Protecting the IoT Against Data Leaks through Intra-Process Access Control

Marcela S. Melara, David Liu, Edward W. Felten, Michael J. Freedman
Princeton University

Background

Third-party libraries are extremely prevalent in the IoT:

- ***99%*** of studied IoT applications import at least one third-party library
- Out of top 50 studied Python libraries:

Written in Python	18.0%
Have native dependencies	76.0%

Table 1: Library implementation language.

Run external binaries	40.0%
Use ctypes FFI	40.0%

Table 2: Used dynamic language features.

Problem: Developers do not/cannot inspect and vet imported third-party code
⇒ security & privacy vulnerabilities go undetected

Motivating Threats:

- Our experiments: successfully modified function pointers, local variables in the Python runtime call stack from (native) libraries.
- Reported vulnerabilities in Python libraries 2012-2017:

Vulnerability Class	**# of Reports**
Data Leak	15
Arbitrary Code Execution	12
Symlink Attack	5

Table 3: Top 3 Python library vulnerabilities out of 48 analyzed CVE database reports We identified 35 distinct vulnerable Python libraries.

Our goal: Prevent malicious third-party libraries from accessing sensors or data other than those intended by the developer.

Prior Work: Why can't we apply isolation solutions for Android or IoT?

- Isolate libs into separate apps, apply Android permissions (e.g. [1-3])
  ⇒ Linux-based IoT doesn't have built-in mandatory access control
- Android native library isolation:
  ⇒ SFI [4] requires access to library source code
  ⇒ hardware fault isolation [5] is platform-dependent
- Prior IoT solns: poor usability for developers [6] and end users [7]

References:
[1] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. *AdDroid: Privilege separation for applications and advertisers in Android*. ASIACCS, 2012.
[2] S. Shekhar, M. Dietz, and D. S. Wallach. *AdSplit: Separating smartphone advertising from applications*. USENIX Security, 2012.
[3] M. Sun and G. Tan. *NativeGuard: Protecting Android applications from third-party native libraries*. WiSec, 2014.
[4] E. Athanasopoulos, V. P. Kemerlis, G. Portokalidis, and A. D. Keromytis. *NaClDroid: Native code isolation for android applications*. ESORICS, 2016.
[5] J. Seo, D. Kim, D. Cho, T. Kim, I. Shin, and X. Jiang. *FlexDroid: Enforcing In-App Privilege Separation in Android*. NDSS, 2016.
[6] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash. *FlowFence: Practical data protection for emerging IoT application frameworks*. USENIX Security, 2016.
[7] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernan- des, Z. M. Mao, and A. Prakash. *ContexIoT: Towards providing contextual integrity to appified IoT platforms*. NDSS, 2017.

Approach

Provide an **access control framework** at the granularity of libraries (intra-process) that dynamically adjust privilege based on the execution context.

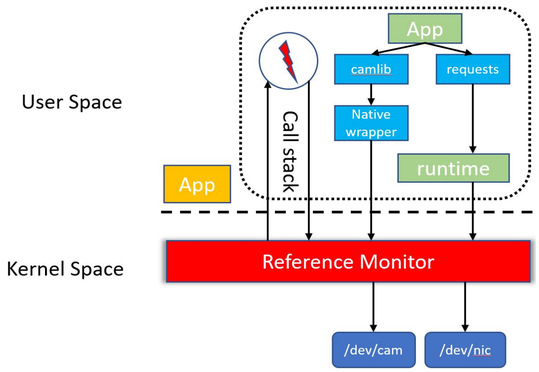


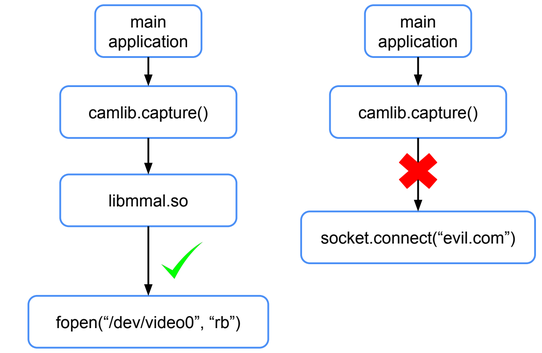
Fig. 1: High-level System Architecture.

Stack Inspection:

- Reference monitor intercepts relevant syscalls and pauses the corresponding app thread
- Stack Tracer thread in runtime collects paused thread's call stack info
  ⇒ pass to reference monitor through a secure comm. channel

Access Control Semantics:

- Ref. monitor makes access decisions on a per-thread basis
- Developer specifies initial permissions of top-level libraries
- Lower-level modules "inherit" the permissions of the closest known caller module at runtime
- System calls are allowed/denied based on the resulting permissions of all libraries in the call stack (i.e. the provenance of the syscall)




```
Developer policy:
camlib CAMERA_READ
requests NETWORK_SEND "mycloud.com"
```

Fig. 2: Two sample call stacks. The developer's policy permits the syscall in the left call stack, but denies the network connection from the camera library.

Native Code Isolation Mechanism:

- Isolate native third-party libraries into their own memory address space
  ⇒ Prevent native libraries from manipulating a thread's call stack info
- Runtime detects calls to third-party native libraries and runs each in a separate process