A Comprehensive Framework For DDoS Resiliency in the Cloud

Mohammad Noureddine
PERFORM Group
UIUC
Introduction

• DDoS attacks are becoming very effective
• Easy to launch attacks
  • DDoS as a Service
    • $38 for one hour/month attack subscription service
  • “Subleasing” botnets containing millions of hosts
    • User machines, mobile devices, IoT devices, etc.
• Largest recorded DDoS assault at 650 Gbps
Motivation

• DDoS is far from being solved
Attack Capabilities

• Majority of botnets are still primitive
The Cloud

• Cloud adoption rates are increasing
  • Cloud providers making it easier
• Provides flexibility and elasticity
  • Efficient use of ever-increasing capacity
• Lucrative targets for DDoS attacks
• Does profit drive security?
  • Telemetry infrastructure unexploited
Traditional Approaches

• Often very intrusive
  • Capabilities require changes to core routers
• Require cooperation between ISPs
  • Unlikely to happen without enforcement
• Require expensive classification of hosts
  • IP traceback is expensive and easily fooled
• Very few make use of the cloud
  • The ones that are often proprietary
Requirements

• We need to detect DDoS attacks
  • Distinguish between high load and DDoS
• We need to effectively respond to attacks
  • Trigger protection mechanism
  • Maintain operation
• We need to scale up horizontally if possible
  • High availability, Content Delivery Networks (CDN)
Approach

- DDoS Protection using Client Puzzles
- Horizontal Scaling and Provisioning
- Monitoring and Detection
- Private Cloud
- Data Collection Bus (Ceilometer, Gnocchi, etc.)
System Model
Data Collection

• Openstack, EC2, Azure provide strong telemetry infrastructure
  • Make use of it for security monitoring

• Openstack ceilometer
  • Complemented with Gnocchi for current and future versions
  • Time series database as a service
Ceilometer
Change Point Detection

• Statistical detection of abrupt changes in normal behavior

• Traditionally, observe
  • $\Delta_t = \{SYNReq_t - SYNRep_t\}, t = 0,1,2, ...$
  • Focuses only on network information
  • Myopic to performance of server instances
In Our Approach

• Ceilometer data collection
  • Disk usage, CPU utilization, Memory utilization, Network utilization
  • [Apache logs]

• Define new sample vector
  • \( \Delta_t = \{\text{disk}_t, \text{cpu}_t, \text{mem}_t, \text{req}_t, \text{rep}_t\}^T \)
  • Provides richer definition of normal behavior
  • Covers larger space of attacks
Client Puzzles

• Efficient stateless proof of work
  • Receive service only after appropriate “payment”
• No puzzle solution required under regular load
• Non intrusive, no infrastructure change required
• Puzzle mechanism initiated upon attack detection
  • Use historical data to select puzzle complexity
  • Mechanism/Incentive design problem
Puzzle Protocol

Under Regular Load

Client

Overlay Server

App Server

Puzzle?

No Puzzle

SYN

SYN ACK

ACK

Under Attack

Client

Overlay Server

App Server

Puzzle?

Puzzle P, T

Solution

Verify solution

Accepted

SYN
Puzzle Construction

• Each puzzle is composed of \( k \) sub-puzzles
  • Each sub-puzzle of length \( m \) bits
• Solve by brute force
Effectiveness

• Stateless mechanism to filter clients
• Serious rate limiting on the client side
• More complexity comes at lower cost for server
Limitations

• Can be DDoS’ed
  • More efficient ways to generate and check puzzles
  • White/Black listing clients based on history
  • Still better than SYN flood attacks

• How to pick $k$ and $m$
  • Currently fixed for all users
  • Increased for everyone when attack intensifies
  • Mechanism/Incentive design problem
Choosing $k$ and $m$

• Provide service for those who actually want it
• Motivation from network pricing
• $x_i$: Request rate for client $i$
• Define utility for each client

$$U_i(x_i, x_{-i}) = \log(1 + x_i) - \text{PuzzleCost}_i - \text{ServiceDelay}$$
Time to Solve a Puzzle

• Expected number of hashes to solve a puzzle is $k \times 2^{m-1}$
Service Delay

- Model application service as an M/M/c queue
  - $c$ is the number of server replicas
- Rate of arrivals $\lambda = \frac{\sum_i x_i}{N}$, $N$ being the number of flows
- Service rate $\mu$ is estimate from ab and other stress testing tools
- Analytical solution for expected wait time $W$
  - Function of $\rho = \frac{\lambda}{c\mu}$
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 34.210.20.7 (be patient)

Server Software:       nginx/1.12.0
Server Hostname:       34.210.20.7
Server Port:           80
Document Path:         /
Document Length:       52757 bytes
Concurrency Level:     100
Time taken for tests:  44.630 seconds
Complete requests:     500
Failed requests:       0
Total transferred:     26532500 bytes
HTML transferred:      26378500 bytes
Requests per second:   11.20 [#/sec] (mean)
Time per request:      89.260 [ms] (mean)
Time per request:      89.260 [ms] (mean, across all concurrent requests)
Transfer rate:         580.56 [Kbytes/sec] received
Stackelberg Game (Mechanism Design)

- \( U_i(x_i, x_{-i}) = \log (1 + x_i) - k_i 2^{m_i-1} x_i - W \)
- \( W \) is a function of \( x_i \) and \( x_{-i} \)
- Solve for equilibrium rates \( x_i^* \)
  - \( U_i(x_i^*, x_{-i}^*) \geq U_i(x_i, x_{-i}^*) \quad \forall x_i, \forall i \)
- The cloud’s design problem is to find \( k_i^* \) and \( m_i^* \)
  - \( \arg\max_{p_i,k_i \in \mathbb{N}} \sum_i \log(1 + x_i^*) - \sum_i (m_i \times k_i) x_i^* \)
Horizontal Scaling Ideas

• Adding more replicas naively
• Give networking and compute budgets
  • Solve optimization problem
• Think in terms of the queueing model
  • Scale up to keep $\rho < 1$
OpenStack Deployment
Infrastructure and Applications

• Deployment composed of 4 servers and 6 commodity machines
• Running Wordpress server replicas with MySQL backend
• NginX load balancer
Steps for Evaluation

• Evaluate change point detection mechanism
• Simulations to evaluate puzzle difficulty selection
• Comprehensive evaluation
  • Simulate normal and heavy load using stress testing tools
  • Simulate attack using botnet simulator
Major Challenge

• We have tried multiple deployment tools
  • Each claim to be the “Chuck Norris” of deployment
• Lack of documentation for troubleshooting deployment errors
• Biggest success with Ubuntu Autopilot so far
Conclusion

• Comprehensive design for resilient applications in a private cloud
• Uses telemetry infrastructure for monitoring
• Uses client puzzles for DDoS protection under attack
• Provides horizontal scaling to guarantee performance