Boundary detection and containment of local worm infections

Diego Zamboni*, James Riordan, Milton Yates†
IBM Zurich Research Laboratory
{dza,rij}@zurich.ibm.com, milton.yates@loule.info

May 15, 2007

Abstract

We propose a system for detecting scanning-worm infected machines in a local network. Infected machines are detected after a few unsuccesful connection attempts, and in cooperation with the border router, their traffic is redirected to a honeypot for worm identification and capture. We discuss the architecture of the system and present a sample implementation based on a Linux router. We discuss future improvements for increasing the detection abilities and coverage of the sensor. While the system was developed based on the Billy Goat worm-detection system, it can easily be used with other honeypot systems.

Keywords: intrusion detection, Internet worms, information security, honeypot.

1 Introduction

One of the greatest threats to security of networked systems comes from automatic self-propagating attacks, including viruses and worms. The presence of these attacks is not new, but the damage that they are able to inflict and the speed with which they can propagate have become paramount. Increases in connectivity and complexity only threaten to exacerbate their virulence.

This paper describes an approach to detection and containment of local infections by network-scanning worms. The process is triggered by monitoring for error conditions resulting from failed connection attempts such as ICMP Unreachable messages, refused connections and timeouts.

When a threshold of similar incompleted connections of local origin is detected, the local router is reconfigured to redirect all traffic corresponding to those connections to a local honeypot system, such as a Billy Goat [7]. The infected machine thus believes that it has successfully connected to a remote system, when it is actually connected to a local honeypot that can diagnose its activities to determine the nature of the infection, and to potentially raise alarms or take other actions. Regardless of the activities performed by the honeypot system, the redirection of the traffic, by itself, has the effect of reducing unwanted infection-spreading traffic leaving the local network.

2 Related work

The detection of local worm-infected machines is a problem that has been explored extensively. Most of the schemes proposed focus on monitoring connections established from the local network to determine which machines are scanning the outside, and possibly limit those connections [e.g. 8, 10], but do not attempt to identify or further diagnose the nature of the infection.

Honeypots and honeynets have been used to look for worms, and to trick them into revealing their nature or even their code. Examples of such systems include Billy Goat [7], Nepenthes [3], HoneyStat [6] and the Potemkin Virtual Honeyfarm [9]. These systems have better diagnosis abilities, but require traffic to be sent to them for analysis. This is commonly done by statically routing unused sections of the network to the honeypot system.

The use of ICMP error messages to detect worm-infected machines has been proposed by Berk et al. [4, 5]. They propose an Internet-wide monitoring system that depends on many instrumented routers throughout the network to collect information that is then analyzed at a central location. While Internet-wide detection is useful from the perspective of global awareness and analysis, it does little to help network administrators with local infections and to limit further propagation.

*Contact author.

†Work done while this author was in temporary employment with IBM.

3 Design and architecture

Router-based Billy Goat (RBG) [7] is a mechanism that adds dynamic discovery of external *unused* or *unreachable* IP addresses and redirects traffic to them to a honeypot for processing and response. This dynamic assignment vastly extends the monitoring abilities of the honeypot.

In our work we have used Billy Goat, a specialized worm-detecting honeypot, as the device to which traffic is redirected. However, the detection and redirection mechanism is generic and can be used with other honeypot devices, such as Nepenthes [3].

The benefits offered by our solution include:

- It allows the Billy Goat to dynamically spoof unused IP addresses outside the local network, instead of relying on static assignment of unused address blocks. This allows for much larger address coverage, leading to more efficient detection of infected machines.
- Local infections are detected locally, providing timely information to network administrators and eliminating the need for complex alarm redistribution mechanisms.
- It is able to counter advance scanning strategies, such as those necessary for scanning IPv6 networks. Such techniques include: detection of connections to existing addresses and scanning the local network around them, or scanning for additional services on detected existing machines. These techniques are not normally detected by honeypots, since they scan allocated address ranges.
- It helps in automatically preventing infected hosts from scanning outside the network, thereby reducing unwanted outgoing traffic.

3.1 Requirements

We defined the following high-level requirements for the RBG design and implementation:

- Consider failure modes, which is particularly important in a system that modifies the usual behavior of the network, to ensure it does not interfere with critical infrastucture. We have considered ways in which the system may induce failure, and mechanisms by which these failures can be avoided or mitigated. One such mechanism is the use of white and black lists for both source and destination addresses.
- Ensure extensibility, by keeping clear separations of duty and designing with well-designed interfaces.
- Ensure security, since RBG has a critical place on the network. Special care has been given to security aspects at the design and implementation levels to avoid race conditions, system compromise and Denial of Service attacks.
- Use open source software, to make it easier and less expensive to build, extend and maintain the system.

3.2 Triggering mechanisms

The idea of RBG is to trigger traffic redirection upon detection of failed connection[1] attempts. Such attempts can be detected by the following mechanisms:

- Receipt of ICMP-Unreachable messages. The disadvantage of using only this mechanism is that not all routers produce such messages. Berk et al. [5] performed a test of random IP addresses on the Internet which indicated that in only 6.2% of the cases were ICMP Unreachable messages received.
- Timed-out initial connections. In this scheme, the instrumented router keeps track of initial packets in each connection (for example, TCP SYN packets). Traffic redirection is triggered when no response is seen for a packet after a certain period of time. This mechanism was also described by Berk et al. [4] as a way to dramatically increase the coverage.
- Detection of refused connections. While this mechanisms does not indicate an unused address, it can be used to perform finer-grained per-port redirection of traffic.

3.3 Overall solution behavior

Under normal conditions, when a host tries to contact an unreachable destination or service, one of the three error conditions mentioned in Sec. 3.2 occurs.

When using RBG, the error condition is intercepted. For example, in the case of an ICMP error message, the following sequence (illustrated in Fig. 1) takes place:

1. The internal host sends the first packet of the connection.

[1] By connection, we do not imply only TCP connections.

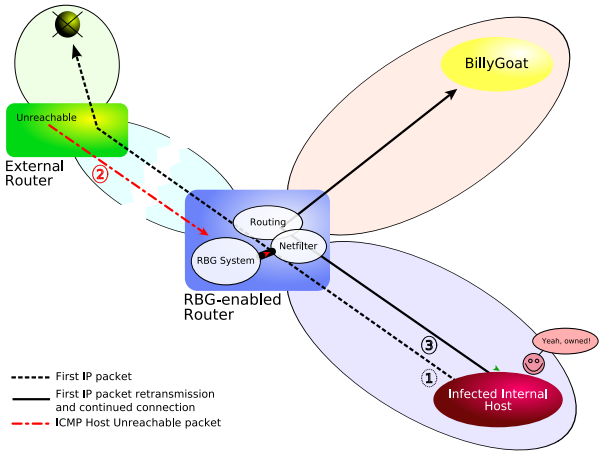

Figure 1: The "unreachable destination" behavior using the RBG architecture

2. The external router sends back an ICMP Unreachable message. The local router intercepts it and automatically generates a rule to route future packets to this unreachable destination, to the honeypot and also sends the original packet to the honeypot.
3. The Billy Goat system receives the packet and replies to it, spoofing the destination host. The internal host gets the reply he wanted and will consider the destination host as being up.

3.4 Placement in the network

The ideal place to put the RBG logic and mechanisms is in a border router. We have used this approach in our implementation using a Linux-based router. However, it would also be possible to implement RBG as a bridge placed in front of the router, monitoring traffic and remotely reconfiguring routes. This mode of deployment would make it easier to adopt RBG without modifying deployed routers.

3.5 Routing rules

In response to the error conditions detected, the RBG has to create routes to redirect the appropriate traffic to the honeypot. We could create one routing rule for each host to redirect, but this would quickly create a large routing table and introduce latency in the routing process for *every* packet.

We chose policy-based routing as a more scalable solution. Modern routers have the ability to mark packets or connections, and to make routing decisions based on those marks using a single rule. This keeps overhead to a minimum, and uses the built-in functionality of the router instead of crafting a specialized mechanism.

4 Implementation

We have implemented and tested a prototype of our RBG system. The system consists of the following main components:

Filtering and routing device: We used Linux as our implementation platform, relying extensively on the Linux Advanced Routing project and the Netfilter Framework.

Control components: The user-space components control the overall logic and flow of the system, providing management of the filtering and routing rules, policy decisions and configuration, and administrative control. We chose to implement these components in Perl [2].

Our current implementation handles only triggering on receipt of ICMP Unreachable packets and relies upon TCP automatic retransmission.

4.1 Implementation architecture

The resulting architecture is described in Fig. 2. (0) ICMP Unreachable packets are caught by Netfilter, (1) checked for correctness and rate limited, then (2) passed to the control program, which (3) generates the appropriate packet-marking rules for iptables. Using the policy-based routing facilities of the Linux kernel, (4) the router sends marked packets to Billy Goat while unmarked packets continue on the standard path. This solution keeps the needed customization level of the router very low, by using many preexisting and widely tested components.

4.2 Detailed design characteristics

We can distinguish three main sections in the whole process: static filtering, static routing, and the control module.

4.2.1 Static filtering

We use the MARK and CONNMARK iptables extensions which enable the marking of packets as well as whole connections. The filtering has to be done *before* routing, so we use the PREROUTING chain. The defined static filtering policy is as follows:

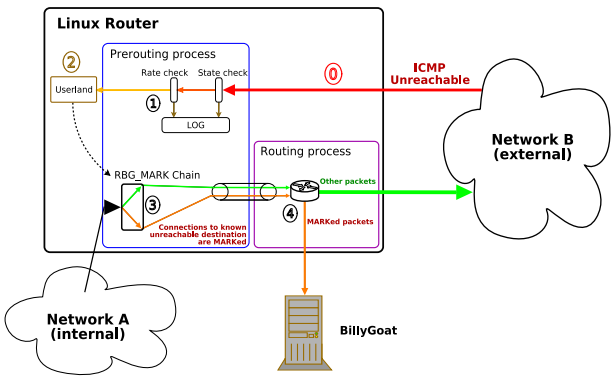

Figure 2: Final RBG architecture

1. We let already established connections pass, except ICMP packets. If one of the packets of the connection is α-marked[2], then all future packets of the same connection will also be marked.
2. A set of "whitelisted" hosts is matched in a separate chain and allowed to pass.
3. Incoming ICMP Unreachable packets are redirected to the RBG_CAP chain, where they are rate-limited, state-checked and sent to the control module via the QUEUE target. State checking is done using the iptables state extension, which allows us to accept only ICMP packets related to a pre-existing connection, automatically dropping forged packets.
4. At this point, only packets initializing a connection remain in the PREROUTING chain.
   (a) A set of "blacklisted" hosts is matched in a separate chain where they are MARKed.
   (b) Packets are sent to the RBG_MARK chain where they are matched against the rules produced by the control module. Those that match are MARKed.

After passing through the PREROUTING chain, remaining packets enter the routing process.

4.2.2 Static routing

Packets marked at the filtering level need to follow a specific route. To achieve this, we add to the existing routing environment:

[2]Using the MARK and CONNMARK iptables extensions, packets are marked using a user-given `unsigned long integer` value. As we only need one kind of mark, and for legibility purposes we will use the α mark.

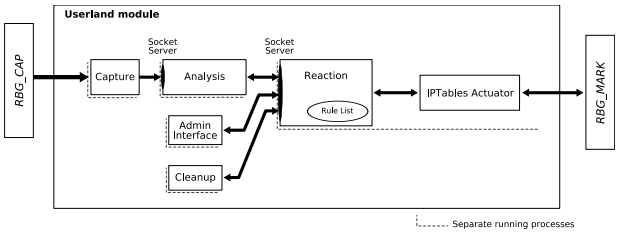

Figure 3: Userland Module internals

- A new routing table, with a default route to the Billy Goat.
- A new rule in the default table, sending α-marked packets to the above table.

4.2.3 Control module

The control module is split into simple, independent, self-recovering and general-purpose modules. It is responsible for:

- Capturing packets queued by the iptables RBG_CAP chain using the QUEUE iptables extension.
- Analyzing these packets and deciding if a rule needs to be generated, according to the policy.
- Creating and handling the set of redirection rules in the RBG_MARK chain.

These tasks have been implemented as a set of running processes connected in a chain, as described in Fig. 3.

Components of the Userland Module

STP module (Simple Text Protocol): This module provides a generic way for all the modules to communicate. It relies on Unix domain sockets to communicate and uses an entirely text-based, human-readable and writable protocol.

Capture module: Responsible for getting packets from the iptables QUEUE extension. After decoding, relevant information is sent to the Analysis module.

Analysis module: Receives information about packets, and processes them according to the policy. If a rule needs to be created, the necessary information is sent to the Reaction module.

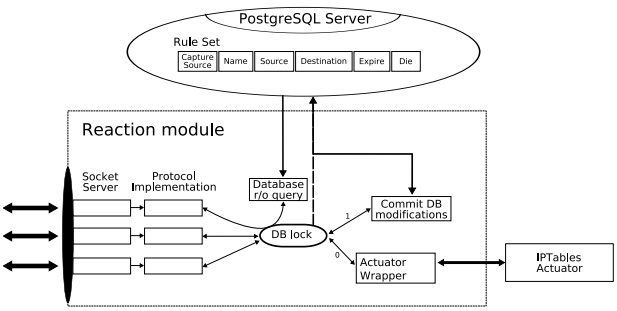

Figure 4: Reaction module internals

Reaction module: Receives commands to apply to the rule sets. It does atomic operations to the whitelist, blacklist and packet marking rule sets. It controls lower-level actuators via a generic rule management API, to interact with different filtering mechanisms (iptables, IOS, ipf, etc.). The IPTables Actuator module is an implementation of this API. The internal structure of the Reaction module is shown in Fig. 4.

Cleanup module: Removes expired rules from the lists.

Administration console: Allows a human administrator to start/stop the system, add/remove rules in the different lists, check the state of the system, etc.

4.2.4 Implementation of the rule sets

The rule sets (the α-marking, whitelist and blacklist sets) are stored in a PostgreSQL table. We use Postgres' table-locking facilities to ensure ordered, atomic operations on the tables. They all share the same table structure:

pktname: The name of the rule.

capsrc: The mechanism by which the error condition was detected.

srcip, dstip: Source and destination IP addresses.

expire: Rule expiry time. This field can be increased according to the policy.

die: Rule death time. This field cannot be increased once the rule is created, providing a simple mechanism to ensure rules do not remain forever.

4.2.5 Extended functionalities

Multiple matching: When duplicate unreachable packets are received, the `expire` field of the corresponding rules is increased according to policy.

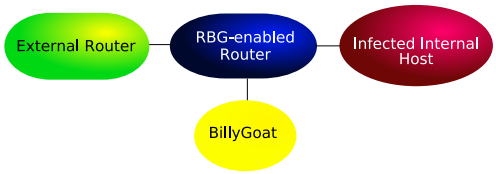

Figure 5: Testing setup

Source flooding: When the same source address generates more than a predefined number of rules, it is isolated by setting a rule that sends all its traffic to Billy Goat for a given time. This increases the amount of traffic captured and slows down the spread of the worm outside the local network.

Destination flooding: When a predefined number of rules with the same destination address are generated, an aggregate rule is generated for that destination.

4.3 Experimental results

We constructed a test environment using the User Mode Linux [1] virtualization system. It emulates a deployment of the RBG architecture as shown in Fig. 5. We performed the following tests to ensure the functionality of the system:

Ping: Sending ICMP Echo Request packets from the internal host to non-existent hosts in the external network, which causes the router to return an ICMP Host Unreachable packet to the requesting host, in turn causing RBG to generate the appropriate redirection rules.

Forged ICMP Unreachable messages: A host from the internal (or external) network sends forged ICMP Unreachable messages at a high rate (we used 10 packets/sec). The goal is to stress test the system by trying to generate a large number of useless rules. ICMP packets are rate-limited (by default 5 packets/sec), and in addition packets that are not related to an already established connection are discarded, so no bogus rules are generated (Sec. 4.2.1).

TCP connection: This test tries to establish a TCP connection from the internal host to an unreachable host. In the implementation, TCP connection redirection works flawlessly, thanks to the TCP retransmission features.

UDP scanning: This test assumes the internal host is infected a worm that propagates or scans using ICMP or UDP messages. In this situation,

only one packet per destination is sent. RBG will not catch the first packet sent to each host, but it will create one rule for each unreachable host. When the source-scanning threshold (5 by default) is reached, RBG will create a rule redirecting all the traffic from the worm-infected machine to Billy Goat. Our tests show that this behavior can be very effective if an entire unreachable network is scanned, as it will generate a large number of rules in a short period of time, and thus reaching the source scanning limit very quickly.

Worm capture and diagnosis capabilities in all cases can be improved by the addition of router-based retransmission of captured packets, as described in Sec. 6.1.

5 Conclusions and limitations

We have introduced a general purpose mechanism that shows promising results to increase coverage, speed and accuracy of honeypot-based worm detection systems. It offers the significant benefit of detecting local infections locally, providing a valuable tool to network administators, and it helps perform local containment of worm infections, thereby preventing unwanted traffic from leaving the local network. It also allows capturing advanced scanning techniques, such as those needed to scan IPv6 networks.

5.1 Limitations

We now describe the limitations of the current implementation of RBG, and their workarounds if existent.

Detection of scanning worms only: By design, RBG will only detect and redirect traffic produced by hosts that are scanning nonexistent IP addresses. Hitlist worms, email worms and other types of malware that direct their attacks against existing machines and services will not be detected by RBG.

IP spoofing: Using IP address spoofing, an attacker inside the local network could abuse RBG and make it isolate a local IP address from the outside, using the source flooding detection feature of RBG. This attack may be mitigated using MAC address checking and filtering.

Liveness checking problem: RBG breaks network diagnostic methods that rely on reachability testing (for example, by pinging a remote host). We have included whitelists in the design to address this concern.

White and blacklists: Large white or blacklists could have a negative impact in routing performance. The number of entries in these lists should normally be kept small, both to minimize the performance impact and to maximize the coverage of RBG.

6 Future work

We describe some possible extensions to the current RBG implementation.

6.1 Packet ring buffer

Currently the largest gap between our conception and the implementation is the lack of router-based retransmission of the initial packet of a connection. To address this limitation, a production system should implement a "first-packet ring buffer". The router would push the first packet of each connection into a ring buffer. When a connection to an unreachable host or service is detected, RBG could search in this buffer for the first packet of the connection, and if found, retransmit it to Billy Goat after creating the redirection rule, to ensure that the full connection is captured by Billy Goat.

6.2 Capturing additional packet types

The current implementation of RBG only intercepts ICMP Unreachable messages, but it would be fairly simple to capture any other kind of packets, to detect other error conditions as described in Sec. 3.2. For example, one could catch ICMP Port Unreachable messages to detect connections on closed UDP ports, or TCP Reset packets to detect connections on closed TCP ports.

Adding these extensions would require writing new Capture modules and implementing the corresponding logic in the Analysis module.

6.3 Other detection mechanisms

RBG could also use events from different sources as triggers to generate redirection rules. For example, it could read events from an IDS and generate redirection rules to isolate hosts using RBG.

This way, we would combine the power of an IDS with the RBG architecture to exploit the advanced capabilities of a WDS like Billy Goat, or to produce a appropriate response. It would also allow malicious

activity other than worms to be redirected to appropriate capture and analysis devices.

6.4 Multiple actuators

RBG has been designed to be used with different types of routing devices. Our current implementation has an actuator for a Linux-based router using iptables, but it would be easy to write actuator modules for other routing devices.

6.5 Blocking initial traffic

In case of a high-volume worm infection, the time until the RBG detects and redirects the traffic could still allow a significant number of infection attempts to leave the local network. A solution to this problem, at the expense of largely increased resource usage in the router, would be for the router to keep track of addresses from which legitimate responses have been seen (i.e., valid existing addresses). When a connection to a new address is seen, the initial packet is allowed through, but all other traffic to that address would be queued or dropped until a response is received, or until a certain timeout occurs.

References

[1] The User-Mode Linux homepage. `http://user-mode-linux.sourceforge.net/`.

[2] Perl homepage. `http://www.perl.com/`.

[3] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. C. Freiling. The Nepenthes platform: An efficient approach to collect malware. In D. Zamboni and C. Krügel, editors, *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 4219 of *Lecture Notes in Computer Science*, pages 165–184. Springer, 2006. ISBN 3-540-39723-X. URL `http://dx.doi.org/10.1007/11856214_9`.

[4] V. Berk, G. Bakos, and R. Morris. Designing a framework for active worm detection on global networks. In *Proceedings of the IEEE International Workshop on Information Assurance*, Darmstadt, Germany, Mar. 2003. URL `http://people.ists.dartmouth.edu/~vberk/papers/iwia03.pdf`.

[5] V. H. Berk, R. S. Gray, and G. Bakos. Using sensor networks and data fusion for early detection of active worms. In *Proceedings of the SPIE Aerosense conference*, Apr. 2003. URL `http://people.ists.dartmouth.edu/~vberk/papers/spie03.pdf`.

[6] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. Honeystat: Local worm detection using honeypots. In *Proceedings of the Seventh International Symposium on Recent Advances in Intrusion Detection*, Sept. 2004. URL `http://www.cc.gatech.edu/~wenke/papers/honeystat.pdf`.

[7] J. Riordan, D. Zamboni, and Y. Duponchel. Building and deploying Billy Goat, a worm-detection system. In *Proceedings of the 18th Annual FIRST Conference*, June 2006.

[8] S. E. Schechter, J. Jung, and A. W. Berger. Fast detection of scanning worm infections. In E. Jonsson, A. Valdes, and M. Almgren, editors, *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 3224 of *Lecture Notes in Computer Science*, pages 59–81. Springer, 2004. ISBN 3-540-23123-4. URL `http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=3224&spage=59`.

[9] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. Snoeren, G. Voelker, and S. Savage. Scalability, fidelity and containment in the Potemkin virtual honeyfarm. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 148–162, Brighton, UK, Oct. 2005. ACM.

[10] C. Wong, S. Bielski, A. Studer, and C. Wang. Empirical analysis of rate limiting mechanisms. In A. Valdes and D. Zamboni, editors, *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 3858 of *Lecture Notes in Computer Science*, pages 22–42. Springer, 2005. ISBN 3-540-31778-3. URL `http://dx.doi.org/10.1007/11663812_2`.