CS 5434 Assignment 4

Due via CMS on Wednesday Nov 18th by 11:59pm

Problem 1

Consider a worm attempting to spread stealthily on an internal class B network via the following strategy. It makes scanning probes once per hour. 99% of the time, it picks a random address on the same class C network as the currently infected machine except that it ignores the network address (.0) and the broadcast address (.255). 1% of the time, it picks a random address from anywhere in the class B, except again it ignores the broadcast and network addresses of each of the 256 class C subnets that the class B is divided into.

Assume that 2% of addresses are vulnerable (all with equal likelihood). Assume also that there are 256 intrusion prevention systems (IPSs) that control access in and out of each of the class C subnets and to all the others. The IPSs attempt to prevent scanning between the subnets. There are no IPSs or other devices controlling access within each class C subnet.

a) Should the IPS for each class C subnet focus on blocking inbound or outbound probes?

b) What is the maximum average number of probes per infected IP that the IPSs can allow through, before blocking, in order that they contain the worm below the epidemic threshold?

Problem 2

Consider a network intrusion detection system that produces one false positive alert per million packets, on average. Consider this system deployed on a 10Gbps link, which is on average 60% utilized. Assume that average packet size is 700 bytes including all preamble/postamble.

a) How many false positives per week must the security operations team investigate?

b) If it’s only permissible for the IDS to produce one false report per day, how many packets must go past per false report?

[see over]

Problem 3

Consider a DDOS botnet that is controlled in the following manner. Each bot makes HTTP connections on port 80 to the command and control server (thus likely evading firewall rules at many sites). The command and control server responds with a plain text file containing lines of the following form

<pkt-type> <ip>/<net>[:<port>] <rate> <duration>

Here <pkt-type> is one of 'SYN', 'ICMP', 'GET', 'DNS', <ip>/<net> is a specification of an address range to flood, the destination <port> number is optional, and <rate> is the number of packets to flood per second to random destinations in the target range for <duration> seconds. The command language allows for multiple lines per file, with the bot carrying out all of them simultaneously (infected machine bandwidth permitting).

Write a snort rule, or set of rules, to identify these commands on the wire. Consider how to reduce false positives against the terms in this command language occurring naturally in web documents.

Problem 4

Consider this pcap:

http://www.cs.cornell.edu/courses/cs5434/2014fa/capture.pcap

Analyze the address ranges and ports occurring in it. For example, you could use tcpdump, cut, grep, sort, and uniq on the Unix/Linux command line, or some other tools of your choice. Based on your analysis, construct a set of firewall rules that would allow all the traffic in your sample, while allowing as little other traffic as practical.

For syntactic definiteness, use the rule syntax of pf as at

http://www.openbsd.org/faq/pf/filter.html

Provide enough detail of your analysis of your traffic that the TA can assess the correctness of your rules. Compact solutions that display an understanding of the traffic will receive more credit than very verbose and unrealistic rules that are auto-generated.

Problem 5

Consider the description of Blackhole v2 at http://blog.spiderlabs.com/2012/09/blackhole-exploit-kit-v2.html with a particular focus on the initial obfuscated javascript described there. Develop one or more snort signatures to detect this javascript.

Do you think this is a good strategy to detect Blackhole exploit attempts. Why or why not?

Problem 6

Take the heapspray code from http://www.thegreycorner.com/2010/01/heap-spray-exploit-tutorial-internet.html and get it to work in your browser (eg by loading an HTML file from disk). Note, not the exploit itself (which is very unlikely to work in a current browser) just the heapspray with enough initialization code to get it to run. Now vary the number of iterations in the main for() loop, and measure the size of the browser as a function of this loop count (eg using ps on a unix/Linux system) over a wide range of values. Can you determine the increments in which the browser gets memory from the OS? How much overhead is there between successive copies of the nop-sled/payload combination?