CANSCID: Combined Architecture for Stream Categorization and Intrusion Detection

MEMOCODE Design Contest 2010

1 Overview

The objective of the 2010 MEMOCODE Hardware Software Codesign Contest is to inspect a stream of ethernet packets to see if they match against given regular expressions (deep packet inspection). Commonly, deep-packet inspection has two different main uses: stream categorization (such as L7-filter, `l7-filter.sourceforge.net`), and intrusion detection (such as snort, `www.snort.org`). The goal of this contest is is to construct a packet inspector which performs both of these tasks. Thus we name this design CANSCID: Combined Architecture for Stream Categorization and Intrusion Detection.

When a new stream is detected, a set of regular expressions is used to categorize it based on the first 3 packets (similar to L7-filter). When a stream has been identified as belonging to a certain category, a second category-specific set of regular expressions is run on the stream to scan for intrusion dectection (similar to snort). Additionally, certain patterns must be scanned against all streams, no matter their category.

Contest scoring: Regular-Expression patterns in this contest are categorized as *mandatory* or *optional*. All mandatory patterns must be supported in order for a submission to be accepted. Additionally, designs should attempt to process packets fast enough to meet a target *line rate* of 500 Mb/s. If no submitted design can meet the target rate, then the winning design will be the design which implements all the mandatory patterns at the fastest processing rate. If one or more submitted designs meets the line rate, then the winning design will be the design which implements the most optional patterns while meeting the line rate.

The base reference design includes 25 mandatory and 125 optional patterns. Teams which demonstrate they can successfully match against these patterns at the target rate may request more patterns from the contest organizers (see Section 3.1).

2 CANSCID Specification

A high-level overview of CANSCID is presented in Figure 1. The system consists of 3 main regular expression units, which are described in the following sections.

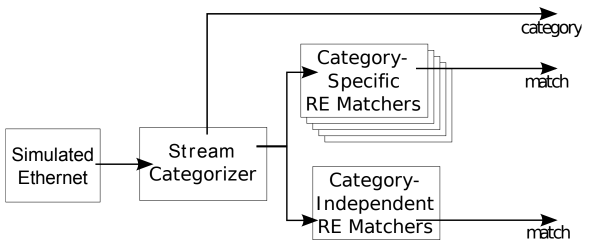
category
Category-Specific RE Matchers
match
Simulated Ethernet
Stream Categorizer
Category-Independent RE Matchers
match

Figure 1: Overview of CANSCID deep-packet inspector.

2.1 Simulated Ethernet

In order to make the design contest more accessible to a wide number of platforms, we have decided to simulate interactions with Ethernet. The Simulated Ethernet block presents the CANSCID design with a stream of packets. For the purposes of this contest assume that all packets are TCP and are delivered to the design in the correct order. Each packet is delivered to as a sequence of 32-bit flits, together with an 8-bit tag which indicates flit meaning:

```
typedef enum
{
    FLIT_SRC = 0,       // 32-bit IP address of the source
    FLIT_DST = 1 ,      // 32-bit IP address of the destination
    FLIT_PORT_PAIR = 2, // 32-bit number representing 2 16-bit port ids (src, dst)
    FLIT_FLAGS = 3,     // TCP Flags. Used only to check for the FIN flag which
                        // marks that no more packets from a stream will be seen.
    FLIT_PLEN = 4,      // The Packet length. The simulated ethernet inserts
                        // a PLEN 0 Flit to indicate end-of-packet.
    FLIT_SEQNO = 5,     // The TCP Sequence number. Unused for the contest.
    FLIT_PAYLOAD = 6    // 32-bit payload to be passed to the regex matchers.
}
PACKET_FLIT_TYPE;
```

Figure 2: Simulated Ethernet flit tag output.

The reference design includes a C++ implementation which is known to run on a Xilinx Microblaze, as well as x86 Linux, and should be portable to other platforms. Contest participants are allowed to alter the Simulated Ethernet block in order to port it to a new platform, or to implement it using a custom memory controller. However contestents are *not* allowed to alter it in such a way that breaks the abstraction of an incoming stream of packets (IE looking into the future, compressing the stream, or delivering flits in a different order). If you have any questions about if a proposed change is legal please contact the contest organizers.

2.2 Stream Categorization

When the system detects a new TCP connection (based on host IP and port, and destination IP and port) it attempts to categorize the stream based on the payloads of the first ten packets. This functionality is similar to the software package L7-filter, which also provided the regular expressions for this part of the contest. The regular expressions define patterns which may cross packet boundaries. For instance, the regular expression for detecting an SMTP mail session:

```
^220[\x09-\x0d -~]* (E?SMTP|[Ss]imple [Mm]ail)
```

With this pattern the "220" string does not necessarily need to reside in the same packet as the "Simple Mail" string for a match to occur. Therefore the design must support some scheme for storing the state of the the regular expression matchers associated with a particular stream, and swapping that state in when a packet of the stream is detected (based on host IP and port, and destination IP and port). For the purposes of this contest the design must be able to handle up to 64 simultaneous open connections.

When a stream is succesfully categorized, the system records a histogram of the number of streams seen of each type. (A real network might use this data to enforce policies such as quality-of-service, however this is beyond the scope of this contest.) Additionally, the category of each stream must also be remembered, so that it can be used for intrusion detection.

2.3 Category-Specific Intrusion Detection

When a stream has been categorized, its packets should be inspected for intrusions or other malicious behavior. This functionality is similar to the software package snort, which also provided the regular expressions for this part of the contest. For instance, here is a snort regular expression to detect an SMTP vrfy decode attempt:

```
vrfy\s+decode
```

This is simply the string “vrfy”, followed by one or more whitespace characters, followed by “decode”. Checking non-SMTP streams for this pattern would waste resources and could result in false positives (for instance, if a user loaded a webpage describing the vulnerability). Combining many regular expression parsers can result in large and inefficient automata which can limit a packet sniffer’s bandwidth.

Snort avoids this problem by hardcoding which IP addressess and ports represent SMTP servers, and only checking connections to those locations for these patterns. This approach can lead to problems—for instance if new servers are added without the configuration files being updated—or if an attacker manages to convince a computer to open an SMTP server on an unexpected port. Additionally, attack attempts which *originate* in the administered domain can be missed.

The CANSCID approach works around these limitations by using the stream categorizer described above. The categorizer employs regular expressions to identify protocols which are host- and port-independent. Once a stream has been categorized, CANSCID only needs to apply the appropriate category-specific regular expressions against that stream for the remainder of its lifetime.

When a match does occur the packet sniffer outputs a message identifying the connection destination and host, category, regular expression matched, and the offending location in the packet stream. (A real packet sniffer might have different responses based on the severity of the intrusion attempt, but this is beyond the scope of this contest.)

Note: When the stream-categorizer unit identifies a match, the packet which triggers the match should be sent to the associated category-specific checker. However there is no requirement that packets from earlier in the stream need to be buffered and checked.

2.4 Category-Independent Malicious Behavior Detection

In addition to the category-specific patterns, CANSCID includes a set of patterns that should be run on every stream, regardless of their category. These include patterns which represent things such as *shellcode* — executable code masquerading as ASCII text. Snort disables these patterns by default because of the large performance hit which they can entail. Matches against these patterns are reported in the same manner as category-specific patterns.

3 Contest Patterns and Scoring

Category	Number of Patterns	
	Mandatory	Optional
finger	1	5
ftp	1	10
http	1	10
imap	1	10
netbios	1	10
nntp	1	10
pop3	1	10
rlogin	1	5
smtp	1	10
telnet	1	10
all*	5	10
other**	0	25
total	15 + 10 categories = 25	125

* The “all” category refers to category-independent patterns that must be run on all streams.

** The “other” category refers to other protocols which the packet sniffer can identify, but does no further checking on.

Figure 3: Overview of patterns officially supported by the contest

For the contest we have identified patterns as shown in Figure 3. There are 5 mandatory patterns which must be run on each stream regardless of category. Each category then requires 1 pattern to identify streams of that protocol, and has 1 associated mandatory pattern which must be run on streams of that type. As there are 10 categories, the minimum requirement for a functioning submission is to implement 25 patterns (10 category-matching patterns, 10 category-specific patterns, and 5 category-independent patterns).

The first goal of this contest is to construct a design which can handle these 25 patterns while meeting the target line rate of 500 Mb/s. (Note that in network rate calculations Mb means 1,000,000 bits, not 2^{20} bits.) If no submission is able to meet the line rate, then the winner will be the team which acheives the fastest overall rate while implementing all mandatory patterns.

Once line rate has been acheived, there are additional optional patterns that designs can implement: 90 category-specific patterns, 10 category-independent patterns, and 25 additional categorization patterns. These optional categorization patterns have no category-specific patterns associated with them. For instance, it may be useful for a packet sniffer to identify a World-of-Warcraft stream in order to perform network-policy enforcement. However such a stream does not need to be checked for intrusion detection (beyond the checks which are applied to all streams). The submission which implements the most patterns while meeting the line rate will win the contest (assuming any submission manages to meet the line rate).

3.1 Requesting More Patterns

Once a team has demonstrated that it can implement all the base mandatory and optional patterns while meeting the line rate, they may request more patterns from the contest organizers. This process can be repeated as long as the design continues to meet the line rate. Therefore there is no upper bound on how many patterns a submission can implement. Once line rate is acheived teams should focus on implementing as many patterns as possible within the time of the contest.

4 Regular Expression Parser and DFA generator

In order to help contestants get started, the contest organizers have supplied a regular-expression parser constructed in Java using JLex. This parser outputs a separate Deterministic Finite Automaton (DFA) for each pattern, represented as either a C, Verilog, or Bluespec finite state machine. These FSMs are in no way intended to represent any kind of optimal implementation. Rather they represent a beginning approach, and a useful parser which can be adapted to the user's needs. See the README file for more information.

5 Acknowledgments

The 2010 design contest was coordinated by Joel Emer and Forrest Brewer. It was implemented by Michael Pellauer, Asif Khan, Muralidaran Vijaraghavan, Abhinav Agarwal, and Man Cheuk Ng. The idea was inspired by a class project by Sam Gross, Ben Gelb and Adam Lerer.

6 Suggested Reading

- Brodie, Cryton, and Taylor. "A Scalable Architecture for High-Throughput Regular Expression Matching." ISCA 2006.
- Bispo, Sourdis, Cardoso, and Vassiliadis. "Synthesis of Regular Expressions Targeting FPGAs: Current Status and Open Issues." ARC 2007.
- Mitra, Najjar, and Bhuyan. "Compiling PCRE to FPGA for Accelerating SNORT IDS." ANCS'07.
- Yu, Chen, Diao, Lakshman, and Katz. "Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection." ANCS'06, 2006.