1 Problem Statement

Distributed systems are difficult to build, verify and extend. Part of the problem is the gap between specification and implementation: systems that may be specified very concisely often require implementations that are several orders of magnitude larger than the specification text in LOC (lines of code). Hence the burden of manually comparing the specification to the program text is too large to be carried out with confidence. Worse still, the imperative languages that are often used for implementations are not amenable to automatic verification, due to general issues like the state explosion problem and specific semantic issues like pointer aliasing.

These issues are worrisome enough for any distributed system, but particularly for secure systems. In general, we can partition the correctness properties of a distributed system into two principal classes: safety properties and liveness properties. Informally, a safety property specifies that certain bad things never happen, while a liveness property states that certain good things must eventually happen. Security properties will usually represent a subset of both partitions. For example, a security safety property might state that a system will never pass unchecked, externally-provided parameters to a vulnerable subroutine, while a liveness property might state that the system must provide a request to a given response. The former property, if it holds, protects against command injection attacks, while the latter ensures that the system is robust to denial of service attacks.

We propose a new high level logic language called Doughface, based on Datalog. Like its ancestor distributed logic languages, Doughface is based on a subset of first-order logic, lacks control structures like loops and conditionals, and avoids complex data structures and pointers in favor of a flat, relational representation of program state. This makes it extremely easy to specify safety properties as assertions over state, without concern for the evaluation order of the program. Moreover, its explicit dataflow and metaprogramming capabilities make applications like taint tracking straightforward, in both the (conservative) static and dynamic cases. Unlike previous languages, Doughface includes language extensions that explicitly deal with time and the details of state per-
sistence over time. This enables the assertion of liveness properties that were
difficult or impossible to express in those languages.

2 Evaluation

Doughface is largely a theoretical effort: while it can be transformed into an
Overlog program via a rewrite and evaluated using that interpreter, some of its
clean semantics are lost in translation. Our evaluation of Doughface will begin
with proofs of the language’s equivalence to Datalog-N with extent in time, and
with algorithms for transforming an arbitrary Overlog program into a Dough-
face program with unambiguous semantics. With this foundation established,
we’ll provide examples of Doughface applications, and demonstrate safety and
liveness proofs that ensure that critical security properties, including protection
from command or SQL injection attacks, and robustness to denial of service
attacks.

3 Related Work

Doughface is based on Overlog [4], a distributed logic language developed at
Berkeley and originally intended for declarative specification of network proto-
cols. Overlog itself is based on Datalog [5], a recursive database query language.
Recently, we published a paper that describes our experience implementing the
Paxos consensus protocol [3] in Overlog, and reasoned about the appropriate-
ness of the language to capturing safety and liveness properties in distributed
systems. Taint tracking in imperative languages has a long history in the liter-
ature [2, 6, 1], though our language sidesteps many of the difficult issues raised
in that work.

References

ACM Workshop on Secure Web Services, November 2009.
tical approach to defeat a wide range of attacks. In 15th USENIX Security