Real-time Detection of Known and Unknown Worms

Sumeet Singh, Cristian Estan, George Varghese, Stefan Savage
University of California, San Diego
{susingh,cestan,varghese,savage}@cs.ucsd.edu

Abstract— **Worms are a major threat to the security and reliability of today's networks. Because they can spread rapidly from computer to computer, to effectively contain them we need automated methods to very quickly identify and filter new worms before they grow into a massive epidemic. In this paper we propose such an automated approach based on identifying in real time the traffic characteristics common to all worms: highly repetitive packet content, going from an increasing number of infected hosts to very many random IP addresses of potential new victims. Our preliminary results on a small network show that our automated approach of identifying new worms is promising: it identified three confirmed worms with an encouragingly low percentage of false positives when configured with good parameters.**

I. Introduction

Current worm detection technology is both *retroactive* (i.e., only after a new worm is first detected and analyzed by a human, a process that can take days, can the containment process be initiated) and *manual* (i.e., requires human intervention to identify the signature of a new worm). Such technology is exemplified by say the Code Red and Slammer outbreaks for which it took days of human effort to identify the worms, followed by containment strategies in the form of turning off ports, applying patches, and doing signature based filtering in routers and intrusion detection systems.

The difficulties with these current technologies are:

1, Slow Response There is a proverb that talks about locking the stable door after the horse has escaped. Current technologies fit this paradigm because by the time the worm containment strategies are initiated, the worm has already infected much of the network. In fact, [1] shows that unless containment strategies are initiated in around 30 seconds (as opposed to days), the worm will spread relentlessly.

2, Constant Effort Every new worm requires a major amount of human work to identify, post advisories, and finally take action to contain the worm. Unfortunately, all evidence seems to indicate that there is no shortage of new exploits and worse, simple binary rewriting and other modifications of existing attacks, can get around simple signature based blocking (e.g., as used by Snort).

Thus there is a pressing need for a new worm detection and containment strategy that is real-time (and hence can contain the worm before it can infect a significant fraction of the network) and is able to deal with new worms with a minimum of human intervention (some human intervention is probably unavoidable to at least catalog detected worms, do forensics, and fine-tune automatic mechanisms).

Thus the goals of a good automatic worm detection and containment strategy are (roughly in order of importance):

- *Real-time Detection:* The detection system should detect the worm in seconds, and potentially initiate some containment strategy in a similar time frame. Clearly, the reason for this is to prevent widespread infection before it is too late [1].
- *Content Agnosticism:* The detection system should not rely on external, manually supplied input of worm signatures. Instead, the system should *automatically* extract worm signatures even for new worms that may arise in the future. Clearly, the reason for this is to reduce the constant human and other overhead involved in detecting a new worm.
- *Versatile Deployment:* The scheme should be deployable at any point in the network including routers (edge and core), Intrusion Detection systems running on local area networks and even on end systems themselves.
- *Low Memory Usage:* The scheme should ideally use a low amount of memory especially in incarnations deployed at routers, both core and edge with emphasis on core routers. This is because high speed memory on such devices is expensive, and also because it may be important to communicate detection state across multiple participants and low memory algorithms into faster information exchange times. In particular, we wish to avoid keeping per-flow state at routers.

- *Small Processing per packet:* There have been recent market studies that have shown the importance of doing real-time intrusion detection at 10 Gbps and faster. Ideally, the real-time detection system should be deployable (especially at routers, but also at IDS on fast gigabit LANS) and keep up with data rates. (In particular, string matching schemes as exemplified by Snort, may be hard to do at such speeds, and they do not even provide our goal of not requiring manual input).

The following secondary goals may also be desirable:

- *Public Knowledge of Basic Scheme:* While some deployments may use secret heuristics, and ideal containment scheme against new worms should be public (as with public key encryption scheme) so that others can make improvements and find weaknesses that can be eliminated in future versions of the design.
- *Resilience to Simple Worm changes:* We can expect no system to provide absolute safety (even public key encryption after years of failed attacks falls into this category), but one should be able to argue that the worm detection strategy is resilient to simple countermeasures adopted by worm authors.
- *Ability to Handle Asymmetric Routing:* While in many places (e.g., end systems) one can observe both directions of a traffic flow, it would be desirable to have a system that could be deployed at say a core router where due to asymmetric routing the deployment point can only observe one direction of a traffic flow. (50% of Internet routes in 1999 were asymmetrical.)
- *No use of active probing:* Ideally, the containment system should passively monitor the traffic (at least for detection purposes) as opposed to doing active probing of addresses. For instance, if an IDS system sent challenge queries to various IP addresses to distinguish their traffic from malicious traffic, this extra traffic could bother network managers and could cause further problems (such as extra strain on the network in busy times, triggering other IDS's, etc.). However, a small amount of traffic between cooperating agents of the IDS may be reasonable. Clearly, the containment stage may also require some traffic to be sent by the IDS at least to a manager.

II. The Scheme

We define a worm to have the following abstract features which are indeed discernible in all the worms we know, even ones with such varying features as Code Red (massive payload, uses TCP, and attacks on the well known HTTP port) and MS SQL Slammer (minimal payload, uses UDP, and attacks on the lesser known MS SQL port).

- *F1, Large Volume of Identical Traffic:* These worms have the property that at least at an intermediate stage (after an initial priming period but before full infection) the volume of traffic (aggregated across all sources and destinations) carrying the worm is a significant fraction of the network bandwidth.
- *F2, Rising Infection Levels:* The number of infected sources participating in the attack steadily increases.
- *F3, Random Probing:* An infected source spreads infection by attempting to communicate to random IP addresses at a fixed port to probe for vulnerable services.

Thus we propose the following detection strategy that automatically detects each of these abstract features with low memory, small amounts of processing, works with asymmetric flows, does not use active probing. We can also make a simple argument that our scheme is resilient to changes by worm authors who know the system in that such changes would require them to invent an attack that differs from a worm in at least one of the abstract categories defined above.

The high-level mechanisms[1] we propose to detect each of these features are:

- *M1, Identifying Large Flows in real time with small amounts of memory :* [2], [3] describe mechanisms to identify flows with large traffic volumes for any definition of a flow (e.g., sources, destinations). A simple twist on this definition is to realize that the content of a packet (or more efficiently, a hash of the content) can be a valid flow identifier, which by prior work, can identify in real-time (and with low memory) a high volume of repeated content. An even more specific idea (that distinguished worms from valid traffic such as peer-to peer) is to compute a hash based on the content as well as the destination port (that remains invariant for a worm).
- *M2, Counting the number of sources :* [4], [5] describe mechanisms using simple bitmaps of small size to estimate the number of sources on a link with small amounts of memory and processing. These mechanisms can be used easily to count sources corresponding to high traffic volumes identified in the previous mechanism.

[1]Each of these mechanisms needs to be modulated to handle some special cases, but we prefer to present the main idea untarnished with extraneous details.

- *M3, Determining Random Probing by counting the number of connections attempts to unused portions of the IP address :* The prefixes from the BOGON list cover unused portions of the address space. A large number of packets sent to addresses in this list are indications of random probing. Note that these addresses are not aside for the IDS system and are, in effect, "shared" across all such IDS system, of which there can potentially be one on every network.[2]

III. Preliminary Evaluation

A. The setup

We have built the system based on the scheme detailed in section II. The system is positioned to look at all traffic flowing in and out of our Local Area Network. The Local Network is comprised of 7 hosts (4 Linux, 2 WinXP, 1 Win2K). Though this setup may not be considered representative of most real-time workloads, it provides us with a good start to help evaluate the high-level mechanisms discussed in section II.

B. Generating Signatures from Packets

As our desire is to monitor the flow of data content between various communicating hosts, our first goal is to generate signatures from the data contained in the packets. We have adopted and are evaluating 3 different mechanisms for generating signatures representative of the data payload contained in the packets.

1) Utilizing the pre-computed *TCP-CHECKSUM* available in the packet header.
2) Computing a *CRC* from the data payload contained in the packet.
3) Computing Incremental *Rabin Fingerprints* from the data payload contained in the packet.

In all three cases the generated signature is augmented with the destination port. The reason we do this is because exploits are typically targeted towards a particular well known service, and the particular service resides on a specific port, therefore we consider the occurrence of a signature to be an anomaly if it is found to be propagating not only to multiple destinations, but also to the same port on the various destinations.

Each of the mechanisms has its own set of complexities as well as shortcomings.

Shortcomings of utilizing the TCP-CHECKSUM,

[2]One can argue that the worm author could modify the worm to randomly probe addressed not in this list. We can address this problem by say adding some random secret addresses shared within one ISP.

- The TCP-CHECKSUM includes pseudo headers when computed, therefore the same content propagating between different hosts may generate different TCP-CHECKSUMS. (We are looking at methods to remove the effect of the pseudo-header from the checksum, by utilizing methods similar to the one proposed for incrementally updating the hop count).
- A similar checksum is not available for UDP packets.
- As the TCP-CHECKSUM is only 16 bits, the false match rate may be higher as compared to a 32 or 64 bit signature.

Shortcomings of utilizing a CRC,

- As the CRC is computed over the complete data contained in the packet payload, which means that even if 1 byte were to change then the computed checksum would be different. Future worms could take elementary countermeasures to evade being detected by a scheme based on CRC's by simply moving a few bytes around each time they propagate.
- Computation overhead involved.

Shortcomings of utilizing Rabin Fingerprinting,

- If we select a small length for generating the Rabin Fingerprints then we may end up generating too many Rabin Fingerprints for each packet this adds to the memory overhead. On the other hand if we select a large length for generating the Rabin Fingerprints then we may ignore too many packets of smaller length this may resulting in missing worms that propagate at a slow rate over multiple packets.
- Computation overhead involved.

C. Detecting Anomalies

A signature is computed from the data payload contained in every packet passing through the system. By utilizing mechanisms described in [][] (sample-and-hold, multi stage filters) we can determine which signatures occur more frequently than others.

For all signatures that occur frequently we need to maintain counters. These counters maintain counts for (i) the number of unique destination IP's, (ii) the number of unique source IP's, and (iii) the number of unique source IP- destination IP pairs. Because there is memory overhead involved with maintaining these counters per individual signature, we only instantiate counters for a particular signature when it occurs more than a minimum threshold number of times.

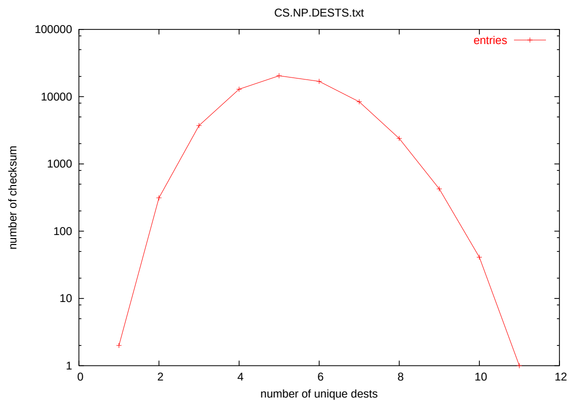

Fig. 1. TCP Checksum Distribution

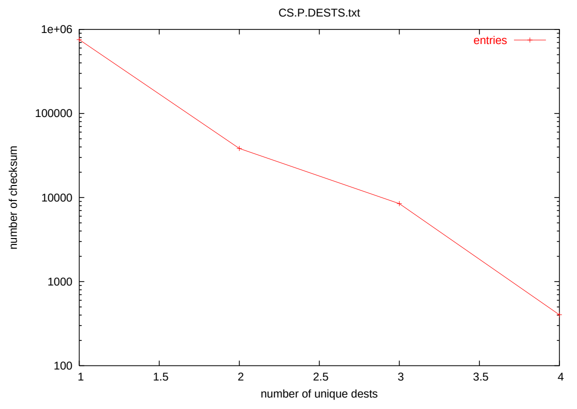

Fig. 2. (TCP Checksum + Destination Port) Distribution

Besides maintaining specific counters per signature, we also maintain global counters, for the number of unique destination IP's, the number of unique source IP's, the number of unique destination Ports, and the number of unique source IP - destination IP communicating pairs across all traffic passing through the system. We propose to utilize these and other such counters to aid in dynamically determining the various threshold levels used by the system to distinguish between the anomalous and the non-anomalous packets.

We have done preliminary experiments by setting the value of the thresholds statically, however we are now considering heuristics to adaptively determine these thresholds. Besides taking into account the volume of various traffic types, and the global counters described above these heuristics also take into account the location where the system has been deployed (i.e. core router, edge router, gateway, host machine).

We observed no packet fragmentation during our analysis. At routers the packets may be fragmented, therefore adequate adjustments need to be made to the algorithm.

We flag packets as anomalies when:

1) We observe packets with identical content being sent to a number of hosts in our local network (of 7 hosts). (Code Red, Port 139 NetBios worm)
2) Packets with identical content are being sent from a large number of hosts (in the Internet) to a single host in our local network(Port 139 NetBios worm)
3) packets with identical content are being sent from a host(s) (within our local network) to a large number of hosts in the Internet. (Linux Slapper)

D. Observations

The observations were made by running our tests on a tcpdump trace file collected over a 9 day period. There are a total of 4 million packets in the trace file.

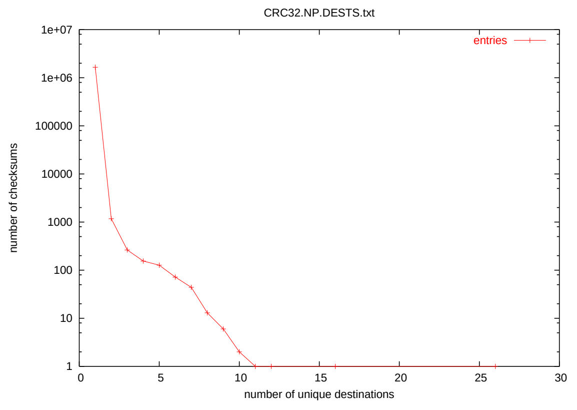

Fig. 3. CRC Checksum Distribution

Based on the graphs we observe that the CRC+Port and the Rabin Fingerprint+Port distributions are not much different from each other.

Further based on these graphs we can observe that the number signatures based on CRC+Destination Port or Rabin Fingerprint ($\beta = 39$)+Destination Port occurring at most once is exponentially higher than the Fingerprints occurring more than once. This leads us to believe that we will need to instantiate counters for a very small fraction of the overall checksums.

On the other hand when we reduce the length of the Rabin Fingerprint to ($\beta = 4$) then we see a large number of signatures occurring more frequently and the TCP CHECKSUM as picked up from the TCP Header provides with no assistance in making any deterministic decisions.

We caught the following three worms: Code Red, Linux Slammer and Port 139 NetBios worm .

There are a number of applications that can lead to frequent false positives without a careful choice of the parameters that control the flagging of worm packets: VNC Sessions, Mailing Lists and SSH Sessions.

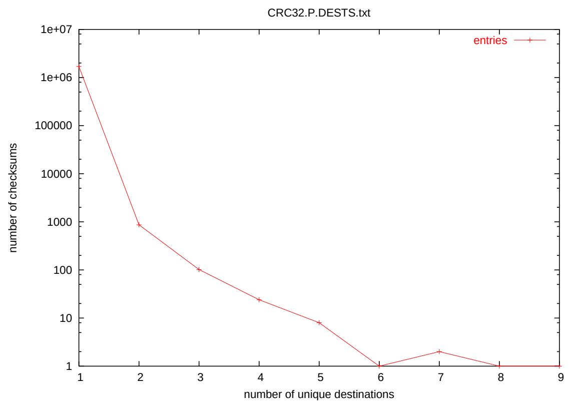

Fig. 4. (CRC + Destination Port) Distribution

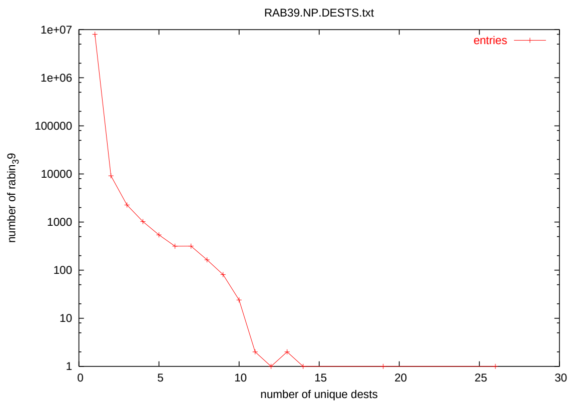

Fig. 5. Rabin Fingerprint (length $\beta = 39$) Distribution

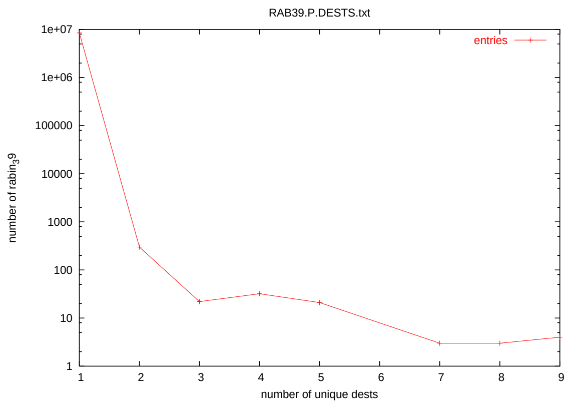

Fig. 6. Rabin Fingerprint (length $\beta = 39$) + Destination Port Distribution

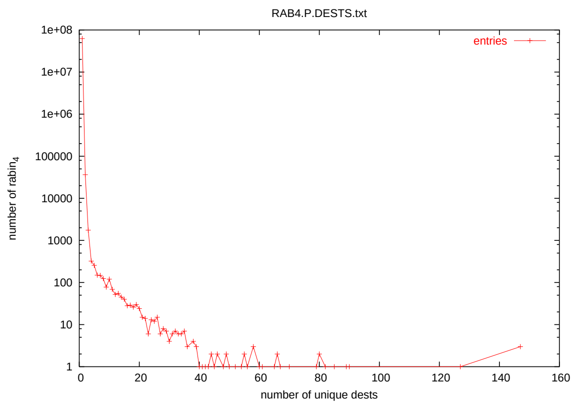

Fig. 7. Rabin Fingerprint (length $\beta = 4$) + Destination Port Distribution

E. Algorithm Refinements

Based on the observations we are now testing a new set of refinements.

Considering the long-term goals of our project, utilizing Rabin Fingerprints seems to be the most suitable option for converting the data contained in the packets into numerical signatures.

However utilizing Rabin Fingerprints is not without it's problems. The length of the sub-strings over which we compute the Rabin fingerprint has (i) a direct impact on the memory required by the Bloom Filter (or any other data-structure) used to keep track of the occurrence of the fingerprints, and (ii) the false positive rates.

If the fingerprint is too small then a) the number of fingerprints generated for every packet may be too many to track, and b) the number of packets the sub string has occurred previously may be too many. On the other hand if the fingerprint is too large then a) we may lose the granularity required to correlate two instances of the same worm, which employ simple techniques to avoid being caught, and b) we may end up ignoring the small packets (packets with fewer bytes than the length of the fingerprint).

Based on these observations we have opted to make some modifications to our algorithm.

Considering that the Rabin fingerprints are computed incrementally, we now propose to track fingerprints of various lengths simultaneously. We now select say two (the final algorithm could use more than two) representative lengths α and β over which we will compute the Rabin fingerprints. Of these two, α is of significantly smaller length than β. As a first step Rabin fingerprints are computed for α length sub-strings and only if a match is found for the smaller length we proceed to compute the Rabin fingerprint for the sub-string of longer length β. We believe that by adopting this method we will only need to compute a few Rabin fingerprints (of small length α) for most packets, which amounts to a significantly reduced processing overhead per packet. This technique will also limit the number of signatures we need to match future packets against thus reducing the memory overhead.

We are also considering computing the Rabin fingerprints for a set of randomly selection sub-strings from the packet and understanding the effects of the same on the false-positive rates.

It is our understanding that we may need to spend considerable effort in determining the effects of selecting various lengths for α and β on various distinct packet streams with substantial variations in the workload.

In order to utilize limited resources we would also like to apply strategies to flush the set of signatures in the filters periodically. At the present time we are studying the effects of the following strategies. (i) Flush all the signatures in the filters, after a period of time or a number of packets, (ii) flush all signatures in the filters while retaining the signatures we have found to be anomalous in the previous window, and (iii) mechanisms to decay the signature entries in the filters as opposed to periodically flushing them.

To reduce the false positive rates, we also need to introduce some adaptive learning so that the system may distinguish between flash events (mailing lists) and worms.

When deployed at the gateway there might be a significant improvement in the effectiveness of the system if we consider different thresholds for the data stream coming into the network and the data stream going out.

F. Validation

For validating future versions of our system deployed on larger networks we will need to estimate the number of worms that actually went by. For this purpose we can use existing IDSes such as Snort that are based on worm signatures.

IV. Acting on the Anomalous packets

While our current system only reports the anomalous packets, the final system will also take adequate action. Because the system cannot always tell with high certainty whether a given packet belongs to a worm or not, it is appropriate to have more than one type of countermeasure in order to limit the amount of collateral damage caused by the system. First of all when the system suspects the appearance of a new worm it alarms the network administrator and provides forensic data to let human experts analyze and classify it. For traffic that looks like a worm but there is still a chance of misclassification the system would take less drastic action such as rate limiting it. For traffic identified with high confidence as worm traffic the system can take more drastic action such as dropping the packets or resetting the TCP connections.

V. Deployment of the system

The deployment of our system on a large enough number of points in the network can significantly slow down the spread of new worms. It can also have the benefit of eliminating traffic generated by older worms that are still active on un-patched systems. There is a large amount of redundant traffic in the Internet, from hosts that remain infected by the worms even after the release of numerous advisories. For example an expected 300,000 hosts are still infected by the much publicized CodeRed worm generating large volumes of traffic as they are still going about their business to infect other hosts. If our proposed Detection Systems were deployed at strategic points in the network, then this redundant traffic could be eliminated.

VI. Conclusion and Future Directions

Worms are a major threat to the security and reliability of todays networks. Because they can spread rapidly from computer to computer, to effectively contain them we need automated methods to very quickly identify and filter new worms before they grow into a massive epidemic. In this paper we propose such an automated approach based on identifying in real time the traffic characteristics common to all worms: highly repetitive packet content, going from an increasing number of infected hosts to very many random IP addresses of potential new victims. Our preliminary results on a small network show that our automated approach of identifying new worms is promising: it identified three confirmed worms with an encouragingly low percentage of false positives when configured with good parameters.

We note that the techniques discussed in this paper could be used to detect other types of malicious traffic prevalent in the Internet: email worms and Spam.

VII. Appendix

A. Rabin Fingerprinting

The Rabin fingerprint for a sequence of bytes $t_1, t_2, t_3, \dots, t_\beta$ of length β is given by

$F_1 = (t_1 \cdot p^{\beta-1} + t_2 \cdot p^{\beta-2} + \dots + t_\beta)\ mod\ M$ where p and M are constants.

The best way to evaluate a polynomial given its coefficients is by Horner's rule:

$F_1 = (p \cdot ((\dots(p \cdot (p \cdot t_1 + t_2) + t_3)\dots)) + t_\beta)\ mod\ M$

If we now want to compute F_2, then we need only to add the last coefficient and remove the first one:

$F_2 = (p \cdot F_1 + t_{\beta+1} - t_1 \cdot p^{\beta-1})\ mod\ M$

We pre-compute a table of all possible values of $(t_i \cdot p^{\beta-1})$ for fast execution of the algorithm.

References

[1] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *Infocom*, Apr. 2003.

[2] P. B. Gibbons and Y. Matias, "New sampling-based summary statistics for improving approximate query answers," in *Proceedings of the ACM SIGMOD*, June 1998, pp. 331–342.

[3] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proceedings of the ACM SIGCOMM*, Aug. 2002.

[4] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, no. 2, pp. 182–209, Oct. 1985.

[5] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," CSE Department, UCSD, Tech. Rep. 0738, Mar. 2003.