Welcome!

Our presentation will begin shortly…

Audio for today's presentation will be streamed through your computer speakers.

All participants are in listen-only mode.

SOURCEfire
Security for the real world.

Performance Rules Creation

VRT Rules Methodology

SOURCEfire
ENTERPRISE THREAT MANAGEMENT

What madness today?

- Review VRT's rule generation methodology
- Examine preprocessors
  - Why they are used
  - How they impact detection
  - Configuration of key preprocessors
- Detection engine
  - How the rules are parsed
  - Performance considerations
  - 2.8.2 Update

What is a DQOH?

Matthew Olney

(irc nick: dqoh)

VRT Security Analyst for two years

Primary Responsibilities:

- Snort rules generation
- QA for SEU and VRT rules feed
- Purveyor of bad ideas

Past life:

- Network and Security Engineer
  - Cisco
  - Snort
  - Open source security products

VRT Rules Methodology

SOURCEfire

ENTERPRISE THREAT MANAGEMENT

The Goal

Write a rule that protects against the triggering conditions of the vulnerability, rather than specific exploits.

Step 1: Research the vulnerability

For example, for an overflow:

- What data structures are involved?
- How are those structures populated?
- What checks protect those structures?
- How do you get data to those structures?
- And of course: What pointers can be overwritten?

Step 2: Modeling the protocol

How is the attack delivered?

- How is data transferred across the network?
  - What overarching protocol and port?
  - How is the data laid out in the packet?
- How do we key in to the part of the data we need to examine?
- What obfuscation and evasions options are available to an attacker?

Step 3: Identify the triggering conditions

- Combine the information from steps 1 and 2
- Make the detection as precise as possible:
  - TCP or UDP
  - Port number
  - Established connection?
  - Direction of traffic (to_server or to_client)?
  - Target the field that contains the problematic data
  - Check that the field will be processed
- Make any modifications necessary to account for false-positives or evasions.

Step 4: Testing and verification

- Analyst test
  - Once the rule is written, the analyst will test it in a limited test environment.
  - Provide final rule to Team Lead.
- Test Suite
  - Automated test cases
  - 16 million checks
  - Looking for:
    - Performance issues
    - False positives
    - False negatives

Detection Methodology Summary

- To write a successful rule we need to know:
  - How the attack affects the targeted system
  - How the data for that system is transferred across the wire
  - What the triggering conditions are
  - Rule has to be functional and not impact performance
- To really do the last point, we need to understand more about how Snort works…

SOURCEfire
Security for the real world.

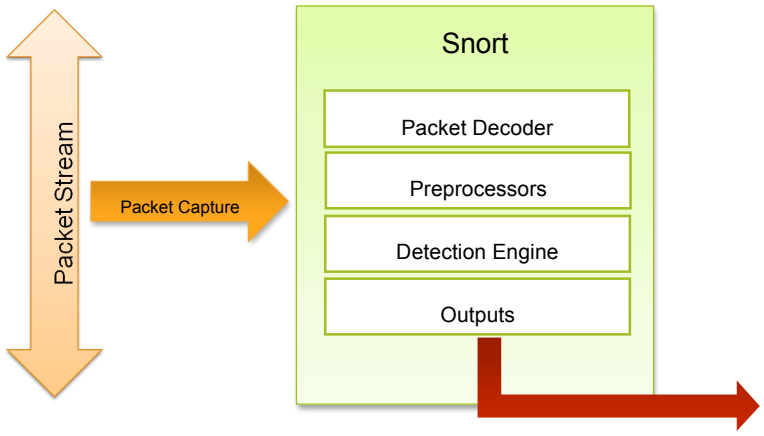
Snort Architecture
(Preprocessors)

SOURCEfire
ENTERPRISE THREAT MANAGEMENT

Why Architecture is Important

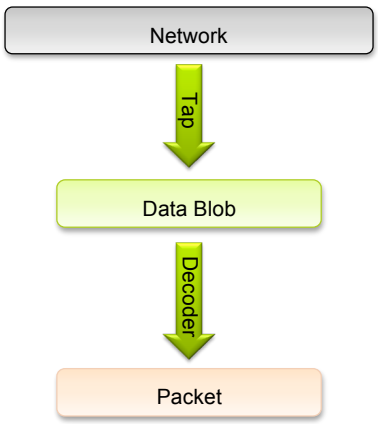
- For a given detection problem there is often more than one solution:
  - Preprocessor
  - Content
  - PCRE (or not…)
- Understanding the architecture enables us to build the correct rule to maintain performance and maximize detection.
- We need to know how to configure Snort to support our detection

Snort Basics Architecture



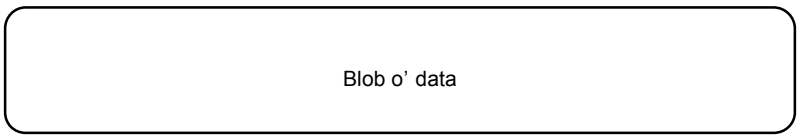
Decoder Functionality

- Decoder receives a blob of data
- Decoder adds pointers to critical data locations
  - Ethernet header
  - IP header
  - TCP header
  - Payload
- Small set of sanity checks are made here



Decoder Function

Blob of data in:



Decoded packet structure out:

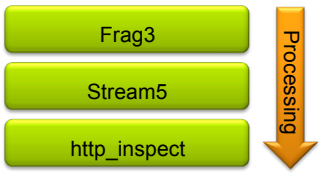
Preprocessors

- Preprocessors do one or more of the following:
- Provide detection for attacks and activity not able to be done by standard snort rules
- Provide normalization services to present data in a standardized format
- Provide reassembly services so that detection can be performed against a complete message

SOURCEfire
Security for the real world.

Preprocessors: Important Thing to Know

- Preprocessors are loaded in the order they occur in your snort.conf (or equivalent)
- Packets flow through preprocessors in the order they are loaded
- Ensure the preprocessors are loaded in a rational manner:



SOURCEfire
Security for the real world.

Frag3

- Provides target based IP defragmentation
- Reassembles fragmented packets into a pseudo-packet
- Pseduo-packet is fed back into decoder for processing
- Original fragmented packets continue through the detection sequence also
- Also provides alerts (GID:123) for certain fragmentation based attacks

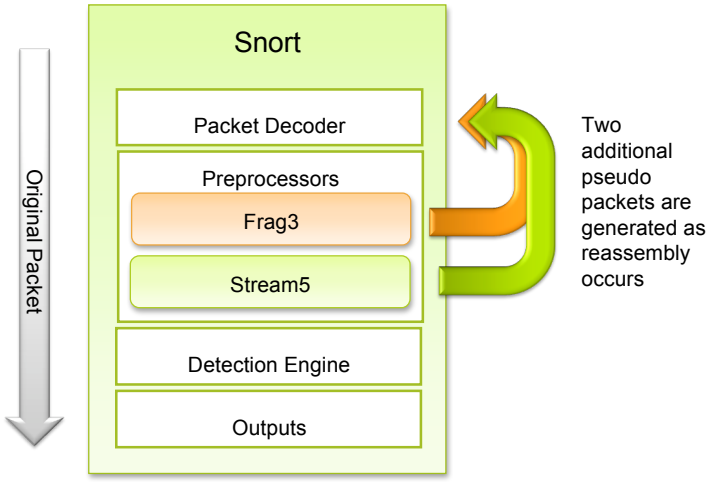
Stream5

- Provides target-based TCP reassembly
- Provides state tracking for TCP, UDP and ICMP
- Reforms TCP messages into a pseudo-packet and forwards back into the decoder for full detection
- Provides alerts for certain TCP reassembly attacks (GID: 129)

Stream5 Configuration

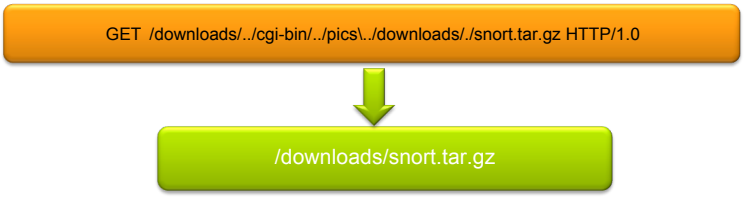
- "Ports"
  - Specifies the ports for reassembly
  - Configuration default:
    - ports client 21 23 25 42 53 80 110 111 135 136 137 139 143 445 513 514 1433 1521 2401 3306
  - Port numbers provided are always server side port numbers
  - 'client' in this case meaning 'traffic originating from the client'
  - To add port 80 client side reassembly:
    - ports both 80
    - ports server 80
- Detection impact
  - Modification to the configuration may be necessary to support additional ports or to reassemble for client-side attacks.

Amazing Snort Packet Generator!



http_inspect

- Provides normalization support for the URI
- Places the normalized version of the uri in the “URI” buffer
- Also provides alerts (GID: 119/120) for a set of evasions and attacks



http_inspect: flow_depth

- Performance tunes snort to ignore portions of the HTTP response (traffic to client only)
- Default value is to look at only the 300 bytes of the response
- Max configurable size of flow_depth is 1460
- flow_depth can cause false negatives
- flow_depth: 0 will cause snort to process the entirety of the HTTP response
- This can lead to performance issues

Detection concerns based on preprocessors

- Preprocessors can significantly impact what you see
  - Normalization
  - Truncation
  - Reassembly
- Preprocessors can also provide detection capability for certain problematic traffic

SOURCEfire
Security for the real world.

Snort Architecture
(Detection Methodology)

SOURCEfire
ENTERPRISE THREAT MANAGEMENT

Optimized Rule Evaluation

- Rules are loaded into data structures built to make Snort run as quickly as possible
- Goal is to evaluate packets only with rules with a chance to fire
- Before Snort 2.0, rules were organized into “rule chains”:
  - Chains of rules were built with common headers:
    - src IP / dst IP / src Port / dst Port
  - Packets only ran on chains with matching headers
- Snort 2.0 introduced the fast pattern matcher

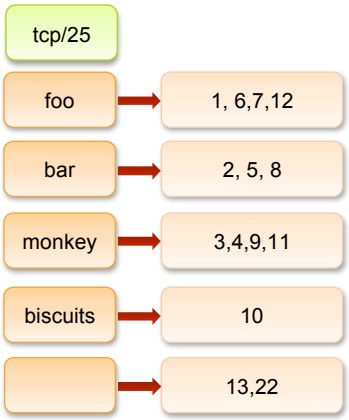
SOURCEfire
Security for the real world.

Fast Pattern Matcher

- Rules are parsed into categories based on the protocol and the destination port (port groups)
- Within any given pair, the fast pattern matcher parses the rule looking for the first, longest, non-negative content match:
  - (content:”a”; content:”bc”; content:”de”; content:!”biscuits”;)
  - Results in the fast pattern using “bc”
- Rules are only run on packets that have matching content

SOURCEfire
Security for the real world.

Fast Pattern Matching Example



- Bottom category is for rules with no content
- So…if we all send an email to the blue monkey bar we will evaluate only:
  - 2,3,4,5,8,9,11,13, 22
- Every packet sent to tcp/25 will be evaluated against the rules with no content

SOURCEfire
Security for the real world.

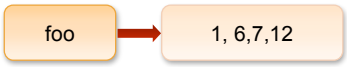
Rule Options List

- Rules are parsed into a sequence of options
- When evaluation occurs, options are checked in sequence
- When developing rules look for ways to “bail early”
- Dsize, flow and flowbit checks are fast ways to terminate rule processing
- Example options list:
  - flow: to_server, established
  - flowbits: isset, haz.biscuits
  - content: “gravy”
  - byte_test:2,>,15,relative;
- Rule header information is also checked:
  - tcp $external_net any -> $home_net any

SOURCEfire
Security for the real world.

This just in…Snort 2.8.2

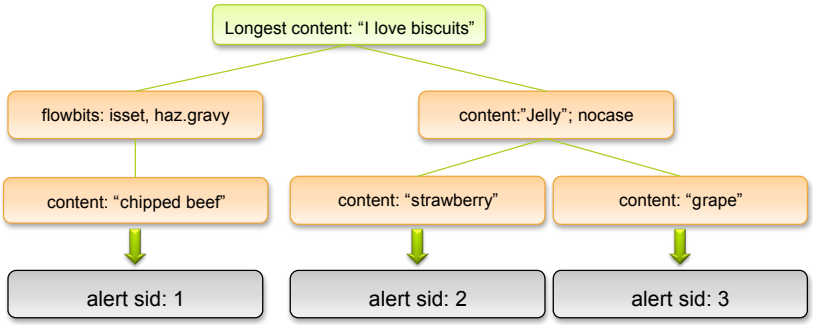
- Instead of a list of rules to process:



- Snort 2.8.2 returns a tree intended to speed through redundancies in detection between multiple rules
- During initialization, for each longest content (and for the set of rules with no content), a tree is built

SOURCEfire
Security for the real world.

If a rule alerted in a forest…



(flowbits: isset, haz.gravy; content:”I love biscuits”; content:”chipped beef”; sid: 1)
(content:”I love biscuits”; content:”Jelly”; nocase; content: “strawberry”; sid: 2)
(content:”I love biscuits”; content:”Jelly”; nocase; content: “grape”; sid: 3)

SOURCEfire
Security for the real world.

Detection criteria based on architecture

- In all cases, if possible, have a content match
- Make the match as long as possible
- Where multiple rules offer similar detection, mirror the detection for as long as possible
- Ensure that checks involving the header or stream state (for example, flow and flowbits) are done first in the rule

Questions?

- If you have questions in general:
  - snort-sigs mailing list
  - snort-users mailing list
  - #snort on freenode irc
  - research@sourcefire.com
- If you have qeustions or comments on this presentation:
  - molney@sourcefire.com

Sourcefire Commercial Products

Sourcefire 3D™ System

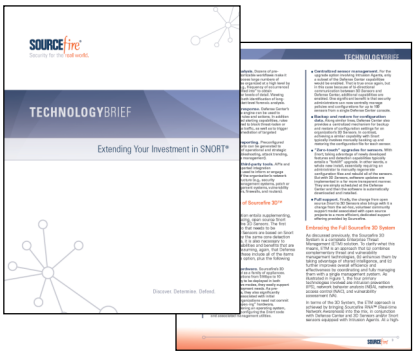
- Sourcefire 3D Sensors
  - Sourcefire IPS™
  - Sourcefire RNA™
  - Sourcefire RUA™
  - Sourcefire NetFlow Analysis
- Sourcefire Defense Center™
- Sourcefire Intrusion Agent for Snort

SOURCEfire
Security for the real world.

For More Information…

Sourcefire 3D System Flash Demo

"Extending Your Investment in Snort" Technology Brief



Available Now on Sourcefire.com

Questions?

Please submit questions via the Q&A interface in the lower-right corner of your screen.

SOURCEfire
Security for the real world.