Program Misuse

Mitigations

Yan Shoshitaishvili
Arizona State University

Security Mitigations

Over time, system designers became resigned to the fact that security errors are here to stay… What to do?

Mitigations! Assuming that a security breach *will* occur, how can systems be designed to limit the damage?

We’ve already seen this with Hacking Team‘s separated networks for tool development.

We’ll see this over and over and over and over throughout the class.

Common theme: mitigations *reduce* but do not *eliminate* the potential for harm.

Example: /bin/sh SUID mitigation

Most command injection vulnerabilities end up hijacking `/bin/sh`.

Mitigation:
If `/bin/sh` is run as SUID
(i.e., `eUID == 0` but `rUID != 0`),
it will *drop privileges* to the rUID
(i.e., `eUID = rUID` and `rUID != 0`),

To disable that: `sh -p`

Example: Wireshark

Wireshark (a popular network sniffer) has historically been the subject of security problems:

- It typically runs as root to have the permissions to sniff network traffic.
- It includes an enormous amount of "protocol parsers" to analyze different types of network traffic, resulting in a very large attack surface.

Mitigation: wireshark‘s developers split it into two programs, one which dumps traffic (dumpcap) and one which analyzes it (wireshark). Only dumpcap needs root privileges.

More reading: https://wiki.wireshark.org/Development/PrivilegeSeparation

# General Mitigations: Program Misuse

How do we mitigate program misuse in general?

Sandboxing: we run the program in a setting that’s cordoned-off from sensitive data and capabilities.

How? We’ll explore this later in the class during the Sandboxing module!