Shadow: Scalable Simulation for Systems Security Research

CrySP Speaker Series on Privacy
University of Waterloo
January 20th, 2016

Rob Jansen
U.S. Naval Research Laboratory
rob.g.jansen@nrl.navy.mil
@robgjansen
Talk Outline

- Shadow and how it works
- Tor research case study: Kernel-Informed Socket Transport
- Future directions
Why should you care?

- Expedite research and development
- Evaluate software mods or attacks **without** harming real users
- Understand **holistic effects** before deployment
- Shadow supports simulation for **new applications**
Thread 0

EXPERIMENTATION OPTIONS
Desirable Properties

- Scalable
- Accurate
- Reproducible
- Controllable

Goal!
# Network Research Methods

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live Network</td>
<td>Hard to manage, lengthy deployment, security risks</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>Hard to manage, bad at modeling, not scalable</td>
</tr>
<tr>
<td>Simulation</td>
<td>Not generalizable, inaccurate</td>
</tr>
<tr>
<td>Emulation</td>
<td>Larger overhead, kernel complexities</td>
</tr>
</tbody>
</table>
Simulation vs Emulation

- **Time (simulation wins)**
  - Real time vs “as-fast-as-possible” execution
  - Emulation time must advance in synchrony with wall-clock time, or the virtual environment may become “sluggish” or unresponsive
  - Easier to slow down than to speed up execution!

- **Realism (emulation wins)**
  - Uses host OS kernel, protocols, applications
  - Can run anything that runs on OS
Thread 1

SHADOW
What is Shadow?

- Parallel discrete-event network simulator
- Models routing, latency, bandwidth
- Simulates time, CPU, OS
  - TCP/UDP, sockets, queuing, threading
- Emulates POSIX C API on Linux
- Directly executes apps as plug-ins
Simulation Environment

Hosts

Logical processing units with independent state
Simulation Environment

Routing elements (nodes, links) and attributes (bandwidth, latency, packet loss)
Simulation Environment

- Hosts
- Network
- Global Clock

Holds current virtual time (distinct from physical time)
Simulation Environment

- Hosts
- Network
- Global Clock
- Event

Processing task for a host at a specific time
Simulation Environment

- Hosts
- Network
- Global Clock
- Event
- Event Queue

Holds events sorted by time (min heap)
Discrete Event Engine

- Facilitate **communication**: exchange events between hosts through the network
- “as-fast-as-possible” execution
Discrete Event Engine

While !end
- Get next event
- Update clock
- Process event
- Enqueue events

- Facilitate communication: exchange events between hosts through the network
- “as-fast-as-possible” execution

- Facilitate communication: exchange events between hosts through the network
- “as-fast-as-possible” execution
Parallel Discrete Event Engine

Host workloads split among physical threads

Each physical thread has event queue

Synchronization problem!
Conservative Synchronization

- Ensure causality
  - events must occur in correct order (not in the past)
Virtual Network Routing

- Network graph model

- If complete:
  - Lookup link
  - Get latency

- Else
  - Compute shortest path
  - Sum link latencies
  - Cache result

Latency and packet loss

Host connection points (IPs)
Executing Applications on Hosts

- Load programs as dynamic shared object libraries

<table>
<thead>
<tr>
<th>Addr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>0</td>
</tr>
<tr>
<td>0xB</td>
<td>0</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Compile with Clang, extract state addresses with LLVM pass

Each program loaded only once per thread
Virtual Process Management

Save default values on initial load

Copy state for each virtual process
Virtual Process Management

Swap state into/out of memory as virtual processes are switched
Virtual Thread Management

Reentrant portable threads (rpth)
• async. thread-safe, user-land non-preemptive cooperative threading
• Uses make/set/get/swapcontext() magic to jump program stacks when EWOULDBLOCK

Virtual thread layer
Virtual Thread Management

Each virtual process has a private rpth instance

Shadow thread
Virtual Thread Management

Spawns an rpth thread to call the program \texttt{main()} function

Shadow thread

“main” thread
Virtual Thread Management

Program may spawn auxiliary threads

Shadow thread

“main” thread

spawned threads
Execution Flow with rpth

- Swap in virtual process and rpth state
- Return to Shadow thread when all spawned rpth threads would block:
- Swap out virtual process and rpth state
Function Interposition

LD_PRELOAD=libpreload.so (socket, write, pthread_create, …)

Function calls are redirected to simulated counterpart
Simulating a Kernel

- Sockets and queuing
- Network protocols – TCP, UDP
- Threading (pthread)
- Randomization (maintain determinism)
- CPU usage
Thread 2

KERNEL INFORMED SOCKET TRANSPORT

With John Geddes, Chris Wacek, Micah Sherr, and Paul Syverson
Anonymous Communication: Tor

- Client
- Tor Circuit
- Tor Relays
- Tor Directory Service
- Internet Service
This Talk

- **Where is Tor slow?**
  - Measure public Tor and private Shadow-Tor networks
  - Identify circuit scheduling and socket flushing problems

- **Design KIST: Kernel-Informed Socket Transport**
  - Use TCP snd_cwnd to limit socket writes

- **Evaluate KIST Performance and Security**
  - Reduces kernel and end-to-end circuit congestion
  - Throughput attacks unaffected, speeds up latency attacks
Outline

- Background
- Instrument Tor, measure congestion
- Analyze causes of congestion
- Design and evaluate KIST
  - Performance
  - Security
Relay Overview
Relay Overview

Tor circuits are multiplexed over a TCP transport
Relay Overview
Relay Internals

Kernel Input  Tor Input  Tor Output  Kernel Output

Tor Circuits

Opportunities for traffic management
Outline

- **Background**

- **Instrument Tor, measure congestion**

- **Analyze causes of congestion**

- **Design and evaluate KIST**
  - Performance
  - Security
Live Tor Congestion - libkqtime

Kernel Input  Tor Input  Tor Output  Kernel Output

Tor Circuits
Live Tor Congestion - libkqtime

Kernel Input → Tor Input → Tor Circuits → Tor Output → Kernel Output

- tag
- match
- tag
- match
Live Tor Congestion - libkqtime
Shadow Network Simulation

- **Enhanced Shadow** with several missing TCP algorithms
  - CUBIC congestion control
  - Retransmission timers
  - Selective acknowledgements (SACK)
  - Forward acknowledgements (FACK)
  - Fast retransmit/recovery
- **Designed largest known private Tor network**
  - 3600 relays and 12000 simultaneously active clients
  - Internet topology graph: ~700k nodes and 1.3m links
Shadow-Tor Congestion – UIDs

Track the UID

UID

Track the UID

UID

UID
Shadow-Tor Congestion – UIDs

Kernel Input  Tor Input  Tor Circuits  Tor Output  Kernel Output
Tor and Shadow-Tor Congestion

Congestion occurs almost exclusively in outbound kernel buffers
Outline

- Background
- Instrument Tor, measure congestion
- Analyze causes of congestion
- Design and evaluate KIST
  - Performance
  - Security
Analyzing Causes of Congestion

Queuing delays in kernel output buffer
Analyzing Causes of Congestion

Problem 1: Circuit scheduling

Problem 2: Flushing to Sockets

Queuing delays in kernel output buffer
Problem 1: Circuit Scheduling

Libevent schedules one connection at a time
Problem 1: Circuit Scheduling

Tor only considers a subset of writable circuits.

Libevent schedules one connection at a time.
Problem 1: Circuit Scheduling

Tor only considers a subset of writable circuits

Circuits from different connections are not prioritized correctly

Libevent schedules one connection at a time
Problem 2: Flushing to Sockets

Tor Output

Tor Circuits

Kernel Output

FIFO

Queuing delays in kernel output buffer
Problem 2: Flushing to Sockets

Worse priority traffic (high throughput flows)

Tor Circuits

Tor Output

Kernel Output

FIFO
Problem 2: Flushing to Sockets

Better priority traffic (low throughput flows)

Worse priority traffic (high throughput flows)
Problem 2: Flushing to Sockets

- **Worse priority traffic** (high throughput flows)
- **Better priority traffic** (low throughput flows)

Must wait for kernel to flush socket to network (blocked on TCP cwnd)

Kernel Output

FIFO
Problem 2: Flushing to Sockets

Better priority traffic (low throughput flows)

Worse priority traffic (high throughput flows)

Reduces effectiveness of circuit priority

FIFO
Outline

- Background
- Instrument Tor, measure congestion
- Analyze causes of congestion
- Design and evaluate KIST
  - Performance
  - Security
Ask the kernel, stupid!

- Utilize `getsockopt` and `ioctl` syscalls

\[
\text{socket\_space} = \text{sndbufcap} - \text{sndbuflen}
\]

\[
\text{tcp\_space} = (\text{cwnd} - \text{unacked}) \times \text{mss}
\]
Kernel-Informed Socket Transport

- Don’t write it if the kernel can’t send it; bound kernel writes by:
  - Socket: \( \text{min(socket\_space, tcp\_space)} \)
  - Global: upstream bandwidth capacity

Solution to Problem 2
Kernel-Informed Socket Transport

- Don’t write it if the kernel can’t send it; bound kernel writes by:
  - Socket: min(socket_space, tcp_space)
  - Global: upstream bandwidth capacity

- Choose globally from all writable circuits

Solution to Problem 1
Kernel-Informed Socket Transport

- Don’t write it if the kernel can’t send it; bound kernel writes by:
  - Socket: \( \text{min(socket\_space, tcp\_space)} \)
  - Global: upstream bandwidth capacity

- Choose globally from all writable circuits

- Try to write again before kernel starvation
KIST Reduces Kernel Congestion
KIST Increases Tor Congestion

![Graph showing cumulative fraction over time with different lines for vanilla, global, and KIST.]
KIST Reduces Circuit Congestion

![Graph showing the comparison between vanilla, global, and KIST methods over time. The graph plots cumulative fraction on the y-axis against time (ms) on the x-axis. The graph demonstrates that KIST reduces circuit congestion compared to the vanilla and global methods.](image)
KIST Improves Network Latency

![Graph showing Time to First Byte (s) vs Cumulative Fraction for different configurations (vanilla, global, KIST)]
Outline

● Background

● Instrument Tor, measure congestion

● Analyze causes of congestion

● Design and evaluate KIST
  – Performance
  – Security
Traffic Correlation: Latency

Goal: narrow down potential locations of the client on a target circuit

Hopper et.al. CCS’07
Traffic Correlation: Latency

-Inject redirect or javascript
-Start timer
Traffic Correlation: Latency

Hopper et.al. CCS’07
Traffic Correlation: Latency

- Stop timer
- Estimate latency

Hopper et.al. CCS’07
Latency Attack
| estimate – actual |

Cumulative Fraction

Difference (ms)

vanilla
KIST

0
50
100
150
200
250

0.0
0.2
0.4
0.6
0.8
1.0
Latency Attack

num pings until best estimate

Cumulative Number of Pings vs. Cumulative Fraction

- Vanilla
- KIST
Traffic Correlation: Throughput

Goal: find guard relay of the client on a target circuit

Mittal et.al. CCS’11
Traffic Correlation: Throughput

Probe throughput of all guard relays

Mittal et.al. CCS’11
Traffic Correlation: Throughput

Correlate throughput between exit and probes

Mittal et.al. CCS’11
Throughput Attack Results

![Graph showing cumulative fraction over correlation score with two lines, one labeled vanilla and the other KIST.]

- Throughput Attack Results
- Graph showing cumulative fraction over correlation score with two lines, one labeled vanilla and the other KIST.
Summary/Conclusion

- Shadow
- Where is Tor slow?
  - KIST complements other performance enhancements, e.g. circuit priority
- Future work
  - Optimize Shadow threading algorithms
  - Distribute Shadow across processes/machines

shadow.github.io
github.com/shadow
robgjansen.com, @robgjansen
rob.g.jansen@nrl.navy.mil

think like an adversary