Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm

Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, Stefan Savage

Collaborative Center for Internet Epidemiology and Defenses (CCIED)
University of California, San Diego

UCSD

Background

- Large-scale host exploitation a serious problem
  - Worms, viruses, bots, spyware…
  - Supports an emerging *economic* criminal enterprise
    - SPAM, DDoS, phishing, piracy, ID theft…
    - Two weeks ago, one group arrested—controlled 1.5 M hosts!
- Quality and sophistication of malware increasing rapidly

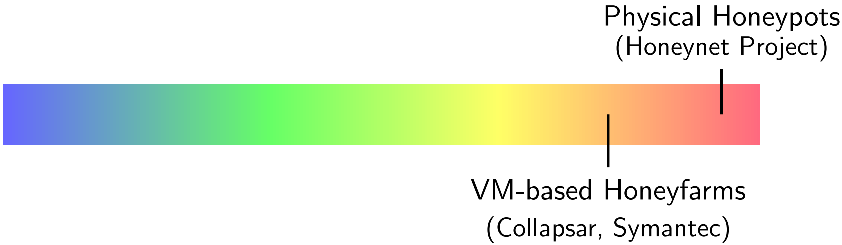
Motivation

- Intelligence about new threats is critical for defenders
- Principal tool is the *network honeypot*
  - Monitored system deployed for the *purpose* of being attacked
- *Honeyfarm*: Collection of honeypots
  - Provide early warning, accurate inference of global activity, cover wide range of software
- Design issues
  - Scalability: How many honeypots can be deployed
  - Fidelity: How accurately systems are emulated
  - Containment: How well innocent third parties are protected
- Challenge: tension between scalability and fidelity

Honeyfarm Scalability/Fidelity Tradeoff

High Scalability **High Fidelity**

Honeyfarm Scalability/Fidelity Tradeoff

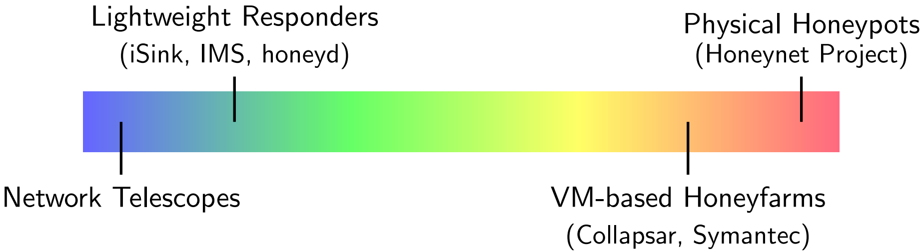


High Scalability

High Fidelity

Execute real code

Honeyfarm Scalability/Fidelity Tradeoff



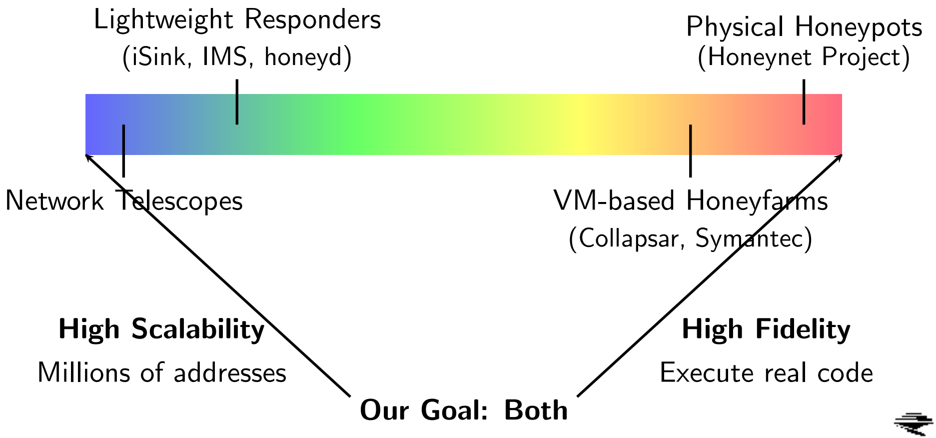
High Scalability

Millions of addresses

High Fidelity

Execute real code

Honeyfarm Scalability/Fidelity Tradeoff



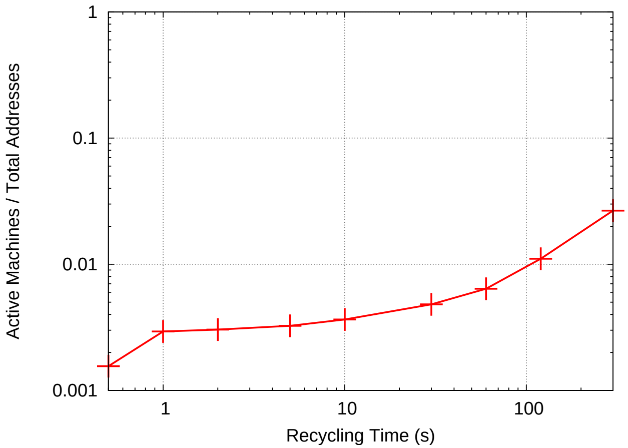
Approach

- Dedicated honeypot systems are overkill
- Can provide the *illusion* of dedicated systems via aggressive resource multiplexing at network and host levels

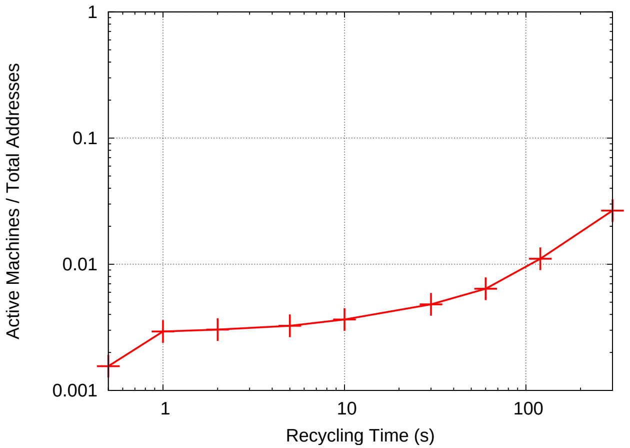
Network-Level Multiplexing

- Most addresses don't receive traffic most of the time
  - $\Rightarrow$ Apply late binding of IP addresses to honeypots
- Most traffic that is received causes no interesting effects
  - $\Rightarrow$ Allocate honeypots only long enough to identify interesting behavior
  - $\Rightarrow$ Recycle honeypots as soon as possible
- How many honeypots are required?
  - For a given request rate, depends upon recycling rate

Effectiveness of Network-Level Multiplexing



Effectiveness of Network-Level Multiplexing

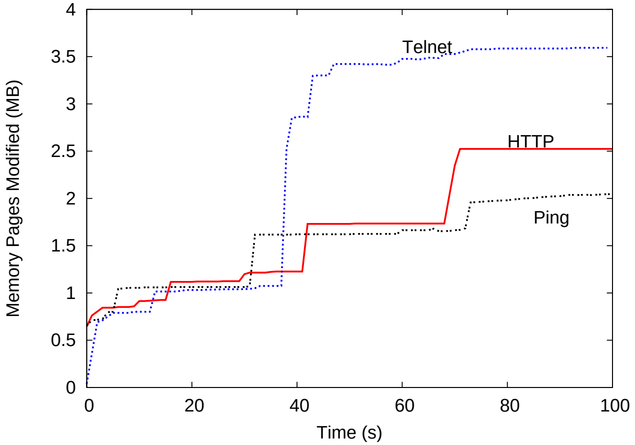


2–3 orders of magnitude improvement!

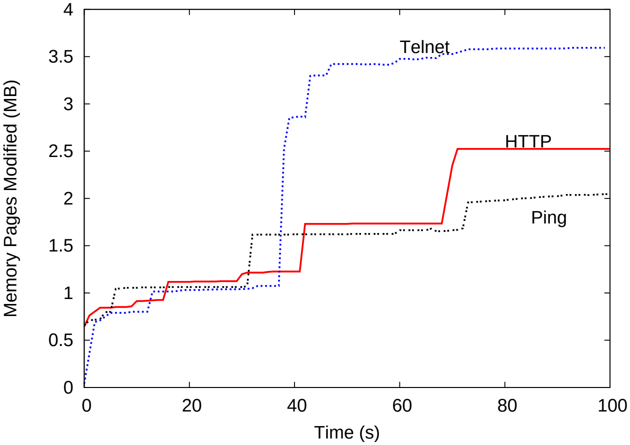
Host-Level Multiplexing

- CPU utilization in each honeypot quite low (milliseconds to process traffic)
  - ⇒ Use VMM to multiplex honeypots on a single physical machine
- Few memory pages actually modified when handling network data
  - ⇒ Share unmodified pages among honeypots within a machine
- How many virtual machines can we support?
  - Limited by unique memory required per VM

Effectiveness of Host-Level Multiplexing

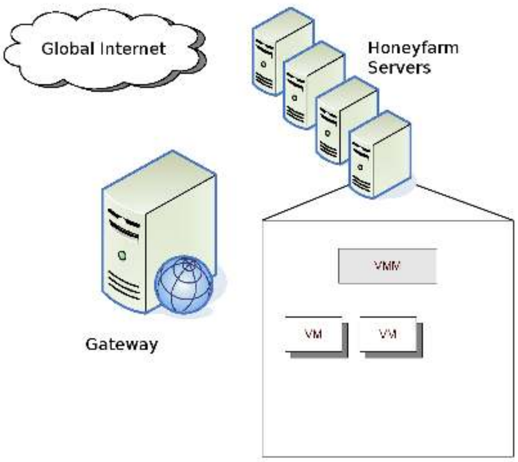


Effectiveness of Host-Level Multiplexing



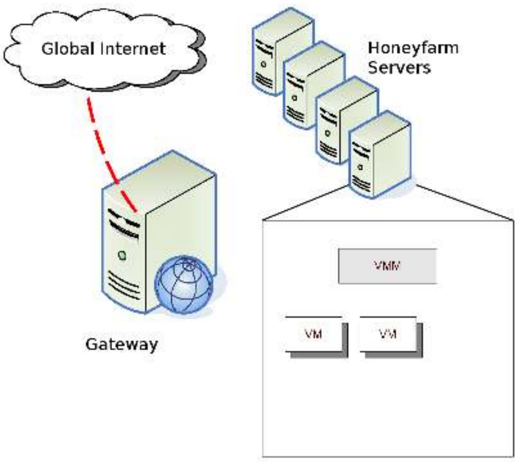
Further 2–3 orders of magnitude improvement

The Potemkin Honeyfarm Architecture



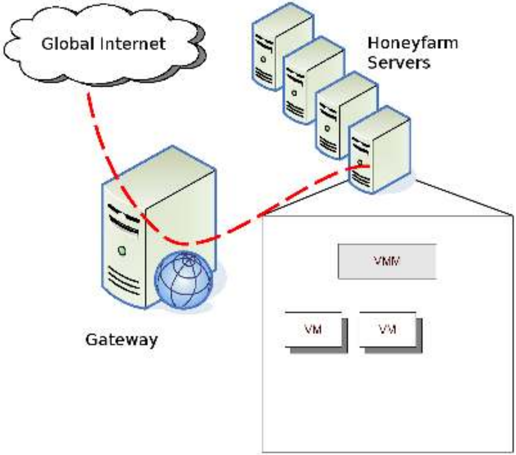
- Two components:
  - Gateway
  - VMM

The Potemkin Honeyfarm Architecture



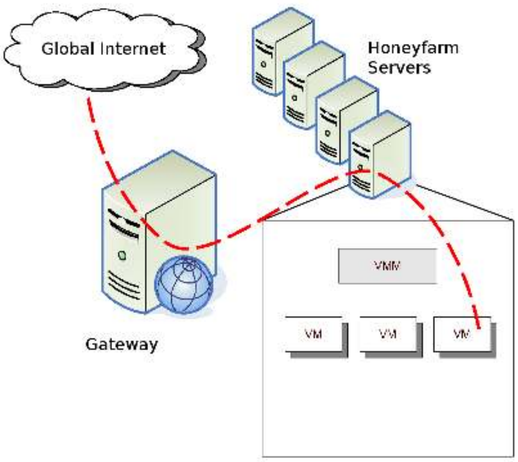
- Two components:
  - Gateway
  - VMM
- Basic operation:
  - Packet received by gateway

The Potemkin Honeyfarm Architecture



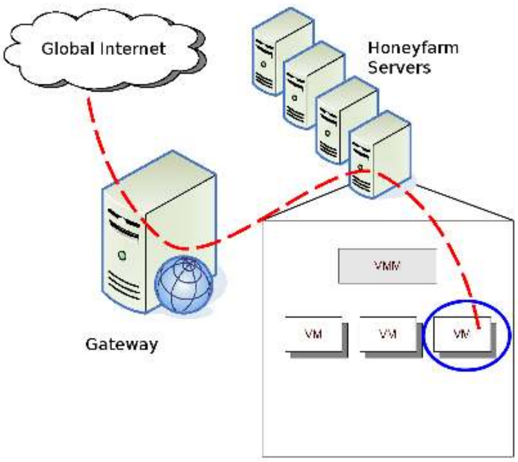
- Two components:
  - Gateway
  - VMM
- Basic operation:
  - Packet received by gateway
  - Dispatched to honeyfarm server

The Potemkin Honeyfarm Architecture



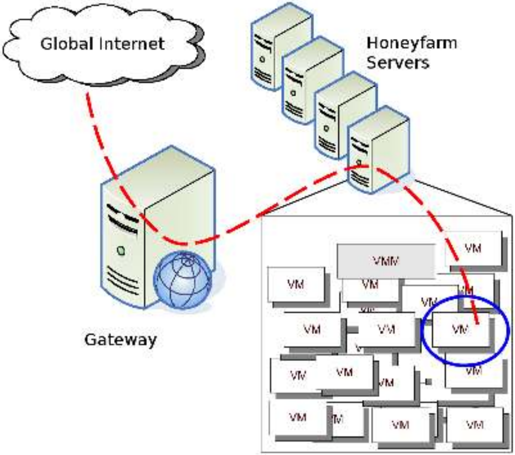
- Two components:
  - Gateway
  - VMM
- Basic operation:
  - Packet received by gateway
  - Dispatched to honeyfarm server
  - VM instantiated
    - Adopts IP address

Potemkin VMM Requirements



- VMs created on demand
  - VM creation must be fast enough to maintain illusion

Potemkin VMM Requirements



- VMs created on demand
  - VM creation must be fast enough to maintain illusion
- Many VMs created
  - Must be resource-efficient

Potemkin VMM Overview

- Modified version of Xen 3.0 (pre-release)
- **Flash cloning**
  - Fork copies from a reference honeypot VM
  - Reduces VM creation time—no need to boot
  - Applications all ready to run
- **Delta virtualization**
  - Copy-on-write sharing (between VMs)
  - Reduces per-VM state—only stores unique data
  - Further reduces VM creation time

Flash Cloning Performance

Time required to clone a 128 MB honeypot:

Control tools overhead	124 ms
Low-level clone	11 ms
Device setup	149 ms
Other management overhead	79 ms
Networking setup & overhead	158 ms
Total	521 ms

0.5 s already imperceptible to external observers unless looking for delay, but we can do even better

Flash Cloning Performance

Time required to clone a 128 MB honeypot:

Control tools overhead	124 ms
Low-level clone	11 ms
Device setup	149 ms
Other management overhead	79 ms
Networking setup & overhead	158 ms
Total	521 ms

0.5 s already imperceptible to external observers unless looking for delay, but we can do even better

Delta Virtualization Performance

- Deployed using 128 MB Linux honeypots
- Using servers with 2 GB RAM, have memory available to support $\approx$ 1000 VMs per physical host
- Currently tested with $\approx$ 100 VMs per host
  - Hits artificial resource limit in Xen, but this can be fixed

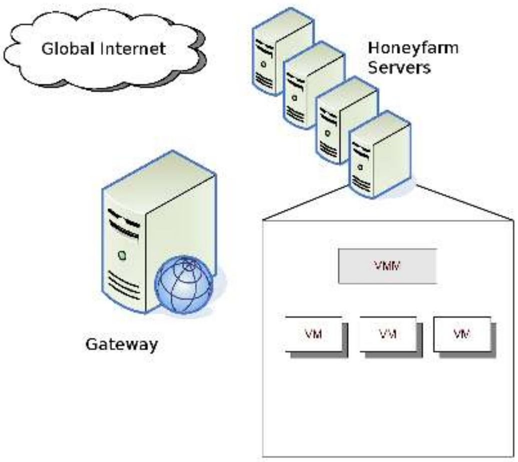
Containment Policies

- Must also care about traffic going out
- We deliberately run unpatched, insecure software in honeypots
- Containment: Should not permit attacks on third parties
- As with scalability, there is a tension between containment and fidelity
- Various containment policies we support:
  - Allow no traffic out
  - Allow traffic over established connections
  - Allow traffic back to original host
  - ...

Containment Implementation in Gateway

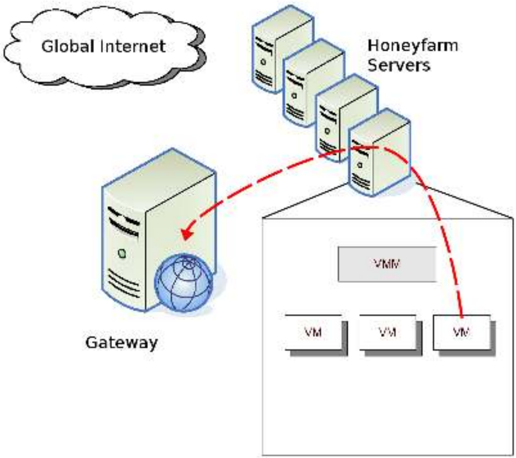
- Containment policies implemented in network gateway
- Tracks mappings between IP addresses, honeypots, and past connections
- Modular implementation in Click
- Gateway adds insignificant overhead ($\ll$ 1 ms)

Traffic Reflection



Example gateway policy: Redirect traffic back to honeyfarm

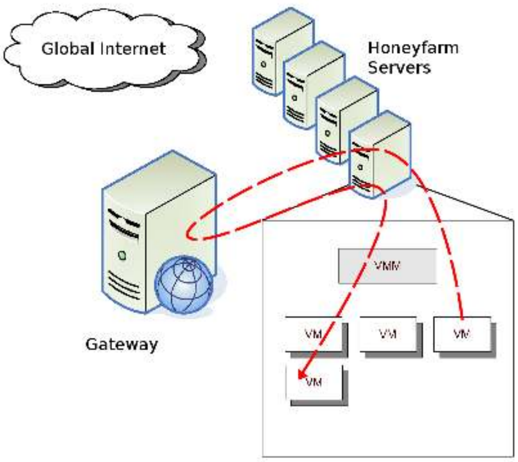
Traffic Reflection



Example gateway policy: Redirect traffic back to honeyfarm

- Packets sent out to third parties. . .

Traffic Reflection



Example gateway policy: Redirect traffic back to honeyfarm

- Packets sent out to third parties. . .
- . . . may be redirected back into honeyfarm

Reuses honeypot creation functionality

Challenges

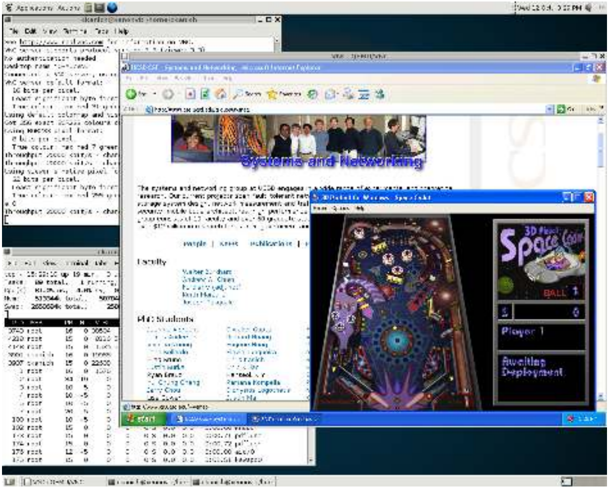
- Honeypot detection
  - If malware detects it is in a honeypot, may act differently
    - How easy it is to detect virtualization?
    - VMware detection code used in the wild
  - Open arms race between honeypot detection and camouflage
- Resource exhaustion
  - Under high load, difficult to maintain accurate illusion
    - Large-scale outbreak
    - Honeypot denial-of-service
  - Challenge is intelligently shedding load

Summary

- Can achieve both high fidelity and scalability
  - Sufficient to provide the *illusion* of scale
- Potemkin prototype: 65k addresses → 10 physical hosts
  - Largest high-fidelity honeypot that we are aware of
- Provides important tool for study of and defenses against malware

For more information:
http://www.ccied.org/

Windows on Xen



Camouflage

Malware may detect honeypot environment in various ways:

- Detect virtualization
  - Via incomplete x86 virtualization
  - Searching for characteristic hardware configurations
  - More complete virtualization can mitigate these leaks
- Detect monitoring tools
  - Network, VM-instrospection tools harder to detect
- Detect network environment
  - Containment requirement places some limits on camouflage effectiveness
  - Network security trends may be in our favor here

Honeypot Monitoring

Various means to monitor honeypots for interesting activity

- Network-level monitoring: Network intrusion detection systems, Earlybird-like detectors, ...
- Host-level intrusion detection
- Virtual machine introspection