Protecting against DoS Attacks by Analysing the Application Stress Space

Cornel Barna\textsuperscript{1} \quad Chris Bachalo\textsuperscript{2}
Marin Litoiu\textsuperscript{1} \quad Hamoun Ghanbari\textsuperscript{1}
Mark Shtern\textsuperscript{1}

\textsuperscript{1}Computer Science and Engineering
York University
\textsuperscript{2}Juniper Networks

February 1, 2015
Denial of Service (DoS) Attacks

Mitigating DoS

Adaptive DoS Mitigation

Stress Space
Motivation

• in the last years, there is a rise of DoS attacks on internet (RIAA, MPAA, Operation Payback, attack on North Korea, etc.);
• attackers developed easy-to-use toolkits: Low Orbit Ion Cannon (LOIC);
• motivation behind the attacks:
  • financial;
  • political;
  • ideological;
  • it’s ”cool” to do it.
DoS Attacks

Network Protocols were designed to be functional, not secure.

- flood the system with requests;
- designed to make the system unresponsive;
- legitimate users cannot receive service;
- methods to fix the problem:
  - add more and better hardware;
  - use a firewall to filter malicious traffic.
- it’s a challenge to defend against them;
- the current state of the art methods do not fully mitigate the DoS attacks.
Types of Attacks

- Regular Denial of Service (DoS)
  - attacker uses one computer to launch the attack
- Distributed Denial of Service (DDoS)
  - the attacker uses multiple computers
- Reflected Denial of Service (RDoS)
  - the attacker spoofs the source address
  - uses intermediaries (reflectors) to amplify the attack
- Distributed Reflected Denial of Service (DRDoS)
  - same as RDoS, but the attacker uses multiple computers
Application Layer DoS

- targets a specific service (HTTP, HTTPS, SMTP)
- less traffic required to saturate the service
- malicious HTTP headers are not distinguishable from legitimate ones
- HTTP POST DDOS attack
  - send the "Content-Length" header with a very large value
  - send the body of the request very slow (1 byte / 110 seconds)
- slowloris
  - keep open as many connections possible for as long possible
  - send partial requests
  - saturates the connection pool
Adaptive DoS Mitigation

If the traffic exceeds our ability to manage it, there is a DoS attack.

- we created a framework to mitigate (D)DoS attacks;
  - dynamic firewall;
  - decision engine;
  - analyzer.
- statistical anomaly detection to create the filtering rules;
- a performance model fine-tunes the filtering rules;
- the protected system is continuously monitored;
- traffic filtering rules are dynamically added/removed;
The architecture

The framework.

Dynamic Firewall
Analyzer
Reverse
Proxy
Decision
Engine
Web
Application
Regular
traffic
Filtered traffic
Performance Monitoring Data
(feedback)
Incoming
traffic
Outgoing
traffic
Challenge
Response
Analyzer
Regular traffic
Regular traffic
The architecture

Not a filtered request: forward it to the web application
The architecture

Filtered request: issue a challenge (CAPTCHA)
The architecture

Challenge solved: forward the request to the web application

Performance Monitoring Data
(feedback)

Incoming traffic

Reverse Proxy

Decision Engine

Dynamic Firewall

Filtered traffic

Analyzer

Regular traffic

Web Application

Regular traffic
The architecture

The framework.

Performance Monitoring Data
(feedback)

Incoming traffic

Outgoing traffic

Reverse Proxy

Decision Engine

Dynamic Firewall

Analyzer

Web Application

Regular traffic

Filtered traffic

Challenge Response

Regular traffic
The decision engine

- **protection loop**: monitors the system under protection;
- **decision loop**: create/remove filtering rules.

![Diagram of the decision engine](image)

- Decision Engine loop
  - Decision Controller
  - Performance Model
  - Kalman Filter

- Protection loop
  - Request Filter Rules
  - System under protection
  - Monitor and Estimator

- Performance goals
  - Incoming requests
Results

Reactive Systems

Emulated DoS attack on two scenarios using LOIC.
The Performance Stress Space

For proactive systems, we need better prediction!
Consider a system with two hardware resources, two software resources and two classes of service.

Hardware Constraints
• Linear equations:

\[ U_K^h = \sum_{\forall C \in C} \frac{D_{K,C}}{D_{K,r,C}} \times U_{K_r,C}, \quad \forall K \in K^h \]

• Feasible space \( \rightarrow OABC \)

Software and Hardware Constraints
• Non-linear equations:

\[ U_K^s = \sum_{\forall C \in C} R_{K,C} \times U_{K_r,C}, \quad \forall K \in K^s \]

• Feasible space \( \rightarrow OABDE \)
The Performance Stress Space

For proactive systems, we need better prediction!

Consider a system with two *hardware* resources, two *software* resources and two *classes of service*.

\[
\begin{align*}
U_h^1 &= a \\
U_h^2 &= b \\
U_s^1 &= c \\
U_s^2 &= d
\end{align*}
\]

**Hardware Constraints**

- **Linear equations:**
  \[
  U_h^K = \sum_{\forall C \in C} \frac{D_{K,C}}{D_{K,C}} \times U_{K,C}, \quad \forall K \in \mathcal{K}^h
  \]
  - Feasible space $\rightarrow OABC$

**Hardware and Software Constraints**

- **Non-linear equations:**
  \[
  U_s^K = \sum_{\forall C \in C} \frac{R_{K,C}}{D_{K,C}} \times U_{K,C}, \quad \forall K \in \mathcal{K}^s
  \]
  - Feasible space $\rightarrow OABDE$
The Performance Stress Space

For proactive systems, we need better prediction!
Consider a system with two hardware resources, two software resources and two classes of service.

- System becomes saturated when close to EDBA lines $\rightarrow$ likely to crash/deny service
- We should predict where the workload is heading and how close are we to EDBA lines
- How to calculate the shortest distance to the boundary? Perpendicular to the border lines?
Analytically find the scenarios that should be filtered.

- **Web Application with 6 scenarios**
- **first row**—user ratio when the attack started
- **2+ rows**—stress vectors (46 total, only showing a few)
- **last column**—euclidian distance between the stress vectors and user ratio at the moment of attack
Conclusion

• use a predictor for workload and identify system overload before it happens

• challenges:
  • prediction is hard
  • how close is a given workload to the edges of feasible space?
  • how do we validate the method?

• benefits for very large scale systems: reduce the search space to find
  • when the system will saturate/crash
  • what scenarios to filter
  • what types resources to add
  • dynamic decision making (not development time)
1. How close is a given workload to the edges of feasible space?
   This need to be found automatically, at runtime.
2. What is the interpretation for the distance to the edges?
3. Could the feasible space be useful to solve other problems?