality or integrity of data handled by the service, nor does it reveal details about the service implementation.

O4: Performance overhead. LibSEAL imposes a low performance overhead with respect to native service execution.

Design. LibSEAL acts as a drop-in replacement for existing TLS libraries (O1). The omnipresence of TLS means that LibSEAL can be applied to many existing Internet services (O2). Once a TLS-enabled service links against LibSEAL, LibSEAL terminates client connections and transparently records information from client requests and service responses in an audit log. The integrity of code and data, including LibSEAL itself, the audit log, and service requests and responses, is ensured by executing security-sensitive parts of LibSEAL inside a hardware-protected Intel SGX enclave (O3). In addition, LibSEAL cryptographically signs the audit log when storing it on disk. Integrity violations are expressed as violations of invariants over the audit log in terms of simple SQL queries (O2). LibSEAL avoids costly transitions between enclave and non-enclave code by permanently associating threads with the enclave (O4). Further, LibSEAL can be configured to write the log to disk asynchronously, reducing the impact on the critical path (O4).

Secure and efficient TLS termination. LibSEAL ports LibreSSL to SGX enclaves, executing and maintaining security-sensitive code and data inside the enclave. This includes code related to the TLS protocol, as well as any private keys and TLS session keys. LibSEAL reduces the number of SGX enclave transitions by (i) allocating memory for non-sensitive data in bulks, (ii) using the SGX provided thread locks implementation instead of pthread synchronization primitives, and (iii) ensuring that non-sensitive service data is stored outside of the enclave. LibSEAL further reduces the cost of enclave transitions by performing calls into the enclave asynchronously. For this, LibSEAL implements dedicated user-level lthread tasks inside the enclave.

Audit logging and checking. LibSEAL generates the audit log based on client requests and service responses. It observes all messages exchanged in a TLS connection by monitoring the TLS functions `SSL_read()` and `SSL_write()`. To prevent data loss under failure, LibSEAL writes the audit log to local persistent storage. To avoid logging every request and response in its entirety, LibSEAL employs service-specific modules to (i) parse the service specific requests and responses, (ii) extract the information required to verify the service invariants, (iii) append the data to the audit log in a relational format, and (iv) specify invariants—usually soundness and completeness properties—as SQL queries over the relational schema. LibSEAL triggers invariant checks after configurable time intervals, but clients may also trigger invariant checks explicitly.

Evaluation. We evaluate the security and performance of LibSEAL using the Git version control service, the ownCloud collaborative document service, and the Dropbox file storage service. Results show that: (i) invariants are simple to write yet strong enough to detect integrity violations, such as the soundness and completeness of files and documents served by Dropbox and ownCloud; (ii) LibSEAL prevents log bypassing and is secure against enclave interface attacks; and (iii) LibSEAL has an acceptable performance overhead of at most 32% for asynchronous logging; depending on the service and its invariants, invariant checking takes 40–200 ms for several thousand log entries.

[1] Presenter [2] Student

LibSEAL: Detecting Service Integrity Violations Using Trusted Execution

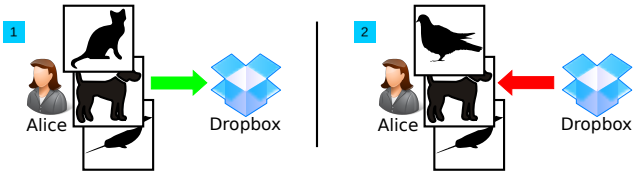
Pierre-Louis Aublin, Florian Kelbert, Dan O'Keeffe, Divya Muthukumaran, Christian Priebe, Joshua Lind and Peter Pietzuch
{p.aublin,f.kelbert}@imperial.ac.uk
Imperial College London

Robert Krahn and Christof Fetzer
TU Dresden

David Eyers
University of Otago

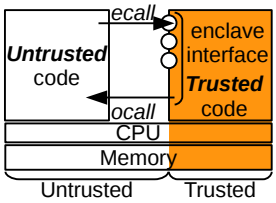
Motivation

- Internet services do **not guarantee integrity**
  e.g. Git, Dropbox
- Data can be lost or corrupted
- **Goal: Discover and prove** integrity violations



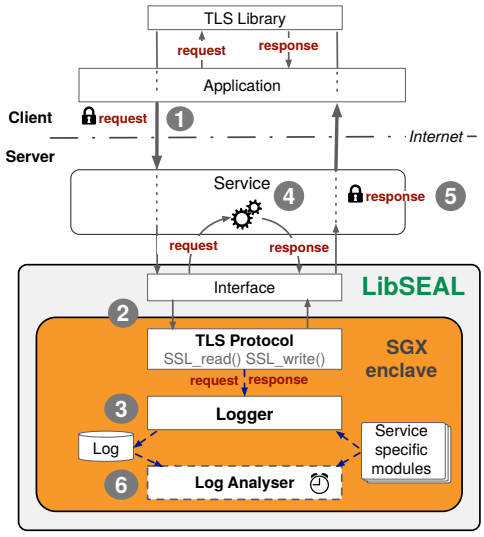
Background: Intel SGX

- **Security instructions** on Intel CPUs
- **Enclaves** isolate code and data
- Protects against malicious OS and hardware



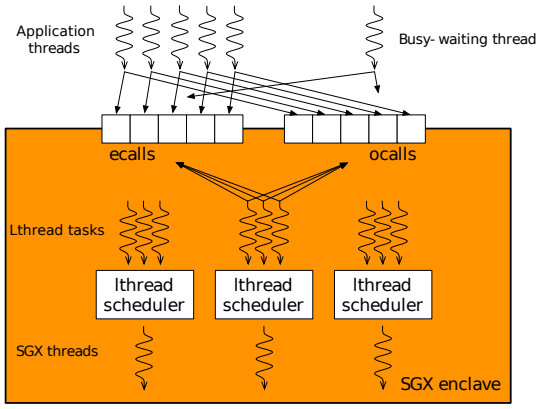
LibSEAL: SEcure Auditing Library

- **Drop-in replacement** for existing TLS libraries
- **Terminate TLS** inside an SGX **enclave**
- **Securely log** service **requests** and **responses**
- **Periodical** log-auditing by **checking invariants**



Improving performance

- **Asynchronous** enclave transitions
- **User-level** scheduling



Audit logging and checking

- **Service specific modules** log relevant data
- **Relational database** stores the log

```
updates(time, repo, branch, cid, type)
advertisements(time, repo, branch, cid)
```

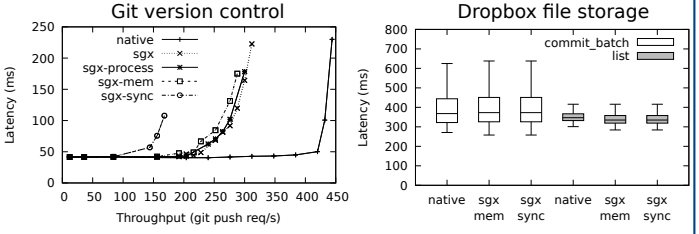
- **Asynchronous writes** improve performance
- **Signatures** to detect log tampering
- **SQL statements** specify service invariants

```
SELECT time, repo FROM advertisements
NATURAL JOIN branchcnt
GROUP BY time, repo, cnt HAVING COUNT(branch) != cnt;
```

- **Persistent counters** to detect rollback attacks

Performance results

- Implementation based on LibreSSL



- **Performance overhead** of at most 32%
- **Invariant checking** takes less than 200ms