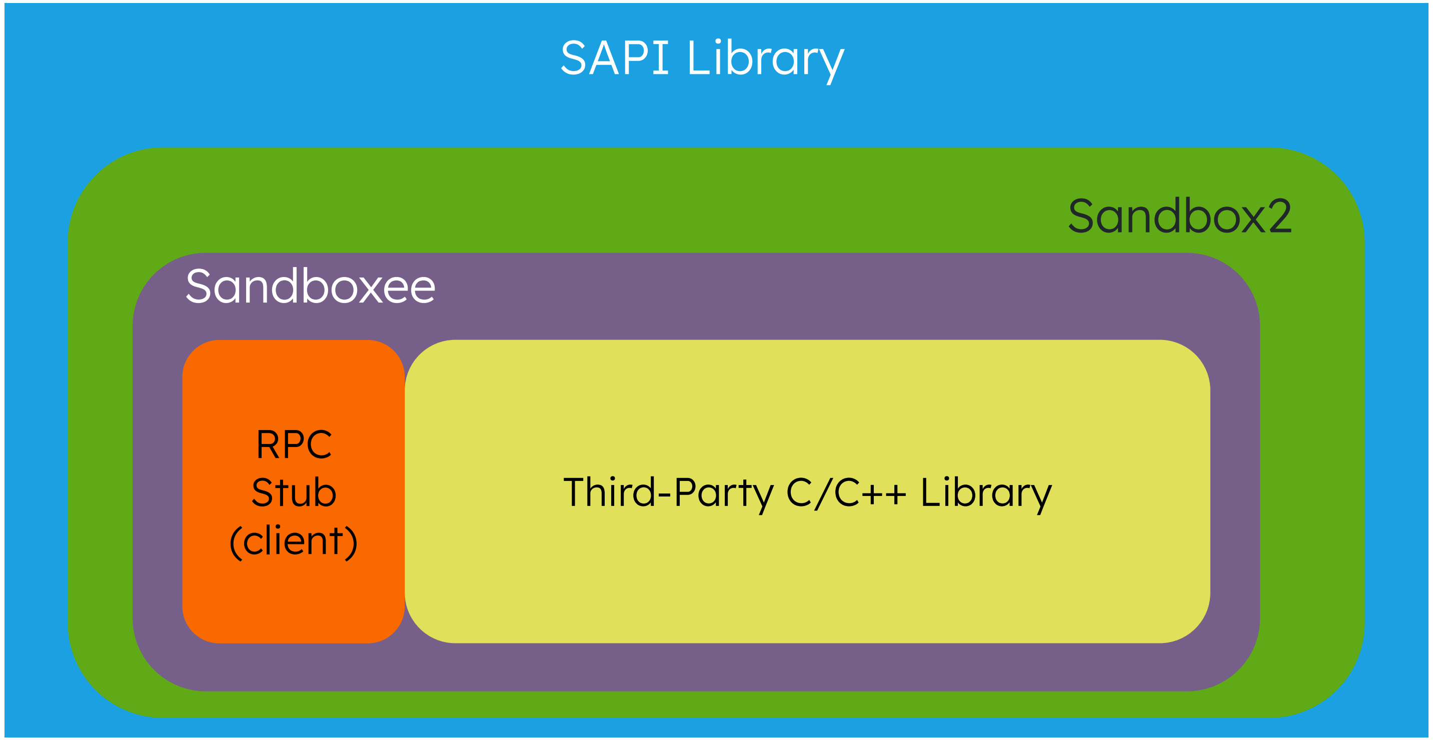
Library Sandboxing with SAPI

Oliver Kunz - BSides Lisbon 2024

Situation Overview



Introduction to Sandbox2

Overview: Sandbox2

- Developed by the ISE Sandboxing Team
- Open-Sourced in 2019
- Use case: Full binary sandboxing
- Deferred sandboxing is possible
  - Sandboxee must call `SandboxMeHere()`[1]
- For CPU-intensive initialisation
  - Users can define a Custom Forkserver [2]

Used as the sandboxing driver for SAPI

Documentation at: developers.google.com/code-sandboxing/sandbox2

Advantages

- Good for third-party binaries
  - Precompiled or not
- Good for your source controlled binaries
  - With third-party libraries or not

Disadvantages

- Syscall filter can become quite large and complex
  - Leads to a too coarse-grained sandbox policy
- Limited architectures and OS supported [3]

Building Blocks of Sandbox2

- Builds on well-known and established technologies
- Namespaces: IPC, Network, Mount, PID, User, UTS
- seccomp-bpf for syscall filtering
- Ptrace for monitoring
  - Alternative monitoring via seccomp user notify
- Sockets for communication

Sandbox2 Policybuilder

- API to define a sandbox policy in code [4]
- Provides functions to:
  - Map files or directories into the sandbox
  - Convenience functions to allowlist multiple syscalls:
    - `AllowExit()` => exit, exit_group
    - `AllowDynamicStartup()`
  - Allowlist individual sycalls
  - Create specific filter policies on syscall arguments

Documentation at: developers.google.com/code-sandboxing/sandbox2/full-getting-started#2_create_a_sandbox_policy

Executor & Sandboxee

Sandbox Executor [5]

- Part of the Hostcode
- Trusted process, not sandboxed
- Controls the resource limitations of the Sandboxee (configurable)
- Sets up the Monitor which tracks the Sandboxee

Documentation at: developers.google.com/code-sandboxing/sandbox2/getting-started/executor

Sandboxee

- Phase I - Trusted Execution without sandbox
    - Sandbox Executor sends policy to Sandboxee via IPC
    - The client [6] object applies the policy
- Phase II - Untrusted Execution with sandbox
    - The process now becomes the Sandboxee
    - Execution of the actual sandboxee binary via execve
    - Executes the untrusted code

Introduction to SAPI

Overview: SAPI

- Developed by the ISE Sandboxing Team
- Addresses an observed user pattern from Sandbox2
- Open-Sourced in 2019
- Use case: Individual library sandboxing
- Relies on Sandbox2 as sandboxing driver

Documentation at: developers.google.com/code-sandboxing/sandboxed-api

Overview: Advantages & Disadvantages

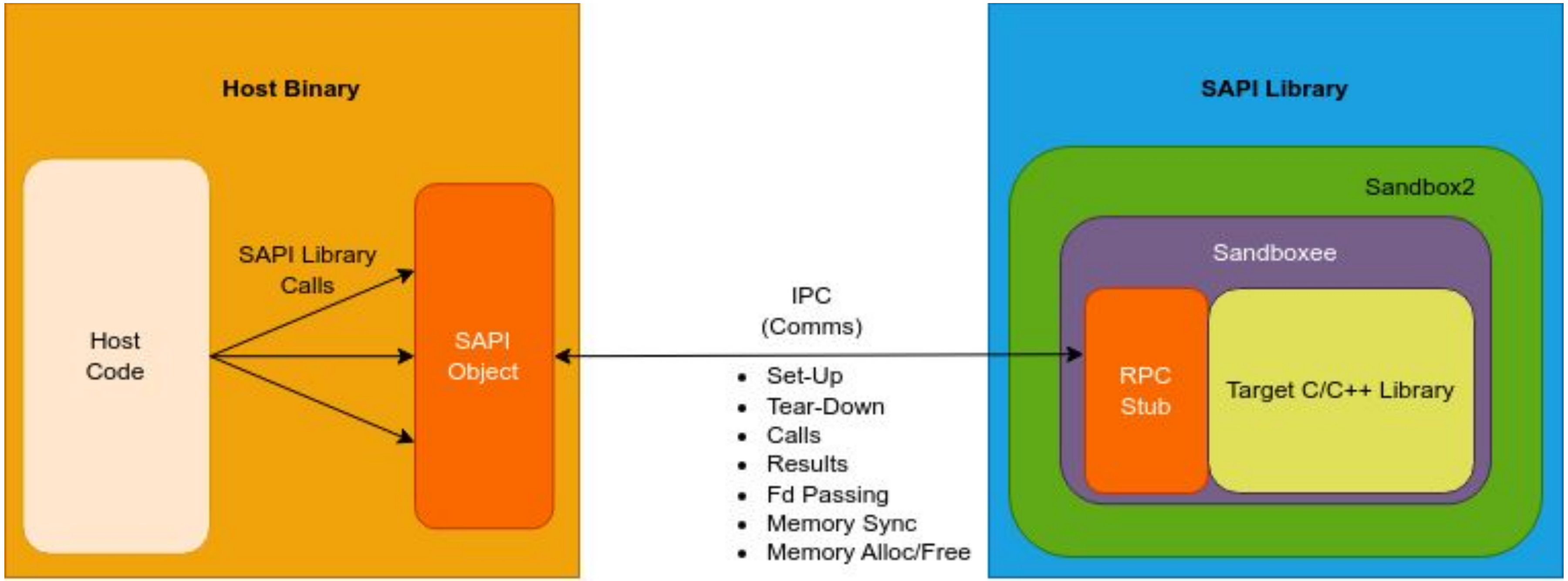
Advantages

- Sandbox once, use anywhere
- Reduce the sandboxing footprint to only untrusted API calls
- Reduces the syscall filter complexity
- Good for untrusted third-party libraries
- Good for wrapper libraries around multiple third-party libraries

Disadvantages

- Development overhead, worst-case:
  - Library analysis to identify concerning APIs
  - Write C-Wrapper,
- Every library call [7] requires at least one RPC roundtrip
- Data and parameters may need to be passed into the sandbox prior and maybe retrieved after the call.

Architecture



Build Rule: sapi_library

- Defines a sandboxed library target
- Will build a sandboxee binary, combining
  - The target library and RPC stub (client)
- Will build some additional intermediate artefacts
- Will execute the SAPI code generator

```
sapi_library(
   name,
   functions,
   generator_version,
   lib,
   lib_name,
   namespace,
   deps,
)
```

Supported Build Systems:

- Bazel [8]
- Cmake [9]

SAPI Code Generator

- Functionality:
    - Analyse the target library
    - Extract type information
    - Translate types to SAPI Types
    - Extract the functions
- End result: emits the `*.sapi.h` header
- Two generators are available
    - Python generator [10]
    - clang generator (libtooling) [11]

SAPI Types

- Ensures hostcode can access variables and memory blocks in the sandboxee process
- Comprehensive set of C++ classes
  - `#include "sandboxed_api/vars.h"` header [12]
- Native C types supported by default
- Examples:
  - `sapi::v::Int`
  - `sapi::v::Array<uint8_t>`
  - `sapi::v::Struct<MyStruct>`
  - `sapi::v::Void`
  - `sapi::v::RemotePtr`

Documentation at: developers.google.com/code-sandboxing/sandboxed-api/variables

Pointer Synchronization

Problem

- Hostcode and Sandboxee have different address space
- Impossible to access a pointer from within the other context
- Requires to synchronise data at pointer between the contexts

Solution

- Passing the SAPI Type pointers in the RPC call should be obtained using these functions
- Pointer Synchronization Modes [13]
    - PtrNone
    - PtrBefore
    - PtrAfter
    - PtrBoth

How to sandbox a library

How to sandbox a library

1. Analyse the target API
   a. Understand how the library is being used and for what
   b. Identify functions that need to be sandboxed
2. Attempt to have SAPI generate the entire API
   a. Create the `sapi_library` target with an empty `functions` list
   b. Build the target and investigate the Build output
      i. Check the *.sapi.h that all your functions of concern are present.
3. If (2) fails, then specify the `functions` list and retry
4. If (3) fails, then an intermediary C-Wrapper is needed

How to sandbox a library: C-Wrapper

1. For each of the functions of concern, create a C-Wrapper
2. Change the `lib` parameter of the `sapi_library` build rule to the C-Wrapper library
3. Attempt to generate the sandboxed version for the default configuration
4. Check again if the `*.sapi.h` header contains all the functions.

Example: sapi-librot13

```
# librot13 build target
cc_library(
    name = "rot13",
    srcs = ["rot13.cc"],
    hdrs = ["rot13.h"],
)
```

```
# sandboxed librot13 build
target
sapi_library(
  name = "rot13_sapi",
  functions = [
     "Rot13File",
  ],
  input_files = ["rot13.cc"],
  lib = ":rot13",
  lib_name = "Rot13",
)
```

```
// Unsandboxed Rot13File function of librot13.
int Rot13File(const char* input_file, const char* output_file) {
  try {
    std::ifstream in(input_file, std::ios::binary);
    in.exceptions(std::ifstream::failbit);
    std::ofstream out(output_file, std::ios::binary);
    out.exceptions(std::ofstream::failbit);
    std::transform(std::istreambuf_iterator<char>(in),
                   std::istreambuf_iterator<char>(),
                   std::ostreambuf_iterator<char>(out), Rot13);
  } catch (std::ios_base::failure& fail) {
    std::cerr << "ROT13 error: " << fail.what() << std::endl;
  }

  struct rusage rusage;
  if (getrusage(RUSAGE_SELF, &rusage) != 0) {
    return -1;
  }
  return rusage.ru_utime.tv_sec * 1000000 + rusage.ru_utime.tv_usec;
}
```

Generated SAPI Header

- The SAPI Header is one the outputs of the `sapi_library` rule
- Include this header to use the sandboxed library

The left shows part of the generated rot13_sapi.sapi.h of the example shown later.

```
class Rot13Sandbox : public ::sapi::Sandbox {
    [...] // Removed.
};

class Rot13Api {
public:
 explicit Rot13Api(::sapi::Sandbox* sandbox) : sandbox_(sandbox) {}
 ::sapi::Sandbox* sandbox() const { return sandbox_; }

 // int Rot13File(const char *, const char *)
 absl::StatusOr<int> Rot13File(::sapi::v::Ptr* input_file,
                               ::sapi::v::Ptr* output_file) {
   ::sapi::v::Int ret;

   SAPI_RETURN_IF_ERROR(sandbox_->Call("Rot13File", &ret,
                                       input_file, output_file));
   return ret.GetValue();
 }

private:
 ::sapi::Sandbox* sandbox_;
};
```

Extending Default Policy

1. Extend `Rot13Sandbox`
2. Overwrite `ModifyPolicy`
   a. Uses default syscall filter [14]
   b. Extends default to add the input and output files

The outcome of this policy is that the two files, which are created outside of the sandbox, are accessible inside the sandbox.

```
class Rot13SapiSandbox : public Rot13Sandbox {
public:
 Rot13SapiSandbox(
     const std::string& in_file, const std::string& out_file)
     : in_file_(in_file), out_file_(out_file) {}

 std::unique_ptr<sandbox2::Policy> ModifyPolicy(
     sandbox2::PolicyBuilder* builder) override {
   builder->AddFile(in_file_)
       ->AddDirectoryAt(
           dirname(&out_file_[0]), "/output", /*is_ro=*/false)
       ->BuildOrDie();
 }

private:
 std::string in_file_;
 std::string out_file_;
};
```

Using a Sandboxed Library

1. Create the sandbox and API objects
2. Prepare the SAPI types
3. Make the SAPI call
4. Check the SAPI call return status and the sandboxed API's return value
5. Get the data from the sandbox and return the values

```
// Create sandbox and SAPI objects.
Rot13SapiSandbox sandbox(in_file, out_file);
CHECK_OK(sandbox.Init());
Rot13Api api(&sandbox);

// Prepare SAPI Type variables.
sapi::v::ConstCStr in_file_var(in_file.c_str());
std::string out_file_in_sandboxee(std::string("/output/") +
                                   basename(&out_file[0]));
sapi::v::ConstCStr out_file_var(out_file_in_sandboxee.c_str());

// Call the sandboxed Rot13File function.
absl::StatusOr<int> res =
   api.Rot13File(in_file_var.PtrBefore(),
out_file_var.PtrBefore());
CHECK_OK(res);
CHECK_NE(res, -1);
```

Q & A

Resources

Documentation

- General Sandboxing: https://developers.google.com/code-sandboxing
- Sandbox2: https://developers.google.com/code-sandboxing/sandbox2
- SAPI: https://developers.google.com/code-sandboxing/sandboxed-api

Code

- https://github.com/google/sandboxed-api

Examples

- https://github.com/google/sandboxed-api/tree/main/contrib
- https://github.com/google/sandboxed-api/tree/main/oss-internship-2020
- https://github.com/google/sandboxed-api/tree/main/sandboxed_api/examples
- https://github.com/cblichmann/sapi-librot13

Links

[1] https://github.com/google/sandboxed-api/blob/main/sandboxed_api/sandbox2/client.h#L54
[2] https://developers.google.com/code-sandboxing/sandbox2/full-getting-started#method_3_custom_forkserver...
[3] https://github.com/google/sandboxed-api/blob/main/sandboxed_api/config.h
[4] https://github.com/google/sandboxed-api/blob/main/sandboxed_api/sandbox2/policybuilder.h
[5] https://github.com/google/sandboxed-api/blob/main/sandboxed_api/sandbox2/executor.h
[6] https://github.com/google/sandboxed-api/blob/main/sandboxed_api/sandbox2/client.h
[7] https://github.com/google/sandboxed-api/blob/main/sandboxed_api/call.h
[8] https://github.com/google/sandboxed-api/blob/main/sandboxed_api/bazel/sapi.bzl#L234
[9] https://github.com/google/sandboxed-api/blob/main/cmake/SapiBuildDefs.cmake#L92
[10] https://github.com/google/sandboxed-api/tree/main/sandboxed_api/tools/python_generator
[11] https://github.com/google/sandboxed-api/tree/main/sandboxed_api/tools/clang_generator
[12] https://github.com/google/sandboxed-api/blob/main/sandboxed_api/vars.h
[13] https://developers.google.com/code-sandboxing/sandboxed-api/variables#sapi_pointers
[14] https://github.com/google/sandboxed-api/blob/main/sandboxed_api/sandbox.cc#L90