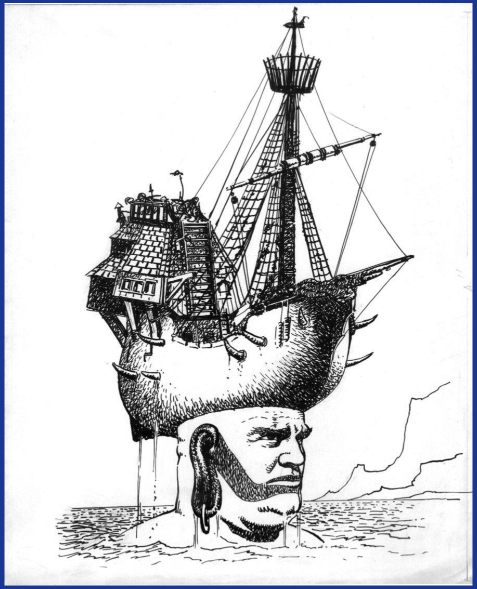


HTTP Time Bandit

About US

- We fix stuff & accidentally break things
- Interested in time travel
- Love to tri (swim/bike/run)

- Tigran Gevorgyan
  Engineering Manager
  Qualys

- Vaagn Toukharian
  Principal Engineer
  Qualys

What?

Yet another application layer DOS attack that strives for resource starvation through asymmetric resource utilization.

- Method
- Tool
- Stats
- Defence
- Usage possibilities

Why?

DOS Classification

- Crash, non-resource attack, degrading IT capabilities
- Resource consumption attack
  - Network resource exhaustion
  - Infrastructure device resource exhaustion
  - Target resource exhaustion
    - OS or network layer (e.g. SYN flood)
    - Application layer
    - Business logic “layer”

From DoS Attack Taxonomy [1]

Classic Application Layer DOS/DDOS

DDOSing blindly

- GET index.html
- 10000 x of the GET
- No feedback
- Near-Symmetrical load

Smarter Bots

- SlowLoris
- SlowHttptest
- SlowRead
- PKI abuse
- SQL wildcards
- WebSockets connection hogging

Some Exotic L7 DOS

- Using ‘%’ in the request may cause the DB to fetch every row in the DB (use genetic algorithm to figure out a payload that makes the server to work the hardest?)
- Business logic - “above L7 attacks”
  - Too many items in the cart
  - Too much logging caused by invalid inputs
  - Too many temporary objects in memory (attachments for webmail)

Get Flooding With Spice

- Is not exotic
- It ain’t Slow*
- Not going for exhaustion of 20k HTTP connections
- Resource consumption is asymmetrical by nature, just trying to get bigger divide
- Just a Get flood, with some analysis done before flooding takes place

The Proposed Method

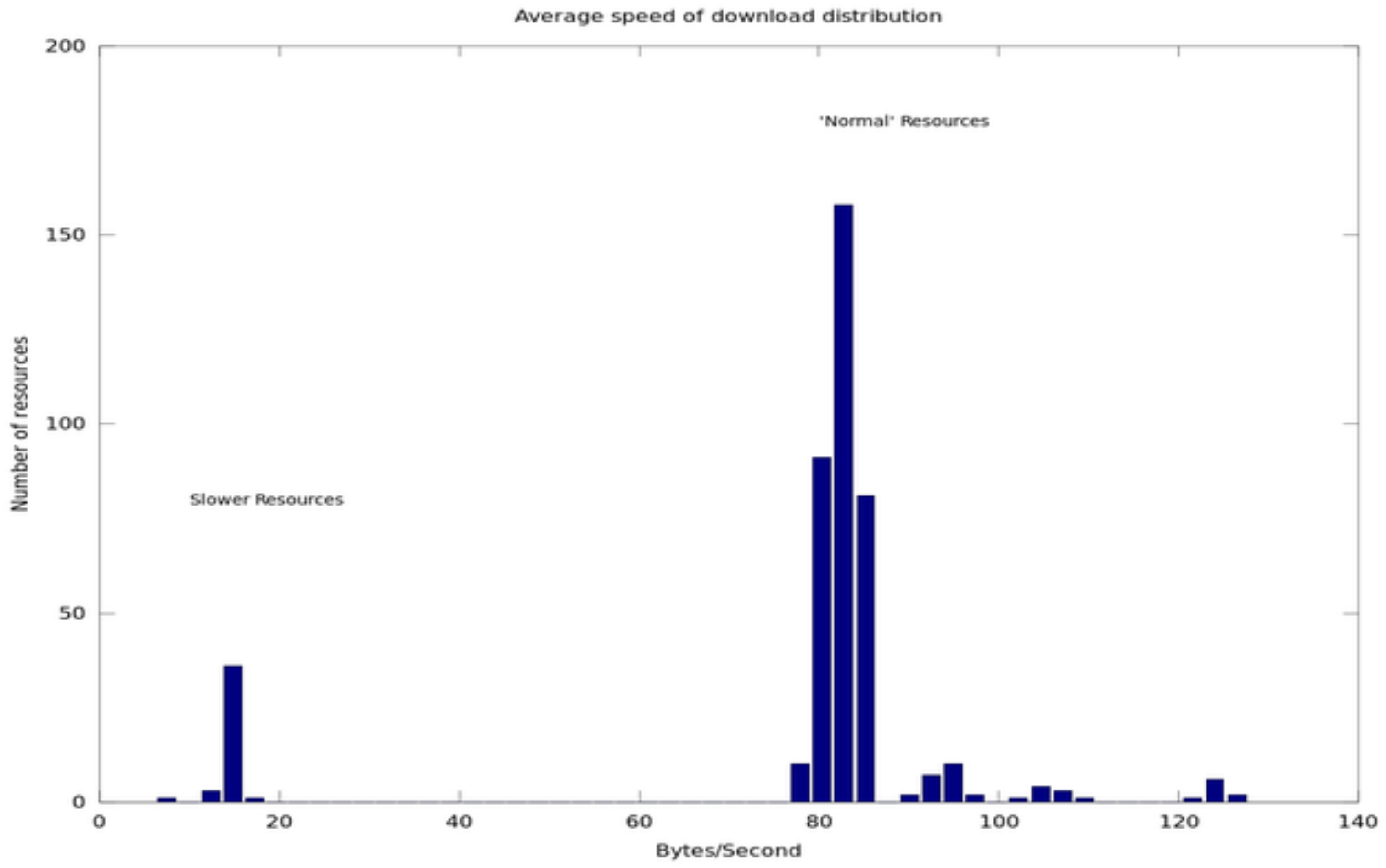
Method of detection of the critical resource

- Spider over the web site and collect transfer times for each resource
- Calculate the average speed and distribution of transfers
- Identify the resources that have slower average transfer times

Transfer time's correlation with load

- CPU intensive resources take more time to response
- Resource size is not significant

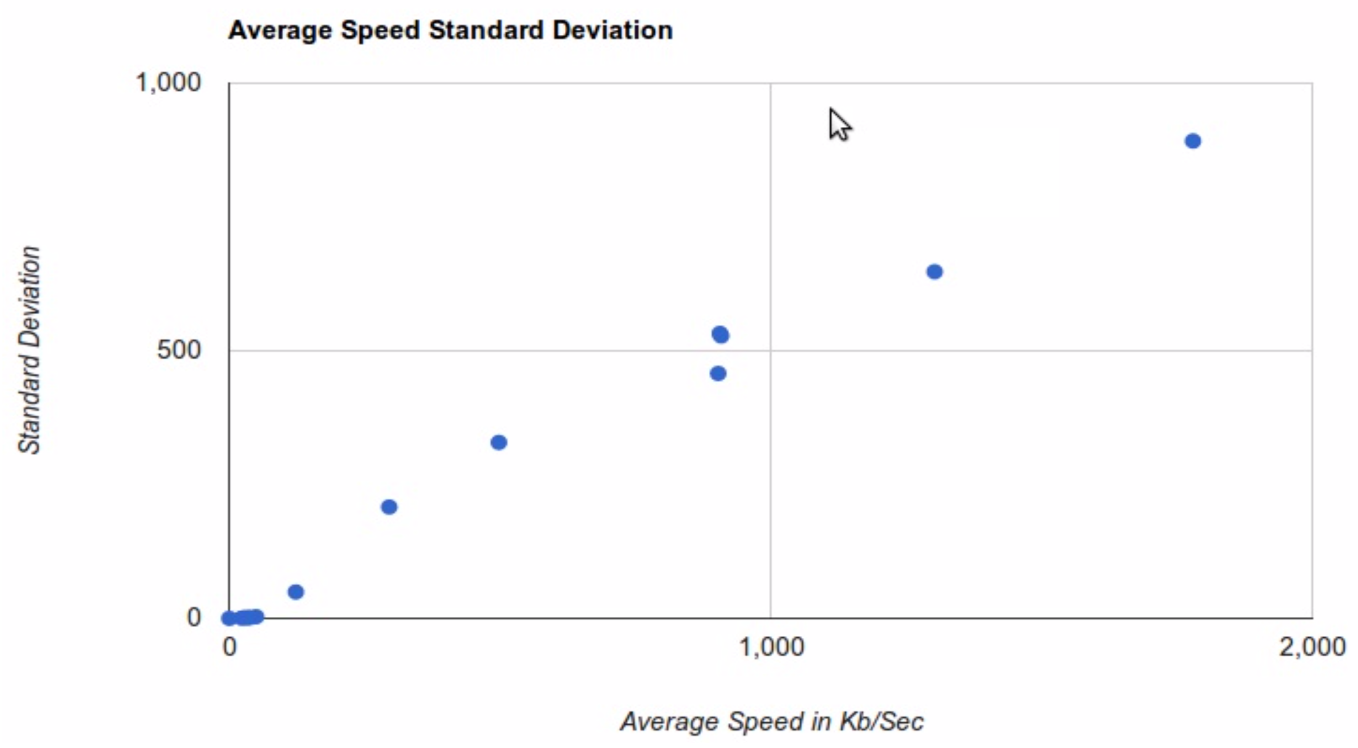
Lies, Dirty Lies and Statistics



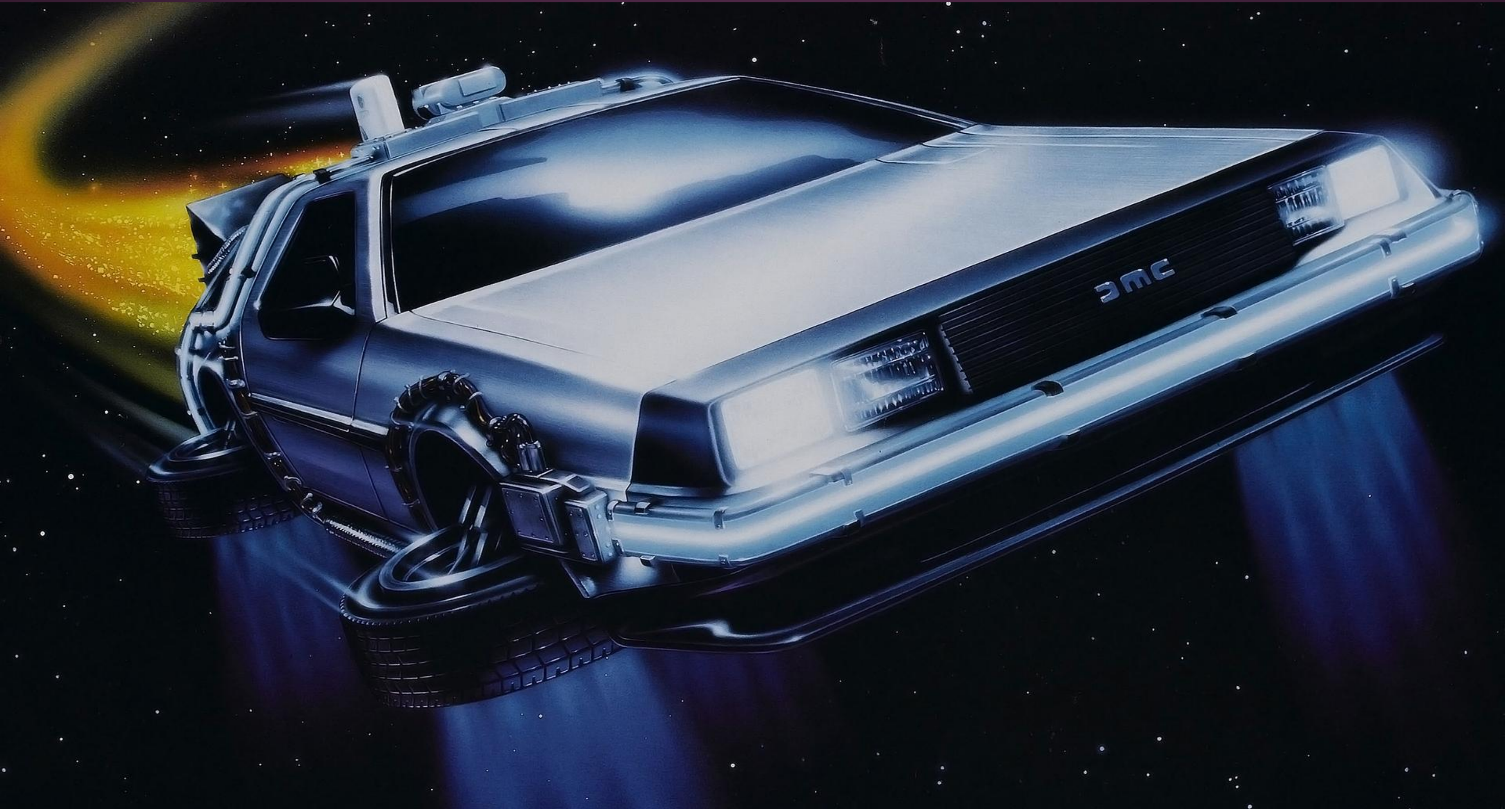
Using Statistics to Normalize the Data

- Mean as the measure of central tendency
  - Calculate the mean of all resource download speeds
  - Calculate the means of each resource download speeds
  - Select the resources whose download speeds are less (slower) than the mean of all download speeds
- Selecting resources with lower mean
- Discarding resources with large variance

Speed Distribution



Demo



Attack Like Stage of Testing

Measurement of service degradation while doing a hard test for narrowing down the choice of links

```
$
./crwlr --url http://10.12.0.3/Concrete5/Concrete5-6.0/ --verbose 1
--depth 3 --count 10 --xml concrete.xml
$
crwlr --count 100 --in concrete.xml&
crwlr --count 100 --in concrete.xml&
crwlr --count 100 --in concrete.xml&
...
http://10.12.0.3/Concrete5/Concrete5-6.0/index.php/blog/
original mean/sdev: 23.039/3.531 stress mean/sdev: 28.058/6.272
original mean/sdev: 23.039/32.531 stress mean/sdev: 27.568/6.039
original mean/sdev: 23.039/3.531 stress mean/sdev: 27.389/5.927
```

	Original mean/sDev	Stressed mean/sDev
Banit_0	23.039/3.531	28.058/6.272
Banit_1	23.039/3.531	27.568/6.039
Banit_2	23.039/3.531	27.389/5.927

Similar Tools

DoSHTTP

- No statistical analysis

JMeter

- Performance measurement
- Extendible

Tsung

- Erlang based many(upto 1M) user simulation

Pylot

- Very close, some statistical analysis
- Not a crawler
- No parallel testing, load measurement

The Art of (D)DOS Defence

"Hard it is, but try we can for DOS at least"

- Load Balancing
- Identify/Fix resource hogs
  - Use our tool for this
- Apache config suggestions
- Other Apache modules
- Advanced mod_security protection

"Fail those will if used is force"

Load Balancers

Stopping Get Floods using:

- Rate-limiters
- Unusual traffic filters
- Source checks

Possible issues

- No real sense of load on the targets
- Internal IP leakage
- If protections are sensed the attacks could be crafted to perform just under the threshold
- If the attack detection is based on similarity of requests mutation could fool it

HAProxy

- Divides the load between the back-end servers
- Different policies for static and and dynamic resources
- Can set some thresholds[2]

```
...
  tcp-request content reject if { src_get_gpc0 gt 0 }
  http-request deny if { src_get_gpc0 gt 0 }
...
  use_backend bk_web_static if { path_end .jpg .png .gif .css .js }
...
  acl abuse src_http_req_rate(ft_web) ge 10
  acl flag_abuser src_inc_gpc0(ft_web)
  http-request deny if abuse flag_abuser
```

Commercial Protection Services

- Few players using limiters for:
  - Resource rate
  - Connection
  - Originating IP
- Some Slow* defences
- mod_security like measures against SQLi and XSS
- **Good cloud based solutions cost >$150/m**
- **" would not use the full-blown solution because don't want to degrade the user experience"**
- **Those could fail as described in** Universal-DDOS-Mitigation_Bypass[3]

Using the Tool for Good

- Identify/Fix resource hogs
  - Use our tool for this
  - Manual(intelligent) tweaking of the request to get possible higher stress
  - Confirm the high resource usage by stressing the “finds” with parallel requests and measuring the degradation
- In ideal world the tool would generate conf files for DOS protection modules

Playing with Apache Configs

Baseline, no protection

- 1 client running 10x parallel requests of the most expensive resource
- 3% CPU on the client machine
- Server: i7, 4 core, 8 gb
- **98% CPU** utilization on the server

Standard config measures ?

Nothing that would really help Get Floods, but there are some setting that would help with Slow* attacks[4]

mod_security

- Simple mod_security protection [5]
  - Requests per IP limit, blocking the violators
  - Effective but too strict
  - Blocks the offensive IP right away.
  - CPU usage goes down to 0%

```
SecRule ip:requests "@eq 50" "phase:1,pass,nolog,setvar:ip.block=1,
expirevar:ip.block=5,setvar:ip.blocks=+1,expirevar:ip.blocks=3600"
```

- Advanced mod_security protection
  - Identification of regular flows
  - Out of ordinary flow filtering
  - State coherence checks
  - Still only a theory

mod_limitipconn

Limits the number of simultaneous downloads permitted from a single IP address [6]

“This module is not designed to prevent denial-of-service attacks.” -README

```
MaxConnPerIP 3
```

Cons:

- A bit crude
- Need to identify the (arbitrary) limit

Pros:

- Limites CPU to 38% CPU

mod_qos

Implements control mechanisms to provide different priority to requests and control server access based on available resources [7]

```
QS_SrvMaxConnPerIP 50
```

Works

- Limites CPU to 38% CPU
- “QS_SrvMinDataRate” will help to fight slow* attacks

mod_bwshare

Accepts or rejects HTTP requests from each client IP address, based on thresholds set by past traffic from a particular IP address[8]

```
BW_tx1debt_max          30
BW_tx1cred_rate         0.095
BW_tx2debt_max          3000000
BW_tx2cred_rate         2500
```

- Tricky with setting the limits
- Sophisticated way of setting a limit

mod_throttle

Is intended to reduce the load on your server, and the data transfer generated by popular virtual hosts, directories, locations, or users.

Discontinued...

The rules:

N/A

The effect:

N/A

mod_evasive

Provide evasive action in the event of an HTTP DOS /DDoS or brute force attack. [9]

```
DOSPageCount        10
DOSSiteCount        100
DOSBlockingPeriod   60
```

- Once detect all the connections from an attacker are dropped
- This really works.
- Our favorite for now

Conflicts with Slow* Attack Protection

- Slow* attack mitigation is an addition
- mod_evasive could not protect from these
- There is no conflict (good news)

We suggest using these apache directives for Slow* attack mitigation:

```
RequestReadTimeout
KeepAliveTimeout
KeepAlive
MaxRequestWorkers
```

mod_httpbl

Not exactly for protecting the server from a DOS attack but is cool as it is leveraging the "Project Honey pot"

- HoneyPot collects a list of offenders
- List of offenders gets blacklisted

httpbl.sourceforge.net

Usage

of HTTP Time Bandit

The Good

Find potential CPU/DB hogs in my web apps

The Bad

Automated iterative analyzer attacker

The Ugly

Probably should not be spelled out:)

Imagine “The Bad” x 1000

Back to the Future

- Understanding Load Balancers
- SQL wildcard usage
- State Reset cost analysis
- Automated Attacker, service degradation measurement

Thank You

tgevorgyan@qualys.com
@tukharian, vtoukharian@qualys.com

https://github.com/Qualys/timeBandit

References

1. http://blogs.gartner.com/anton-chuvakin/2012/06/06/quick-dos-attack-taxonomy
2. http://blog.exceliance.fr/2012/02/27/use-a-load-balancer-as-a-first-row-of-defense-against-ddos
3. https://media.blackhat.com/us-13/US-13-Lee-Universal-DDoS-Mitigation-Bypass-WP.pdf
4. http://httpd.apache.org/docs/current/misc/security_tips.html
5. http://blog.cherouvim.com/simple-dos-protection-with-mod_security
6. http://dominia.org/djao/limitipconn.html
7. http://opensource.adnovum.ch/mod_qos
8. http://www.topology.org/src/bwshare/README.html
9. http://www.tecmint.com/protect-apache-using-mod_security-and-mod_evasive-on-rhel-centos-fedora/