Turret: A Platform for Automated Adversarial Testing of Distributed Systems and Network Protocols

Hyojeong Lee, Jeff Seibert (now with Google), Endadul Hoque, Charles Killian (now with Google), Cristina Nita-Rotaru

Department of Computer Science and CERIAS
Purdue University
Dependable and Secure Distributed Systems
Research Overview

- Overarching goal:
  - Create and build distributed systems and network protocols that achieve **security**, **availability**, and **performance** in spite of failures, misconfigurations, and attacks.

- Approach:
  - Combine theoretical principles and experimental methodologies from distributed systems, cryptography, networking, information theory, and machine learning.
  - Create systems and protocols based on provable guarantees and validated in realistic environments.
Challenges for Building Trustworthy Systems

- **Failures:**
  - Computers and networks fail in many (often unpredictable) ways

- **Attacks:**
  - Computers get compromised

- **Constraints:**
  - Real-time constraints

- **Requirements:**
  - Performance requirements

- **Complexity:**
  - Complex systems fail in complex ways
Towards Robust Systems and Protocols

- **Model checking**
  - Mathematically check if the design meets its goals at every possible state

- **Simulation**
  - Test correctness and performance under various environments

- **Real deployments**
  - Capture traces, use replaying for debugging
But …

- Model checking
  - Design choices made at implementation time are not captured by models
  - New vulnerabilities introduced during implementation

- Simulation
  - Does not capture the interaction between protocols and OS

- Real deployments
  - Difficult and expensive
  - Better to find problems before they occur in the wild
Adversarial Testing of Implementations

- Test software beyond its functionality by
  - Examining edge cases
  - Testing boundary conditions
  - Conducting destructive testing (insider attacks)

- Captures both bugs and vulnerabilities
- Allows testing for insider attacks
- Realistic environments: binaries in native environment with reproducible conditions
A correct but slow system might not be that useful

Slow processing might be an indication of a bigger issue

Slow performance might mean money both for service provider and users

Some sources estimate that Amazon gets $9,823 every five seconds.
Automated adversarial testing for implementations of distributed systems and protocols in realistic environments, looking at significant degradation of performance
Performance Attacks

- **Performance attacks**: attacks that make the system slow down
  - While the network is stable, the distributed system or network protocol does not achieve the performance that it can achieve

- A set of actions that:
  - deviate from the protocol
  - taken by a group of malicious nodes
  - Δ-worse than benign scenario in performance
Malicious Actions through Messages

- Message delivery actions
  - Delay
  - Divert
  - Duplicate
  - Drop

- Message manipulation actions: lie field by field (based on field type range and on original value)
  - Min and max
  - Zero
  - Scaling
  - Spanning
  - Random
Our Previous Work: Gatling

- **Fault injector**: Basic message delivery and manipulation actions
- **Model checker**: Greedy algorithm builds up attack
- **Framework**: Event-based simulator combining model checking and fault injection in the Mace toolkit
- Found 48 attacks in 9 systems

*Gatling: Automatic Attack Discovery in Large-Scale Distributed Systems.*
Gatling Limitations

- Systems need to be rewritten in MACE if they were written in other languages
- Source-to-source compiler to get the deployed binary
- The testing environment does not capture the deployment environment: Vulnerabilities are often related to the environment: OS, libraries, etc
Test unmodified implementation

Environment as close to the deployment environment as possible
  OS, libraries, etc.

Inject malicious actions

Reproducible results
Attack Finding Algorithms: Brute Force

- **Approach**
  - Pre-generate strategies
  - Run each strategy and evaluate performance

- **Pros.**
  - Simple: Does not require control during the execution of an attack scenario, but just the ability to start the system and stop it

- **Cons.**
  - Overhead: Wastes time due to unnecessary executions (system runs in real time)
Attack Finding Algorithms: Greedy

- **Approach**
  - Test all actions for a message type and select greedily (strongest) at each branch point
  - Repeat until confident

- **Pros.**
  - Can build up attacks
  - Saves some unnecessary executions

- **Cons.**
  - Some redundant executions
  - More complex as it requires execution branching
Attack Finding Algorithms: Weighted Greedy

- Approach
  - Prioritize actions based on history

- Pros.
  - More efficient: Removes redundant execution

- Cons.
  - Still requires execution branching
Checkpointing methods
- Log/replay execution
- **Take/load snapshot**

Taking snapshot of the entire distributed system
- State of nodes + messages on the fly
Implementation

Implementation: Malicious Proxy

- Intercept a packet at the tap device
  - TCP: Send up to application layer to maintain socket status
  - UDP: Strip off headers and modify directly
- Message type based attacks
  - Small compiler to parse message and recognize the type
- Message format provided by user
Implementation: Execution Branching

- NS-3 modification
  - Save / Load
  - Event queue, objects and events
  - Freeze / Resume
  - Separate clock

- KVM modification
  - TSC – used to calibrate drift between host and guest
  - Para-virtual clock: report manipulated TSC
Implementation: Page Sharing Aware Snapshot Management

- KSM: modify to expose sharing information
- KVM: modify to use sharing information
Running Turret

User must

1. Implement a simulation driver
2. Provide an impact score
3. Provide the layout of messages
4. Set several parameters
5. Find attacks!

The user does not need to provide a malicious implementation.
How We Use Turret

- As our "Red Team" for testing protocols we create and implement – we use it our protocols developed in the MRC Darpa Grant
- To test distributed systems: tested 5 systems (PBFT, Zyzyva, Aardvak, Steward, Prime), found 30 total, 24 not previously reported problems
- To test wireless routing protocols (the benefit of emulation): tested 5 different protocols (AODV, ARAN, OLSR, DSDV, BATMAN), found 37 attacks and 3 bugs
- As a projects and testing platform in the Distributed Systems class Spring 2013, plan to do it again in Fall 2014
Ongoing Work

- Research:
  - Take into account information about the protocol state machine
  - Scaling to larger systems
- For use in class
  - Integrate better debugging and feedback tools
  - Developing more projects
  - Provide it as a service
- Code and projects will be released in the next few weeks
Conclusion

- Introduced Turret, which automatically finds performance attacks on distributed systems implementations
  - Implemented using KVM and NS3
- Applied Turret to intrusion tolerant systems and wireless routing
- Used it as teaching tool