High-Speed Parallel Processing of Protocol-Aware Signatures[1]

Jordi Ros-Giralt, James Ezick, Peter Szilagyi, Richard Lethin
Reservoir Labs, Inc.
{giralt, ezick, szilagyi, lethin}@reservoir.com

Summary

Intrusion Detection and Prevention (IDP) systems serve an important gate guard role in proactively preserving the integrity of computer networks under cyber attacks. IDPs are built around the concept of a signature: a boolean function S(M) that returns true if the incoming network message M is an attack and false otherwise. Thus, a main task of an IDP is to resolve a set of signatures $\{S_i\}$, each covering a specific exploit or vulnerability of the system, as fast as possible, to determine which traffic is malicious.

A key to the design of IDP systems is the idiomatic richness in which an incoming message M can be expressed. For instance, traditional IDP systems are based on simple pattern-matching/protocol-agnostic signatures, $S^t(M^t)$, that treat the input message M^t as an array of opaque bytes. Such signatures are simple to implement and can run at very high speed, but their coverage is limited and they suffer from a high number of false negatives because attackers can evade them by using polymorphic variances of the original attack message. Instead, a protocol-aware IDP system is capable of processing protocol-aware signatures. A protocol-aware signature, $S^p(M^p)$, differs from the traditional one, $S^t(M^t)$, in that its input M^p is expressed in terms of the protocol header fields of the incoming message. Such signatures are considered more powerful than the traditional protocol-agnostic ones [1] because, by being resilient to polymorphic attacks, their coverage is much higher. (In some cases providing zero false negatives.)

Efficient execution of signatures is key since IDP systems frequently operate on network trunks handling large amounts of traffic (up to 100 Gbps). The situation is further exacerbated by the fact that real deployments need to typically handle on the order of thousands of signatures. While traditional IDP solutions based on pattern-matching can be executed in parallel using hardware optimizations such as embedded DFA engines, to the best of our knowledge no equivalent solution has been produced for the processing of protocol-aware signatures on a parallelized hardware engine. For instance, [4] presents relevant results on the acceleration of protocol-aware vulnerability-based signatures based on the independent optimization of each signature's protocol parser, but it leaves the problem of single-pass parallel processing of all the signatures as open. In this paper we present a method, based in part on previously developed techniques for optimizing boolean functions for processing by Satisfiability (SAT) solvers, that exploits redundant computational elements across protocol-aware signatures to efficiently process their logic.

Exploiting Signature Redundancies

In any protocol-aware signature, $S^p(M^p)$, the number of protocol header fields that its input M^p can potentially depend on is bounded. This is unlike the traditional pattern-matching signatures, whose inputs are defined as functions of any random sequence of bytes derived from the incoming packets. Because of this bounded nature, one should expect to find redundant computational elements across the signatures. Furthermore, the number of redundancies should increase with the number of signatures loaded into the IDP system. Consider, for instance, the optimal protocol-aware signatures for the Atphttpd and GhttpdLog cyber attacks [1] shown in Figure 1. These two signatures have *bit1* as a redundant computational element.

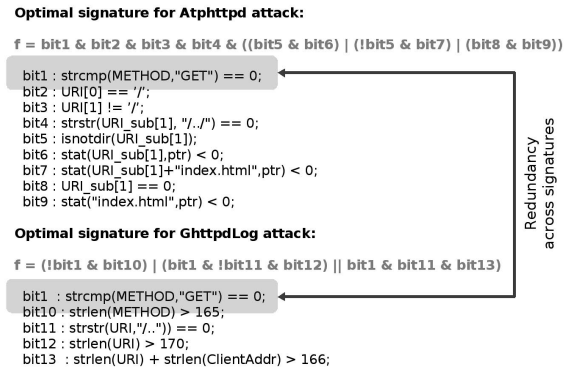

Figure 1: Protocol-aware signatures and their redundancies

The above property provides one key to the design of a signature processing engine that can scale up with the number of signatures. Our objective is to design a method that exploits such types of redundancies while allowing for the parallel processing of the signatures.

Parallelization Method

To mathematically exploit redundancies, we use the OR operator to generate a new logical signature from the set of base signatures handled by the IDP system. The OR signature can be understood as a boolean function that returns true if and only if one or more base signatures are triggered. Since the OR signature simplifies away logical redundancies across the base signatures, this provides a natural way to implement a fast path, as shown in Figure 2.

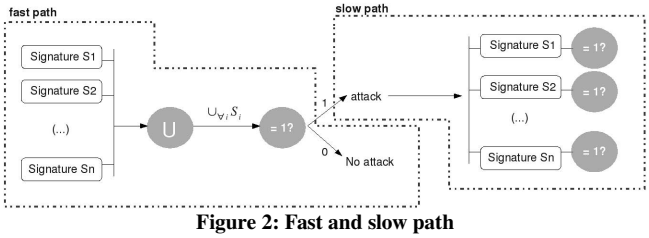

Figure 2: Fast and slow path

The fast path S_u is defined as a signature equal to the union of all the base signatures $\{S_i\}$ loaded in the system:

$$S_u = U_{\text{for all } i} S_i \quad (1)$$

1This work was funded by the US Department of Energy, contract number DE-FG02-08ER85046.

If S_u resolves to false on a specific flow F, then no signature is triggered and no further work needs to be done for F. Otherwise, at least one signature is being triggered, in which case the flow is passed on to the slow path which is programmed to resolve each signature individually.

We have devised a method to obtain hardware representations of protocol-aware fast path (OR) signatures using Binary Decision Diagrams (BDDs) with the computational elements (bit results) as the decision points. To obtain the BDD representation of a signature, we use Salt™ [2] (or Satisfiability Application Logic Translator), our constraint logic language and optimizing translation tool for SAT applications that converts Boolean functions into simplified Conjunctive Normal Form (CNF) files. The method we propose is based on the following three-step procedure: (1) Use Salt to generate a simplified CNF representation for each signature, (2) convert each CNF expression into a BDD and (3) generate the fast path by OR'ing all the BDDs.

Consider as an example the signatures in Figure 1. We start by first composing a Salt representation of each signature; by running the Salt translator we then obtain a simplified CNF representation for each signature (Figure 3).

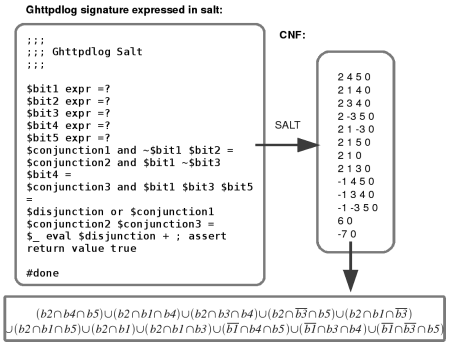

Figure 3: Salt/CNF representations of the Ghttpdlog attack

Using a standard BDD software suite, from each CNF we obtain a BDD graph describing the logic of each signature, as shown in Figure 4. (In our illustrations we indicate a true or false transition in the BDDs with a continuous or discontinuous edge, respectively.)

Each BDD corresponds to a DFA implementation of each signature. The final step involves the simplification of nodes that appear multiple times in the fast path (OR) BDD, since such nodes are a manifestation of redundancies across signatures. In our example, node 1 appears twice in the fast path BDD and therefore the second instance can be eliminated (Figure 5). This simplification step effectively eliminates *bit1* as a redundant element in the original signature pair (as displayed in Figure 1). In practice, this optimization is part of the standard BDD package OR operation. The end result of the above procedure is a BDD specification of the fast path that accounts for simplifications of redundancies across signatures. Since BDDs can be represented as DFAs, this provides a natural method for fully offloading the signature processing from the core processors.

The Salt tool has built-in operator support for the bit manipulations that occur in the signatures and handles optimization of the boolean functions, including detecting and optimizing certain logical implications. The clauses emitted as the CNF representation act as the building blocks for constructing the BDDs. Since the clauses can be arbitrarily partitioned and since each partition can be transformed to a parallel BDD, a single signature or signature set can be divided and balanced among an arbitrary number of DFA engines. The BDD package then provides optimized union (OR) that is faster to generate than the product-space automata generated by OR'ing DFAs. This apparent "free lunch" is due to the fact that the BDDs interpreted as DFAs are acyclic.

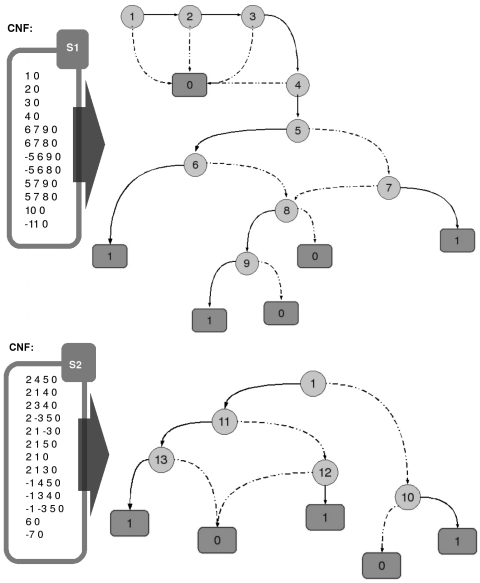

Figure 4: Generation of the signature's BDDs

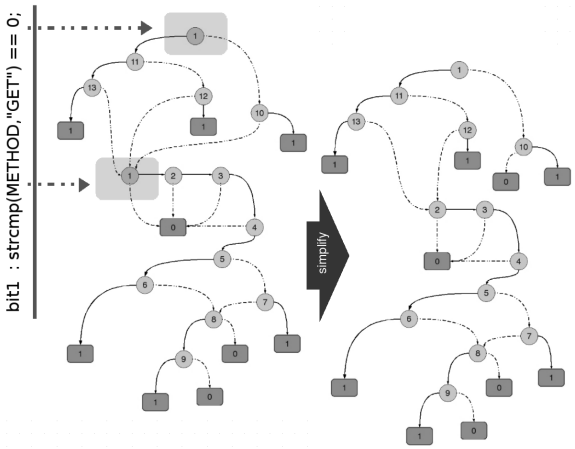

Figure 5: Generation of the fast path signature's BDD

References

[1] J. Caballero, Z. Liang, P. Poosankam, D. Song, "Towards Generating High Coverage Vulnerability-based Signatures with Protocol-Level Constraint-guided Exploration", 2008.

[2] J. Ezick, "Salt™ 1.5 Closing the Programming Gap for Boolean Satisfiability Solvers," Reservoir Report, 2007.

[3] R. Pang, V. Paxson, R. Sommer and L. Peterson, "BinPAC: a yacc for Writing Application Protocol Parsers," Proceedings of ACM Internet Measurement Conference, October, 2006.

[4] N. Schear, D. R. Albrecht, N. Borisov, "High-speed Matching of Vulnerability Signatures," Symposium on Recent Advances in Intrusion Detection, 2008.