RSA CONFERENCE ASIA PACIFIC 2013

REMOVING THE MYSTERY OF SECURITY ENGINES AND THEIR EFFECT ON YOUR NETWORK

Philip Trainor
Senior Manager – Applications and Security
Ixia Communications

Session ID: SPO-T02

Session Classification: Intermediate

BYOD, Cloud Computing, and switching from wired to wireless is a network performance and security nightmare!

- Complex network traffic exponentially increases the workload on Network infrastructure. This lecture will discuss:
  - What function does a security engine or content aware device perform in a network?
  - What are their choke points when handing complex traffic?
  - How is malicious traffic handled differently than normal user traffic?
  - How can a network ensure performance under ALL conditions?

Modern network Size: Mobile

- In the last year there were more smart phones & tablets activated than there were babies born
- 1.3 million android devices are activated every day…300K babies born every day
- The average smartphone has 41 apps
- The average person checks their phone 150 times per day (or once every 6.5 minutes)
- By the end of 2013 there will be an estimated 1.82 billion active smartphones globally.

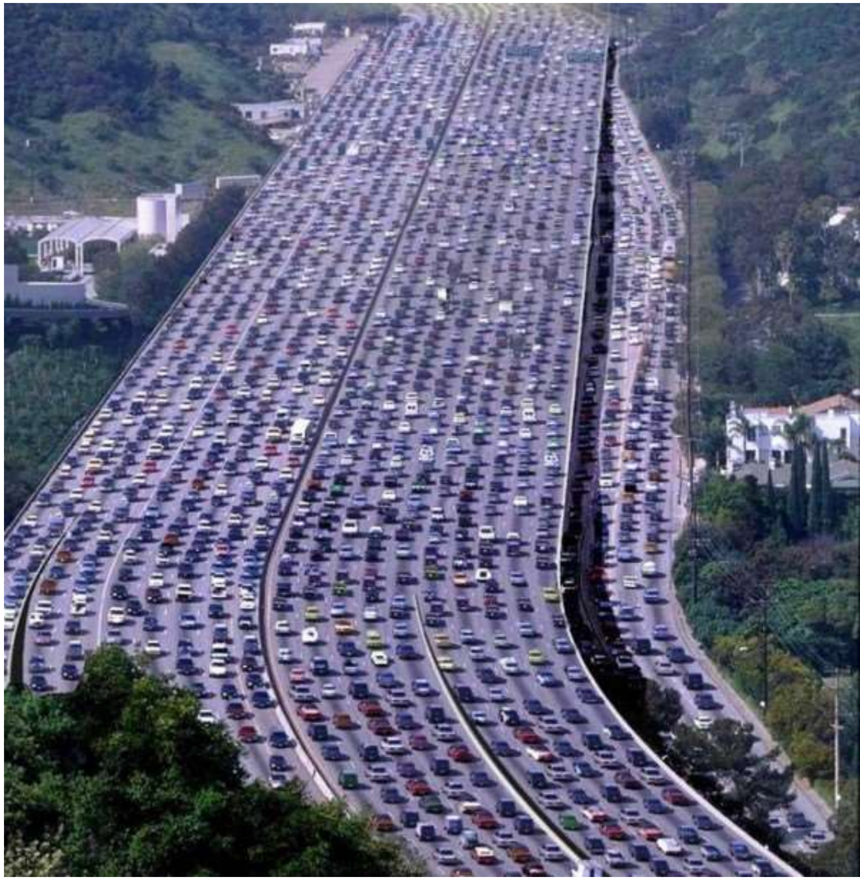
8.7 Billion Wired Devices online

- According to Cisco, in 2012 there were 8.7 billion devices connected to the internet
- There are approximately 7 billion people in the world today…

http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

That makes 10 Billion Networked Devices Globally

- These devices access both Sensitive Data (Personal and non-personal) as well as untrusted networks.
- The content they view is highly complex and requires high bandwidth
- The devices move from Network to Network **CONSTANTLY**



BYOD (Bring Your Own Device

- Many work places allow for their employees to simply bring ANY machine they want to work
- Why is this a problem?
  - Taking a machine out of a controlle environment will potentially expose the system to malicious events
  - The user will then BRING their compromised machine to work where it will be in contact with other critical machines and pieces of network infrastructure.

Cloud Computing Security Issues

- Interconnecting disparate networks creates environments where the exposure to compromised machines is almost certain
- How do you ensure that opening up your network to that level of exposure will not result in a security event?!?!??

Networks rely on security devices!

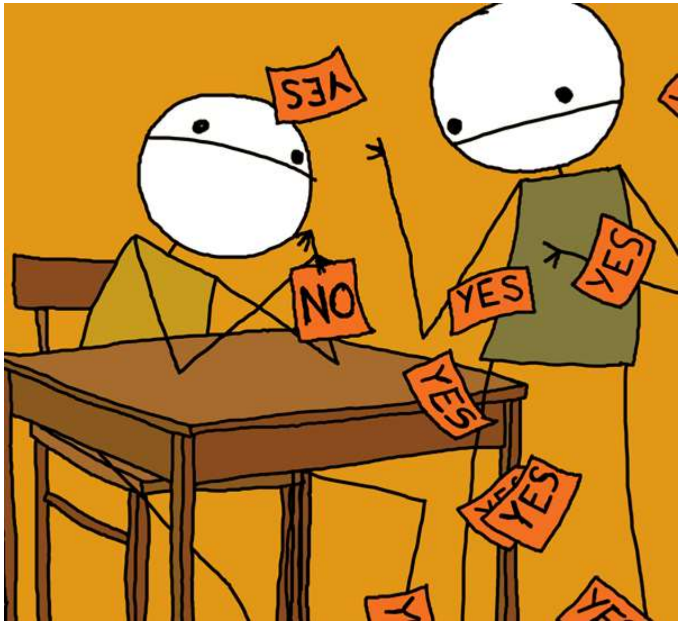
- The burden on ensuring security rest squarely on the deployment of security solutions!
- Let's remove the mystery of how they work so we can better understand the problems we face.

The job of security and content aware devices is daunting!

- These devices must make decisions based on parsing information within traffic flows.
- Processing complex flows by content aware devices ***WILL*** increase network latency
- The development of new applications is rapid and constant
- The development of new security attacks is also rapid and constant

What does a *content aware* security engine do?

- Inline security devices are a full content wire-tap. They sit inline on a network and make decisions.
- Is the traffic allowed from the UNTRUSTED zone to the TRUSTED zone?
- Do we log the event or block the network traffic?
- Is this an isolated event or part of a massive attack?



How do security devices make those decisions?

- Security devices look at packets and run ***regular expressions*** to parse real user traffic versus potentially malicious traffic.

No.	Time	Source	Destination	Protocol	Length	Info
1761	46.4859400	192.168.1.3	192.168.1.4	SSH	106	Encrypted response packet len=52
1762	46.4860070	192.168.1.4	192.168.1.3	TCP	54	57228 > ssh [ACK] Seq=10525 Ack=6
1763	46.5412650	192.168.1.3	192.168.1.4	SSH	154	Encrypted response packet len=100
1764	46.7409600	192.168.1.4	192.168.1.3	TCP	54	57228 > ssh [ACK] Seq=10525 Ack=6
1765	47.1110490	192.168.1.4	192.168.1.255	NBNS	92	Name query NB RWTDOM<1b>
1766	47.8610480	192.168.1.4	192.168.1.255	NBNS	92	Name query NB RWTDOM<1b>
1767	48.8550230	66.196.120.100	192.168.1.4	HTTP	92	HTTP/1.1 200 OK
1768	48.8605440	192.168.1.4	66.196.120.100	TCP	1404	[TCP segment of a reassembled PDU
1769	48.8605550	192.168.1.4	66.196.120.100	HTTP	260	GET /v1/pushchannel/phil_trainor?
1770	48.9171920	66.196.120.100	192.168.1.4	TCP	60	http > 56888 [ACK] Seq=39 Ack=155
1771	49.6110860	192.168.1.4	192.168.1.255	NBNS	92	Name query NB RWTDOM<1e>
1772	50.3619080	192.168.1.4	192.168.1.255	NBNS	92	Name query NB RWTDOM<1e>
1773	51.1119560	192.168.1.4	192.168.1.255	NBNS	92	Name query NB RWTDOM<1e>
1774	51.8624070	192.168.1.4	192.168.1.255	BROWSER	216	Get Backup List Request
1775	51.8624710	192.168.1.4	192.168.1.255	NBNS	92	Name query NB QATEST<1b>
1776	52.6120010	192.168.1.4	192.168.1.255	NBNS	92	Name query NB QATEST<1b>

What is PCRE?

- Perl Compatible Regular Expressions:
  - The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl.
- Individuals write pattern matching logic in PCRE for security engines to match against malicious network traffic

- Let's look at one now:

This Snort signature uses PCRE:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"OS-OTHER Cisco IOS HTTP configuration attempt"; flow:to_server,established; content:"/level/"; http_uri; pcre:"/\x2flevel\x2f\d+\x2f(exec|configure)/iU"; metadata:ruleset community, service http; reference:bugtraq,2936; reference:cve,2001-0537; reference:nessus,10700; classtype:web-application-attack; sid:1250; rev:21;)

What is the rule trying to prevent?

- A crafted string to certain Cisco devices will bypass the authentication and allow the attacker to execute any command on the device.

Let's dissect the rule further:

- We make a security alert **IFF** criteria is matched:

1. tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS;
2. flow:to_server,established;
3. content: "/level/";
4. pcre:"/\x2flevel\x2f\d+\x2f(exec|configure)/iU";

A deeper look at the PCRE Section:

- "/\x2flevel\x2f\d+\x2f(exec|configure)/iU"

- \x2f is the hex encoding for a backslash (/)
- level is just ASCII “level”
- \d+ is any number of digits (6422423)
- (exec|config) is either ASCII “exec” of ASCII “config”

- The corresponding attack to the regex is:
- GET /level/16/exec//show HTTP/1.1
- See how they match up?

This is how network traffic will look when the pattern is matched and the engine blocks

- This is the perspective from the untrust zone.
- After the socket is made the exploit is attempted and the connection is swiftly reset

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	MS-NLB-PhysServer-2	Broadcast	ARP	60	Who has 14.0.0.1? Tell 14.0.0.11
2	0.000059	Sonicwal_c0:47:a4	MS-NLB-PhysServer-2	ARP	60	14.0.0.1 is at 00:17:c5:c0:47:a4
3	0.013267	14.0.0.11	15.0.0.10	TCP	62	23652 > http [SYN] Seq=0 Win=16383 Len=0 MSS=1460
4	0.015735	15.0.0.10	14.0.0.11	TCP	62	http > 23652 [SYN, ACK] Seq=0 Ack=1 Win=16383 Len=
5	0.058570	14.0.0.11	15.0.0.10	TCP	62	23652 > http [ACK] Seq=1 Ack=1 Win=16383 Len=0 MS
6	0.119253	14.0.0.11	15.0.0.10	HTTP	278	GET /level/16/exec//show HTTP/1.1
7	0.119479	15.0.0.10	14.0.0.11	TCP	60	http > 23652 [RST, ACK] Seq=1 Ack=1 Win=16383 Len=

Here is the trust side perspective

- Notice that the HTTP GET Method with the *malicious content* **NEVER** made it to trust:

1	0.000000	MS-NLB-PhysServer-2	Broadcast	ARP	60	Who has 15.0.0.1? Tell 15.0.0.10
2	0.000084	Sonicwal_c0:47:a5	MS-NLB-PhysServer-2	ARP	60	15.0.0.1 is at 00:17:c5:c0:47:a5
3	0.013414	14.0.0.11	15.0.0.10	TCP	62	23652 > http [SYN] Seq=0 Win=16383 Len=0 MSS=1460 W
4	0.015674	15.0.0.10	14.0.0.11	TCP	62	http > 23652 [SYN, ACK] Seq=0 Ack=1 Win=16383 Len=0
5	0.058613	14.0.0.11	15.0.0.10	TCP	62	23652 > http [ACK] Seq=1 Ack=1 Win=16383 Len=0 MSS=
6	0.119484	14.0.0.11	15.0.0.10	TCP	60	23652 > http [RST, ACK] Seq=1 Ack=1 Win=16383 Len=0

- A TCP Socket was made from UNTRUST but ***NONE*** of the attack that matched the regex crossed into TRUST.

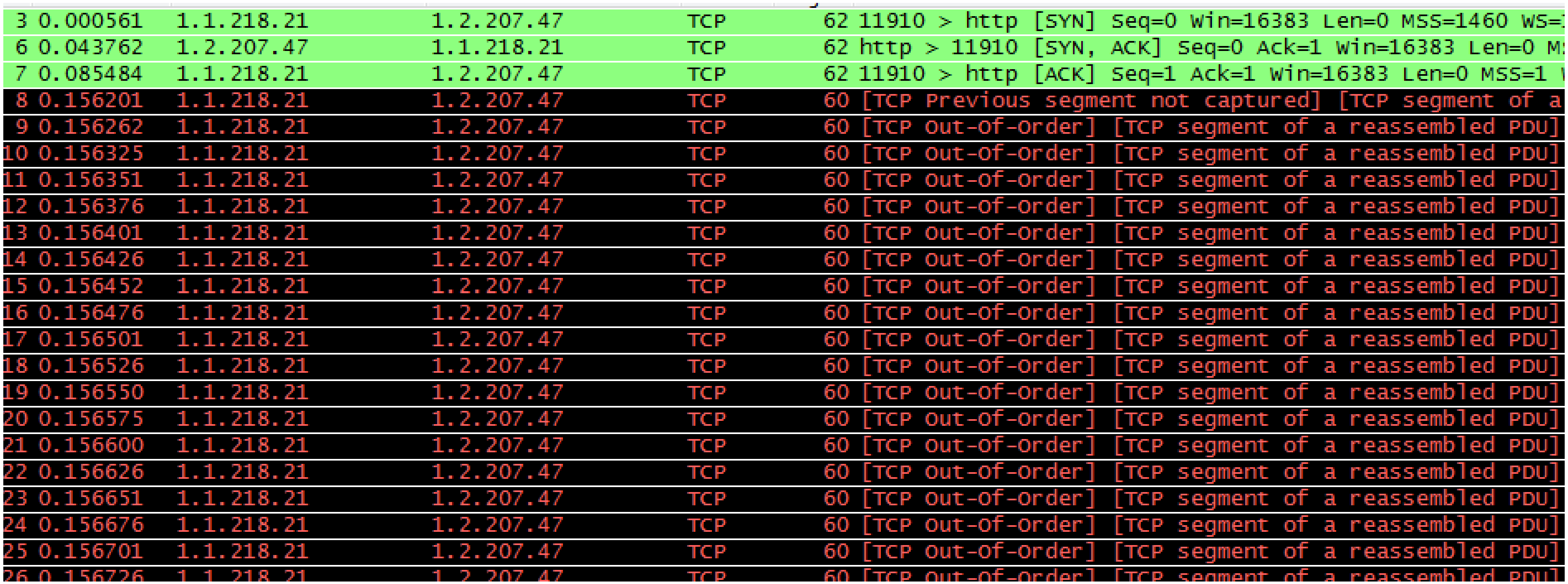
But what if we change how we send the attack with an evasion technique?

- An evasion example is the backslash character "\", U+005C. Under the original UTF-8, it could be represented by a hex 5C, C19C and E0819C. All these represent the exact same Unicode code point: when one applies the transformation algorithm, one gets the same value. Many older applications that support UTF-8 will accept the three values and perform the transformation to the backslash.

14.0.0.3	15.0.0.9	TCP	62	31481 > http [SYN] Seq=0 Win=16383 Len=0 MSS=1460 WS=
15.0.0.9	14.0.0.3	TCP	62	http > 31481 [SYN, ACK] Seq=0 Ack=1 Win=16383 Len=0 M
14.0.0.3	15.0.0.9	TCP	62	31481 > http [ACK] Seq=1 Ack=1 Win=16383 Len=0 MSS=14
14.0.0.3	15.0.0.9	HTTP	220	GET %5clevel%5c16%5cexec%5c%5cshow HTTP/1.1
14.0.0.3	15.0.0.9	TCP	60	31481 > http [FIN, ACK] Seq=167 Ack=1 Win=16383 Len=0
15.0.0.9	14.0.0.3	TCP	60	http > 31481 [FIN, ACK] Seq=1 Ack=167 Win=16383 Len=0
15.0.0.9	14.0.0.3	TCP	60	http > 31481 [ACK] Seq=2 Ack=168 Win=16383 Len=0
14.0.0.3	15.0.0.9	TCP	60	31481 > http [ACK] Seq=168 Ack=2 Win=16383 Len=0

Another Evasion: 1-byte TCP Segmentation with out of order Packets

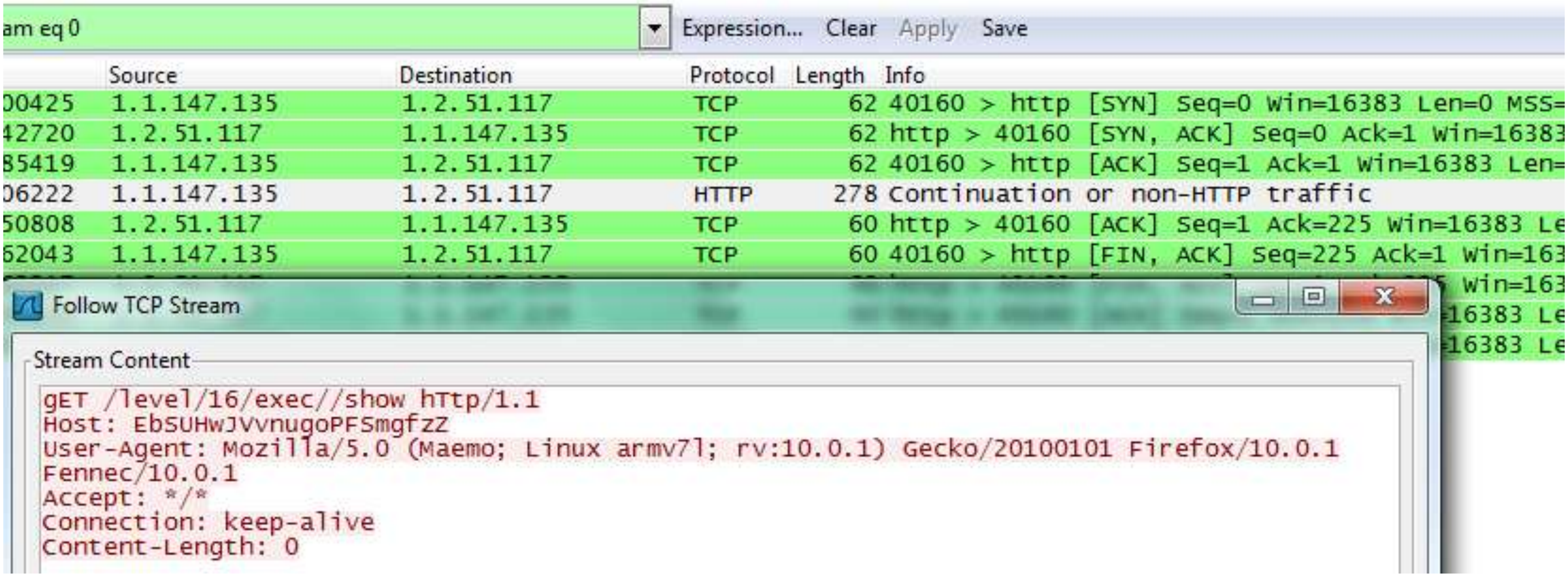
- The security system will have to reassemble the flow in order to perform the regex



3	0.000561	1.1.218.21	1.2.207.47	TCP	62	11910 > http [SYN] Seq=0 Win=16383 Len=0 MSS=1460 WS=
6	0.043762	1.2.207.47	1.1.218.21	TCP	62	http > 11910 [SYN, ACK] Seq=0 Ack=1 Win=16383 Len=0 M
7	0.085484	1.1.218.21	1.2.207.47	TCP	62	11910 > http [ACK] Seq=1 Ack=1 Win=16383 Len=0 MSS=1
8	0.156201	1.1.218.21	1.2.207.47	TCP	60	[TCP Previous segment not captured] [TCP segment of a
9	0.156262	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
10	0.156325	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
11	0.156351	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
12	0.156376	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
13	0.156401	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
14	0.156426	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
15	0.156452	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
16	0.156476	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
17	0.156501	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
18	0.156526	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
19	0.156550	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
20	0.156575	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
21	0.156600	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
22	0.156626	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
23	0.156651	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
24	0.156676	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
25	0.156701	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]
26	0.156726	1.1.218.21	1.2.207.47	TCP	60	[TCP Out-Of-Order] [TCP segment of a reassembled PDU]

How well written is your RegEx?

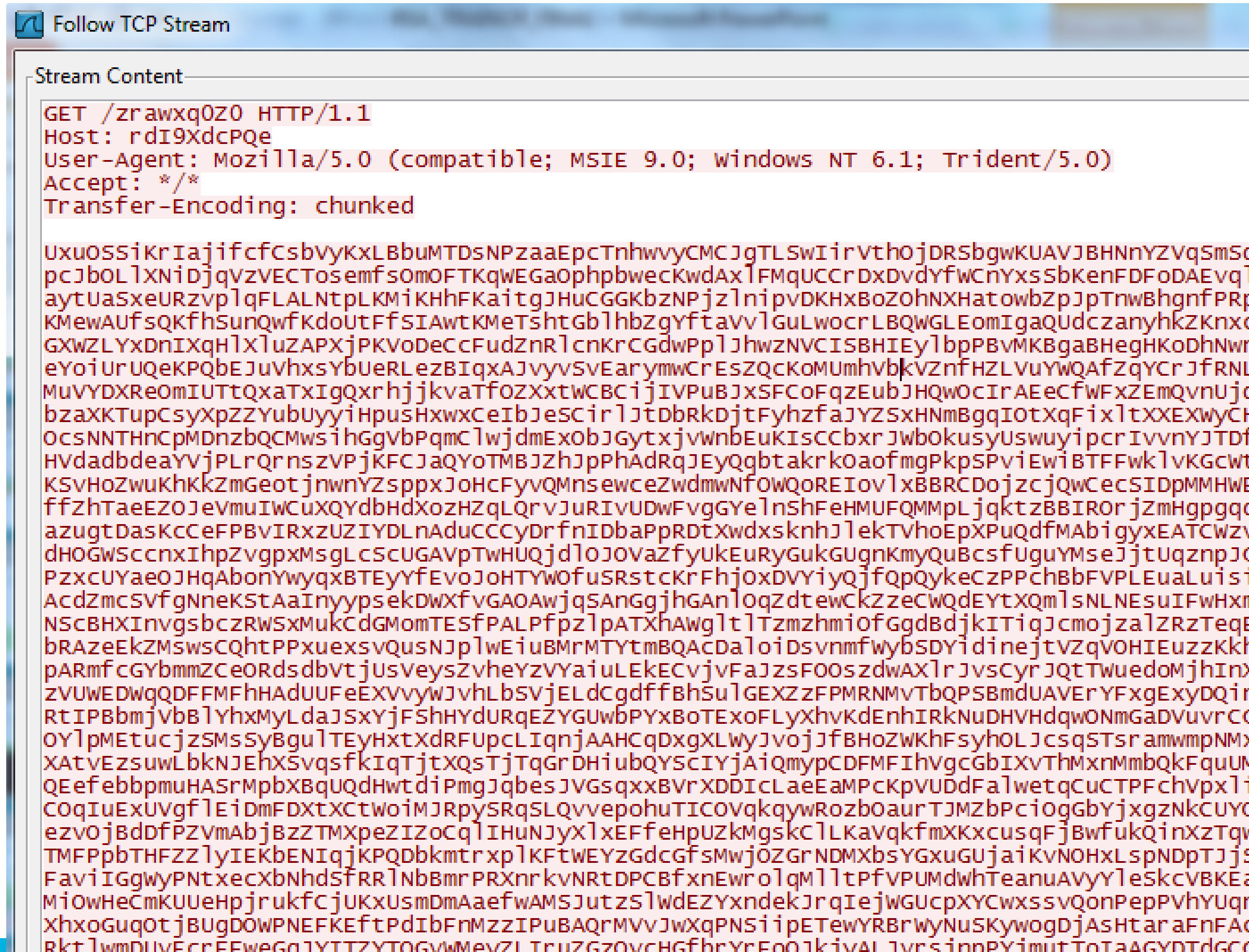
- gET is the exact same action as GET
- hTtp is the exact same action as HTTP
- Will your RegEx account for that scenario?



OK…that attack was old

- …but it was a good example ☺
- Now lets look at a new attack from 2013 and look at the difficulties a security engine will have to deal with!!
- Lets look at CVE 2013-2028 Nginx HTTP Chunked Buffer Overflow
- Nginx is a new and popular webserver that is open source

This is how the attack would look in wireshark:



CVE-2013-2028 in depth

- The attack I crafted to create a stack based buffer overflow a worker process on NginX HTTP Server involved a 7,560 byte chunked-encoded payload
- NginX Version Not vulnerable: 1.5.0+, 1.4.1+
  Vulnerable: 1.3.9-1.4.0
- Discovered on May 7th, 2013

What is Chunked Encoding?

- **Chunked transfer encoding** is a data transfer mechanism in HTTP 1.1
- data is sent in a series of "chunks".
- It uses the Transfer-Encoding HTTP header in place of the Content-Length header
- **Because the Content-Length header is not used, the sender does not need to know the length of the content before it starts transmitting a response to the receiver.**

http://en.wikipedia.org/wiki/Chunked_transfer_encoding

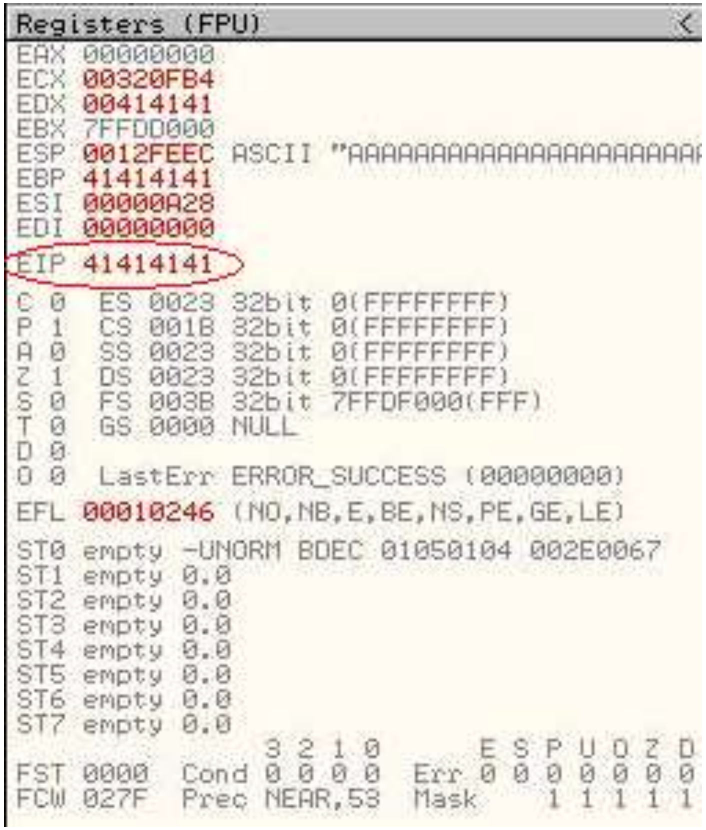
Size Matters!!

- Since we're using chunked encoding the server DOES NOT KNOW how much data we are going to send until we finish!!!
- This is why we are able to over-run the stack and cause a segmentation fault
- If we were clever we would not simply crash the worker process…we would TAKE OVER the worker process and have control over the machine running NginX

How does a stack based buffer overflow work?

Give this program 100 A's…

```
void foo(char *user_str)
{
char local_str[64];
strcpy(local_str, user_str);
}
int main(int argc, char *argv[])
{
 if(argc!=2)
        { printf("usage: %s <in_string>\n", argv[0]); return 1; }
foo(argv[1]);
return 0;
}
```



Background on Stack Pointers

- EIP (Extended Index Pointer) -> Is a 32-bit register that points to the memory address which the processor will next attempt to execute.
- When we push more data onto the stack than was intended we overwrite what instruction happens next.
- If there is garbage in the EIP (\x41\x41\x41…) the program will simply crash
- A clever hacker will figure out how to put his OWN CODE as part of the payload and use a NOP sled to set the EIP value to the start of their code.

I could go on for hours on assembly code but we (unfortunately) do not have time…

More Complexity for Security Engines: Client-Side Attacks!

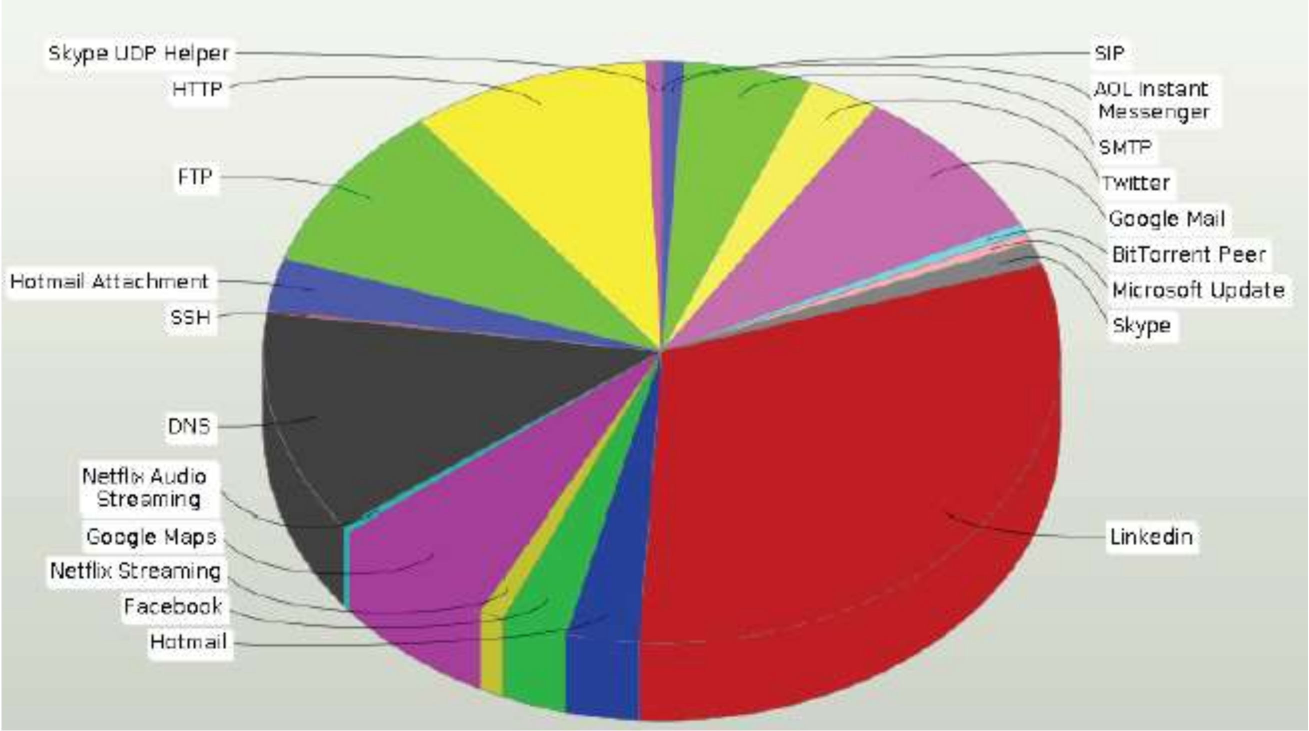
- What is a client side attack?
  - When a user, inside of the trusted zone, contacts a server hosting malicious code
  - This code is then run on the clients machine, typically in a web browser
- Client Side attacks are difficult for security engines to handle for several reasons:
  - The socket is created from INSIDE the trust zone
  - Often times HUGE amounts of data precede the attack which requires security engines to buffer large amounts of data

Back to Business…Security Engines!

- Security Engines have their work cut out for them when identifying attacks…but what happens when the attacks are 0.0001% of the network traffic??!?!?!?!
- The end result is latency and false positives

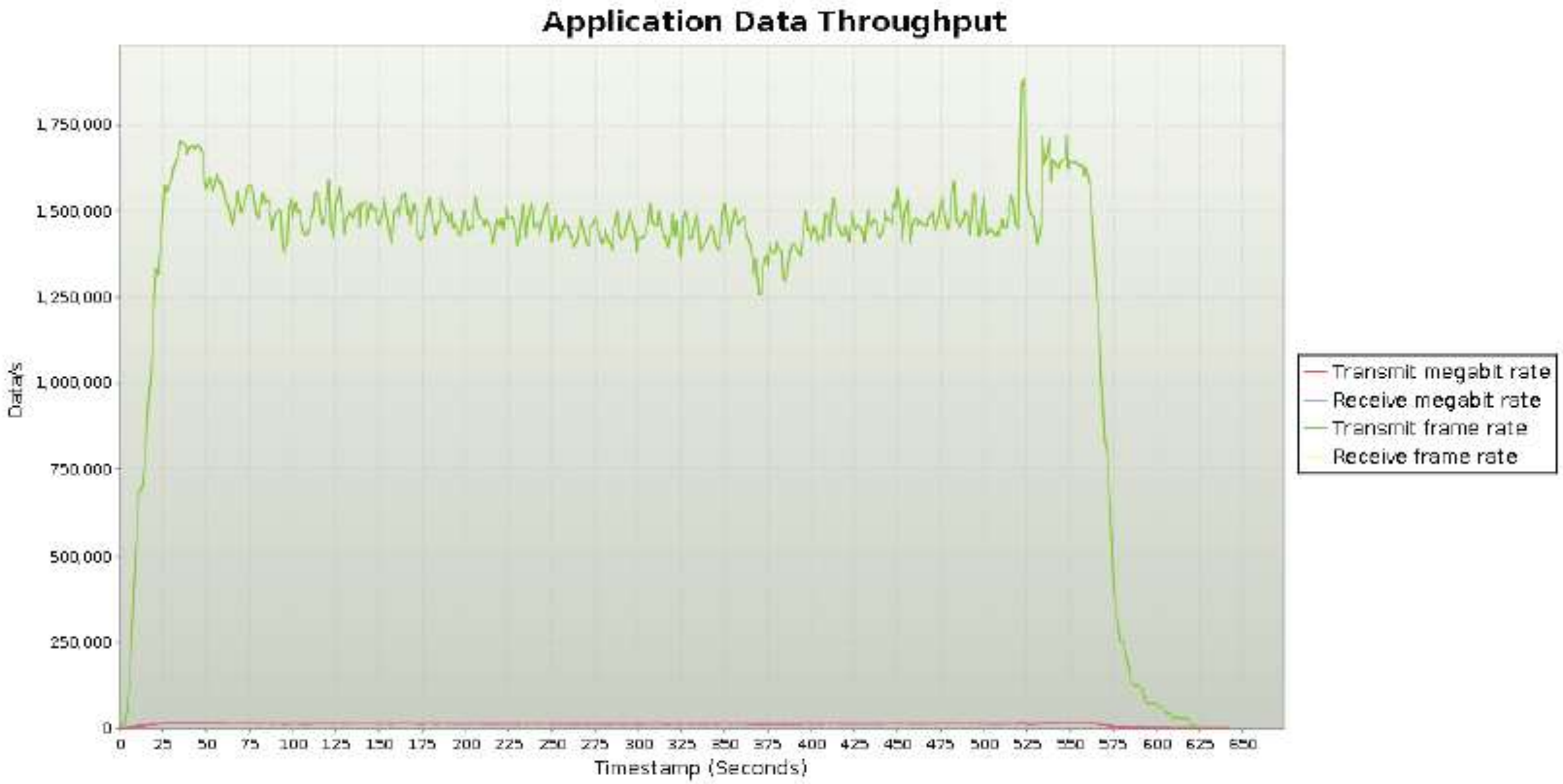
A typical network is running dozens to hundreds of unique protocols at high speeds…

- Finding attacks in a sea of real users is one of the biggest challenges of security engines



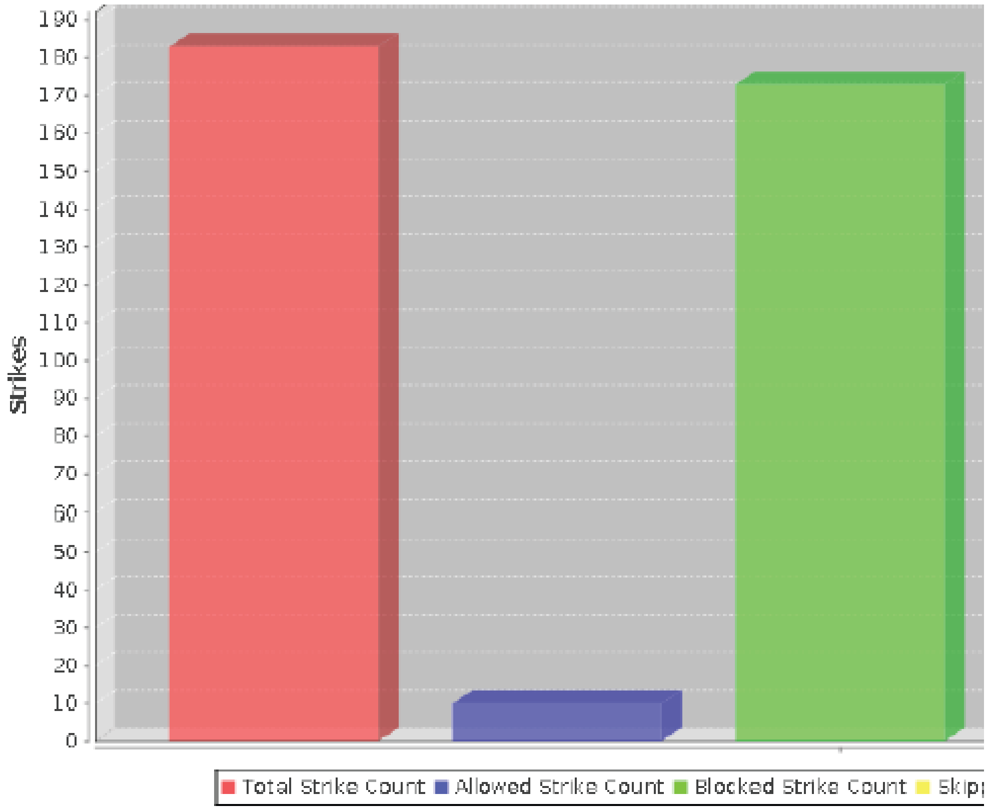
Network Traffic in 2013 at major endpoints and backbones is in the terabytes…or greater

7.21.2. Application Data Throughput



Under this kind of network load even good security engines will miss attacks

Missing 10 Attacks is 10 Attacks too many!!



Measurement	Value (Strikes)
Total Strike Count	183
Allowed Strike Count	10 5.464%
Blocked Strike Count	173 94.536%
Skipped Strike Count	0
Errored Strike Count	0

Why did the attacks get through?

- The larger the network the faster the engine must work because security devices are traffic agnostic…they cannot assume a flow is good or bad. ***EVERY*** flow must be scrutinized.

Modern applications are complex

- It's very easy to get a security engine stuck chasing it's tail when it comes to Application Layer Security **BECAUSE** a RegEx must be applied to every flow
- Even simple evasion techniques require the security engine to allocate MORE resources and limit their effectiveness and **increase latency**

Why haven't we discussed DDoS????

- Distributed Denial of Services have become an increasing issue that security engines have to account for when securing networks.
- These type of attacks are now more complex as they attack Applications and not simply throw TCP SYN packets or ICMP Echo Request packets at the target
- This problem is growing…

DDoS Effects EVERY kind of organization…

- 27% Governments
- 9% Law Enforcement Agencies
- 7% Educational Institutions
- 6% Online Games
- 6% Internet Services
- 5% Sport
- 4% News
- 4% Financial Institutions
- 4% Technology

***2012 statistics

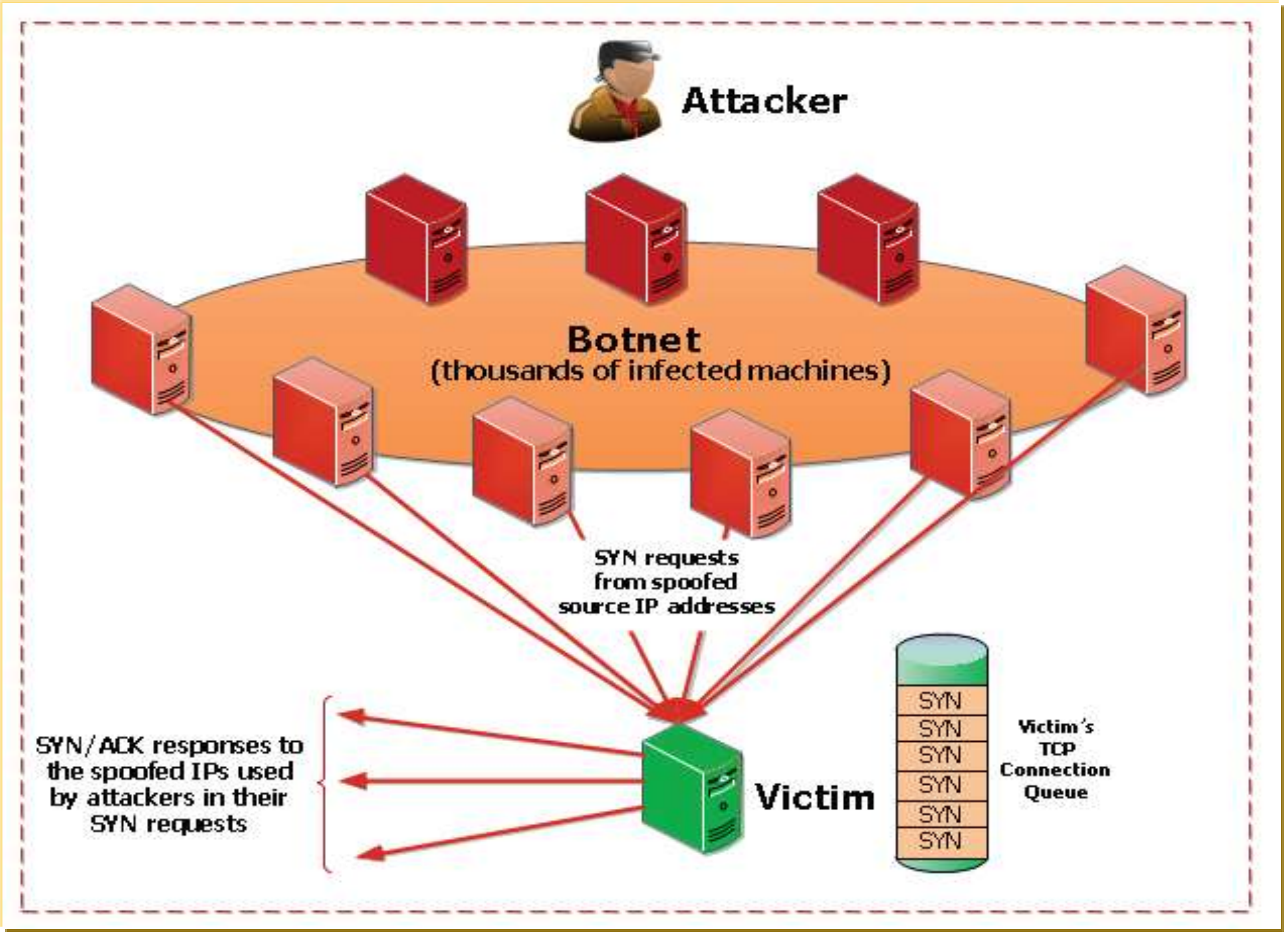
Recent targets…

- A large scale coordinated attack by Anonymous directed at Isreal took place on April 7th of this year named #OpIsrael
- Websites were defaced…
- Critical GPS information was stolen and is being held hostage…
- Attack was done very publicly

The new fad: Coordinated DDoS attacks!

- Layered attacks where there are multiple contingency plans
  - attempt attack 1, if success exploit, else
  - attempt attack 2, if success exploit, else
  - …
- DDoS can be used as a subterfuge
- Log files chocked with incidents bury the real attack

What does a DDoS Look Like?

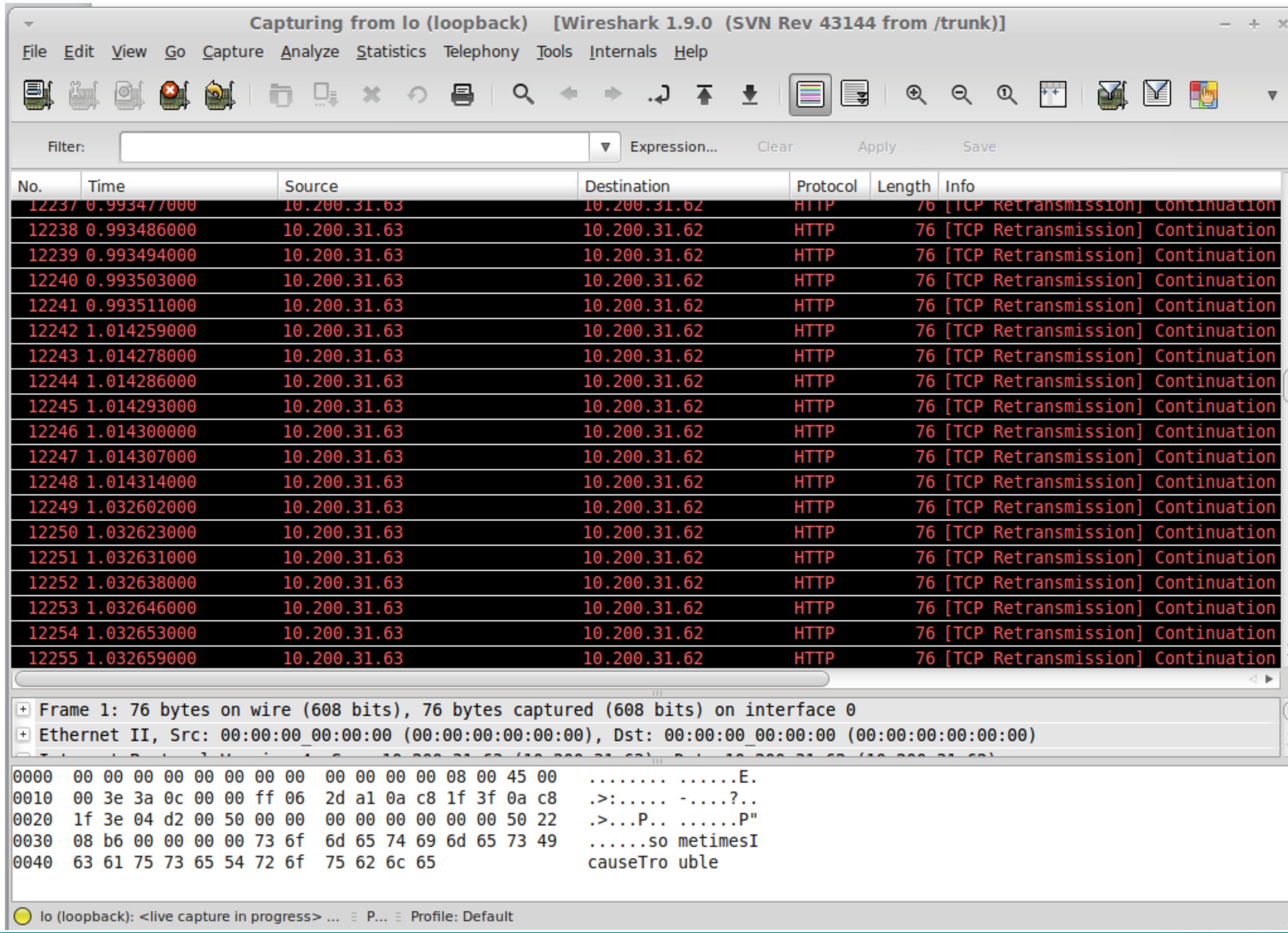


The source of DDoS Attacks?

- Individual PC's
- Some even **<u>voluntary</u>** due to hacktivism!!
- But how can individual PC's cause any damage to a well thought out network that is secured with the latest technology?

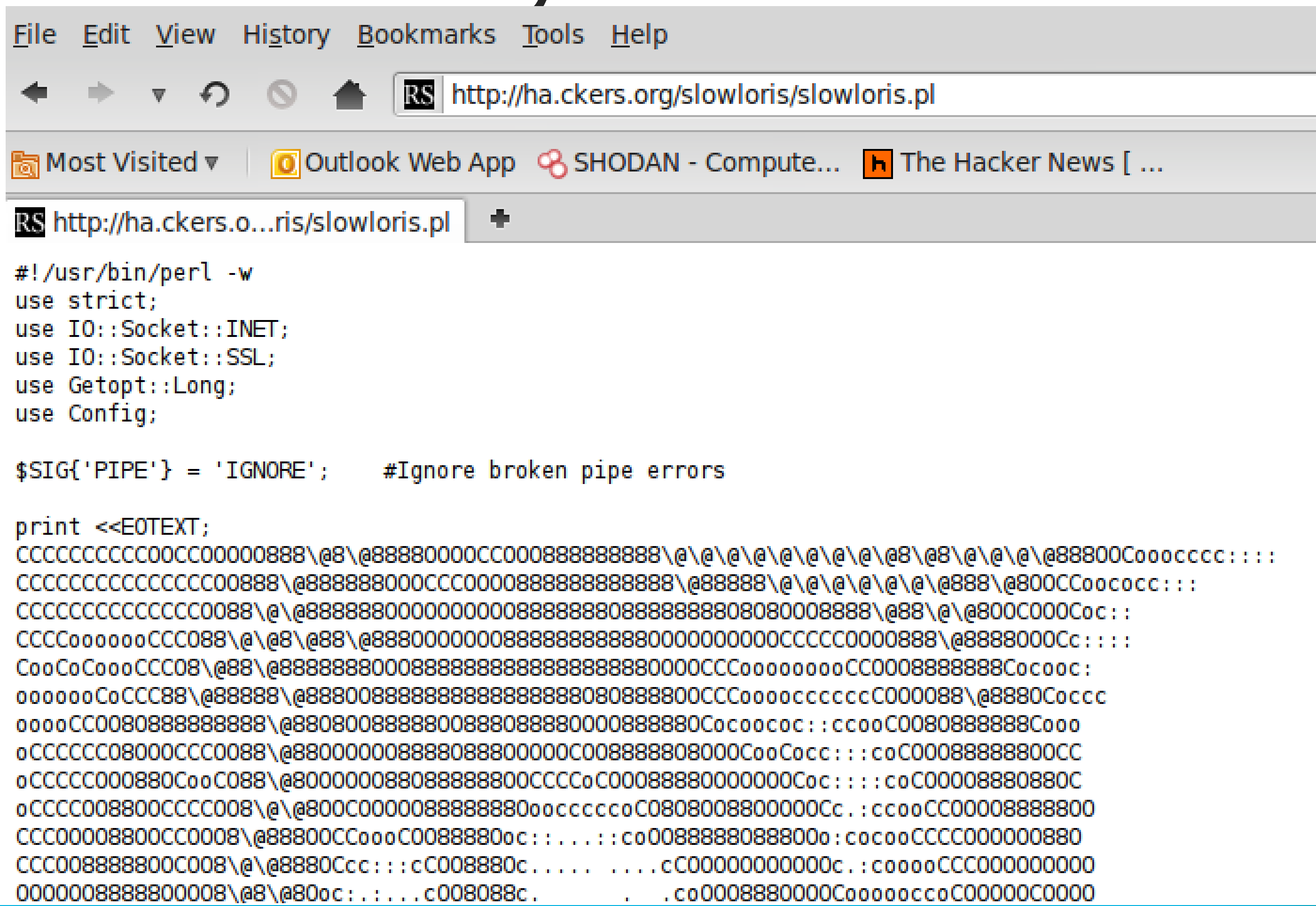
Very Easily…PC’s are more powerful than most people think…

- My 5 year old i5 laptop put out a 12K Packet Per Second DDoS
- Multiply that by even a few thousand machines…

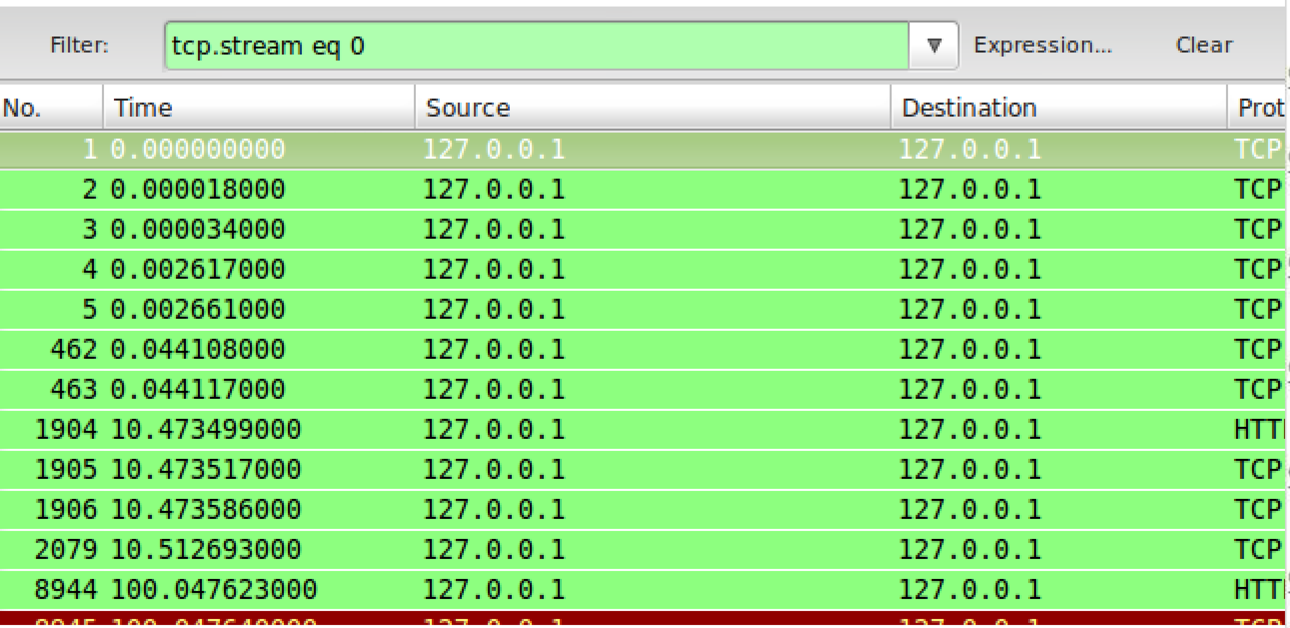


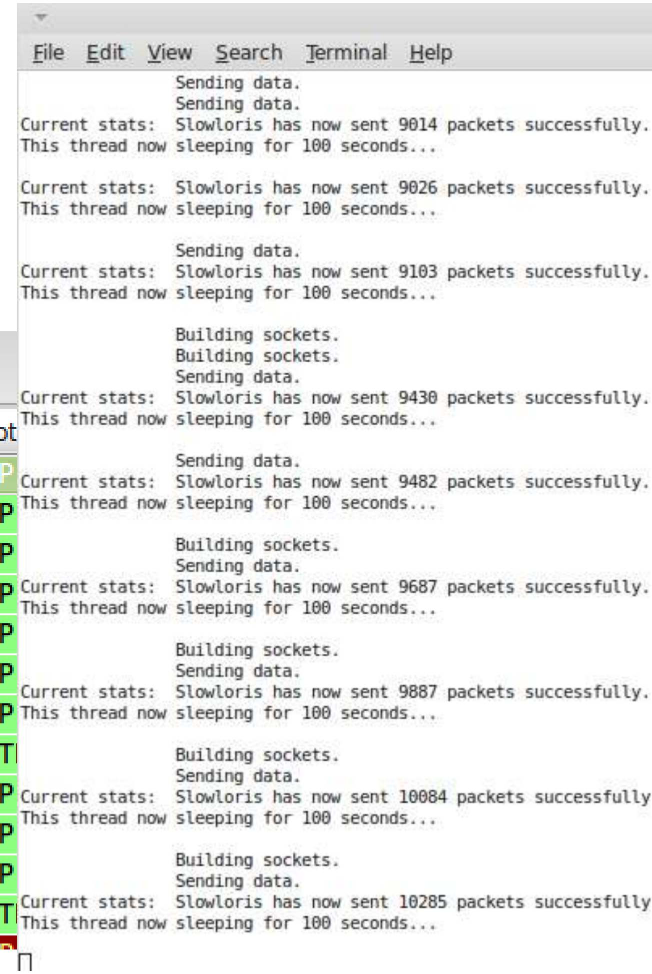
Earlier I mentioned Application Layer DDoS…

- Welcome to Slowloris - the low bandwidth, yet greedy and poisonous HTTP client



This will clutter a security log file up with 50,000 machines atttacking…



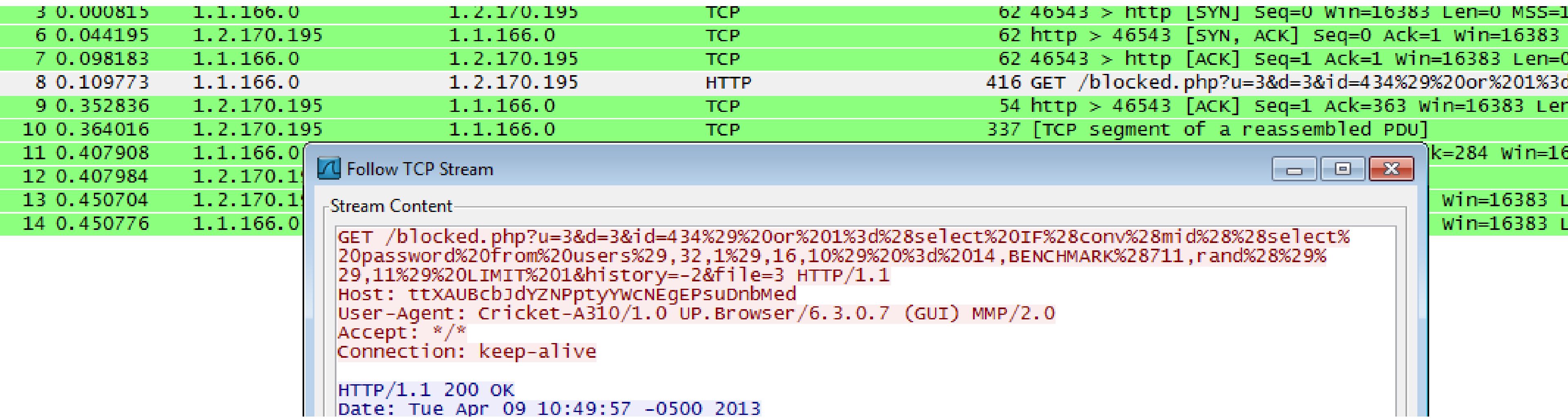


Why are DDoS Attacks especially difficult for Security Engines?

- DDoS attacks are often a subterfuge for the REAL ATTACK
- Clogging the security engine with millions of repetitive alerts for SlowLoris, High Orbit Ion Cannon, or RUDY (R U DEAD YET) paves the way for a precise and successful attack!

Like an SQL Injection!

- SQLi is often used during targeted attacks to make false entries into databases, drop tables, and steal sensitive data



So how do you build a security engine that can handle everything we just mentioned?

- You have to test the device under real conditions!
- Millions of real users running current applications!
- Network Throughput commensurate to the load that will be handled when deployed
- …then you bombard the device with real attacks and real DDoS scenarios

RSA CONFERENCE ASIA PACIFIC 2013

Thank You!
Questions?