Fine-grained Program Partitioning for Security

Zhen Huang*, Trent Jaeger#, Gang Tan#

* School of Computing, *DePaul University*

School of Electrical Engineering & Computer Science, *Pennsylvania State University*

Outline

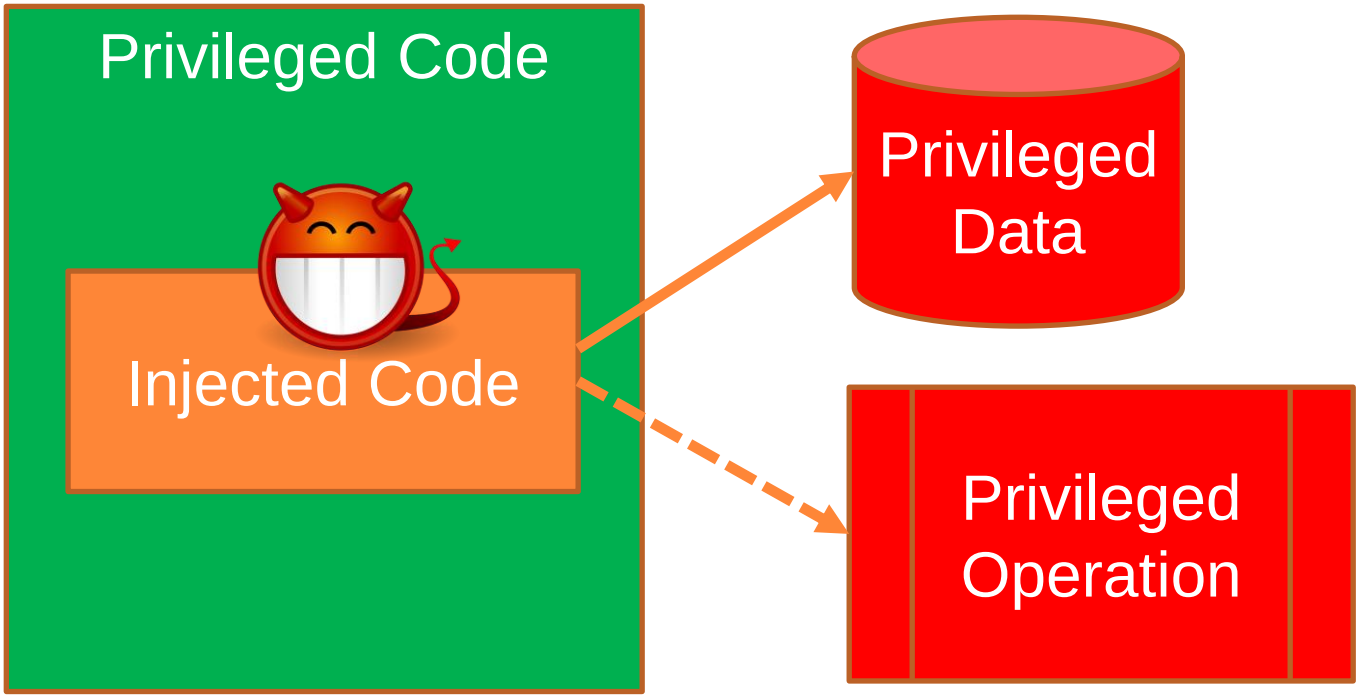
- Program Partitioning For Security
- Fine-grained Program Partitioning
- Implementation
- Evaluation
- Conclusion

Software Security

- Software vulnerabilities remain a critical issue for software security.
  - Over 53,000 vulnerabilities were disclosed for the last three years.
  - A Russian-based espionage campaign compromised U.S. federal agencies in 2020.

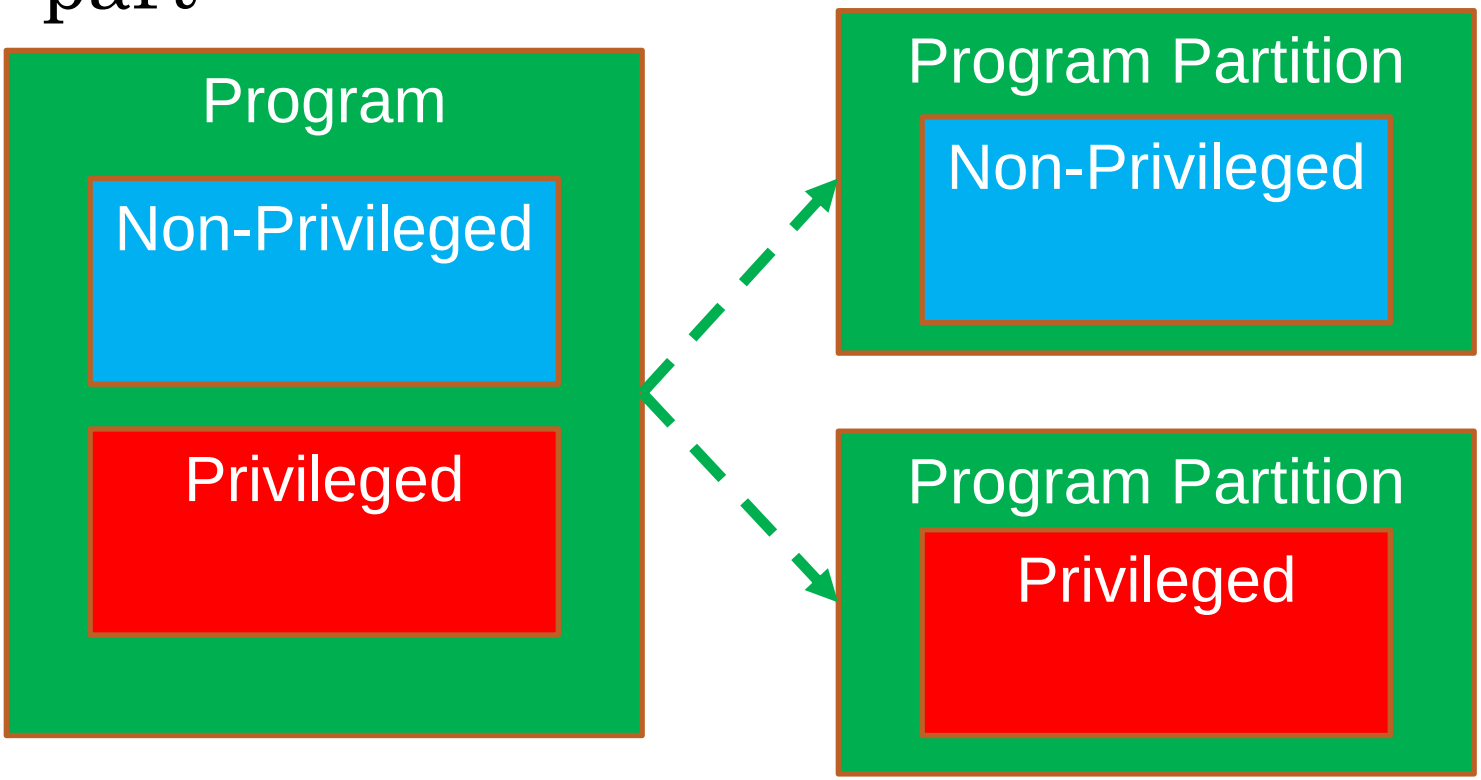
Vulnerabilities in Privileged Code

- Exploiting vulnerabilities in privileged code can cause the most severe damages



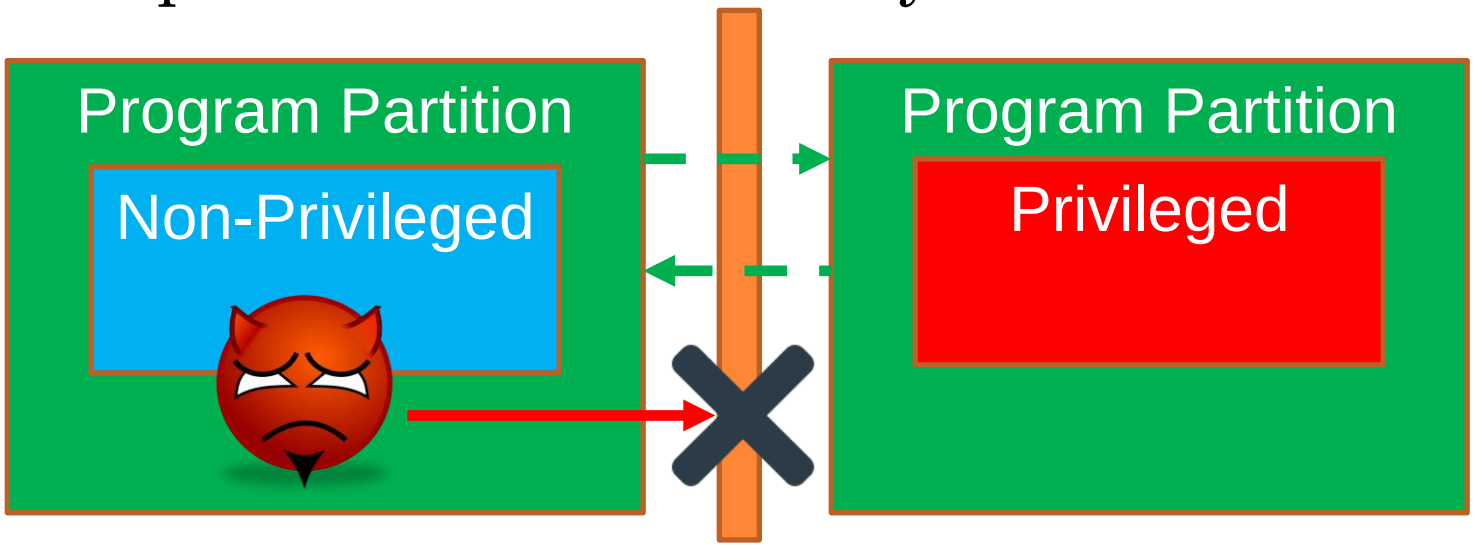
Principle of Least Privilege

- Separating a program into a privileged part and a non-privileged part



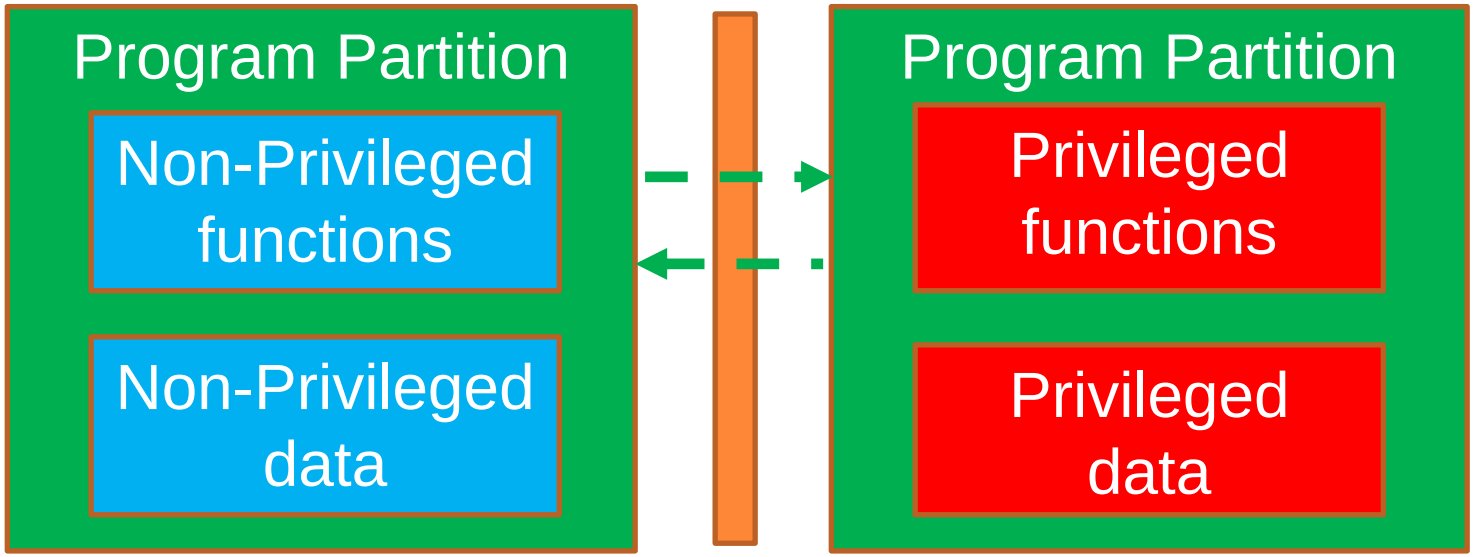
Program Partitioning

- Each program partition can run in its own address space
- Partitions communicate via a guarded interface
- Improves software security



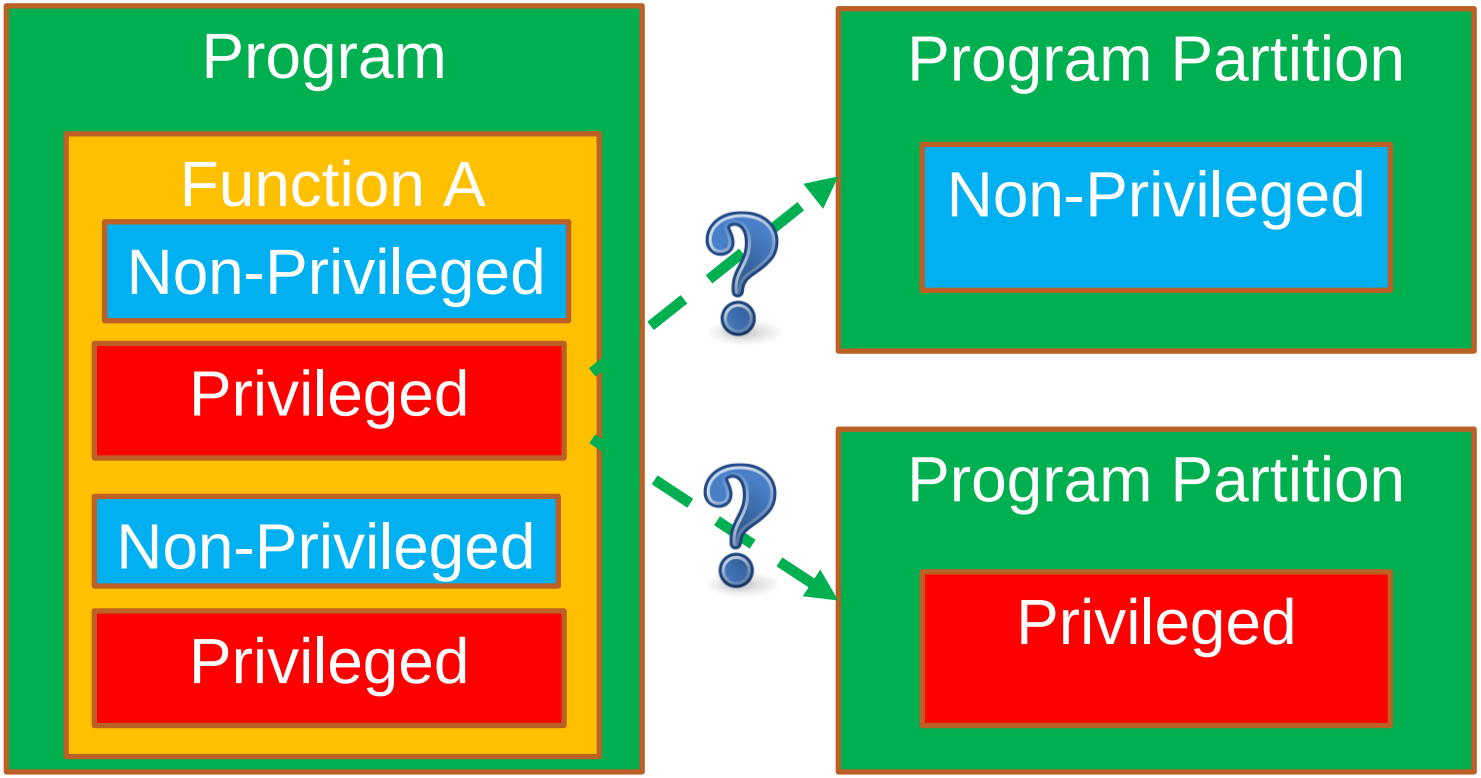
Automatic Program Partitioning

- Each partition implemented as an separate program
- Communication implemented using RPC function calls
- Partitioning at function level



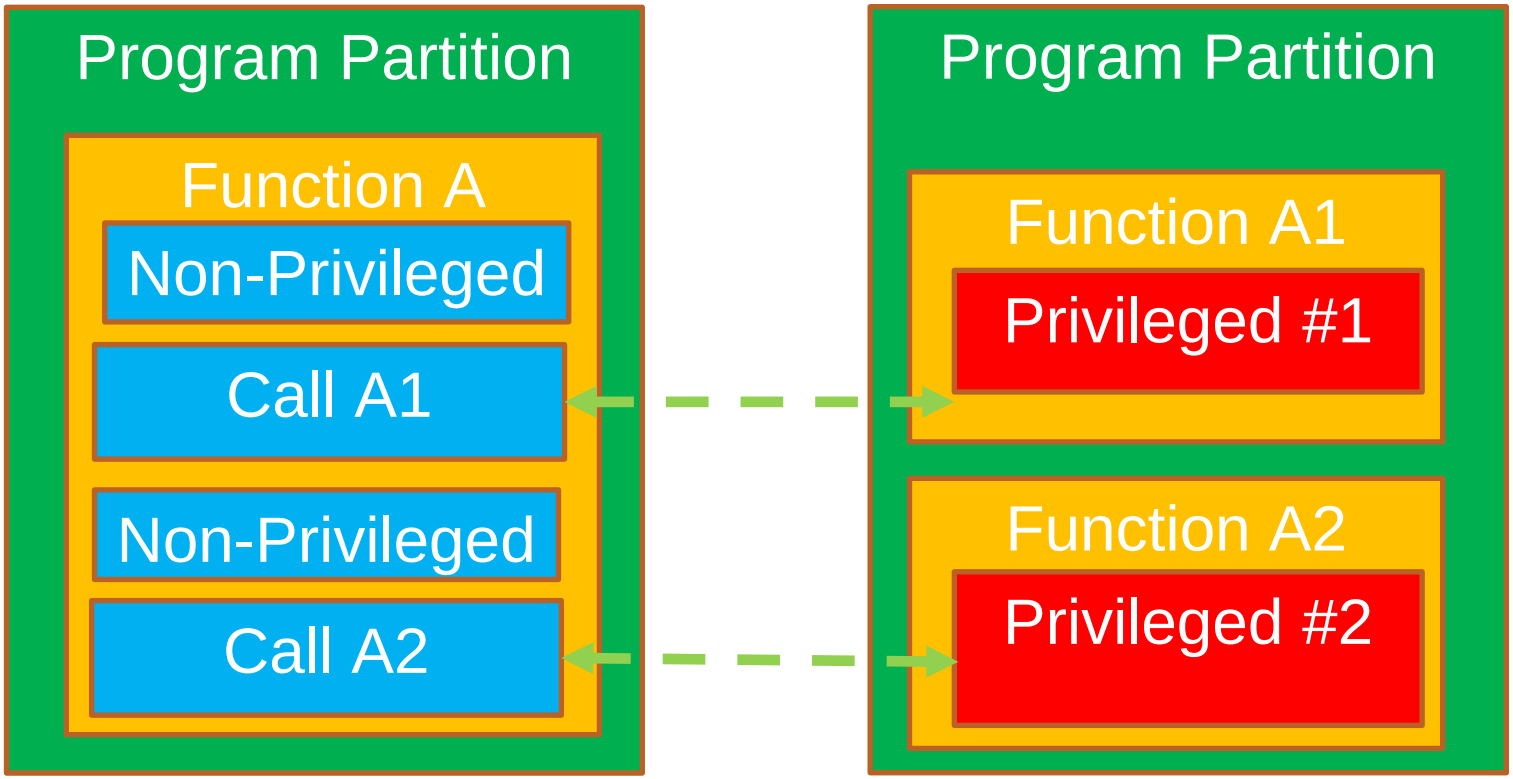
Issue with Function-Level Partitioning

- How do we partition functions containing *intertwined privileged code and non-privileged code*?



A Naïve Solution

- The naïve solution can result in a high number of RPC calls between partitions.

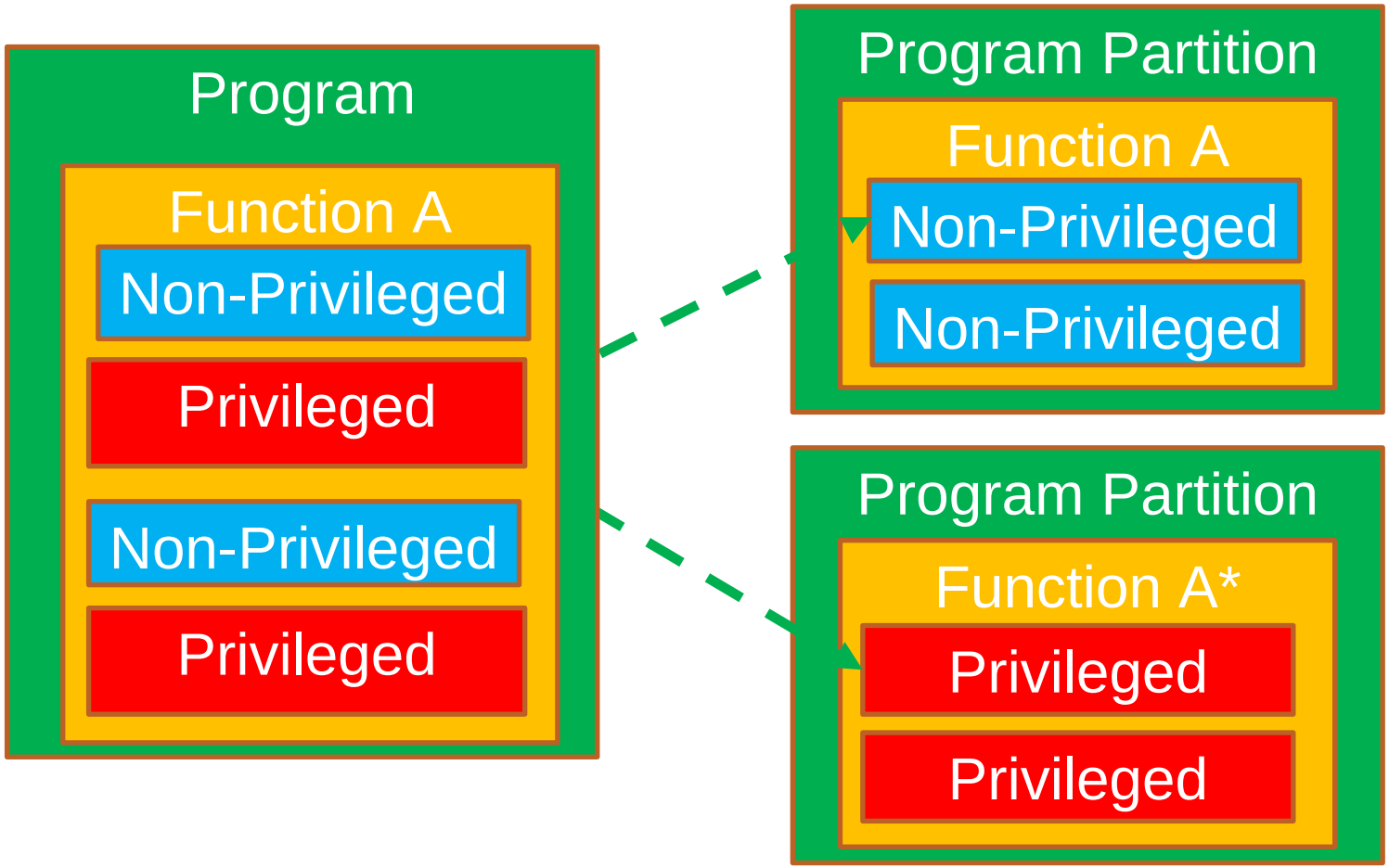


Outline

- Program Partitioning For Security
- **Fine-grained Program Partitioning**
- Communication between Partitions
- Evaluation
- Conclusion

Fine-Grained Program Partitioning

- Partitioning within a function

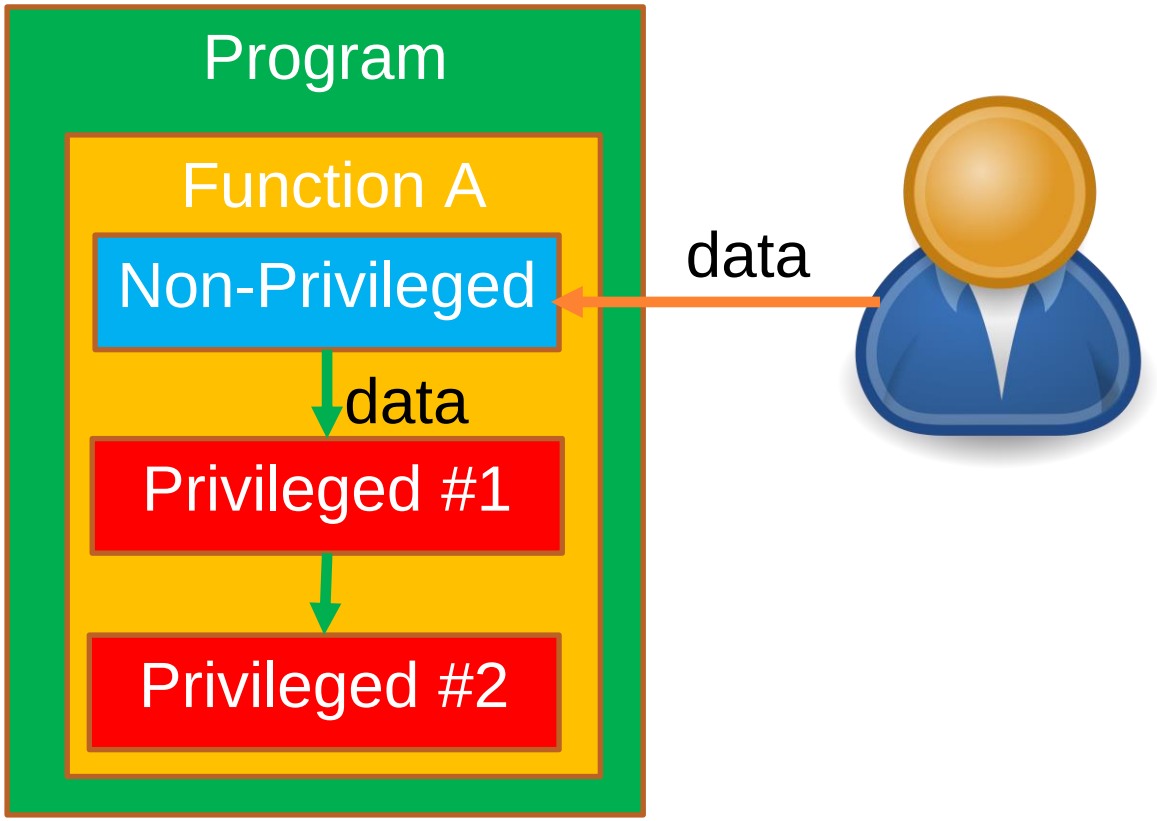


Fine-Grained Program Partitioning

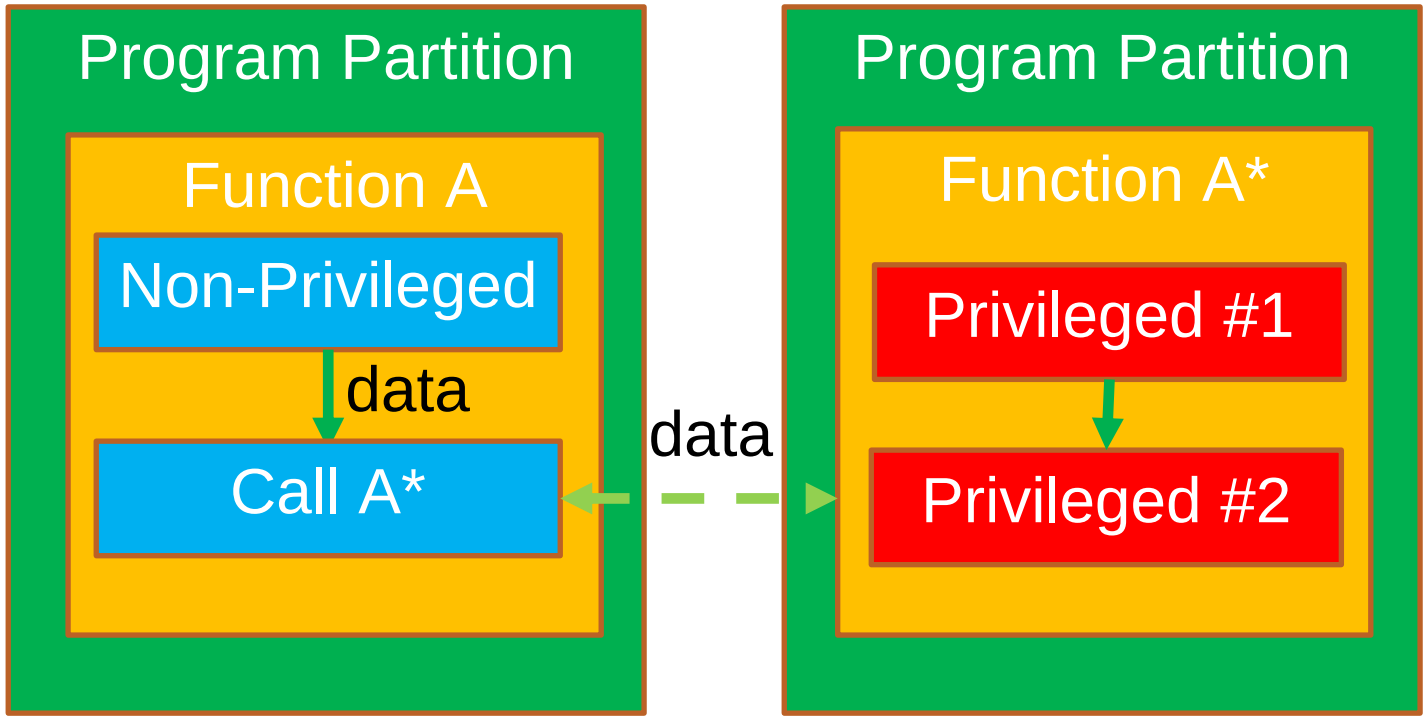
- Using static program analysis to partition functions in existing programs
- Focusing on two *hot spot* programming patterns
- Merging code together to reduce the number of RPC calls

Pattern #1: Non-privileged to Privileged

- Non-privileged code followed by privileged code

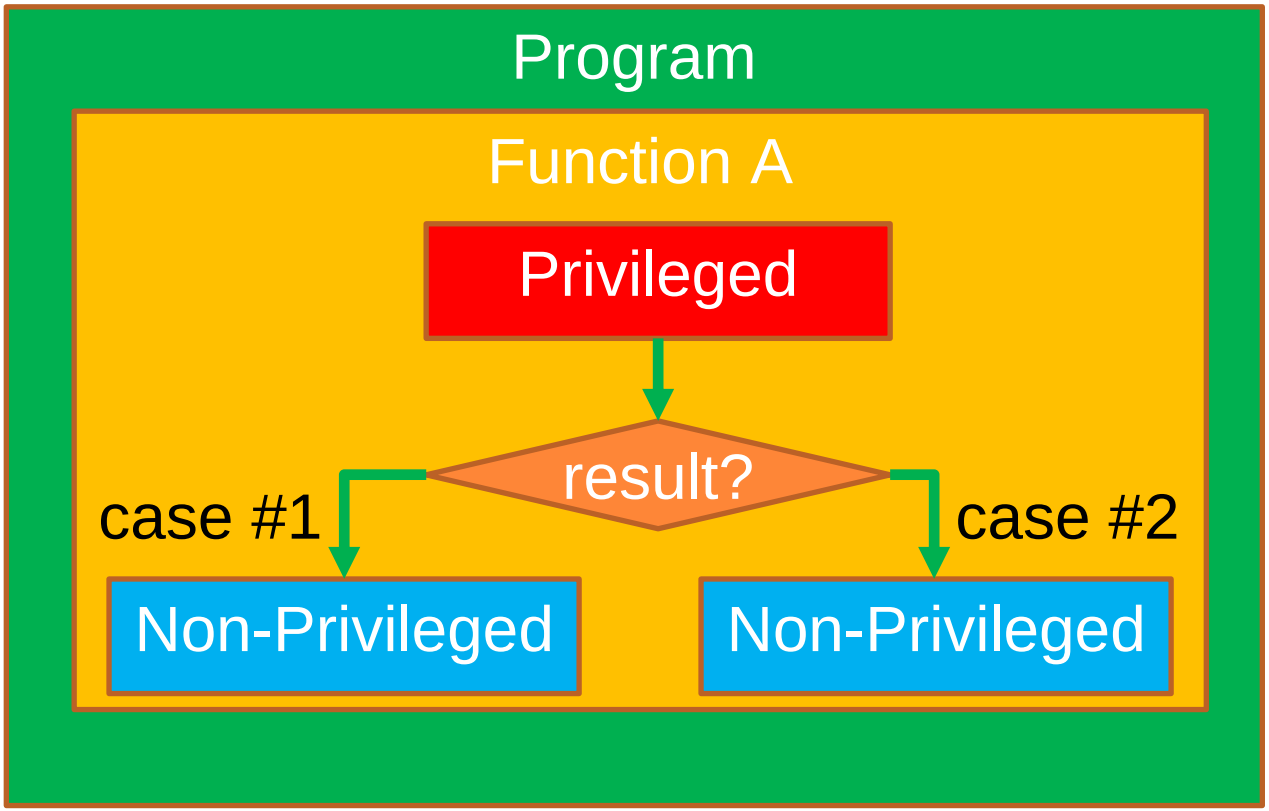


Solution #1: Non-privileged to Privileged

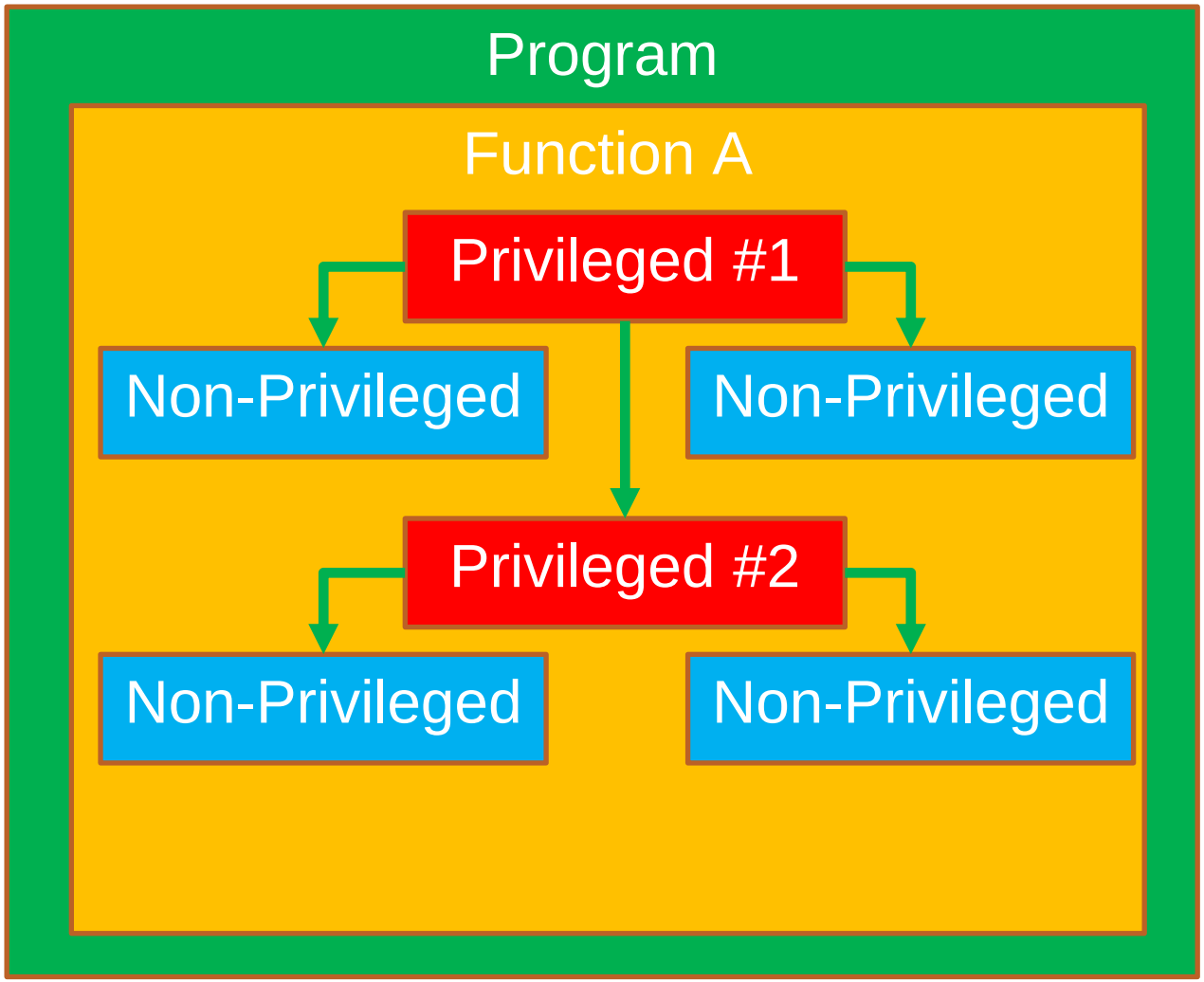


Pattern #2: Privileged to Non-Privileged – Simple Case

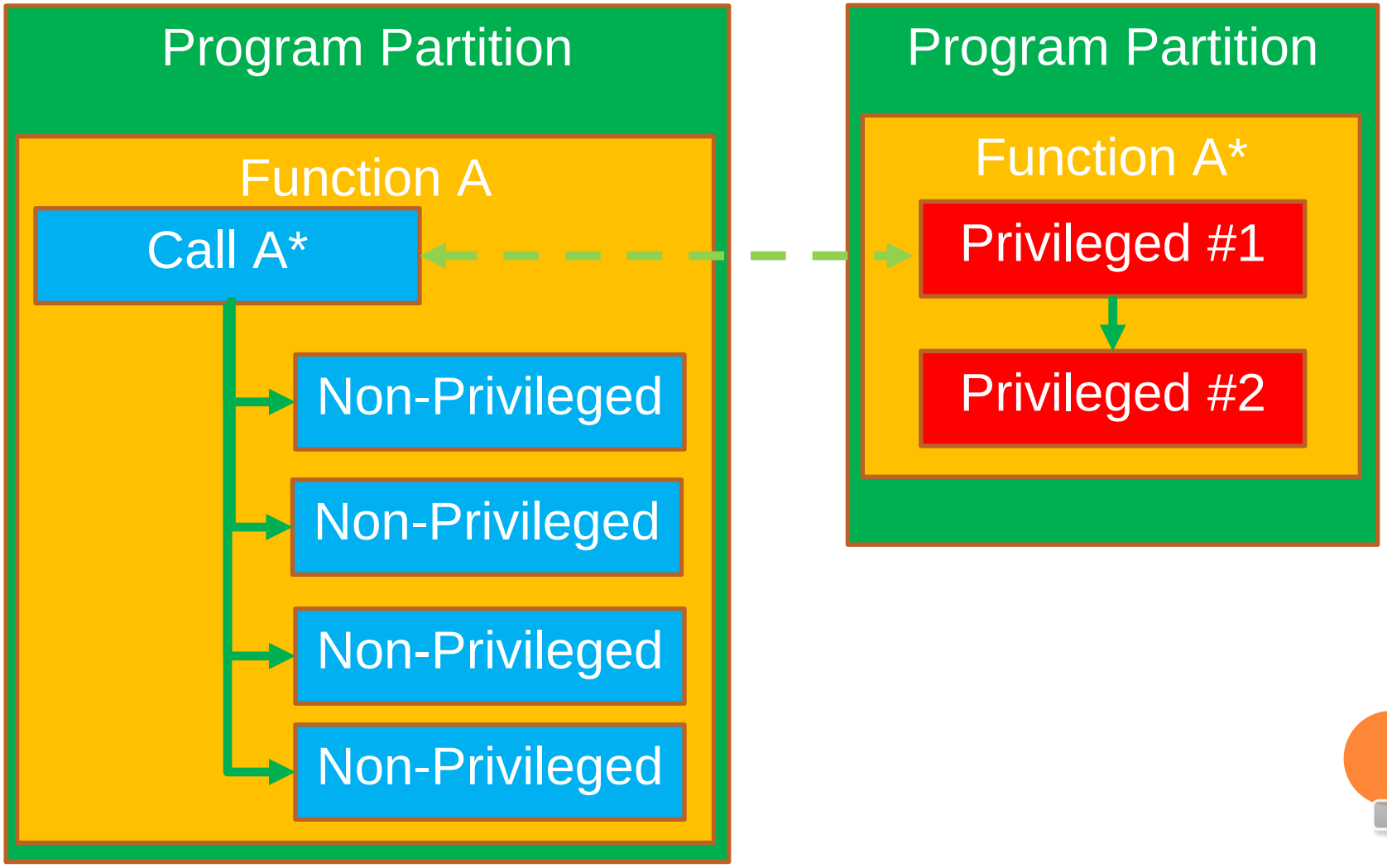
- Privileged code followed by non-privileged code



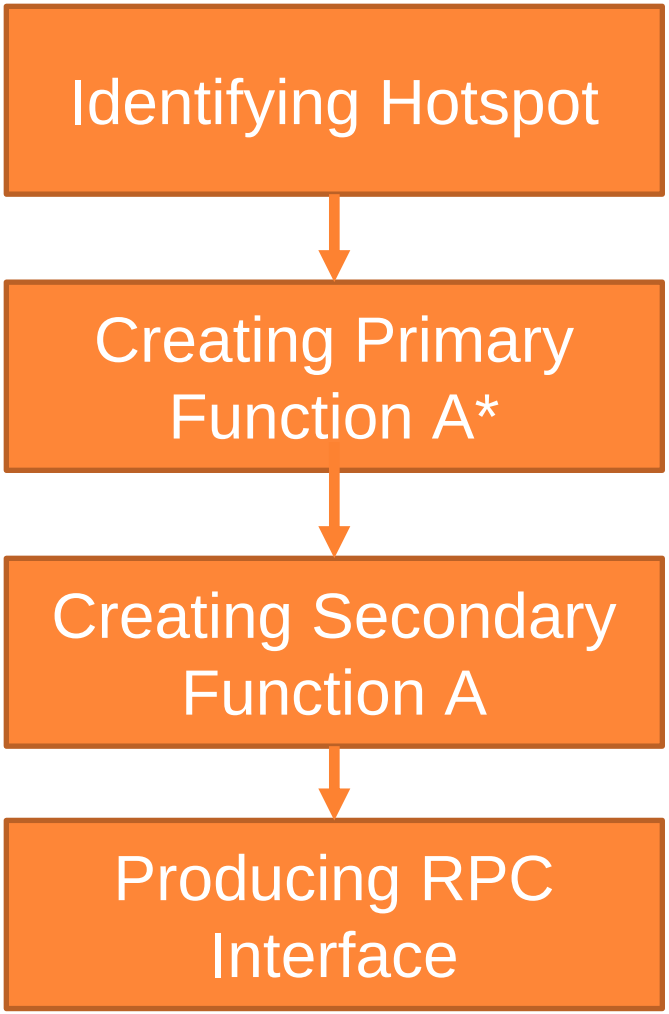
Pattern #2: Privileged to Non-Privileged – Complex Case



Solution #2: Privileged to Non-Privileged



Program Partitioning Steps



Implementation

- We implemented a prototype that partitions C/C++ programs.
  - identifies hotspots
  - creates primary functions
  - creates secondary functions
- All primary and secondary functions are automatically created in the form of source code.

Evaluation - Benchmarks

- The prototype is evaluated on 10 networking and interactive programs.
  - ssh
  - wget
  - Eight Linux shadow utilities, e.g. chsh, passwd, useradd, userdel, etc.

Evaluation Results

- The prototype is effective for all benchmark programs.
  - identifies hotspots
  - creates primary functions
  - creates secondary functions
- The mean runtime overhead introduced by partitioning is 5.2%.
- Merging code results in 1.38x speed up.

Conclusion

- Fine-grained partitioning enables separation of intertwined privileged code and non-privileged code.
- It improves performance of partitioned programs.

Thank You!

zhen.huang@depaul.edu