Stabilizing Trust and Reputation for Self-Stabilizing Efficient Hosts in Spite of Byzantine Guests

Shlomi Dolev and Reuven Yagel, Ben-Gurion University of the Negev, Beer-Sheva, Israel
dolev,yagel@cs.bgu.ac.il

Host Example: Virtualization

- Growing popularity, e.g., in server farms
  - Better resource allocation (consolidation)
  - Better separation and protection
- Terms: Host-VM server, Guest, VMM-hypervisor, VM
- Various levels
  - Hardware/OS/Application
- Major players: VMware, MS Virtual Server, Xen, OpenVZ
  - Recently VT-technology, KVM

Recovery and Byzantines

- A Byzantine guest might take advantage of
  - Soft errors
  - Hypervisor bugs
  - OS level security/robustness issues (e.g. unaware user)
- Following the above, a self-stabilizing host needs to recover and gain control
- Suppose we can always detect a Byzantine guest. Can we just remove it?
- Must we continue running the Byzantine forever?

Self-stabilizing OS

- Black-box approach based on restarts
- Tailored kernel
  - processor scheduling (SAACS04)
  - Memory management (SSS05, SOSP05)
  - Device drivers (SSS06)
- Guarantee process stabilization and fairness
- Assuming no Byzantine behavior!

Previous Work: A Self-Stabilizing Computing Stack

- Self-Stabilizing Program
- Self-Stabilizing Middleware
- Stabilization Preserving Compiler
- Self-Stabilizing Operating System
- Self-Stabilizing Processor

Settings and Requirements

- Every configuration (processor registers/ memory/IO controllers) is possible
- Some programs might exhibit Byzantine behavior
- Achieving:
  1. Guest stabilization preservation
  2. Efficiency guarantees
Stabilizing Trust & Reputation

- Give a chance to change reputation, both ways...
- Maybe the reputation history is corrupted
- Constantly fading old reputation and accumulating new reputation (e.g. [7])
- Grant resources according to stabilizing trust and reputation

Byzantine Behavior Detection

- Defining host-guest contracts
- Offline scan on program's code - Plug-in framework
- Injection of sanity checks and access restrictions - sandboxing
  - Preserving program semantics
  - Reloading programs and updating reputation

Main Concepts

- Secure booting of a minimal TCB
- Stabilization of host kernel
- Offline Byzantine behavior detectors
- Runtime anti-Byzantine contract enforcers
- Stabilizing trust and reputation
  - Starting from ant state, host stabilization leads to efficient execution of non-Byzantine guests

Update Trust and Reputation - Increase and Decay

Increase-Bad-Reputation (process_recrod)
1. process_recrod.reputation[0] = process_recrod.reputation[0] + 1
2. return

Decay-Reputation (process_recrod)
1. for i in (HISTORY_SIZE - 1) .. 1
2. do process_recrod.reputation[i] = process_recrod.reputation[i - 1]
3. process_recrod.reputation[0] = 0
4. return

Detection Example

1. add ax, bx
2. mov ds, ax
3. mov word [0x292], 3
4. a. or ax, SEGMENT_MASK
   b. jz afterSanityCheck
   c. call IncreaseBadReputation
   d. mov ax, SEGMENT_MASK
   e. afterSanityCheck:

Implementation

- Pentium in real-mode, single address space
  - Simple
  - common for sensors/microcontrollers
  - Protected mode & VM mechanisms can be handled accordingly
- Kernel size: ~4K
  - TinyOS (sched. Only) ~1K, VxWorks ~10K, Linux (default xconfig) ~4M
- Fault injection with the Bochs simulator
- Prototype only
Efficiency

- Tuning of activation time for code refresh and enforcers
- Fast access restrictions and reducing consistency checks rate
- Using auxiliary processors

Conclusion

- We presented a general design for stabilizing host systems (e.g., virtualization)
- Self-stabilizing reputation gains efficiency in spite of Byzantine programs
- By supplying an infrastructure for practical self-stabilizing systems, robust and dependable systems can be achieved

http://www.cs.bgu.ac.il/~yagel/sos

Thank you