

TALISMAN: Tamper Analysis for Reference Monitors

Frank Capobianco¹, Quan Zhou¹, Aditya Basu¹, Trent Jaeger^{1,2} and Danfeng Zhang^{1,3}

¹Penn State, ²UC Riverside, ³Duke University

29 February 2024



Reference Monitors [Anderson, 1972]

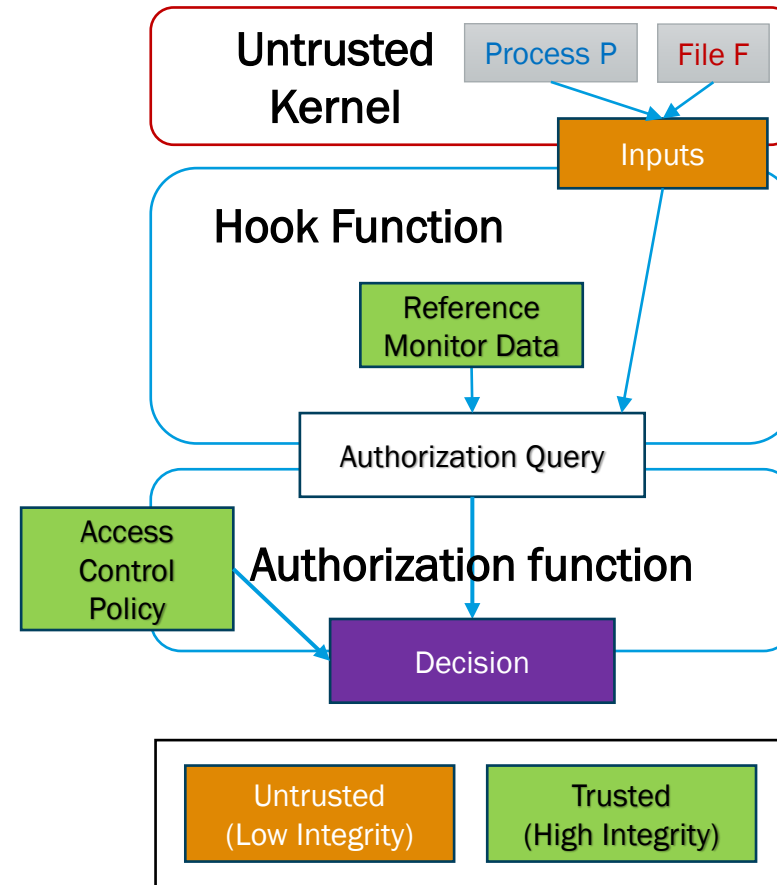
- Definition: Enforcement of *access control policy* in a system over *subjects*' ability to perform *operations* on *objects*.
- Properties
 - Complete mediation
 - Tamper-proof
 - Verifiable

Reference Monitors [Anderson, 1972]

- Definition: Enforcement of *access control policy* in a system over *subjects*' ability to perform *operations* on *objects*.
- Properties
 - Complete mediation
 - *Tamper-proof* ← *Very few prior work*
 - Verifiable

Reference Monitors

- Linux Security Module (LSM)
 - An example: granting or denying the request from *process P* of *writing to file F*.

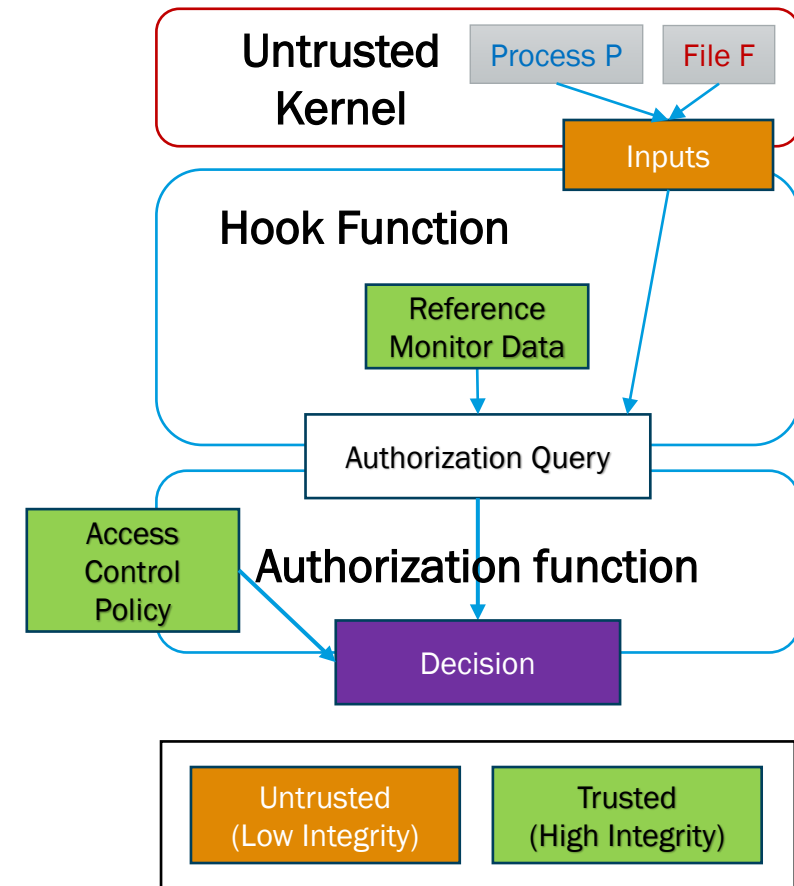


Tamper-proof Reference Monitors

- Existing tampering of RM
 - Security identifiers stored in host program
 - Incorrect use of request input
 - Authorization bypass

Problem Definition

- A static tamper analysis to detect violations of the tamper-proof requirement in LSM.
- A systematic endorsement mechanism for benign violations.

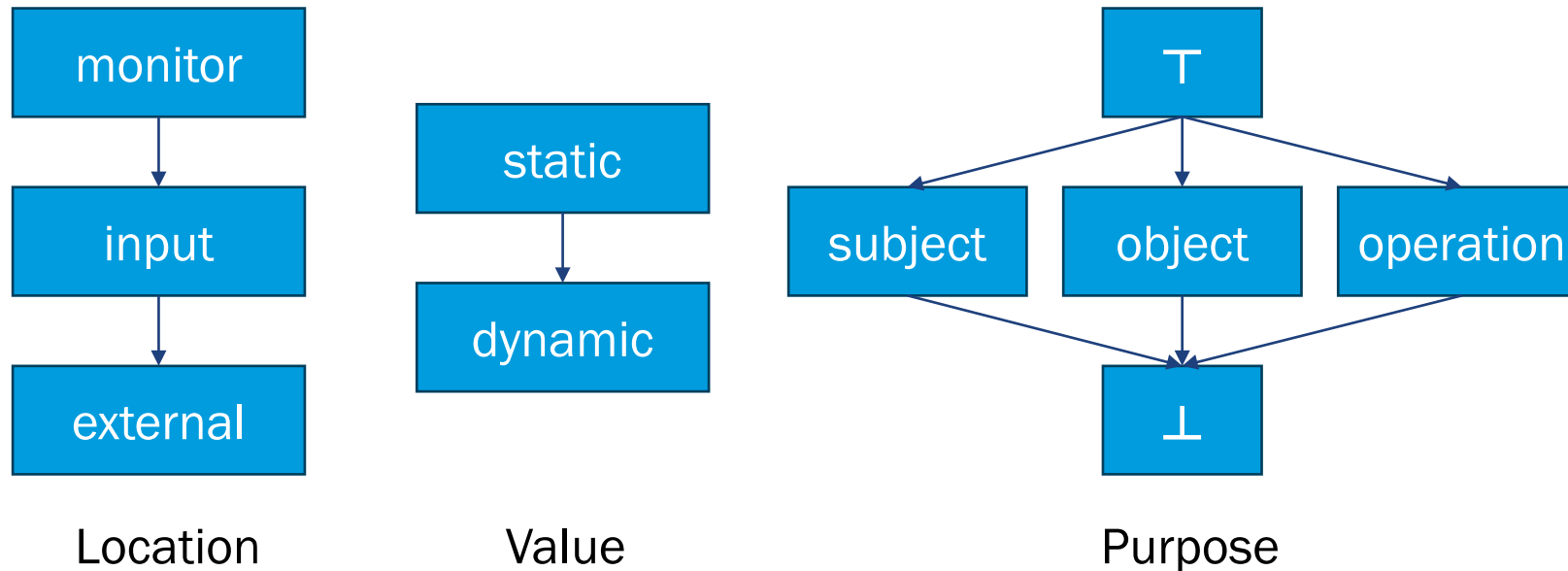


Approach Overview

- Modeling the tamper-proof property as an information flow control problem.
- Challenges
 - Standard noninterference reports too many false positives.
 - Manually endorsing each benign violation is impractical and error prone.

Tamper Analysis as an Information Flow Problem

- Three dimensions of the integrity lattice.
- Source: Any data flowing into the authorization functions.
- Sink: Arguments of the authorization functions.

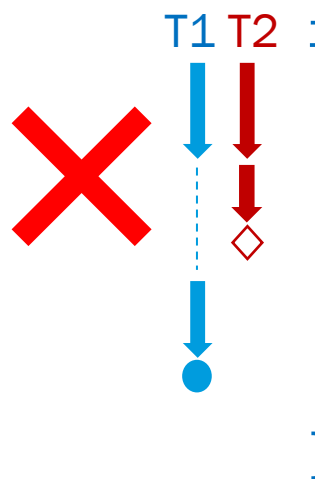


Strict Noninterference

- For *any* two different executions where only **low-integrity sources** change, the **high-integrity sink function calls** must stay intact.
 - The sink function parameters must not change.
 - Sink functions calls must match across different executions.

Strict Noninterference

- For *any* two different executions where only **low-integrity sources** change, the **high-integrity sink function calls** must stay intact.
 - The sink function parameters must not change.
 - Sink functions calls must match across different executions.
- **Strict noninterference rejects benign code!**



```
T1 T2 int apparmor_ptrace_traceme(task_struct *parent) {  
    int error = cap_ptrace_traceme(parent);  
    if (error)  
        return error;  
    else  
        return aa_ptrace(parent, current, PTRACE_MODE_ATTACH);  
}
```

Relaxation of Strict Noninterference

- Allow return of non-zero error code without a sink function call.

```
int apparmor_ptrace_traceme(task_struct *parent) {  
    int error = cap_ptrace_traceme(parent);  
    if (error)  
        return error;  
    else  
        return aa_ptrace(parent, current, PTRACE_MODE_ATTACH);  
}
```

Justification

- The request fails preliminary checks.
- No permission will be granted.

Relaxation of Strict Noninterference

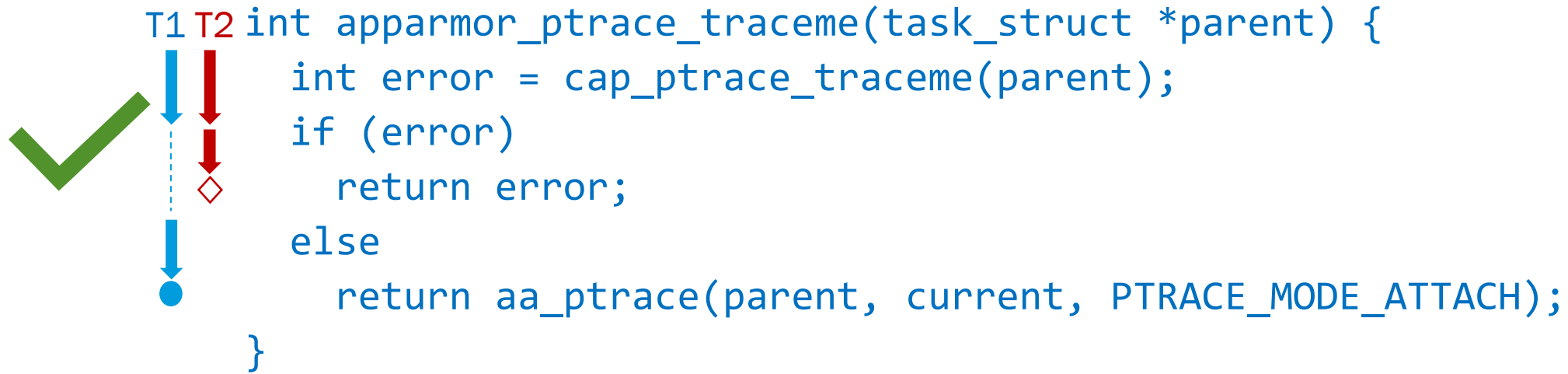
- Allow inputs to influence the sequence of sink function calls.

```
int apparmor_ptrace_traceme(task_struct *parent) {  
    int error = cap_ptrace_traceme(parent);  
    if (error)  
        return error;  
    else  
        return aa_ptrace(parent, current, PTRACE_MODE_ATTACH);  
}
```

Justification

- The intended functionality of the authorization process requires calls to different sink functions or even skip sink functions based on its parameters.

Relaxed Noninterference



Integrity guarantees

- Untrusted inputs cannot bypass all sink calls and return success.
 - Low-integrity inputs affecting authorization decisions are captured.
- * *Formalization provided in the paper.*

Endorsing Benign Violations

- Allow intended authorization bypass returning 0 (success).

```
int apparmor_path_chmod(path *path, umode_t mode) {  
    if (!mediated_filesystem(path->dentry->d_inode))  
        return 0;  
    ...  
}
```

Static endorsement

- Publicly accessible objects: `!mediated_filesystem(inode)`, `unlikely(IS_PRIVATE(inode))`, etc.
- Subjects with admin privilege

Endorsing Benign Violations

- Endorse input data from the untrusted kernel with dynamic integrity check.

```
int apparmor_ptrace_traceme(task_struct *parent) {  
    endorse_record(parent); // record low-integrity data  
    ...  
    return aa_ptrace(parent, current, PTRACE_MODE_ATTACH);  
}  
  
// in aa_ptrace, before decision is made  
    endorse_verify(parent); // verify recorded data is unmodified
```

Dynamic Endorsement

- Verify security identifiers' integrity status with reusable endorser templates.

Endorsing Benign Violations

- More complex cases
 - Multiple authorization checks.
 - Intertwined data on the purpose dimension.
- Solution
 - Manual justification based on system requirements and policy.
 - Manual insertion of endorsers.

Evaluations - Relaxed Noninterference

- 145 hook functions across 3 different LSM implementations

Integrity violations found by Relaxed NI (Strict NI) in each dimension

	Purpose	Value	Location	Total	FP Reduction
SELinux	348 (917)	28 (51)	429 (949)	805 (1917)	58%
Tomoyo	31 (292)	11 (149)	31 (329)	73 (770)	91%
AppArmor	1 (120)	7 (9)	20 (145)	28 (274)	90%

Relaxed noninterference removes **69.4%** of false positives.

Evaluations - Endorsement

- 145 hook functions across 3 different LSM implementations

Remaining flawed hook functions after endorsement and their root cause

	Hook Verification			Violations			
	Analyzed	Verified	Remaining Flawed	Case I	Case II		
				Return 0	Subject	Object	Operation
SELinux	102	63	39	29	23	31	17
Tomoyo	23	0	23	23	5	23	4
AppArmor	20	0	20	12	0	20	0

Evaluations - Endorsement

- 145 hook functions across 3 different LSM implementations

Remaining flawed hook functions after endorsement and their root cause

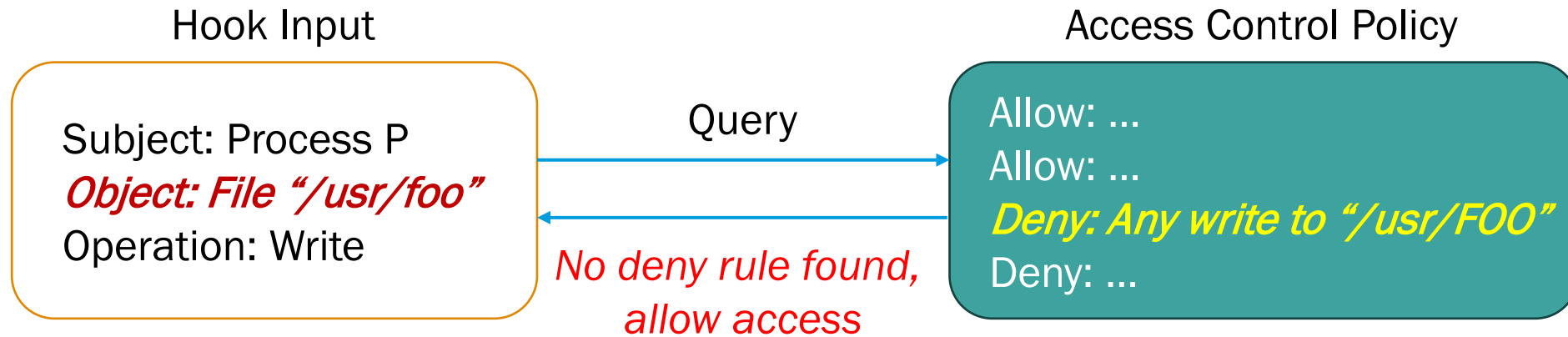
	Hook Verification			Violations			
	Analyzed	Verified	Remaining Flawed	Case I	Case II		
				Return 0	Subject	Object	Operation
SELinux	102	63	39	29	23	31	17
Tomoyo	23	0	23	23	5	23	4
AppArmor	20	0	20	12	0	20	0

Remaining Flaws

- Bad coding style
 - Returning 0 on failed null-pointer checks → 53 hooks
 - Returning 0 on invalid operations → 25 hooks
- Exploitable vulnerabilities
 - The object lookup vulnerability → All 43 hooks in AppArmor and Tomoyo
- Unknown
 - Require justification from system requirements and policy specification → 37 hooks

Identified Vulnerabilities

- The object lookup vulnerability in AppArmor and Tomoyo
 - Mismatch between case-insensitive file-system and case-sensitive canonicalization in LSMs



Summary

- Tamper-proof property can be modeled as an information flow integrity problem.
- Vanilla noninterference too strict for practical use, especially when implicit flows must be considered.
- Relaxation of strict noninterference removes most false positives in the tamper analysis.
- Further endorsement of information flow can be applied to benign integrity violations.
- TALSIMAN identifies *exploitable* security vulnerabilities in reference monitors.

THANK YOU!

