GLOWS: A High Fidelity Worm Simulator

Shad Stafford, Jun Li, Toby Ehrenkranz, and Paul Knickerbocker
Department of Computer Science
University of Oregon
{staffors, lijun, tehrenkr, pknicker}@cs.uoregon.edu

***Abstract*—This work presents our GLOWS (Gateway Level Oregon Worm Simulator) simulator, designed to produce realistic worm traffic over a broad range of scenarios. GLOWS simulates the spread of a worm across the Internet and its propagation into a single domain with the goal of capturing the worm traffic that crosses the gateway point separating the monitored domain from the Internet.**

I. Introduction

A key element in evaluating the effectiveness of network level worm detectors is running them against worm traffic and measuring their ability to detect it. To do so is tricky however. You must have a way to release real worms in a controlled environment, a recording of real worm traffic, or realistically simulated worm traffic. The DETER Testbed [1] is a real step towards the first solution, but it lacks realistic background traffic and requires a heavy investment of resources to use. Recordings of worm traffic are problematic because they are difficult to come by for a broad range of worm propagation techniques. This leaves the final option, simulated worm traffic, as the only reasonable solution. This work presents our GLOWS (Gateway Level Oregon Worm Simulator) simulator, designed to produce realistic worm traffic over a broad range of scenarios. GLOWS simulates the spread of a worm across the Internet and its propagation into a single domain with the goal of capturing the worm traffic that crosses the gateway point separating the monitored domain from the Internet.

II. How GLOWS Works

GLOWS is designed to simulate the outbreak of a worm in an individual administrative domain such as that of a university or corporation. It uses a finite-state model to simulate the behavior of each vulnerable host in both the Internet and the internal network. This provides a greater degree of accuracy than probabilistic simulators, particularly for worms such as the topological scanning worm which must know of other hosts running the service. Modeling each vulnerable host is feasible for a number of reasons including: the number of such hosts is only a small fraction of the total hosts in the Internet, we don't model packet level interactions we only generate flow-level connections, and we don't model congestion effects or background traffic within the simulator. GLOWS loses a small level of accuracy due to ignoring the effects of congestion, but if experiments are focused at the beginning of the infection cycle where relatively few hosts are infected, this impact is minimal.

To support studying a broad variety of worm scenarios, GLOWS supports the following parameters:

- *Connection type.* Connections can be either TCP-based or UDP-based.
- *Scan speed.* The speed at which the worm scans can be varied.
- *Scan type.* Using previously presented worm studies by Staniford *et al.* [2], we were able to model the following scanning types: random-scanning, sequential-scanning, permutation-scanning, partition-scanning, local-preference-scanning, and topological-scanning

The various scanning techniques work as follows: *random-scanning* worms are the simplest as they simply choose each new target address completely randomly. *Sequential-scanning* worms choose a random address to start from and scan sequentially from there. *Permutation-scanning* worms permute the entire address space and sequentially

This research is partially supported by a research grant from Intel Corporation.

scan that. *Partition-scanning* worms partition the address space and scan only their partition. *Local-preference* worms choose target addresses randomly, but with a preference towards choosing from the local subnet: they choose an address from their host's class B address space with a 50% probability, from its class A address space with a 25% probability, and from the Internet as a whole with a 25% probability. And finally the *topological-scanning* worm starts with a list of roughly 500 addresses known to be running the target service, though not necessarily vulnerable. Once these have been contacted, it reverts to a pure random-scanning worm.

GLOWS can model the network topology of the monitored domain based on host information taken from a network trace, allowing it to create realistic worm traffic for a given monitored domain. Addresses where a host is active in the internal network are derived from detected host activity in the real trace, and external hosts are probabilistically allocated. Not all hosts run the service that the worm is attacking; we assign service runners probabilistically in both the internal and external network.

GLOWS accurately models the connection-level interaction between two hosts during an infection attempt, right down to setting appropriate TCP flags. Scanning attempts to non existent hosts result in SYN/RST exchanges as one would expect to see in the real world, and the worm client can disconnect at any point leaving the connection in an arbitrary state. The payload and target port number are also configurable.

III. Implementation Details

GLOWS is implemented as a Java program comprised of 20 classes and approximately 4000 lines of code. The internal network and the external network (*i.e.*, the rest of the Internet) are implemented in separate classes allowing the internal network to be modeled more accurately and the external network to use a less accurate but higher speed implementation. There are four possible states for every IP address: no host, host, host running service, and host running vulnerable service.

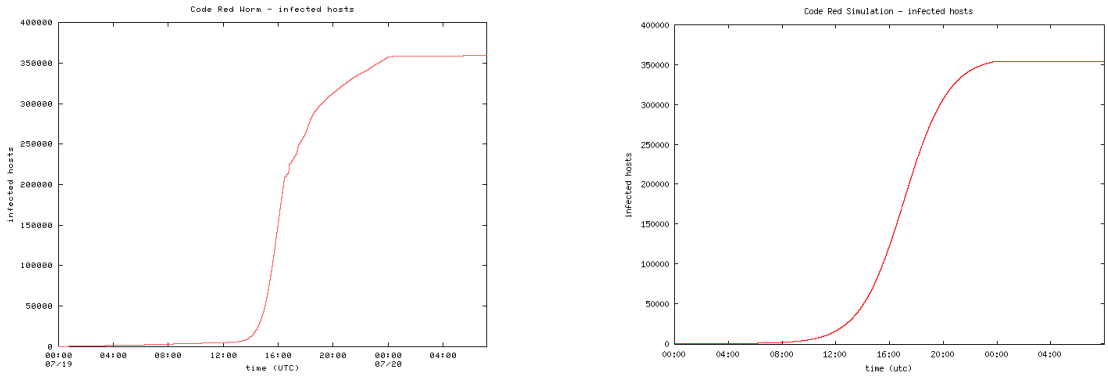
For the addresses in the internal network, we load the list of hosts from a configuration file produced by analysis of the real network trace that the internal network is modeled after. Service runners and vulnerable hosts are chosen randomly from within this set. Every host in the internal network is an instance of our ActiveHost class, allowing each host to respond independently to connection requests.

For the external network, the sets of vulnerable addresses and service runner addresses are generated by a random number generator and are stored as simple lists of 32-bit integers to conserve memory. A given address in the Internets is determined to have a host or not by the following procedure. First check the list of vulnerable addresses, then the list of service runner addresses; if it is present in neither one of these lists, it may still have an active host. Whether or not this address has a host is randomly determined, but must be consistent throughout a given simulation: a random number is generated after seeding the random number generator with the address in question plus a simulation specific salt value. If the random number is below the configured value for the percent of hosts in the external network, then this address has an active host. This process ensures that throughout a simulation the set of addresses with hosts is consistent, but for multiple simulations we will get different sets of addresses with hosts. It can be performed efficiently through the use of a Mersenne Twister [4] implementation as our random number generator, which has excellent performance and very low overhead for setting the seed.

Interactions with hosts in the external network are resolved by the state of the address except that when a vulnerable address is infected an ActiveHost instance is created for that address to respond to future interactions.

At each time tick, the ActiveHost instances in both the internal and external networks are polled for activity. Those that are infected and actively sending worm scans will generate their next scan based on the worm implementation currently active. Because each ActiveHost has its own instance of the worm, individual hosts can have unique neighbor lists, network partitions, or other host specific info. The worm implementation is pluggable, allowing us to simulate a number of different scanning and propagation schemes.

At the end of the time tick all of the worm scans



(a) Measured (from [3])

(b) Simulated

Fig. 1. A comparison of the spread of the measured and simulated Code Red worms showing the cumulative count of infected IP addresses.

are processed. There is a complex set of possible results for each connection attempt, as the worm implementation can specify all of the connection attributes and TCP flags for each kind of connection: to a dark address, to a non-service running host, to a service running host, to a vulnerable service running host, and to an already infected host. Connections from internal to internal hosts or external to external hosts are first processed, and then those connections that cross the gateway are processed. The complete set of attributes for connections that cross the gateway are logged to a file.

IV. Simulator Performance

The speed with which a complete simulation can be processed is of course directly correlated with the number of vulnerable hosts and the rate at which hosts make connections. The following performance numbers are measured when running simulations on a 1.73 GHz Pentium M Laptop with 512 MB of RAM.

A basic scenario for our experiments is the high-speed (100 connections per second) random scanning worm with a vulnerable population of $300,000$ hosts, of which $3,000$ start out as infected. We run this simulation until $100,000$ worm connections have crossed the gateway (roughly 160 seconds of simulated time), and it takes approximately 24 minutes to complete.

This contrasts with the same scenario but run for a low-speed (1 connection per second) worm where our simulation terminates after $3,600$ seconds of simulated time, and only $4,500$ connections crossing the gateway, but which runs in just over 1 minute.

TABLE I
Code Red Simulation Parameters

Parameter	Values
Vulnerable Hosts	360,000
Initial Infected Hosts	3
Connection Rate	2 conns/sec
Scanning Strategy	Random Scan

Larger simulations take correspondingly longer to run. Our simulation of the Code Red v2 worm on the same hardware took 56 hours when run until 99% of the vulnerable population of $360,000$ hosts were infected. This simulation generated more than $12,901,600$ connections which crossed the gateway (and of course substantially more that didn't) over more than 24 hours of simulated time.

V. Simulator Validation

Of course a simulator like GLOWS is not particularly useful if it does not accurately simulate what would happen in a worm outbreak. To validate GLOWS we attempt to simulate a known worm and compare the propagation of the worm in the simulator with the measured propagation of the worm in the real world.

In figures 1(a) and 1(b) we compare the spread of a simulated version of the Code Red worm with its spread during the real world outbreak as measured by Moore *et al.* in [3]. The simulation was run with propagation and vulnerable population parameters similar to the real Code Red v2 Worm (see Table I). In both curves we see a similar basic shape with the

inflection point occurring around 16 hours after the start of the outbreak. The curves begin to diverge after hour 18 most likely due to the impacts of both patching infected systems and network wide congestion. Our experiments are focused around the very early outbreak of the worm, however, where our simulation appears to be quite accurate.

VI. Conclusion

In this paper we have introduced our GLOWS worm simulator, providing a platform capable of capturing simulated worm traffic as it would appear to a gateway router for an administrative domain. GLOWS has been shown to simulate the Code Red v2 worm with high accuracy during the initial phase of the infection and can be configured to simulate a variety of worm types. By properly configuring the network properties and host-list of the domain to be simulated, the resulting worm traffic can be integrated with an actual traffic trace from the same domain yielding a combined trace that contains real background traffic with a realistic worm outbreak. This combined trace is a critical tool in the analysis of the effectiveness of network level worm detectors.

References

[1] "Deter: A laboratory for security research," http://www.isi.edu/deter/.

[2] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the Internet in your spare time," in *Proc. USENIX Security Symposium*, August 2002.

[3] D. Moore, C. Shannon, and kc claffy, "Code-Red: A case study on the spread and victims of an Internet worm," in *Proc. ACM Internet Measurement Workshop*, 2002.

[4] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," in *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, 1998, pp. 3–30.