Take advantage of randomness

Frank Tse

Nexusguard

Agenda

1. What is random
2. Some applications of random
3. Detecting anomalies from randomness
4. Mitigating ‘random’ attacks
5. Visualizing randomness

About::me

From Hong Kong
Researcher in DDoS

I like RFC

General IT security vs DDoS

IT Security

Identify them correctly
Take actions accordingly

Block the known bad
Verify the known good
Track the uncertain
Challenge the suspicious

DDoS:
Good Human
 > Adult, Kid, Infant
Bad Human
 > Smart, not-so-smart
Good Bot (inhuman)
Bad bot (inhuman)

/dev/random

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Entropy: initial seeds for random number generation

Initializing seed for random during boot up (HW)

```
[    0.000000] e820: [mem 0x40000000-0xffffffff] available for PCI devices
[    0.000000] Booting paravirtualized kernel on bare hardware
[    0.000000] setup_percpu: NR_CPUS:512 nr_cpumask_bits:512 nr_cpu_ids:64 nr_no
de_ids:1
[    0.000000] PERCPU: Embedded 28 pages/cpu @ffff88003e200000 s85888 r8192 d206
08 u131072
```

kern.random.sys.seeded	non-blocking while reading
kern.random.sys.harvest.ethernet	LAN traffic
kern.random.sys.harvest.point_to_point	P2P interface
kern.random.sys.harvest.interrupt	HW interrupt (Mouse, keyboard)
kern.random.sys.harvest.swi	SW interrupt (exceptions)

Entropy: initial seeds for random number generation

If I'm running on VM

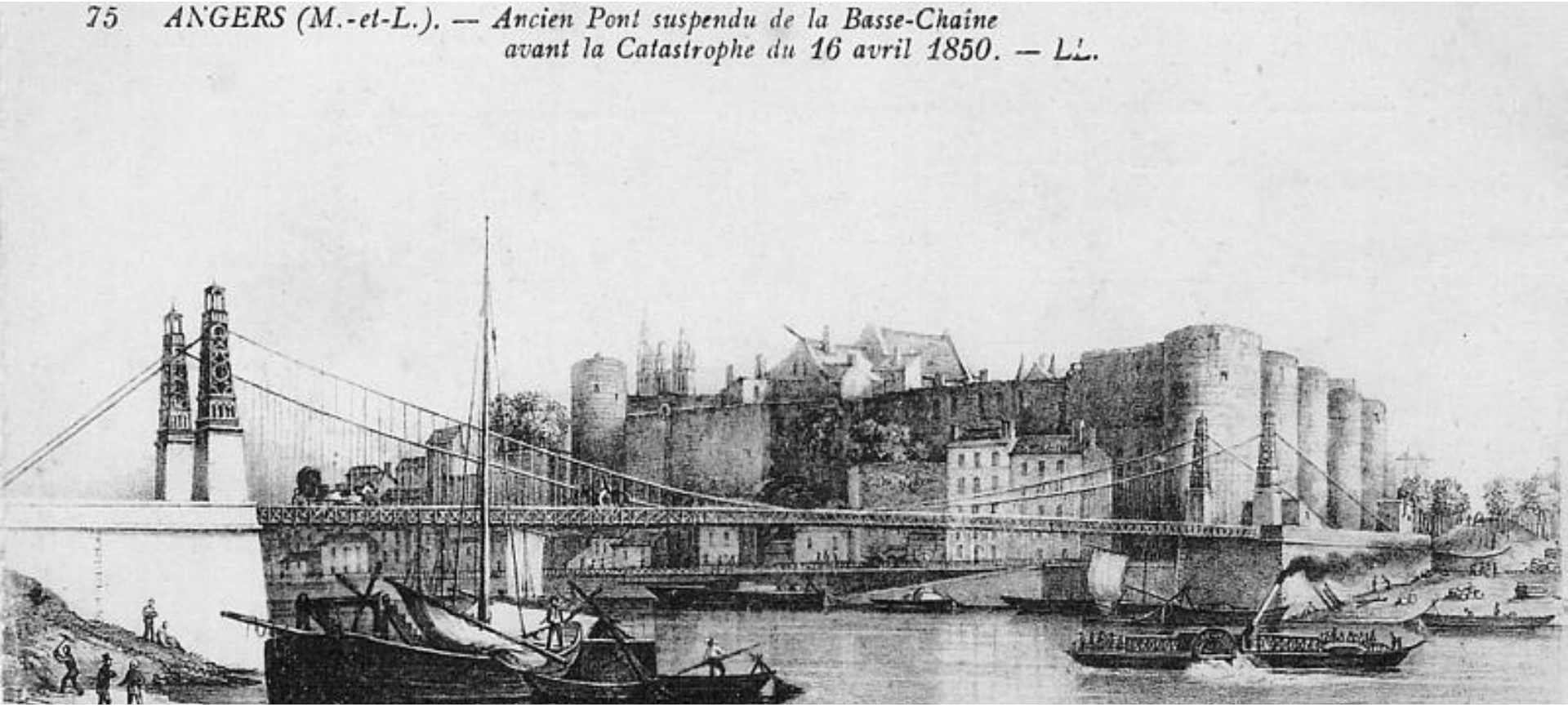
[0.000000] Booting paravirtualized kernel on KVM

virtio-rng: a driver for feeding entropy between VM guest and host

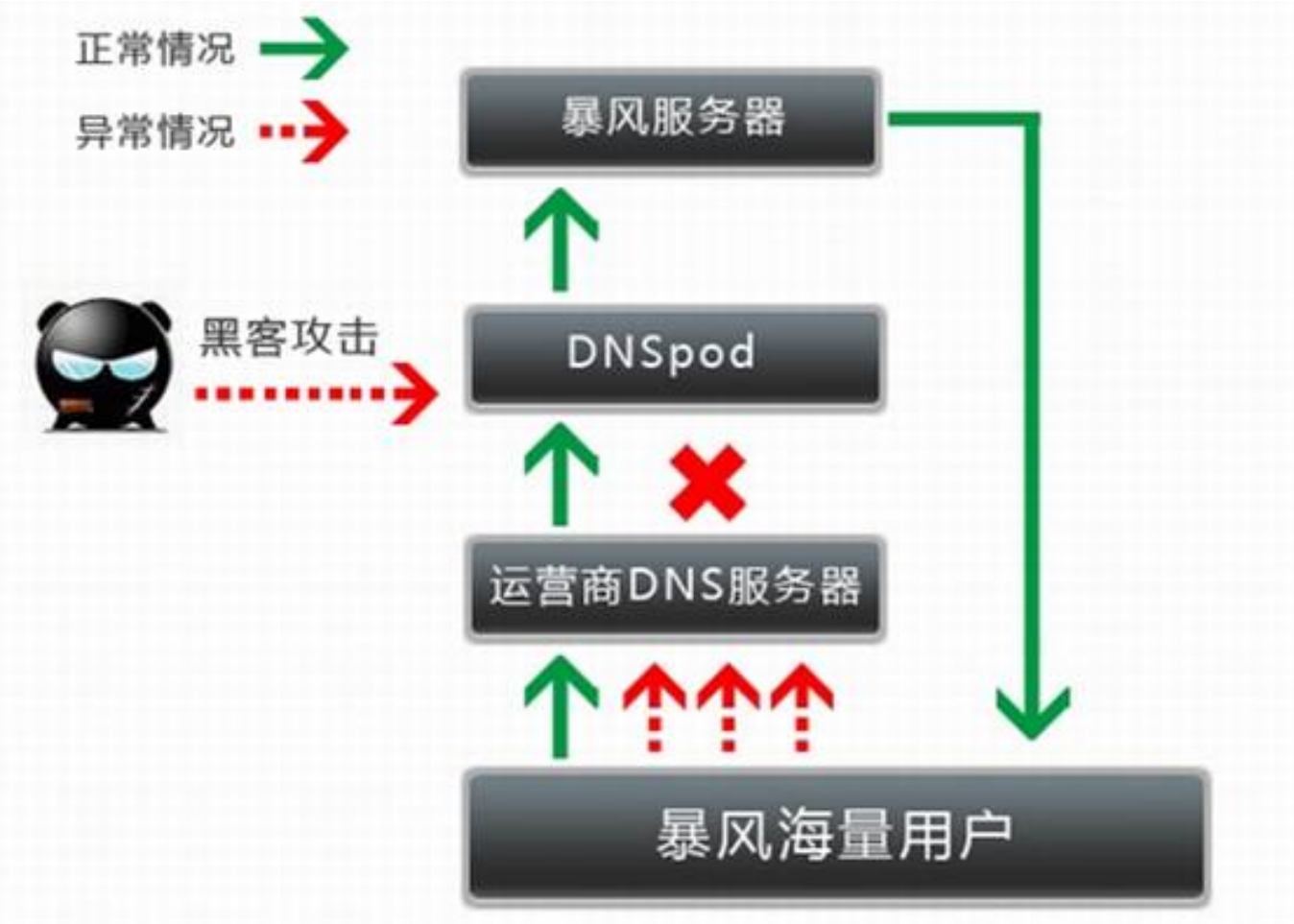
Problem: I don't trust virto-rng
Solution: entropy from remote server

entropy.ubuntu.com

Angers Bridge, collapsed on Apr-16, 1850, due to soldiers marching across it.
aka. “Stuck in synchronization”



2009 MAY 19, Storm Codec [Baofeng] (暴风影音) brings down DNSpod.
Due to lack of random back-off and sleep mechanism



Routing protocol randomized hello timers to avoid ‘stuck in synchronization”

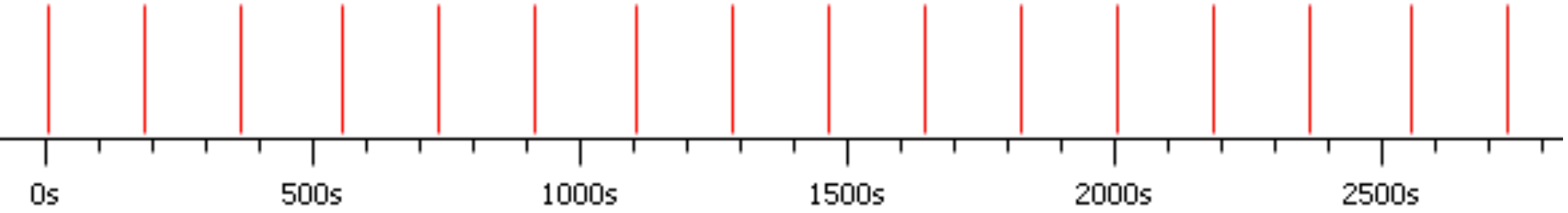
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.1	224.0.0.10	EIGRP	60	Hello
2	4.688281	10.0.0.1	224.0.0.10	EIGRP	60	Hello
3	4.420263	10.0.0.1	224.0.0.10	EIGRP	60	Hello
4	4.356262	10.0.0.1	224.0.0.10	EIGRP	60	Hello
5	4.788305	10.0.0.1	224.0.0.10	EIGRP	60	Hello
6	4.664273	10.0.0.1	224.0.0.10	EIGRP	60	Hello
7	4.266695	10.0.0.1	224.0.0.10	EIGRP	60	Hello
8	4.393835	10.0.0.1	224.0.0.10	EIGRP	60	Hello

RFC4271 – Border Gateway Protocol v4

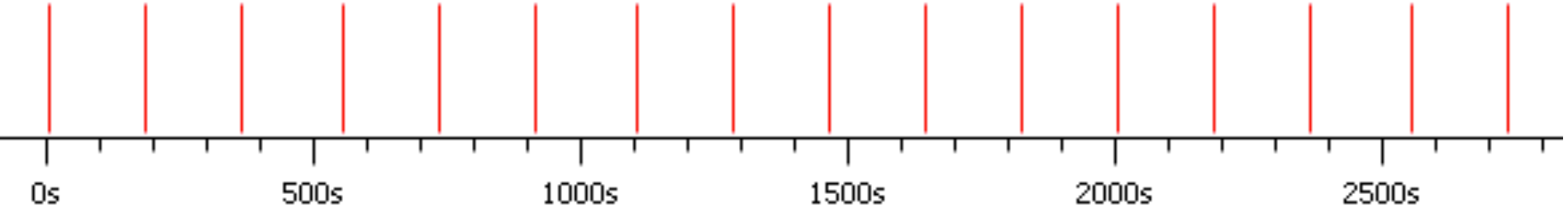
To minimize the likelihood that the distribution of BGP messages by a given BGP speaker will contain peaks, jitter SHOULD be applied to the timers associated with MinASOriginationIntervalTimer, KeepaliveTimer, MinRouteAdvertisementIntervalTimer, and ConnectRetryTimer. A given BGP speaker MAY apply the same jitter to each of these quantities, regardless of the destinations to which the updates are being sent; that is, jitter need not be configured on a per-peer basis.

The suggested default amount of jitter SHALL be determined by multiplying the base value of the appropriate timer by a **random factor**, which is **uniformly distributed** in the range from 0.75 to 1.0. A new random value SHOULD be picked each time the timer is set. The range of the jitter's random value MAY be configurable.

C&C Communication



Software update check



Generating Randomart from SSH host key fingerprint

```
$ ssh root@myhost -o VisualHostKey=yes
Host key fingerprint is ce:7f:ee:de:c0:87:bb:63:8b:ae:d3:6d:08:4d:d4:8f

+--[ RSA 2048]----+
|            .    |
|           . .   |
|          .   o  |
|           . E . |
|         So      |
|        o. .. .  |
|         oo o+ . |
|         ..o.*=  |
|         .++BB+. |
+-----------------+
```

Without randomness

CVE-2008-1447: DNS Cache Poisoning Issue

allow remote attackers to spoof DNS traffic via a birthday attack that uses in-bailiwick referrals to conduct cache poisoning against recursive resolvers, related to insufficient **randomness** of **DNS transaction ID**s and source ports, aka "DNS Insufficient Socket Entropy Vulnerability" or "the Kaminsky bug."

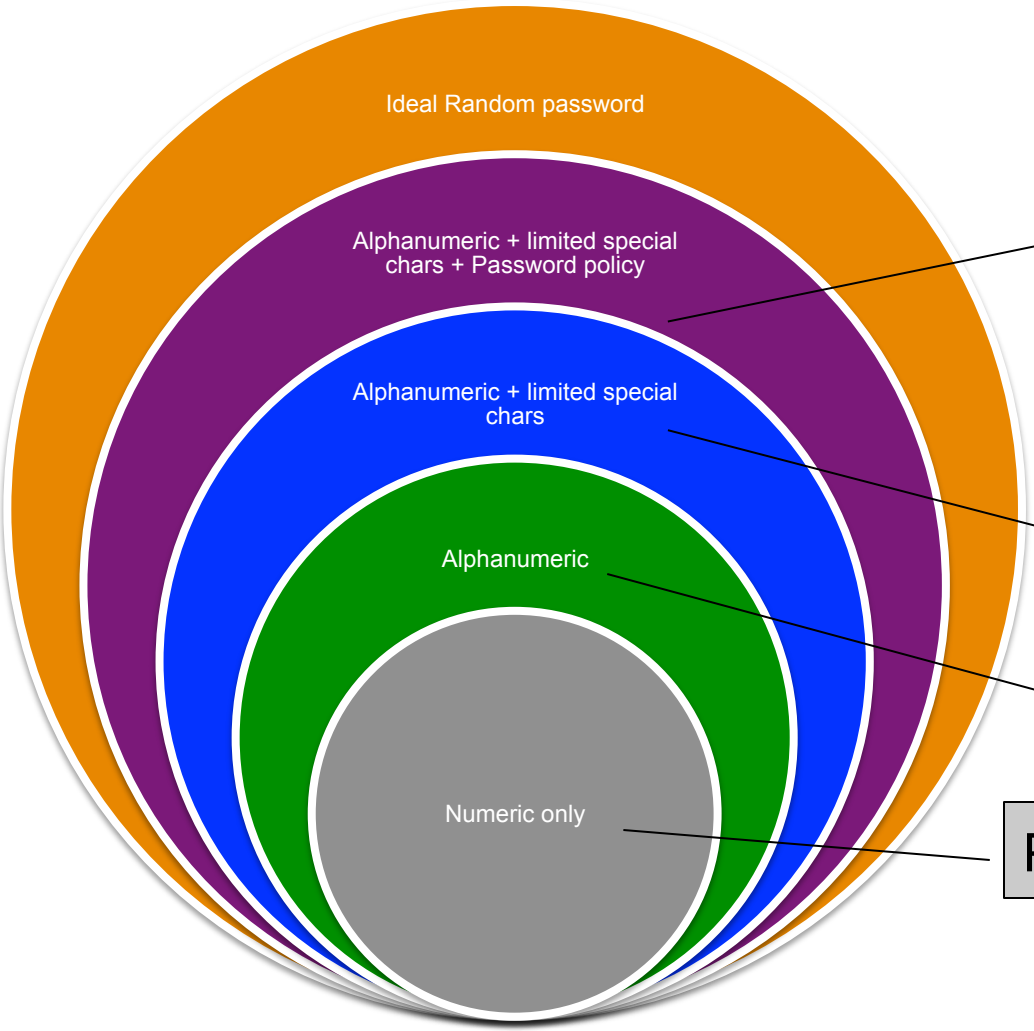
Without randomness

TCP Reset attacks / predictable TCP source port

The easiest way to implement ‘random TCP src port’ is counter++
OSX keep TCP source port++ for each new request, same as Windows

293	0.000101000	10.2.45.15	72.21.202.183	TCP	62241 > 80 [SYN] Seq=4097806945 Win=65535 Len=0
294	0.000048000	10.2.45.15	72.21.202.183	TCP	62242 > 80 [SYN] Seq=1631722621 Win=65535 Len=0
295	0.000034000	10.2.45.15	72.21.202.183	TCP	62243 > 80 [SYN] Seq=4046561166 Win=65535 Len=0
303	0.000216000	10.2.45.15	176.32.103.111	TCP	62244 > 80 [SYN] Seq=353198172 Win=65535 Len=0 M
304	0.000068000	10.2.45.15	176.32.103.111	TCP	62245 > 80 [SYN] Seq=1562545554 Win=65535 Len=0
306	0.000001000	10.2.45.15	176.32.103.111	TCP	62246 > 80 [SYN] Seq=4232495449 Win=65535 Len=0
307	0.000036000	10.2.45.15	176.32.103.111	TCP	62247 > 80 [SYN] Seq=1673477838 Win=65535 Len=0
308	0.000057000	10.2.45.15	176.32.103.111	TCP	62248 > 80 [SYN] Seq=2960773473 Win=65535 Len=0
309	0.000177000	10.2.45.15	54.230.85.29	TCP	62249 > 80 [SYN] Seq=3229015903 Win=65535 Len=0
310	0.000057000	10.2.45.15	54.230.85.29	TCP	62250 > 80 [SYN] Seq=106953361 Win=65535 Len=0 M
311	0.000061000	10.2.45.15	54.230.85.29	TCP	62251 > 80 [SYN] Seq=3239052700 Win=65535 Len=0
313	0.000021000	10.2.45.15	54.230.85.29	TCP	62252 > 80 [SYN] Seq=2774669737 Win=65535 Len=0
314	0.000042000	10.2.45.15	54.230.85.29	TCP	62253 > 80 [SYN] Seq=35904795 Win=65535 Len=0 MS
315	0.000043000	10.2.45.15	54.230.85.29	TCP	62254 > 80 [SYN] Seq=67609313 Win=65535 Len=0 MS
317	0.000146000	10.2.45.15	54.230.84.204	TCP	62255 > 80 [SYN] Seq=3899784881 Win=65535 Len=0
318	0.000050000	10.2.45.15	54.230.84.204	TCP	62256 > 80 [SYN] Seq=3416634072 Win=65535 Len=0

How online services support random password



Variants by languages & site owners, + **Totally insane RANDOM randomness policy**

Variants by languages & site owners

Lazy administrators

Phone compatible services

DDoS attacks – the art of evasion

Attack goes undetected is getting harder

- 0-days on protocol are getting harder to dig out
- Detections are implementing closer to bots
- Security awareness increased by site owners
- DDoS tools are mostly open sources
- Signatures of DDoS tools can be easily implemented
- Websites are behind mitigation filters or CDNs

A successful DDoS attacks is

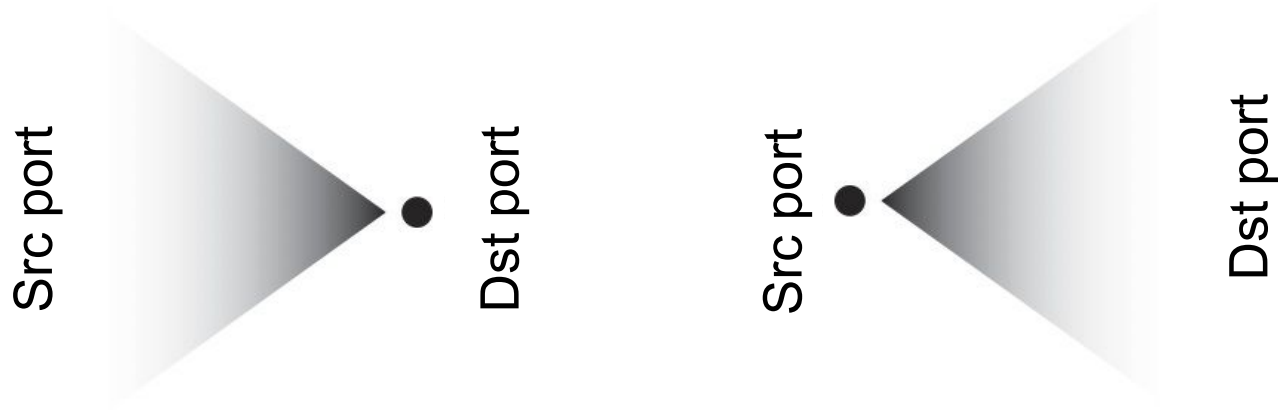
- Make as many false possible as possible
- Detection and mitigation filter never trigger
- Real server believes it is from a legitimate user

Level 0.0 – Bandwidth attacks

100% stateless, even initiated in TCP
99.99% chance of being block since the port is not open
99% chance of being block from source
Your botnet may disconnect from command updates

Level 0.1 – Bandwidth attacks Reflected

100% stateless, mostly works with UDP
Attack power relies on intermediate victim servers
Attack efficiency relies on amplification factor
It's easy to detect, and it's from fixed source port ☺



Normal Traffic

Attack traffic

Level 1.0 – TCP SYN Flood

100% stateless
99.99% using spoof IP
99% complies with RFC but not exists in real world

RFC 793 (TCP) is 33 years old

- it didn't say what you should not spoof
- it didn't say what TCP ACK you should pick during TCP handshake
- It didn't say how many TCP Options you should include during handshake

Level 1.0 – TCP SYN Flood

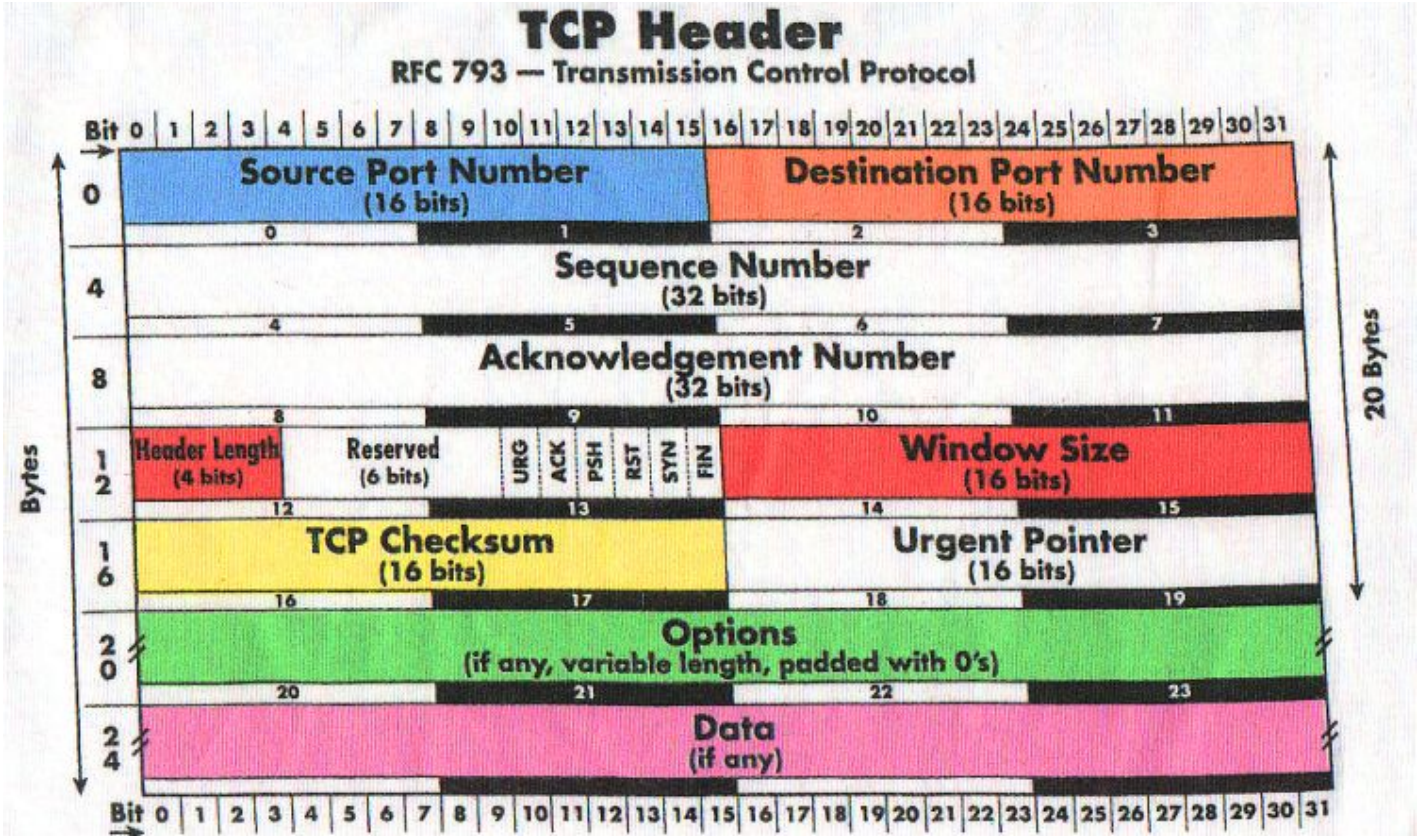
```
Sendtcp.c (hping3-20051105)

/* sequence number and ack are random if not set */
tcp->th_seq = (set_seqnum) ? htonl(tcp_seqnum) : htonl(rand());
tcp->th_ack = (set_ack) ? htonl(tcp_ack) : htonl(rand());

sequence++;     /* next sequence number */
        if (!opt_keepstill)
                src_port = (sequence + initsport) % 65536;
Main.c
/* set initial source port */
        if (initsport == -1)
                initsport = src_port = 1024 + (rand() % 2000);
```

It's easy to spot HPING from source port and non-zero tcp_ack #

Level 1.0 – TCP SYN Flood

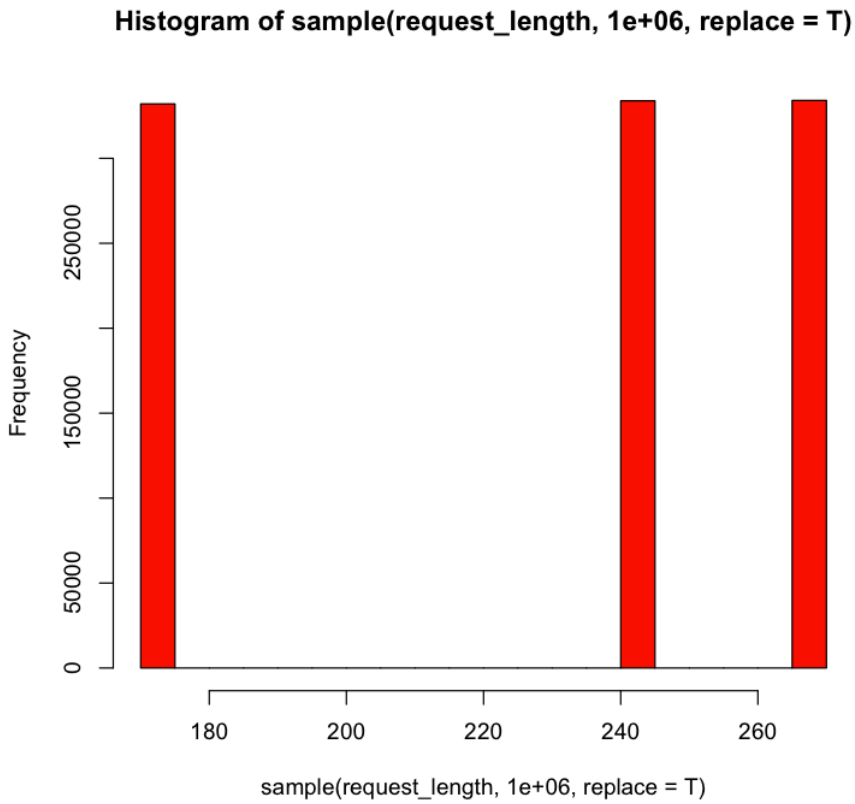
Randomness detection can be based on COMBINATION of fields

Insane packet can be dropped: tcp.flags == 0x02 && (ip.len – 40)%4 !=0



Level 2.0 – HTTP GET Flood - static

```
for ((i=0;i<100;i++)) do `wget target.com &`; done
```



It's is legitimate but
- it's dummy and static
- it's HTTP/1.0
- lack of HTTP headers

Distribution of requests are
- spectrum like
- not as random as expected

How to mitigate
- block tcp.flags == 0x18 and ip.len < 100 and tcp.dstport == 80

Level 2.1 – HTTP GET Flood – static random

This is legitimate request

```
GET / HTTP/1.1
Host: www.nexusguard.com
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5)
AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31
Referer: https://www.facebook.com/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Level 2.1 – HTTP GET Flood – static random

This is how attacker try to variety

```
GET / HTTP/1.1
Host: www.nexusguard.com
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: $VARIABLE

Referer: https://www.facebook.com/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

```
Hulk.py
#builds random ascii string
def buildblock(size):
        out_str = ''
        for i in range(0, size):
                a = random.randint(65, 90)
                out_str += chr(a)
        return(out_str)
```

Level 2.1 – HTTP GET Flood – static random

Hulk.py

```
# generates a user agent array
def useragent_list():
          global headers_useragents
          headers_useragents.append('Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.1.3) Gecko/
20090913 Firefox/3.5.3')
          headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 6.1; en; rv:1.9.1.3)
Gecko/20090824 Firefox/3.5.3 (.NET CLR 3.5.30729)')
          headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.9.1.3)
Gecko/20090824 Firefox/3.5.3 (.NET CLR 3.5.30729)')
          headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.1)
Gecko/20090718 Firefox/3.5.1')
          headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/
532.1 (KHTML, like Gecko) Chrome/4.0.219.6 Safari/532.1')
          headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; InfoPath.2)')
          headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/
4.0; SLCC1; .NET CLR 2.0.50727; .NET CLR 1.1.4322; .NET CLR 3.5.30729; .NET CLR 3.0.30729)')
          headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.2; Win64;
x64; Trident/4.0)')
          headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/
4.0; SV1; .NET CLR 2.0.50727; InfoPath.2)')
          headers_useragents.append('Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)')
          headers_useragents.append('Mozilla/4.0 (compatible; MSIE 6.1; Windows XP)')
          headers_useragents.append('Opera/9.80 (Windows NT 5.2; U; ru) Presto/2.5.22 Version/10.51')
          return(headers_useragents)
```

Level 2.1 – HTTP GET Flood – static random

DirtJumper v5 User Agent selector

```
CODE:004283E8 aMozilla5_0Wi_5 db 'Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.8.1) VoilaBot B'
CODE:004283E8                                          ; DATA XREF: sub_4225C8+1DD2↓o
CODE:004283E8                 db 'ETA 1.2 (support.voilabot@orange-ftgroup.com) ',0
CODE:00428458                 dd 0FFFFFFFFh, 7Bh
CODE:00428460 aMozilla5_0Wi_6 db 'Mozilla/5.0 (Windows; U;XMPP Tiscali Communicator v.10.0.1; Windo'
CODE:00428460                                          ; DATA XREF: sub_4225C8+1DE0↓o
CODE:00428460                 db 'ws NT 5.1; it; rv:1.8.1.3) Gecko/20070309 Firefox/2.0.0.3 ',0
CODE:004284DC                 dd 0FFFFFFFFh, 75h
CODE:004284E4 aMozilla5_0Wi_7 db 'Mozilla/5.0 (Windows;) NimbleCrawler 1.12 obeys UserAgent NimbleC'
CODE:004284E4                                          ; DATA XREF: sub_4225C8+1DEE↓o
CODE:004284E4                 db 'rawler For problems contact: crawler@healthline.com ',0
CODE:0042855A                 align 4
CODE:0042855C                 dd 0FFFFFFFFh, 53h
CODE:00428564 aMozilla5_0X11U db 'Mozilla/5.0 (X11; U; Linux 2.4.2-2 i586; en-US; m18) Gecko/200101'
CODE:00428564                                          ; DATA XREF: sub_4225C8+1DFC↓o
CODE:00428564                 db '31 Netscape6/6.01 ',0
CODE:004285B8                 dd 0FFFFFFFFh, 70h
CODE:004285C0 aMozilla5_0X1_0 db 'Mozilla/5.0 (X11; U; Linux i686; en-GB; rv:1.7.6) Gecko/20050405 '
CODE:004285C0                                          ; DATA XREF: sub_4225C8+1E0A↓o
CODE:004285C0                 db 'Epiphany/1.6.1 (Ubuntu) (Ubuntu package 1.0.2) ',0
CODE:00428631                 align 4
CODE:00428634                 dd 0FFFFFFFFh, 41h
CODE:0042863C aMozilla5_0X1_1 db 'Mozilla/5.0 (X11; U; Linux i686; en-US; rv:0.9.3) Gecko/20010801 '
CODE:0042863C                                          ; DATA XREF: sub_4225C8+1E18↓o
CODE:0042863C                 db 0
CODE:0042867E                 align 10h
CODE:00428680                 dd 0FFFFFFFFh, 4Ch
```

Level 2.2 – HTTP GET Flood – dynamic random

```
#http request
def httpcall(url):

          request = urllib2.Request(url + param_joiner + buildblock(random.randint(3,10)) + '=' +
buildblock(random.randint(3,10)))
          request.add_header('User-Agent', random.choice(headers_useragents))
          request.add_header('Cache-Control', 'no-cache')
          request.add_header('Accept-Charset', 'ISO-8859-1,utf-8;q=0.7,*;q=0.7')
          request.add_header('Referer', random.choice(headers_referers) +
buildblock(random.randint(5,10)))
          request.add_header('Keep-Alive', random.randint(110,120))
          request.add_header('Connection', 'keep-alive')
          request.add_header('Host',host)
```

Don't do **unreasonable** random for the sake of randomness confusion
Normal HTTP keep-alive range doesn't fall in this range

Level 2.2 – HTTP GET Flood – dynamic random

Uagent.php // random user-agent generator

```
function nt_version()
        return rand(5, 6) . '.' . rand(0, 1);

function ie_version() // IE
        return rand(7, 9) . '.0';

function osx_version() // need to add support for OSX10.10 ☺
        return "10_" . rand(5, 7) . '_' . rand(0, 9);

function chrome_version()
        return rand(13, 15) . '.0.' . rand(800, 899) . '.0';
```

Hint: Predict next version by time (build-in script)

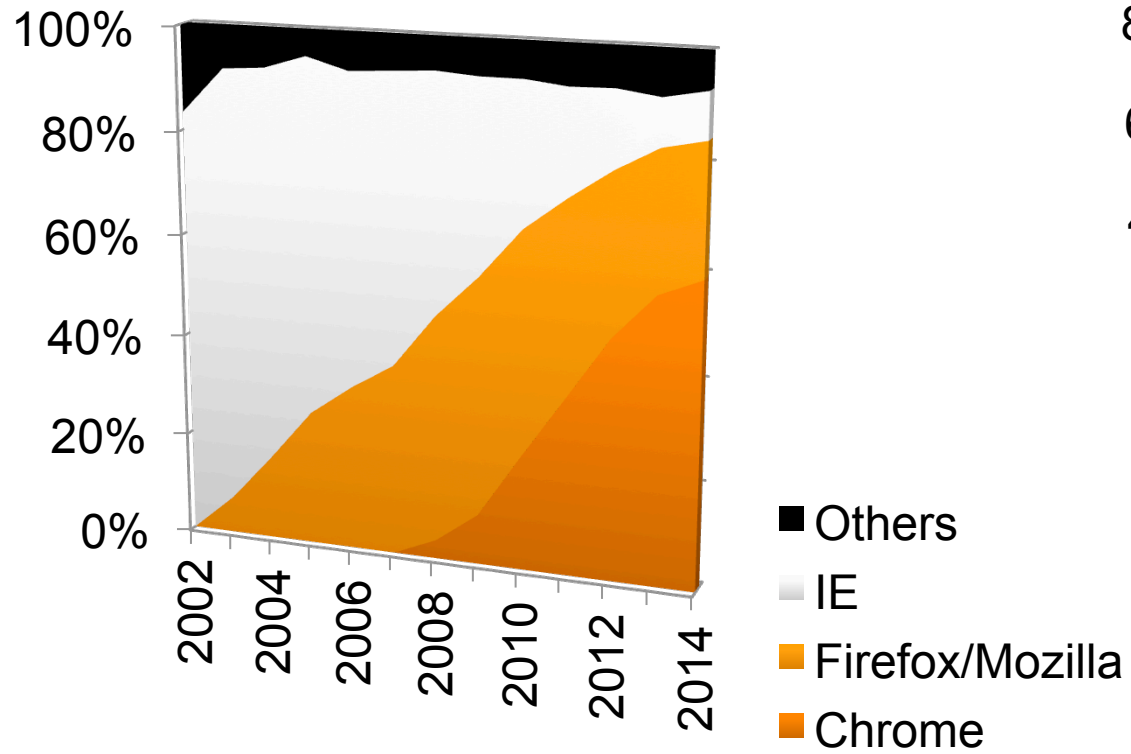
Level 2.2 – HTTP GET Flood – dynamic random

Uagent.php // random user-agent generator

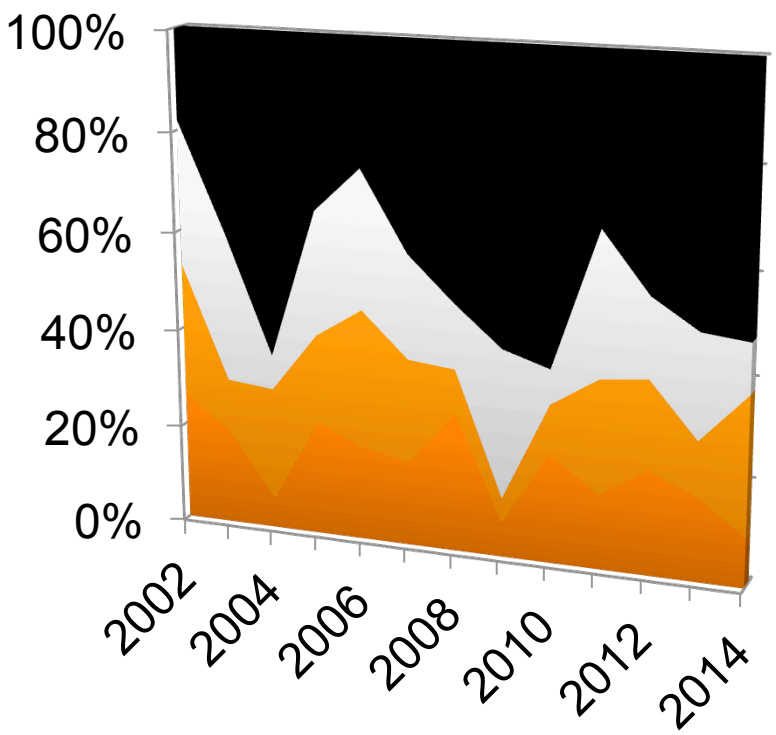
```
function firefox($arch) {
    $ver = array_random(array(
            'Gecko/' . date('Ymd', rand(strtotime('2011-1-1'), time())) . ' Firefox/' . rand(5,
7) . '.0’,
            'Gecko/' . date('Ymd', rand(strtotime('2011-1-1'), time())) . ' Firefox/' . rand(5,
7) . '.0.1’,
            'Gecko/' . date('Ymd', rand(strtotime('2010-1-1'), time())) . ' Firefox/3.6.' .
rand(1, 20),
        'Gecko/' . date('Ymd', rand(strtotime('2010-1-1'), time())) . ' Firefox/3.8’
    ));

    switch ($arch) { // firefox for Linux, Mac and Win with different processers
    case 'lin’:
        return "(X11; Linux {proc}; rv:" . rand(5, 7) . ".0) $ver";
    case 'mac':
        $osx = osx_version();
        return "(Macintosh; {proc} Mac OS X $osx rv:" . rand(2, 6) . ".0) $ver »;
    case 'win’:
    default:
        $nt = nt_version();
        return "(Windows NT $nt; {lang}; rv:1.9." . rand(0, 2) . ".20) $ver »;
    }
}
```

Level 2.3 – HTTP GET Flood – smart random

User-agents are not randomly distributed



Legitimate UA distribution by year

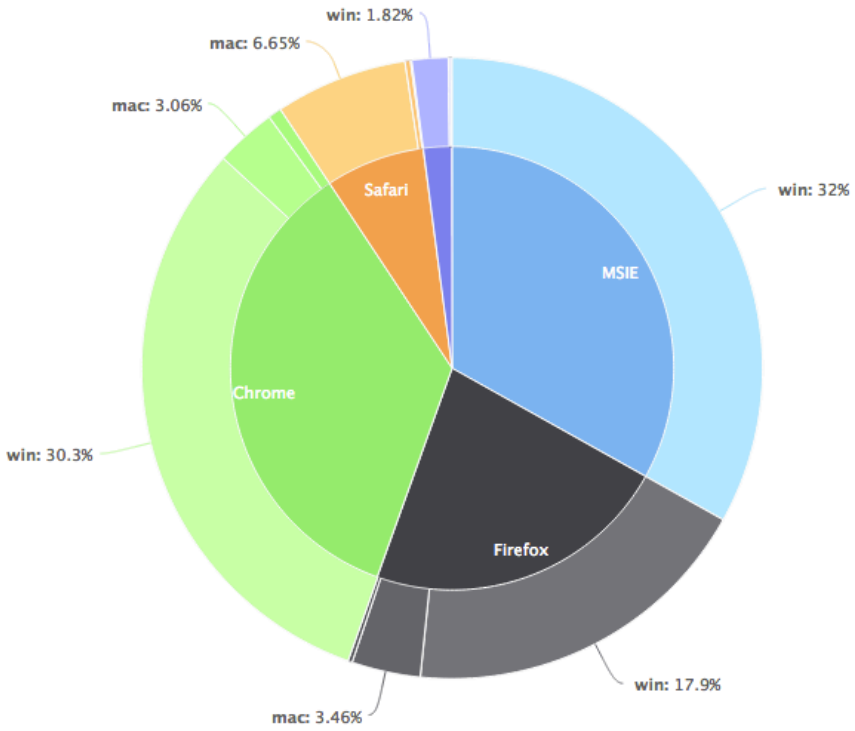


Attack UA distribution by year

Level 2.3 – HTTP GET Flood – smart random

User-agents are not randomly distribute

```
function chooseRandomBrowserAndOS() {
   $frequencies = array(
        34 => array(
            89 => array('chrome', 'win'),
            9 => array('chrome', 'mac'),
            2 => array('chrome', 'lin') ),
        32 => array(
            100 => array('iexplorer', 'win')),
        25 => array(
            83 => array('firefox', 'win'),
            16 => array('firefox', 'mac'),
            1 => array('firefox', 'lin')),
        7 => array(
            95 => array('safari', 'mac'),
            4 => array('safari', 'win'),
            1 => array('safari', 'lin')),
        2 => array(
            91 => array('opera', 'win'),
            6 => array('opera', 'lin'),
            3 => array('opera', 'mac'))
    );
```

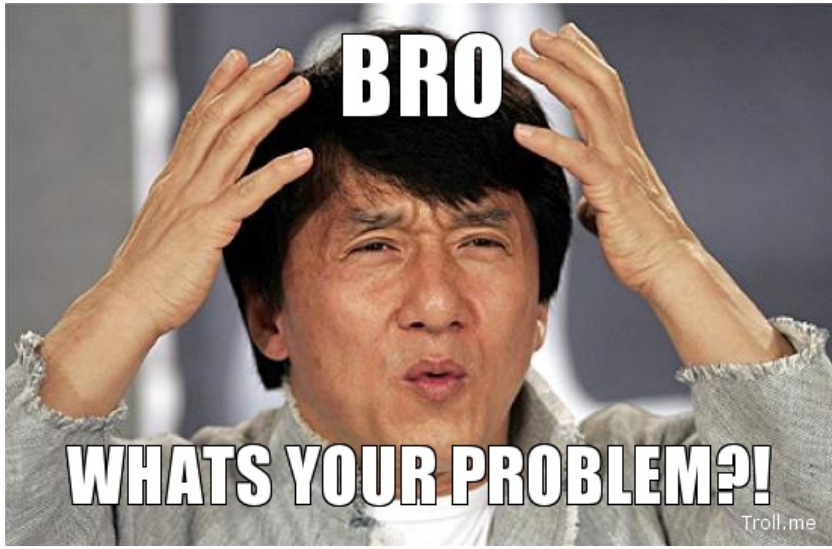


Level 2.3 – HTTP GET Flood – dynamic random

100% predictable URL and parameter
100% predictable HTTP header order
99% purely randomize in pre-defined character space

ADDRESS ORDERS MATTERS

- because RFC2616 HTTP/1.1 only specific required headers, not orders
- implementation of HTTP header order is depending on OS
- Orders can be normalized / corrected by CDN, thank you CDN ☺



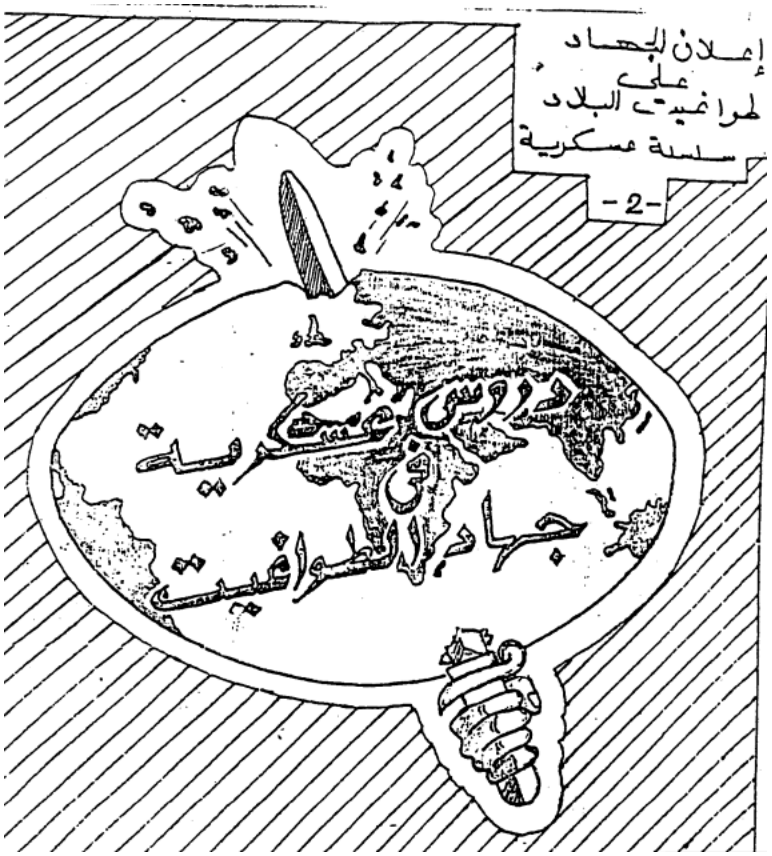
CHARACTER SPACE MATTERS

- Pure random is easy to be detected
- Attack character space didn't fit with distribution of normal request

Level 3.0 – HTTP GET Flood – emulated random

Al Qaeda Handbook
- The Manchester Manual

Lesson 3
Forged Documents
(Identity Cards, Records Books, Passports)



Level 3.0 – HTTP GET Flood – emulated random

Forged Documents (Identity Cards, Records Books, Passports)
The following security precautions should be taken:

1. Keeping the passport in a **safe** place so it would not be ceized by the security apparatus, and the brother it belongs to would have to negotiate its return (I'll give you your passport if you give me information)

2. All documents of the undercover brother, such as identity cards and passport, **should be falsified**.

3. When the undercover brother is traveling with a certain identity card or passport, he should know **all pertinent** [information] such as the name, profession, and place of residence.

Use Proxy
X-forwarded-IP
X-Client-IP

Always spoof User-agent

Behave and react
as claimed, real UA

Level 3.0 – HTTP GET Flood – emulated random

4. The brother who has special work status (commander, **communication** link, ...) should have more than one identity card and passport. He should learn the contents of each, the nature of the [indicated] profession, and the dialect of the residence area listed in the document.

5. The photograph of the brother in these documents should be without a beard. It is preferable that the brother's public photograph [on these documents] be also without a beard. If he already has one [document] showing a photograph with a beard, he should **replace** it.

6. When using an identity document in **different names**, no more than one such document should be carried at one time.

Use anonymous proxy
Use anonymous network (TOR)

Never use real IP to send C&C command or send attack

Don't send too much traffic from a single machine

Level 3.0 – HTTP GET Flood – emulated random

Now attacks are emulating from real users, with

- Low request rate
- From normally distributed source IP (GEO-IP)
- Totally valid TCP and IP headers
- Legitimate user-agents
- Legitimate user-agents with up-to-date distribution
- Correct HTTP headers and orders

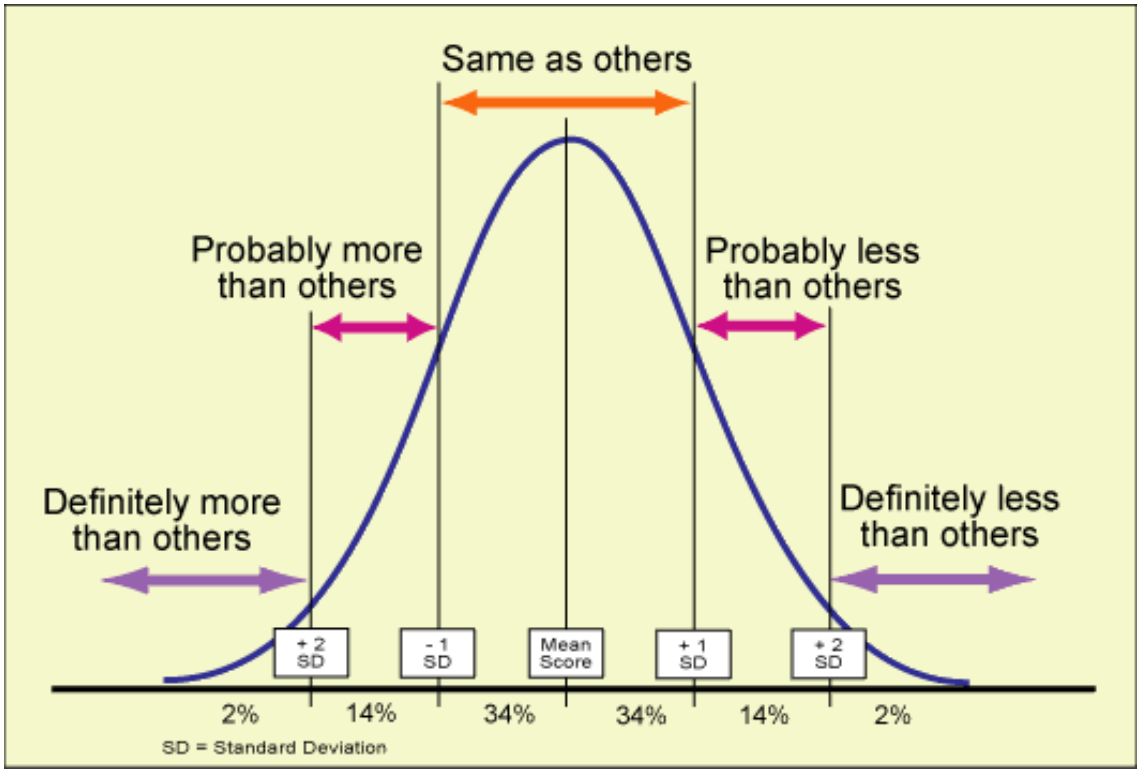
Level 3.0 – HTTP GET Flood – emulated random

p0f

Passive, progressive, layered validation

Level 3.0 – HTTP GET Flood – emulated random

behavior



Progressive, application specific challenge,

Level BOSS – DDoS the legitimate client

Attacker knows your clients' IPs
Attacker knows your detection policies
Attacker knows your mitigation filters

Attacker can launch 'targeted' DDoS by spoofing legitimate client

Proudly Present
"APT Style" DDoS

Level BOSS – DDoS the legitimate client

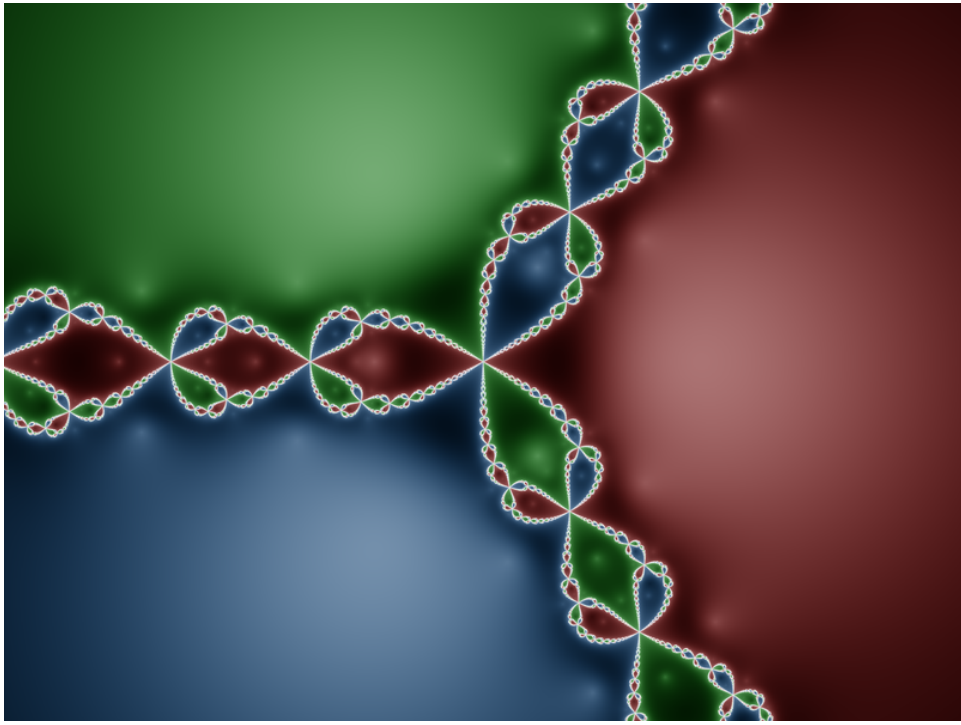
False Positive

False Negative

A + B = Constant

Level BOSS – DDoS the legitimate client

OR



One of the acceptable sample output:
bhvbhdjmnnmbfjnfghjbnvghvbv

Draw this fractal with 2 lines of code
Max. string 200

Questions?

Contact me via ‘random’ e-mail above