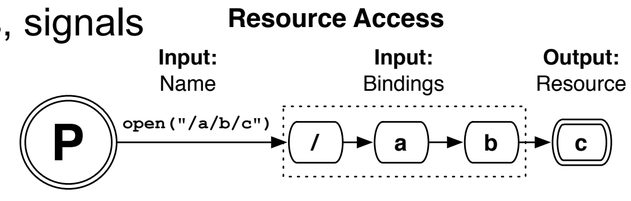
Resource Access Attacks: Detection and Defenses

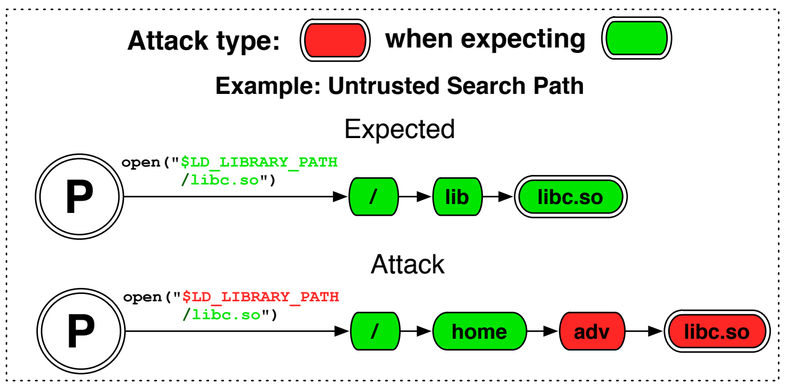
Hayawardh Vijayakumar, Xinyang Ge, Joshua Schiffman, Mathias Payer, Trent Jaeger

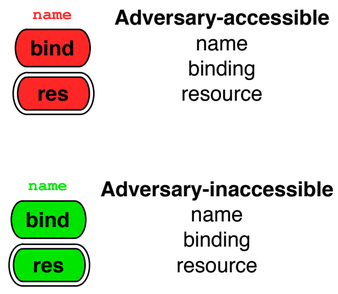
Resource Access and Attacks

- Programs access **resources** from the OS to function. Example: Files, sockets, directories, IPC channels, signals
- Resource access procedure: *inputs*: name, bindings; *output*: resource
- **Resource access attacks** occur when **adversaries control inputs** to resource access and direct programs to **unexpected output resources**

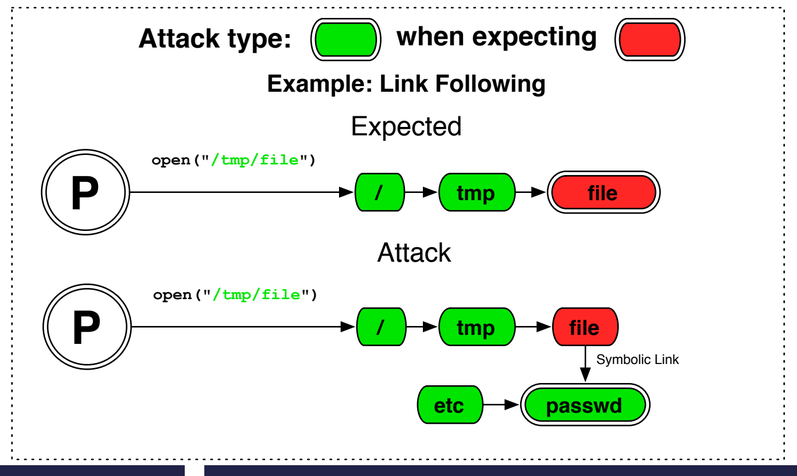


- Adversaries can direct programs to an adversary-accessible resource (Low Integrity) when the program expects an adversary-inaccessible resource (High Integrity)





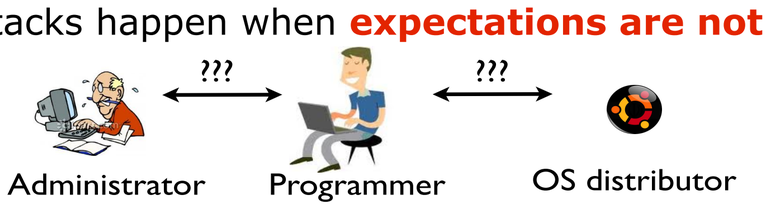
- Adversaries can direct programs to adversary-inaccessible resources (High integrity/secrecy) when program expects adversary-accessible resource (Low integrity/secrecy)

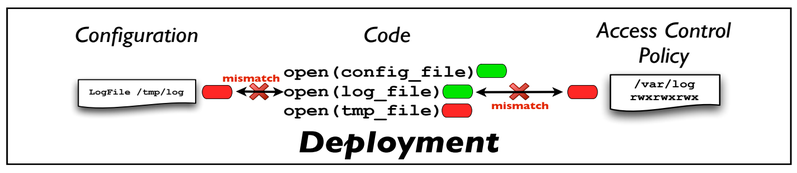


Fundamental Causes

Cause 1. Resource access involves multiple parties. Programmers write code implicitly expecting adversary access at each resource access. OS distributors frame access control policy and administrators configure programs, both of which determines adversary accessibility. Programmer expectations not conveyed to other parties

Attacks happen when **expectations are not met**





Cause 2. Secure program code checks (*filters*) are very **complicated** even where programmer expects adversary accessibility.

```
if ((fdbuf.st_dev != filebuf.st_dev
    || fdbuf.st_ino != filebuf.st_ino
    || fdbuf.st_nlink != 1
    || filebuf.st_nlink != 1
    || (fdbuf.st_mode & S_IFMT) != S_IFREG)) {
    error (_("%s must be a plain file
        with one link"), filename);
    close (fd);
    return EINVAL;
}
```

Defense Principles

Invariant 1. Only allow adversary access in the deployment where programmer expects it.

$$\text{Deployment Attack Surface } S \subseteq \text{Expected Attack Surface } P$$

$$\textbf{Unexpected Adversary Control}(r): \quad (r \in S) \wedge (r \notin P) \implies \mathit{Deny}$$

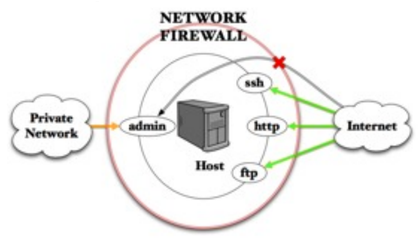
Invariant 2. Where adversary access occurs, limit to adversary accessible resources

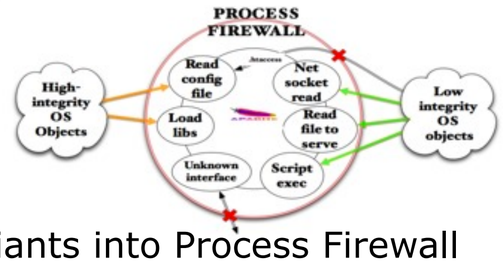
$$\textbf{Confused Deputy}(r): \quad (r \in P) \wedge (r \in S) \wedge (r \notin R) \implies \mathit{Deny}$$

Automatically infer programmer expectations. If programmer has not placed checks for a resource access system call, then system call is not expected to access adversary-accessible resources.

Enforcement

Require enforcement per resource access system call of programmer expectations without changing source code. Analogous to network firewall that constrains individual TCP/UDP ports without changing program configuration or code. Develop the **Process Firewall** that ports **`iptables`** to system call interface.





Convert invariants into Process Firewall rules.

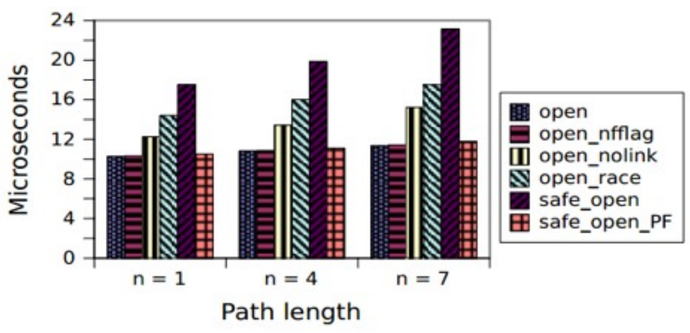
```
pftables --table FILTER             --append INPUT
         --object-label lib_t    --interface 0x15d04
         --vm_area_name /lib/ld.so --binary *
         --process-label *          --jump ACCEPT
```

Results

Programmers have lots of implicit expectations. Mismatches resulted in 2 previously-unknown vulnerabilities in the Apache webserver and 1 default misconfiguration. All these were defended by generated Process Firewall rules

Checks are more efficient as Process Firewall rules than in code.

Program	*Impl. Exp. %*	*Missing*	*Redundant*	*Vulns.*
Apache v2.2.22	65%	2	0	3
OpenSSH v5.3p1	17.6%	0	3	0
Samba3 v3.4.7	62.8%	0	5	0
Winbind v3.4.7	63.3%	0	0	0
Postfix v2.10.0	56.32%	0	9	0



Related Publications

Hayawardh Vijayakumar, Joshua Schiffman and Trent Jaeger, **Process Firewalls: Protecting Processes During Resource Access,** *In Proceedings of the 8th ACM European Conference on Computer Systems (EUROSYS 2013)*

Hayawardh Vijayakumar, Xinyang Ge, Mathias Payer and Trent Jaeger, **Enforcing Expectations to Protect Resource Access,** *In submission*

Sponsored by NSF (CNS-0905343) and AFOSR (FA9550-12-1-0166)