RZ 3609 (# 99619) 05/16/05 (Revised: 11/02/05)
Computer Science 12 pages

Research Report

Billy Goat, an Accurate Worm-Detection System (Revised Version)

James Riordan, Diego Zamboni, and Yann Duponchel

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

{rij,dza,ydu}@zurich.ibm.com

Revised: November 2, 2005

Billy Goat, an Accurate Worm-Detection System

James Riordan, Diego Zamboni, Yann Duponchel
IBM Zurich Research Laboratory
{rij,dza,ydu}@zurich.ibm.com

November 2, 2005

Abstract

Billy Goat is a worm detection system. Its most important feature is its reliability in terms of accuracy, resilience and rapidity in detection and identification of worms without false positives. It is widely deployed throughout IBM [1] and several other corporate networks. We discuss the features and requirements of worm detection systems in general, and how they are addressed by Billy Goat. We also describe the tools and constructions that we have used in the implementation and deployments of the system, and discuss contributions which could be useful in the implementation of other similar systems.

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

1 Introduction

Recent years have brought a continued increase in both the importance of security in networked systems and the difficulty of securing them. Beyond the basic need for integrity, confidentiality and privacy, security has become essential toward providing reliability, safety, and freedom from liability.

One of the greatest threats to security has come 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 a worm detection system that we have built and deployed, throughout IBM and in several customer networks, over the past five years. The deployment within IBM covers the entirety of the corporate intranet, automatically gathering data from approximately 1.2 million virtual sensors, centralizing the data to form a single coherent model of suspicious network activity, and analyzing this model for evidence of worm infections.

The rest of the paper is structured as follows: Section 2 provides an overall description of worm-detection systems in general. Section 3 presents the architecture of Billy Goat, both as a sensor and as a distributed system. Section 4 talks about related work. Section 5 describes the data models used throughout Billy Goat. In Section 6 we describe some of the components we have implemented, and decisions we have taken while building Billy Goat, including some constructs that can be used in other contexts. Sections 7 and 8 describe some of our observations and experiences during Billy Goat deployments. Section 9 concludes the paper, and presents some avenues for future work.

2 Worm Detection Systems

The requirements on a worm-detection system (WDS) are different from those of a general-purpose network-based intrusion-detection system (NIDS). While the latter needs to detect a wide and unpredictable variety of attacks, the former can focus on specific attacks and strategies used by worms. The main purpose of a WDS is to detect infected machines in the network, whereas a general-purpose IDS must detect the attacks themselves. These requirements influence the desired characteristics of the system, particularly in the following aspects:

Accuracy: To offer real utility, a WDS must be able to identify worm-infected machines with a high level of accuracy, so that it can be trusted as the basis for contention and remediation action. A WDS can use highly-specialized techniques to detect worm-infected machines. This enables increased accuracy, at the expense of the ability to detect a wider range of attacks.

Speed: Given the explosive nature of modern worms [7], a WDS should be able to detect an infected machine as quickly as possible, to provide its users a chance to contain the damage, or even to func-

[1] Company name anonymized for review

tion as the basis for an automated response system.

Manageability: At a systems level, installation and update of the components must be automated as much as possible, to be able to deal with monitoring very large networks. At middleware and architecture levels, the base infrastructure must offer sufficient flexibility to enable the rapid creation of new detection capabilities.

Interoperability: A WDS should integrate as much as possible with the existing tools and processes, to avoid unnecessary proliferation of independently-managed security tools.

Resilience: A WDS must operate under extreme conditions in terms of network and processing load, particularly during worm outbreaks. These conditions are more likely to induce failures than other environments. However, a WDS has a specific advantage that is not enjoyed by most other IDSs: because of the repetitive nature of worm activity, the WDS can afford to lose some data without reducing its utility. In practice, this means it is satisfactory to build a system that can "forcefully" recover from failure (for example, by automatically rebooting or even reinstalling itself) rather than trying to resist it.

Graceful degradation: While WDSs may benefit from a distributed architecture, most worm outbreaks have the effect of overloading network links. It is therefore necessary for all sensors to be able to operate on their own, so that even if the global system is impeded, its individual sensors can still be useful during a worm outbreak.

3 Billy Goat architecture

Billy Goat possesses the characteristics described above toward detection of network-service worms; that is, worms that exploit vulnerabilities in existing network services (e.g. HTTP, MS/RPC, MS/SQL, etc.) and that propagate by finding new vulnerable machines in the network and infecting them automatically. Henceforth, we shall refer to these as "worms".

Billy Goat is focused on detecting machines in the network infected with known worms, and in this respect it is free of false positives by construction. It also provides additional information that can be analyzed to detect new worms or other emerging threats in the network.

Billy Goat is designed to take advantage of the propagation strategies of worms. To discover machines to infect, most worms try to connect to IP addresses selected at random or scan entire ranges of addresses. By doing so, they find most of the machines in a network, but they also try to connect to a large number of unused addresses. Billy Goat functions by responding to requests sent to unused addresses, feigning the existence of a large number of machines and services. This approach has three immediate consequences:

- Active feigning of services, rather than the mere recording of connection attempts enables greatly improved understanding of the nature of the connection, and gives Billy Goat an advantage in accuracy.
- The fact that the addresses are otherwise unused and not advertised means that all traffic destined to them is *a priori* suspicious.
- The large number of addresses used gives Billy Goat an extensive view of the network.

Instead of directly "guarding the valuables," as traditional intrusion detection deployments do, Billy Goat guards vast ranges of "nothingness" toward understanding who goes there and why. This is similar to a honeypot, although it lacks the eponymous honey and may have different scope and goals depending on the definition of honeypot used (see also Sec. 4).

This approach, permitted by the clear focus on detecting worms, coupled with the analysis performed on the data (Sec. 6.8) frees Billy Goat from the high rate of false positives produced by most general-purpose IDSs. For the same reason, it is not a replacement for other IDSs but a complement to them. In particular, Billy Goat will not see the traffic directed to existing machines and services, so it is unable to detect attacks against them.

At the core of Billy Goat are a virtualization mechanism and a data repository, shown in Figure 1. The virtualization mechanism (Sec. 6.5) permits services to be written using standard programming models, and respond to multiple IP addresses transparently. This reduces the difficulty of creating new feigning services and of integrating existing ones. The data repository (Sec. 6.2) provides storage for all the IP- and application-level information generated.

The feigned services offered by Billy Goat (Sec. 6.4) include those commonly exploited by worms. Each endeavors to offer sufficient functionality to accurately determine the nature of an attack. All the feigning servers except for SMB (Windows file sharing) are implemented using a specialized framework written in Java (Sec. 6.3) that makes it easy to create new services, and which is carefully audited for security.

To satisfy the requirement of continued function in times of heavy worm activity, when the performance of the network may be dramatically diminished, WDSs require distributed architectures. Each

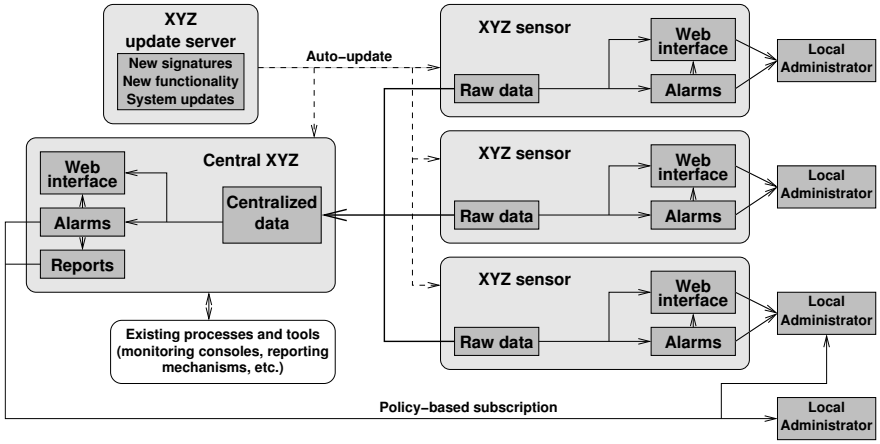

Figure 2: Distributed Billy Goat architecture..

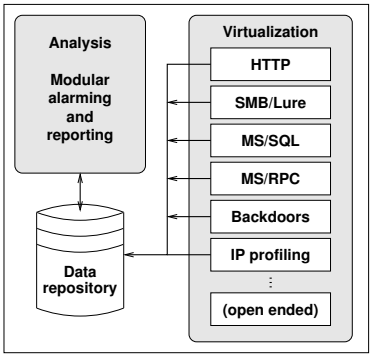

Figure 1: Billy Goat internal architecture.

Billy Goat offers the ability to analyze and report events detected locally, thus providing graceful degradation of the detection service. At the same time the data of all Billy Goats on an intranet is centralized to assemble a more complete view (Figure 2). One effect of this type of monitoring is the feasibility of detecting infected machines on network segments that do not have a Billy Goat sensor installed.

We refer to a machine with Billy Goat installed on it as a "sensor" or "Billy Goat" and to the programs that implement the individual feigned services as "feigning servers".

4 Related work

The idea of observing traffic to unused IP addresses has been used for detection of network noise and scanning behavior. For example, most large Internet Service Providers observe and keep statistics on connection attempts to unused IP addresses within their networks. We do not know of any that actually respond to this traffic.

A similar approach is that of "network telescopes" [6], which also record traffic sent to large blocks of unused IP addresses, but without responding to the traffic, and just recording statistics about those connection attempts.

Billy Goat fits the definition of a low-interaction honeypot [18] in that it pretends to be something that it is not, but without emulating the system in depth.

Projects like HoneyNet [17] have used the concept of deploying fake virtual networks. At the time when Billy Goat began, their focus seemed to be more on accurately simulating those virtual networks, to trick human attackers. Honeyd [12] is a toolkit for building these virtual networks. While these tools are useful for engaging human attackers, such sophistication is not needed to trick worms, which are commonly much dumber.

On the other hand, Billy Goat has focused from the start on detecting automated attacks performed by worms, and for this reason Billy Goat does not go to great lengths to hide what it is. This allows certain foundational constructions resulting in a simpler, more scalable architecture, and in the ability to have a "plug-and-play" implementation, which comes preconfigured to observe a network in a useful fashion, immediately after installation.

Over time, in an example of convergent evolution, honeynets and honeypots have also been used to detect worms [5, 9, 13] and Billy Goat has grown in deceptive sophistication.

The Internet Motion Sensor [2] consists of a distributed network of dark address spaces, with the objective of detecting and measuring threats. The IMS uses generic listeners to capture and respond to connections, which results in less specialized sensors, with the advantage of making it easier to deploy lis-

teners for new services.

Pang et al. [11] have implemented a system for measuring and characterizing network noise, using unused IP address space and protocol-specific listeners. This system uses a non-distributed architecture, and its objective is different from Billy Goat (characterizing the traffic seen, instead of precise identification of infected machines).

Another interesting project is the one reported by Yegneswaran et al. [23]. This project observes traffic to unused IP addresses using a combination of passive monitors, active responders, and virtual networks and hosts. The report describes the use of sampling techniques to deal with the large amount of traffic while still being able to draw meaningful conclusions. Unlike Billy Goat it uses a non-distributed architecture, with the consequent functional and implementation differences.

5 Data Models

At the heart of Billy Goat are several data models used to represent the data gathered in both the IP and applications layers. In this section we describe the types of information that are collected, and the data structure constructed for analysis purposes.

The data models we use address the requirements of a WDS. In particular, we need to address the efficient handling of high volumes of repetitive data; the distributed nature of the sensors and the need for centralizing the information they produce; the need to analyze the data collected by the sensors for evidence of worm infections; the ability to add new feigned services; and the ability to capture varying levels of complexity depending on the data collected.

All data is stored in a relational database, whose structure is described in Section 6.2.

5.1 IP Layer

Data in the IP layer covers TCP, UDP, and ICMP headers. It describes which source addresses have initiated contact with which destination address, along with details that vary with each particular protocol. We gather this data directly from the Linux kernel IPtables facility [3].

5.2 Application layer

Feigning servers gather data in the application layer. We split this data into three aspects: the first describes basic source and time information. The second two aspects address the application-level information. One contains data that, as best we are able to determine, is inherent to the worm (constant through all instances of it) while the other aspect describes the data that varies between different instances of the worm (in many cases the latter aspect is empty).

The distinction between constant and variable data is not uniformly well-defined across all worms. For example, some account-guessing worms have a constant list of account names they use, while others modify the list based on accounts on the infected machines. Feigning servers often have a certain amount of intrusion detection logic to make this distinction. Thus, a feigning server subject to account guessing may have a list of standard account names attacked. If an attacked account name is in the list it is considered "constant," if not it is considered "variable." This allows us to focus analysis on "constant" while still offering the possibility to analyze "variable" for emerging behavior (such as new worm variants).

5.3 Analysis model

The basic data model used for analysis is an aggregation, for each source address, of the above data models over a specifiable time frame. It answers the question "what have we seen from this address recently?" Each per-source model is comprised of the source IP address; the time period covered by the data in the model; the IP addresses of the Billy Goat sensors that have reported the activity; and an aggregation of the IP- and application-level data collected, including full worm capture when possible.

The details of how analysis is performed are covered in Section 6.8.

6 Implementation

This section describes some of the components we have designed and implemented for Billy Goat. We have endeavored to use standard tools, formats, and APIs, and to provide simple, well-documented interfaces by which Billy Goat may be integrated with existing tools and process. We have used many open-source components throughout Billy Goat— in fact its construction would not be possible without them.

6.1 Implementation platform

Billy Goat is implemented as a specialized Linux distribution, which self-installs on a standard PC requiring only basic configuration information. It was our intention to make Billy Goat as appliance-like as possible, so that it can be deployed with minimum effort

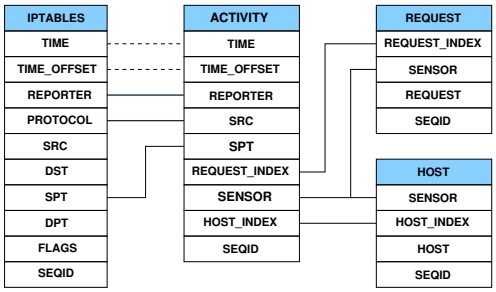

Figure 3: Database tables used in Billy Goat.

throughout a large network. Billy Goat includes extensive self-monitoring and recovery mechanisms that monitor system activity and correct or reinitialize errant components, including the system itself (e.g. reboot).

To support the distributed architecture, Billy Goat includes an automatic update mechanism. This ensures that each sensor is always current with respect to both signatures and software versions, and makes it easier to manage a large distributed infrastructure.

6.2 Database Tables

We split the data collected by Billy Goat (Sec. 5) into four database tables to accommodate the repetitive, and often verbose, nature of the attacks used by worms.

These tables have the form shown in Figure 3, where solid lines indicate external keys (references across tables) and dashed lines indicate temporal proximity (the times are generated in different layers and hence may have slightly different values).

TIME (an SQL timestamp) together with TIME_OFFSET (an unsigned integer that indicates the n^{th} event in a given TIME) create a unique timestamp for each event.

PROTOCOL, SRC, DST, SPT (source port), DPT (destination port) and FLAGS apply to the IP layer, mapping to the three covered protocols (TCP, UDP, and ICMP). FLAGS are void for UDP and ICMP, and the type of ICMP message is stored in both the SPT and DPT fields. Storing the three types of traffic in a single database table makes it easier to extract all the information with a single SQL query.

The full descriptions of the application layer activity, REQUEST and HOST, are expressed in XML and correspond to the previously described constant and variable data aspects. This allows us to meaningfully encode information gathered in the application layer. For example, a simple UDP listener provides a greatly different data model than an extremely complicated protocol like SMB. XML allows us to keep simple descriptions simple while still allowing for complex descriptions.

Rather than the standard practice of using native database external keys for references, we have introduced the use of cryptographic checksums:

$$\text{REQUEST\_INDEX} = \mathbf{md5}(\text{REQUEST})$$
$$\text{HOST\_INDEX} = \mathbf{md5}(\text{HOST})$$

This offers the significant advantage that references depend only on the content of the database record to which they refer. Our experience deploying a large distributed system has shown this technique *greatly* eases data centralization.

REPORTER is the IP address of the Billy Goat sensor that observed the event. SEQID is an automatically incremented value that we use to keep track of which events have been processed by different components. SENSOR is a short string identifying the feigning server that produced the record.

6.3 Event framework

An extensible light-weight Java event framework is used ubiquitously in the construction of Billy Goat: data acquisition, centralization, analysis, alarming and reporting.

The basic components of the framework provide support for buffering, fanning out, predicate expression and evaluation, serialization and de-serialization, various network transports, connections to databases, and several other generically-useful components for event handling.

Events are objects which have the structure of trees, whose nodes are named and have attributes, and whose leaves are arbitrary Java objects. This structure is motivated by XML and, indeed, the serialized form of the objects is valid XML.

Using the event framework for transport makes it easy to inter-operate with existing tools and processes. As examples, we have implemented components for sending events through email, syslog, the Tivoli Risk Manager console [21] and the Arcsight console [1]. Switching to a different event transport mechanism is as easy as changing its definition in a configuration file.

6.4 Feigning servers

As mentioned, the key observation mechanism of Billy Goat is a collection of feigning servers, each covering an infection vector used by worms to propagate.

Real servers may have vulnerabilities in different layers, and often this requires us to write the feigning servers in a way that can detect attacks in different layers. For example, we may need to write the feigning server to be able to detect both low-level buffer

overflow vulnerabilities and application layer vulnerabilities. In general, we try to build the servers to follow the corresponding protocol up to a point that accurate identification of the activity becomes possible. For example:

- The HTTP feigning server accepts and records a single HTTP request, and always responds with an error, before closing the connection.
- The MS/RPC feigning server accepts and records the first 3000 bytes (configurable) transmitted by the client, before closing the connection. This initial payload generally contains either the full code of the worm or an exploit particular to the worm.
- The SMB/Lure server is a special configuration of Samba [16] that appears to be a badly configured machine (open shares, weak passwords, etc.). Because it is a full implementation of the protocol, SMB/Lure can often capture the full code of the worm, as it uploads itself.

The majority of servers are written in Java and produce descriptions of individual interactions. The specific syntax of the records produced is left to each individual server.

Some protocols are sufficiently complex so that it is not feasible, within the scope of our project, to implement feigning versions of them. The most notable example is Microsoft's SMB protocol. In this case, we use an open source implementation of the SMB protocol [16] specially configured to mimic the vulnerabilities of the native Windows implementations. We base our solution on the SMB/Lure system [8], with some ideas from WormCharmer [10], but reimplemented it to address the needs of large-scale virtualization and to fit with the rest of the Billy Goat architecture. This is an example of how existing services may be integrated into Billy Goat thanks to its virtualization infrastructure.

Other feigning servers that currently exist in Billy Goat include MS/SQL, SMTP, DNS, LDAP over SSL, eDonkey, Kerberos, and multiple worm backdoors, including those of MyDoom, Sasser, Dabber and some Beagle variants.

6.5 Address virtualization

Address virtualization transparently maps the large ranges of IP adddresses covered by a Billy Goat to the single "real" address used by the machine. This virtualization allows the Billy Goat feigning servers to be written with no special consideration for the large number of IP addresses that a Billy Goat machine monitors. Address virtualization is handled by the operating system using the IPtables mechanism [3]. One of the mechanisms built into IPtables is Network Address Translation (NAT). Ordinarily, NAT is used to allow several machines inside a network to share a single external address. For Billy Goat, we do the reverse: allow a single machine to respond to a large number of external addresses (Figure 4).

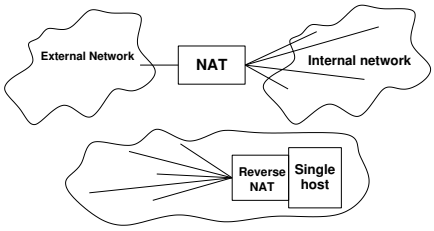

Figure 4: Traditional and reverse NAT.

6.6 Centralized configuration

To maintain local configuration information while maintaining homogeneity across machines, we use a central per-host configuration file in XML from which system configuration files are derived. The file is created by a wizard during first-time installation, and can be later modified by hand or by specialized tools. Having all the information in a single file makes it easy to "back up" an entire sensor.

A related issue is maintaining the configuration for all the distributed sensors in a central place. This enables restoration of a sensor that is completely destroyed (for example, by a catastrophic disk failure), by restoring its configuration to a new machine. It also becomes possible to centrally control the configuration of machines, similarly to network configuration schemes such as BOOTP and DHCP. Based on a unique identifier, the central server can provide each sensor with its configuration information.

This doubly-centralized configuration keeps all the sensors as homogeneous as possible, while allowing for per-sensor configuration in an automated and manageable fashion.

6.7 Data centralization mechanisms

For data centralization, we developed a generic mechanism for one-way transfer of relational tables. This mechanism is built atop the BEEPLite [4] implementation of BEEP [15], a modular, extensible protocol supporting flexible establishment and multiplexing of communication channels. It is designed to tolerate intermittent sensor, server or network failures. Data is automatically transmitted or retransmitted as necessary, without loss or duplication. The mechanism

makes use of the checksum-based external keys and of the monotonically-increasing Sequence-ID field, as described in Section 6.2.

6.8 Data analysis

The data analysis is an iterative process that attempts to determine the types of activity that have been seen by Billy Goat from different source addresses in the network. This process is most complete when done in the central server to which all Billy Goat machines send their data, because it aids in discovery of global behavior that may not be visible at the individual sensors.

Identification of known worms, attacks, and behaviors, is done using a combination of the following methods:

- Capture of the worm itself (for example, SMB worms that upload themselves to Billy Goat). In this case, the MD5 checksum of its code is used to identify the worm with 100% accuracy. As worms become more sophisticated, we anticipate the utilization of analysis techniques from the anti-virus world [20].

- Observation of the exploits used by the worm (for example, an HTTP request containing a buffer overflow). Because Billy Goat is a first-person observer, it can accurately collect the full set of exploits used by a worm, and match them with known worms (for example, we know the full set of exploits used by the Nimda worm).

- Observation of other behaviors indicative of worm activity (for example, horizontal scanning or account guessing). These are weaker indicators of worm activity in the sense that they do not make it possible to precisely identify the worm.

When precise worm identification is possible (when we can give a name to the worm), the findings are labeled as "alarm" and the worm name is given. Clearly suspicious but unidentifiable findings (for example, a large horizontal scan or an exploit set that does not fully identify a worm) are labeled as "warning" and a description is given. All other data is labeled "unknown" and is available via direct query of the database.

Billy Goat is false-positive–free by construction, because "alarms" are only produced when a worm can be unambiguously identified. "Warning" and "unknown" events are also produced, but should not be considered as unequivocal evidence of infection.

6.8.1 Alarm redistribution

We have found that, in large organizations, utilization of existing social structures is essential toward creating mechanisms of sufficient dynamism to address rapidly-emerging security problems. By contrast, static lists tend to become outdated quickly.

We have built a subscription service for Billy Goat that utilizes these structures. It produces alerts based on the centralized Billy Goat data to provide the most complete coverage. It allows individual systems administrators to self-register to receive alerts pertinent to their own network ranges, thereby ensuring that alerts are delivered to someone who can actually do something to fix the detected problems. Open registration allows for a "living" mapping between network and owner. Access control is done via social mechanisms, by notifying the subscriber's manager on registration.

6.9 Modes of deployment

The fundamental premise of Billy Goat is responding to traffic directed to unused IP addresses, as described in Section 2. Different deployment modes can be used and combined to direct such traffic to Billy Goat.

6.9.1 Static routes

This is the standard Billy Goat deployment mode. A specific set of unused IP address ranges is designated for Billy Goat, and the appropriate routers are reconfigured to send traffic destined to those ranges to a Billy Goat sensor. The amount of traffic seen by the sensor depends on the size of the network range assigned. Addresses within non-routable address ranges [14] that are not used locally may also be routed to the Billy Goat, thereby expanding its view of the network.

Advantages: a known set of IP addresses is assigned to Billy Goat, which helps in controlling the amount of traffic it has to process. Only simple configuration changes need to be made to the routers.

Disadvantages: large-enough groups of network addresses must be available and assigned by the network administrator. If the assigned range is too small, the functionality of Billy Goat is limited because it cannot observe much of the network traffic.

6.9.2 ARP spoofing

In a LAN, the machine that has a particular IP address is found by using ARP (Address Resolution Protocol). Using this protocol, machines and routers in the local network that need to send traffic to an

address X broadcast the question "who has address X?" and wait for a response. If no response is received in a certain period of time, the address is considered nonexistent.

ARP is vulnerable to *ARP spoofing* [22], by which a malicious host can "hijack" IP addresses by spoofing ARP responses. This same technique can be used by a Billy Goat device to automatically grab currently unused IP addresses.

Advantages: no previous assignment of IP addresses is needed, so the deployment effort is very low (simply connect the Billy Goat machine to the network).

Disadvantages: ARP spoofing is potentially very dangerous if Billy Goat attempts to spoof the IP address of an existing device. The implementation of this scheme needs to take into account the potential appearance of devices (spoofing must stop immediately when another device with the same address appears on the network).

6.9.3 Billy Goat as default LAN route

Instead of having specific network ranges assigned to the Billy Goat sensor, the router can be configured with a route to Billy Goat for the entire LAN and higher-priority (mask length) routes for the ranges that are being used. This gives Billy Goat all traffic for LAN segments that are not in use.

This scheme can be implemented statically (when the router has a static routing table, and the route to the Billy Goat sensor is added as the default route) or dynamically in conjunction with a routing protocol such as BGP.

Advantages: large network coverage, and ease of configuration (the Billy Goat sensor can be configured to "spoof everything," and it will respond to any traffic it receives).

Disadvantages: potentially dangerous, particularly in conjunction with dynamic routing. In a large network, it is common that certain network segments go offline for short periods of time. If Billy Goat automatically starts responding for them, it may disturb services or automated monitoring systems in the network.

6.9.4 ICMP-based Billy Goat

One of our most recent architectural designs is a mode of deployment in which Billy Goat operates in conjunction with a router to provide automatic utilization of all the unused addresses outside the local network. This is how it works (Figure 5):

1. When an infected machine in the local network tries to contact a remote non-existing address, an ICMP "network unreachable" or "host unreachable" message will be generally sent back.

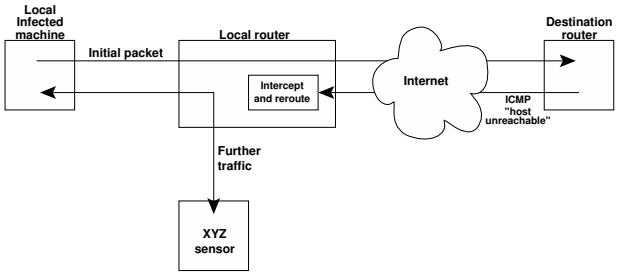

Figure 5: ICMP-based Billy Goat.

2. The ICMP message is intercepted by the router local to the infected machine, which sets up a temporary route for that destination address, with the Billy Goat sensor as its next hop.

3. When the infected machine, after not receiving a response, retransmits its packet, it will be sent to the Billy Goat sensor, which will respond to it.

Many modern worms implement their own TCP/IP stack, often without the automatic retransmit features. By keeping additional state in the router, the retransmit can be spoofed so that those worms are also caught.

Advantages: this mode of operation automatically spoofs every unused address outside the LAN. This provides Billy Goat with a truly expansive view and allows it to quickly identify local infected machines.

Disadvantages: router support is needed to implement this scheme.

7 Environmental effects

One of the areas that we have found most interesting is the effect of Billy Goat on other systems in the networking environment. We describe some of the effects we have observed during our deployment experiences over several years.

7.1 Network discovery

The first aspect that we encountered was the interaction of Billy Goat with devices and software that scan the network legitimately. Normally, Billy Goat responds to these scans for each one of the IP addresses it is spoofing, producing wildly inaccurate results for the scanner. We addressed this problem by adding a mechanism that makes Billy Goat respond "truthfully" to a configurable set of fixed IP addresses. This

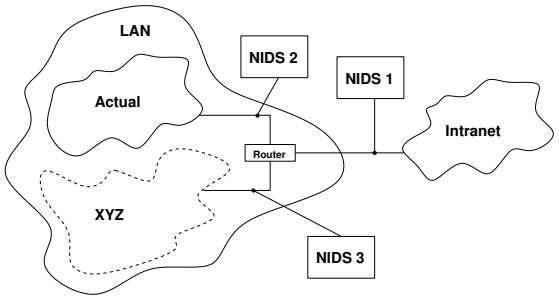

Figure 6: Possible NIDS placements with respect to Billy Goat.

makes Billy Goat appear like a regular machine to authorized scanning devices.

7.2 Network sensors

The second area that we discovered was a sharp increase in the number of alarms coming from network-based IDSs (NIDS) that observe the stream of traffic flowing to Billy Goat. This stems from the fact that Billy Goat, in allowing illegitimate connections to complete, increases the number of real, albeit harmless, attacks seen on the network. The size of the increase depends on the relative sizes of the networks, the saturation level of the network connections, the root cause of the alarm and the signatures and placement of the NIDS.

This effect was first noticed when the Nimda worm was active in a well-connected network whose Billy Goat address space was approximately 100 times larger than the number of actual hosts, and with NIDS at position 1 (see Figure 6). The result was roughly a corresponding 100-fold increase in the number of Nimda-related alarms of attacks from the Intranet against the LAN.

A NIDS at position 2 would see an increase in the number of Nimda-related alarms of attacks originating from machines on the LAN (those against the Intranet *and* those against Billy Goat). A NIDS at position 3 would see attacks from both the LAN and the Intranet and would have greatly increased fidelity stemming from the fact that it does not see any legitimate traffic. We are actively exploring the benefits of these effects both on fidelity and on cost reduction owing to the expanded view that a Billy Goat offers a NIDS.

7.3 Failure Modes

We have observed an interesting failure mode in the default route and Router/ICMP modes of deployment. The problem occurs when a machine or network goes down and Billy Goat automatically starts responding for it. In this case, liveness checking mechanisms, such as ICMP echo (ping), yield deceptive results. This is especially problematic when these checking mechanisms are connected to other systems. In a testing phase, we accidentally induced the crash of a problem-ticketing system which got caught in the cross-fire between two automated systems: one asserting that the network was down, based on the truth, and the other asserting that it was up, based on the fact that the network seemed to be reachable. We take this failure mode, induced by a relatively passive system, as an interesting finding to keep in mind when developing automatic intrusion response systems.

8 Findings

Our deployment of Billy Goat sensors, both in large corporate networks and on the Internet, have afforded us a view into a mass of data about worm-infected machines that is rarely available elsewhere. In this section we share some of our observations.

8.1 Worm flare-ups

Confirming previous findings [19], we routinely see flares of old worms, such as Code Red, Nimda and Sapphire [7]. These reappearances are often brief, due to monitoring and blocking mechanisms that are now aware of them.

We observe that the frequency of such recurrences seems affected by the propagation vectors that each worm uses. For example, Sapphire uses MS-SQL, which is often turned on automatically by other programs (e.g. MS Visio 2000 and MS Project), so it is likely for previously deployed machines to become vulnerable.

8.2 Noise

Billy Goat has been deployed in IBM and customer networks. In addition to identifying worm traffic, we have observed behaviors that vary from network to network. In some networks we have been able to track the causes of these behaviors to misconfigured devices, poorly written software, or other explainable activity. In these cases, the detection of noise is a useful side effect of a Billy Goat deployment, which contributes to general network management and health monitoring.

Some notable examples of explainable phenomena are:

- Machines with an incorrect DNS server configuration, which happens to be in a spoofed range.

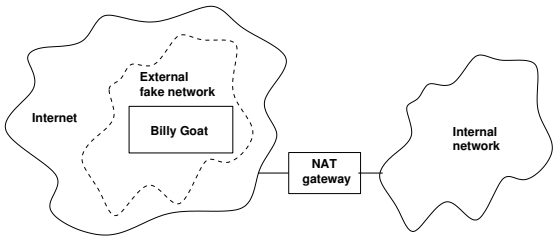

Figure 7: Double-routing scheme for external deployment.

We often observe repeated DNS requests to a specific address, from a single address.

- Broken software implementations that result in network scanning. For example, in one occasion we detected a Linux machine that was scanning the network on SMB ports. Upon examination, it was discovered that an SMB client was running on it, and it was scanning the network trying to find suitable servers, instead of using broadcast packets.

8.3 Internet deployments

We have installed a number of Billy Goat sensors on the Internet for purposes of stress-testing, debugging and data capture. Some of these sensors have been configured to mirror address spaces used in internal corporate networks that are connected to the Internet via NAT (hence, traffic to or from these address spaces should normally not be seen on the Internet).

Such a mirrored deployment is illustrated in Figure 7. It results in an address range that exists both on the Internet (where it is routed to the external Billy Goat sensor) and on the internal network (where it is routed to real machines), in a manner similar to the numerous independent instances of private address ranges [14] (e.g. 10.0.0.0/8). Beyond the expected worm traffic observed on the Internet, this setup has offered us a view of some interesting behaviors, including the following:

- We often see "ghosts" of internal network structures, such as machines on the Internet attempting to connect to internal email servers. This is explained by application caching of DNS information on machines that change networks either physically (laptops brought home and connected to the Internet) or virtually (disconnected VPNs).
- We have also observed unexplained traffic to internal addresses. For example, responses to peer-to-peer traffic, directed to addresses that are not assigned (even internally). One possible explanation for this behavior is that someone is using an internal address range on a separate private network, behind a NAT gateway. If some machine in this private network is identifying itself with its internal address, its peers on the Internet would be attempting to respond to that address, which was previously not routed, but which now goes to the external Billy Goat sensor. This raises questions about the accuracy of identifying and tracing peer-to-peer users through participation in the peer-to-peer protocols.

9 Conclusions and future work

We have built and widely deployed Billy Goat, a worm-detection system specialized in detecting network-service worms. Billy Goat has been designed to be scalable, to operate gracefully in a large distributed environment, and to provide extremely accurate detection of worm-infected machines.

Beyond the base description of Billy Goat, the contributions of this paper include:

- deployment modes for honeypots, in particular the new ICMP-based mode;
- environmental effects of Billy Goat deployements including the effect upon NIDS;
- potential failures of other systems induced by Billy Goat;
- the double routing of an IP address space on the Internet and an intranet connected via NAT, leading to the observation of two varieties of "ghosts": those manifested from internal machines and those unexplained.

Finally, we have described some Billy Goat constructs useful in coherently maintaining a large distributed intrusion detection system:

- cryptographic checksums as external database references,
- doubly-centralized configuration scheme.

9.1 Future work

We are currently studying the use of data mining techniques towards automatic signature generation and automatic creation and deployment of feigning servers. For example, if the IP traffic data shows a marked increase in connections to a certain port where no server currently exists, a generic listener for that port could be automatically instantiated and distributed to all the Billy Goat sensors, to capture the payload being sent to that port. This could greatly

reduce the reaction time in the face of a new worm outbreak, and aid in worm capture.

Finally, having a sensor that produces no false positives for a certain class of attacks might make possible the long-standing dream of intrusion detection: an automated response system. Most such systems to date have been marred by false positives, which often result in the response system causing more damage than good. We are exploring automatic intrusion response mechanisms based on Billy Goat data, with the aim of building a system that accurately and efficiently isolates misbehaving machines, while allowing critical technical and business processes to continue unimpeded, and with an extreme focus on potential failure modes, and how they might be eliminated or mitigated.

References

[1] ArcSight. ArcSight Enterprise Security Manager — product overview, Mar. 2005. URL `http://arcsight.com/product.htm`.

[2] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The Internet Motion Sensor: A distributed blackhole monitoring system. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium*. Internet Society, Feb. 2005. URL `http://www.isoc.org/isoc/conferences/ndss/05/proceedings/papers/ims-nds%s05.pdf`.

[3] M. Josefsson, J. Kadlecsik, H. Welte, J. Morris, M. Boucher, and R. Russell. The netfilter/iptables project. Web page at `http://www.netfilter.org/`, Mar. 2005.

[4] T. Kramp. BeepLite networking layer, Jan. 2004. URL `http://www.alphaworks.ibm.com/tech/beeplite`.

[5] J. Levine, R. LaBella, H. Owen, D. Contis, and B. Culver. The use of Honeynets to detect exploited systems across large enterprise networks. In *Proceedings of the 4th IEEE Information Assurance and Security Workshop*, West Point, NY, USA, June 2003. URL `http://www.tracking-hackers.com/papers/gatech-honeynet.pdf`.

[6] D. Moore. Network telescopes: Observing small or distant security events. In *11th USENIX Security Symposium 2002*, Aug. 2002. URL `http://www.usenix.org/publications/library/proceedings/sec02/tech.html`. Invited talk.

[7] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer worm. *IEEE Security & Privacy*, 1(4):33–39, Aug. 2003. URL `http://www.computer.org/security/v1n4/j4wea.htm`.

[8] J. Morris. Fighting worms in a large corporate environment: A design for a network anti-worm solution. In *Proceedings of the 12th Virus Bulletin International Conference 2002*, pages 56–66, 2002. System available at `http://smb-lure.dnsalias.com/`.

[9] J. Nazario. *Defense and Detection Strategies against Internet Worms*. Artech House Publishers, 2003.

[10] M. Overton. Worm charming: taking SMB Lure to the next level. In *Proceedings of the Virus Bulletin International Conference 2003*, Sept. 2003. URL `http://arachnid.homeip.net/papers/VB2003-Worm_Charming.pdf`.

[11] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of internet background radiation. In *Proceedings of the Internet Measurement Conference 2004*. ACM, Usenix, Oct. 2004. URL `http://www.icir.org/vern/papers/radiation-imc04.pdf`.

[12] N. Provos. Honeyd — A virtual honeypot daemon. In *Proceedings of the 10th DFN-CERT Workshop*, Hamburg, Germany, Feb. 2003. URL `http://niels.xtdnet.nl/papers/honeyd-eabstract.pdf`.

[13] N. Provos. A virtual honeypot framework. In *Proceedings of the 11th USENIX Security Symposium*. USENIX, Aug. 2004. URL `http://www.citi.umich.edu/techreports/reports/citi-tr-03-1.pdf`.

[14] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets. RFC 1918, Network Working Group, Feb. 1996.

[15] M. T. Rose. *BEEP: The Definitive Guide: Developing New Applications for the Internet*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 2002. ISBN 0-596-00244-0. URL `http://www.oreilly.com/catalog/beep`.

[16] Samba. Web pages at `http://www.samba.org`, 2005.

[17] L. Spitzner. The Honeynet Project: Trapping the hackers. *IEEE Security & Privacy*, 1(2):15–23, Mar./Apr. 2003. ISSN 1540-7993. URL `http://csdl.computer.org/comp/mags/sp/2003/02/j2015abs.htm`.

[18] L. Spitzner. Honeypots: Definitions and value. `http://www.tracking-hackers.com/papers/honeypots.html`, May 2003.

[19] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*. USENIX, Aug. 2002. URL `http://www.icir.org/vern/papers/cdc-usenix-sec02/`.

[20] P. Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.

[21] Tivoli. Tivoli SecureWay Risk Manager — product overview. White paper, IBM, Dec. 2000. URL `ftp://ftp.software.ibm.com/software/tivoli/whitepapers/sway_risk_mgr_ov%erv.pdf`.

[22] S. Whalen. An introduction to ARP spoofing. *2600 Magazine*, Fall 2001. URL `http://www.node99.org/projects/arpspoof/`.

[23] V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of internet sinks for network abuse monitoring. In E. Jonsson, A. Valdes, and M. Almgren, editors, *Proceedings of the Recent Advances in Intrusion Detection 2004 workshop*, Lecture Notes in Computer Science, pages 146–165. Springer, Sept. 2004.