

Volleystore: A Parasitic Storage Framework

Kurt Rosenfeld, Husrev Taha Sencar, Nasir Memon
CIS Department
Polytechnic University, Brooklyn, NY 11201

Abstract— We present Volleystore, a filesystem that stores data on network equipment and servers without any authorization, yet without compromising the systems that are used. This is achieved by exploiting the echo functionality present in most standard Internet protocols. Various issues concerning the design of a parasitic storage system are addressed and a practical system based on Internet Control Message Protocol (ICMP) echo messages is demonstrated. We present an analysis of storage capacity and latency limits for various configurations of the system. We also describe a proof-of-concept implementation of the system and show that one can indeed store data using Volleystore for a reasonable lengths of time. Finally, we suggest defenses against parasitic storage, both abstractly in economic terms and concretely in technical terms.

I. INTRODUCTION

Storage is a resource that has value. One of the main goals of security is to maintain control and accounting of resources. Parasitic storage gives the user (the attacker) storage for free, at the expense of others. In essence, poor resource accounting allows goods or services of value to be taken without being paid for. Any system that provides this invites exploitation. Parasitic storage exploits the willingness of Internet-connected systems to echo data back to the sender, possibly after holding it for some time. This echo functionality is usually included by system designers to serve some kind of diagnostic or handshaking purpose. Most current communication protocols including those used on the Internet depend on this at some level. Disabling this behavior is not an option in current protocols. (e.g., TCP three-way handshake is mandatory.) Although the techniques we discuss are harmless in small quantities, widespread deployment of parasitic storage would deplete the resources it uses. We consider the use of these techniques a serious threat, an attack somewhat more dangerous than mere theft of service.

A parasitic storage system can be used for a number of purposes, particularly for storage of sen-

sitive data. Current storage systems secure data by encrypting the data on the disk and on the wire. Although the general framework that governs the design and evaluation of such systems considers a wide variety of issues [3], these approaches cannot meet all types of secure data storage needs. For example, in certain cases, the existence of encrypted data on a disk may pose a potential threat to its owner, regardless of whether anyone can decrypt it. Similarly, under certain circumstances, it may be desirable that the stored data have a natural lifetime such that the stored data cease to exist after a finite time through mechanisms that are not primarily designed to obliterate data.

Conventional disk encryption systems cannot conceal the presence of encrypted data on a disk. In the presence of compulsive mechanisms or under coercive interrogations, such systems cannot ensure the security of data since the owner might be forced to disclose the decryption keys. In order to maintain deniability while storing data, such that without the relevant keys it is impossible to prove whether encrypted data exists, various deniable file systems have been proposed [7], [9], [8]. Essentially, the crux of a deniable file system lies in creating encrypted data that is indistinguishable from random data. However, despite the ingenious use of cryptographic primitives to generate multiple encryption layers (whose existence can be plausibly denied by only revealing parts of a secret) deniable file systems are prone to statistical analysis aiming at discriminating the hidden encrypted layers. In this regard, parasitic storage of data offers a degree of deniability as it deploys the computer in a configuration that uses remote storage, but without a pre-established agreement with any particular file server. The highly distributed nature of the storage medium, the lack of a static and dedicated storage disk space, and the relatively low overhead of the system makes it an appealing choice as a secure storage system.

Another important aspect of a secure storage system is the method of data erasure. Most storage systems today rely on disks as a storage medium. Typically, the data on a disk is extremely difficult to destroy physically. This property may be well suited for many applications, but for security applications this is troublesome. The problem is further exacerbated by the fact that almost all operating system commands designed to delete data or format disks only free up the space that the deleted files consumed and do not, in fact, attempt to remove all of the data. That is, most of the actual data remain on the disk. Although these data might be encrypted and their compromise might not pose an immediate threat, it might later create unnecessary risks. In this regard, a secure storage system should deploy mechanisms (e.g., commands, utilities) that are specifically designed to sanitize data. A parasitic storage system inherently has these properties.

In this work, we study the threat of parasitic storage by analyzing its requirements and capabilities. We describe a secure storage system prototype that is opportunistic in its discovery of potential storage resources. The focal point of our Volleystore model is the Volleystore server, which abstracts accessible storage resources to provide a uniform simple storage interface to clients. The most important two characteristics of this abstraction are that

- The heterogeneous nature of the available storage resources is transformed into a array of uniform storage blocks.
- The short-lived and unreliable nature of the available storage resources is transformed into persistent reliable semantics similar to those of a disk drive.

The proposed parasitic storage framework utilizes some of the standard Internet protocols for storage purposes. Due to its nature, however, a parasitic storage system poses new challenges. For example, each message that is sent out to a reflector has a nonzero probability of being lost. Furthermore, the delays, and therefore the maximum access latencies of the different channels vary widely. Hence, the goal of a parasitic storage system is to unify the storage provided by many disparate channels into a single practical virtual block storage device with reasonable performance.

With this perspective, in this paper, we propose a framework and an analysis methodology to construct parasitic storage systems that exploit the built-in echo functionality of standard protocols

like Internet Control Message Protocol (ICMP)[5], Simple Mail Transfer Protocol, and others. Moreover, we analyze and provide an implementation of parasitic storage system based on ICMP echo messages. In order to obtain theoretical storage capacity limits and latency restrictions in accessing the stored data, we establish a duality between parasitic storage systems and conventional communication systems. In our implementation, a combination of error coding and caching is used to provide high reliability and low average access latency.

II. RELATED WORK AND OUR APPROACH

The use of delay lines as persistent data storage has its root in the EDVAC computer of the 1940's. In his description of this machine John von Neumann wrote [10]:

[The memory components] are essentially delays, which hold a stimulus that enters their input for a time k_t , and then emit it. Consequently they can be converted into cyclical memories, which hold a stimulus indefinitely, and make it available at the output at all times which differ from each other by multiples of k_t . It suffices for this purpose to feed the output back into the input.

Purczynski and Zalewski established the technical foundation of Internet-based parasitic storage in their 2003 Bugtraq posting.[6] They classify parasitic storage resources as being in class A, memory buffers, or class B, disk queues. They estimate the storage capacity at 7TB for a user with 100Mbps Internet link. The paper considers the use of ICMP, HTTP, and SMTP. The issue of reliability and redundancy is not discussed in depth. Zalewski discussed parasitic storage further in 2005.[11]

Barabasi Freeh, Jeong, and Brockman[1] demonstrated in 2001 the possibility of parasitic computing on the Internet. The TCP checksum calculation functionality that is mandated by Internet standards was exploited to perform distributed computation. Their result demonstrated that, in principle, any computation can be offloaded to a set of unaware Internet-connected machines which will perform the calculation for the benefit of the exploiter. The authors clearly state that the attack, in its present form, does not pose a threat, because the overhead outweighs the benefit.

The design space of parasitic storage is a three-way tradeoff between storage capacity, data lifetime, and access speed. There is a necessary tradeoff between the amount data we can store and how quickly it can be accessed. The link capacity of our

network interface is a limited resource which must be divided between by the various data items circulating through the system. Low-delay channels enable low-latency access to data. The delay of the channel provides an upper bound to the access latency. But the use of low-delay channels carries a high cost since it requires frequent volleying of the message, which consumes more link capacity. On the other hand, although high-delay channels allow the system to store more data, the high delay brings with it high latency. Thus there is a tradeoff between storage capacity and access latency.

A parasitic storage system stores data by distributing it over many network elements and nodes. Therefore, it has to provide guarantees regarding the availability, integrity and confidentiality of the stored data. The performance goals that a parasitic storage system strives to achieve can be summarized as follows

1. (*Storage capacity*) Maximum amount of data that can be stored in the system with very low probability of data loss. Storage capacity depends on the error characterization of the protocol which mostly reveals itself in the form of erasures. For example, when ICMP echo messages are used for storage, the main concern is the packet drop rate due to network state. On the other hand, for SMTP it is the administrative configuration settings which might or might not erase an attachment before bouncing a message.
2. (*Availability*) Maximum latency of the system to reconstruct the stored data. Availability of a system can be controlled by increasing the redundancy in the system. Typically, this can be achieved by replication of data or employing denser error correction codes.
3. (*Confidentiality*) Percentage of information revealed if storage nodes are compromised. The use of conventional encryption mechanisms can insure data confidentiality if and when storage nodes are compromised. This can also be achieved through the use of secret sharing schemes which improve the availability of the system as well. [4] Another aspect of confidentiality is that an eavesdropper should not be able know the amount of data stored in the system. This can be controlled through adjusting the redundancy rate of the system.

We will make these requirements more formal in Section 3.

III. ANALYSIS

A parasitic storage system uses standard Internet protocols to store information. Essentially, the

storage capability is due to the time delay it takes information sent by a node to be processed and reflected back to originating node. In its simplest form, this operation involves a *server node*, a *reflector node* and a *path* comprising some sequence of systems and links that connect the server and reflector nodes. Given the error-prone nature of Internet, due to packet loss from congestion and network faults, such a system requires mechanisms to detect and correct errors. In this respect, a parasitic storage system can be viewed as a conventional communications system wherein a transmitter relays information to a receiver through a noisy channel. The statistical characterization of the error process in the channel and the limitations concerning the use of the channel (e.g., power, bandwidth, complexity) are central in determining the performance limits achievable by the communication system. A similar methodology can be extended to study parasitic storage systems where the server node acts as both the transmitter and receiver. The behavior of the reflector node and the nodes along the path constitute the channel, and the primary performance goals are expressed in terms of *parasitic storage capacity* and *latency limits*.

A thorough analysis of a parasitic storage system depends on the specifics of the utilized Internet protocol. Considering the case where ICMP echo (ping) messages are used as the basis of storage, the fundamental parameters that determine the achievable storage capacity and latency bounds can be characterized in terms of the following:

- payload of the messages,
- error rate on the path that connects the server and reflector nodes,
- the speeds at which the two nodes can input and output data, and
- the round-trip time between the two nodes.

It should be noted that in this system, non-malicious errors occur only in the form of data erasures which are due to congestion at relaying nodes. In other words, the server node either retrieves the stored data (payload) successfully or loses it. We now define these ideas more formally.

DEFINITION 1 *A parasitic storage channel, involving a server node S and reflector node R , is a time-delayed discrete memoryless erasure channel with three attributes, denoted by $(\gamma_S, rtt_{S,R}, \gamma_R)$ where γ is the input/output throughput rate of a node and rtt is the round-trip delay time between two nodes. The erasure channel consists of a finite input set*

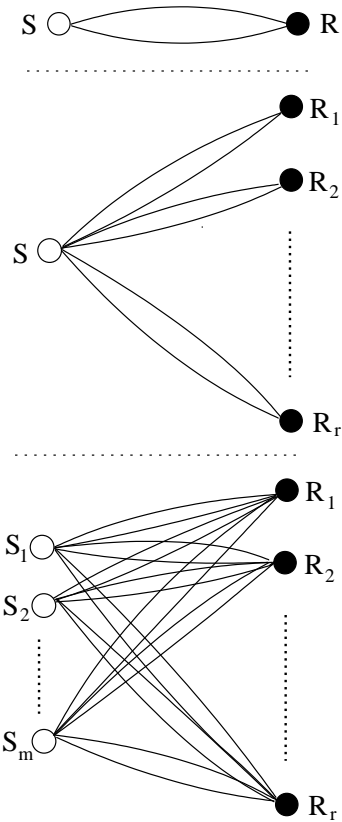


Fig. 1. Three settings for parasitic storage system.

and a collection of probability values which characterize the erasure probability, P_e , for all elements of the input set. The server node S controls the input and output of the discrete channel and stores data through repeated use of this channel, for which the maximum number of use per unit time is constrained by $(\gamma_S, rtt_{S,R}, \gamma_R)$.

The fundamental building block of a parasitic storage system is the parasitic storage channel. Therefore, the achievable amount of storage and the latency in accessing the stored data in such a system can be obtained based on the analysis of a parasitic storage channel(s). For this, we assume without loss of generality that the input finite set is limited to 2^n elements (i.e., n -bit symbols), P_e for each symbol is fixed, and rtt is the *delay threshold* above which the packet will be considered erased. We consider the following three settings as displayed in Figure 1.

A. Single Server and Reflector Nodes

The parasitic storage server has access to a single storage channel described by $(\gamma_S, rtt, \gamma_R)$ and a fixed probability of erasure P_e . To store data, S

sends the n -bit payload (packet) to R and keeps volleying it back and forth to R every rtt until the packet needs to be accessed. However, due to errors in the channel the packet may sometimes be erased, leading to loss of data. To be able to deal with such errors, the system has to have built-in redundancy. In essence, this requires transmission of multiple packets to enable packet loss recovery. The number of packets that can be volleyed continuously between two nodes essentially depends on the minimum of the packet throughput rates γ_S and γ_R since S and R cannot send and receive more packets than what their network connections can handle.

The information capacity of a discrete channel determines the maximum average number of bits that can be sent per channel use with arbitrarily low probability of error. In an erasure channel, each transmitted symbol is erased with probability P_e . The erasure channel has 2^n inputs, all possible n -bit message symbols, and $2^n + 1$ outputs which also includes an erasure symbol as displayed in Figure 2. The capacity of this channel can be computed as [2]

$$C = (1 - P_e) \times n \text{ bits/symbol} \quad (1)$$

which is achieved when each of the 2^n symbols are equally likely to be transmitted. Intuitively, this expression states that since a fraction P_e of the symbols are lost in the channel, at most, a fraction $1 - P_e$ of the transmitted symbols can be recovered. This can be realized by the use of channel coding whose aim is to make a noisy channel behave like a noiseless one via controlled addition of redundancy.

Associating the information capacity result with the storage capacity of a parasitic storage channel requires consideration of additional issues concerning the use of the erasure channel for storage purposes. Storage of a packet requires continuous re-transmission of the packet through the channel. Noting S and R can only send and receive packets at finite rates of γ_S and γ_R , the speed of communication between S and R (in packets per second) is determined by the minimum of γ_S and γ_R . Since rtt is the time delay it takes for a packet to get back to S , its storage requires at least $\frac{1}{rtt}$ retransmissions per second. Hence, the maximum number of distinct packet flows that can be supported between S and R is computed as

$$k = \frac{\min(\gamma_S, \gamma_R)}{\frac{1}{rtt}}. \quad (2)$$

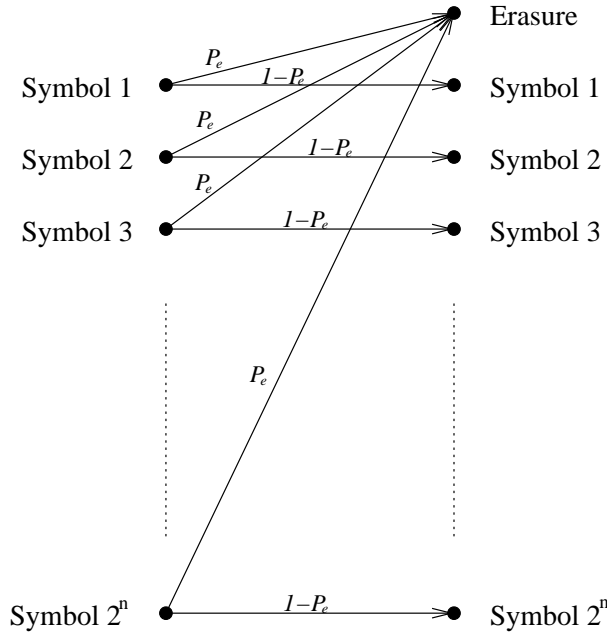


Fig. 2. Erasure channel.

In other words, when the erasure probability is negligible, the achievable storage for this channel is k packets, i.e., $k \times n$ bits. However, due to packet erasures, storage capacity of the channel is lower.

The communication capacity C of the erasure channel indicates the average number of bits that can be sent with each use of the channel. Hence, for k uses of the channel the reliably received amount of information cannot be higher than $k \times C$ bits. When interpreted in the context of storage, this implies that out of every k packets sent, at most $k \times C$ bits can be recovered and retransmitted, yielding storage for only $k \times C$ bits. Consequently, the storage capacity C_s of a parasitic storage channel can be expressed as

$$C_s = rtt \times \min(\gamma_S, \gamma_R) \times (1 - P_e) \times n \text{ bits} \quad (3)$$

This result follows the intuition that storage capacity increases with increasing time delay in the channel and decreasing erasure rate. The other major concern is the latency in accessing the stored data. The worst access time for a packet will occur when the packet is lost and S has to wait other packets to reconstruct it. Since rtt is the threshold delay allowed before a packet is considered lost and the time-delay for a packet to arrive at S , this would take at most $2 \times rtt$. It should be noted that if, at the end of second rtt cycle, redundant data packets are not available, permanent loss of the packet would result on account of a transmis-

sion error in the channel. Hence, the access time for all the data in a parasitic storage channel is upper bounded by $2 \times rtt$.

B. Single Server Node and Multiple Reflector Nodes

The parasitic storage system is composed of multiple storage channels established between a server node S and reflector nodes R_1, \dots, R_r . Each channel is described by $(\gamma_S, rtt_j, \gamma_{R_j})$ and by its probability of erasure P_{e_j} , for $1 \leq j \leq r$. Storage capacity of the system can be obtained by proper throughput allocation. In this regard, there are two possible cases depending on whether or not the throughput of server node is larger than the total throughput of the reflector nodes.

When the throughput γ_S of the server node is larger than the aggregate throughput $\sum_{j=1}^r \gamma_j$ of reflector nodes, S can establish storage channels with all R_j , $1 \leq j \leq r$ and the storage capacity is the sum of the capacities of individual parasitic storage channels obtained as

$$C_s = \sum_{j=1}^r C_{s_j}, \quad (4)$$

where

$$C_{s_j} = rtt_j \times \gamma_{R_j} \times (1 - P_{e_j}) \times n \text{ bits}. \quad (5)$$

Alternatively, γ_S might be smaller than $\sum_{j=1}^r \gamma_j$. In this case, throughput of S cannot support all of the available storage channels and, therefore, it has to allocate its throughput among the *best* storage channels to maximize storage capacity, which can be written as

$$C_s = \sum_{j=1}^r C_{s_j}^*, \quad (6)$$

where

$$C_{s_j}^* = \gamma_j^* \times rtt_j \times (1 - P_{e_j}) \times n, \quad (7)$$

such that

$$\gamma_S = \sum_{j=1}^r \gamma_j^*. \quad (8)$$

Hence, the problem is reduced to finding throughput allotment that maximizes the storage capacity subject to the constraint that $\gamma_S = \sum_{j=1}^r \gamma_j^*$. Since $C_{s_j}^*$ increases linearly with the throughput γ_j^* , S has to utilize channels in order of decreasing $rtt_j \times (1 - P_{e_j})$ as long as its throughput permits.

Let (i_1, \dots, i_m) be a set of ordered indices such that i_1 is the index of the channel that yields the highest $rtt_{i_1} \times (1 - P_{e_{i_1}})$ product and $\gamma_{R_{i_1}}$ is the corresponding throughput of the associated reflector node. Then, corresponding γ^* values can be obtained as

$$\gamma_{i_j}^* = \begin{cases} \gamma_{R_{i_j}}, & \text{if } 1 \leq j \leq t, \text{ s.t.} \\ & \sum_{l=1}^{t+1} \gamma_{R_{i_l}} > \gamma_S \geq \sum_{l=1}^t \gamma_{R_{i_l}} \\ \gamma_S - \sum_{l=1}^t \gamma_{R_{i_l}}, & \text{if } j = t + 1, \\ 0, & \text{if } t + 2 \leq j \leq m. \end{cases} \quad (9)$$

It should be noted that the storage capacity in the former case, (4), is limited by the throughput of the reflector nodes, whereas in the latter case (9) it is due to limitation on the server node. On the other hand, the latency in accessing stored packets depends on highest rtt among all utilized channels, i.e.,

$$\max_j 2 \times rtt_{i_j}, \quad 1 \leq j \leq t. \quad (10)$$

C. Multiple Server and Reflector Nodes

The parasitic storage system involves m server nodes, S_1, \dots, S_m , and r reflector nodes, R_1, \dots, R_r . The resulting $m \times r$ possible storage channels are described by parameters $(\gamma_{S_i}, rtt_{i,j}, \gamma_{R_j})$ and probability of erasure values $P_{e_{i,j}}$ where $1 \leq i \leq m$ and $1 \leq j \leq r$. Since the main limitation on channel use is the available throughput, all of the channels cannot be used and *best* storage channels have to be determined. The achievable storage capacity for this system can be obtained iteratively by selecting the best channel at each step and adding up all the individual storage capacities until all the available throughput for sender or reflector nodes is fully utilized.

Let Γ_S and Γ_R be two vectors consisting of throughput rates for all nodes, i.e., $\Gamma_S = (\gamma_{S_1}, \dots, \gamma_{S_m})$ and $\Gamma_R = (\gamma_{R_1}, \dots, \gamma_{R_r})$, and storage capacity $C_{s_{i,j}}$ between S_i and R_j be defined as

$$C_{s_{i,j}} = rtt_{i,j} \times \min(\gamma_{S_i}, \gamma_{R_j}) \times (1 - P_{e_{i,j}}) \times n \text{ bits}. \quad (11)$$

Then, among all the possible storage channels, the channel that yields the highest storage is selected based on current Γ_S and Γ_R as

$$(i', j') = \arg \max_{i,j} \left\{ \begin{array}{l} C_{s_{i,j}}, \text{ such that} \\ \gamma_{S_i} \in \Gamma_S \text{ and} \\ \gamma_{R_j} \in \Gamma_R. \end{array} \right\} \quad (12)$$

yielding the highest storage capacity of $C_{s_{i',j'}}$. In determining the next best channel, the utilized

part of the throughput have to be excluded. That is, the throughput values in Γ_S and Γ_R have to be updated after each step as

$$\gamma_{S_i}^* = \begin{cases} 0, & \text{if } \gamma_{S_i} = \min(\gamma_{S_i}, \gamma_{R_j}), \\ \gamma_{S_i} - \gamma_{R_j}, & \text{if } \gamma_{S_i} = \max(\gamma_{S_i}, \gamma_{R_j}), \end{cases} \quad (13)$$

and

$$\gamma_{R_j}^* = \begin{cases} 0, & \text{if } \gamma_{R_j} = \min(\gamma_{S_i}, \gamma_{R_j}), \\ \gamma_{R_j} - \gamma_{S_i}, & \text{if } \gamma_{R_j} = \max(\gamma_{S_i}, \gamma_{R_j}). \end{cases} \quad (14)$$

(It should be noted that this cannot be realized by simply subtracting the minimum throughput from Γ_S and Γ_R since storage capacity depends on other parameters as well, see (3).) This procedure is repeated until either Γ_S or Γ_R contains all zeroes, which implies that all server or reflector nodes are sending and receiving packets maintaining full link utilization. Hence, the overall storage capacity is the sum of storage capacities of the channels selected at each step as

$$C_s = \sum_{i',j'} C_{s_{i',j'}}. \quad (15)$$

On the other hand, the latency in accessing a stored packet of data is upper-bounded by the parasitic storage channel that has the highest time delay and delay threshold, similar to (10).

IV. IMPLEMENTATION DETAILS

An ICMP-based parasitic storage system was implemented in our lab and extensively tested on the Internet. The implementation, which we call *Volleystore*, consists of two programs. These two programs, *volleyctl* and *volleybot* are the client and server for *Volleystore*. They communicate with each other over IP using a custom stateless protocol. The system runs on FreeBSD and uses the PCAP library for packet I/O. The kernel IP stack is not used at all.

The implementation distinguishes between *storage blocks* and *volley blocks*. The storage blocks are the chunks of user data. The volley blocks are the actual packet payloads that traverse the network. Storage blocks are expanded using ECC or replication to become volley blocks. The loss of volley blocks is expected. The loss of storage blocks is strongly avoided.

A. *Volleyctl: The Volleystore Client*

The *Volleystore* system is controlled by *volleyctl*. *Volleyctl* supports a fixed set of methods including INJECT DATA, RETRIEVE DATA, DELETE DATA, GET STATISTICS, and TIMECHECK.

The INJECT DATA method reads storage blocks from userspace, chunks them into 1KB storage blocks, and packs them into the payload of ICMP echo request packets. Each of these ICMP packets is tunneled to volleybot in an IP packet, along with some metadata such as the block number of each injected block.

RETRIEVE DATA forwards a request block request to volleybot, receives the data, and delivers it to the user.

DELETE DATA sends volleybot a deletion request for a block. The deleted block is referenced by block number. A deletion request simply causes the server to erase its metadata regarding the deleted block. When a deleted block arrives at volleybot, it is treated as an alien packet and discarded. Maliciously modified or externally injected packets are discarded by the same mechanism.

GET STATISTICS allows the parasitic storage user to inquire about the health of the system. Under normal operation, the loss of volley blocks is handled by volleybot and these low-level loss events are not propagated up to the user. But for diagnostic purposes the GET STATISTICS method is available.

The TIMECHECK method is also available for diagnostic purposes.

B. Volleybot: The Volleystore Server

In our implementation, packets are recirculated and maintained on the wire by the Volleystore server program, volleybot. In addition to listening for control packets from volleyctl, volleybot listens for ICMP packets returning from reflectors. When these packets arrive, their payload verified against a locally stored checksum, packed into the payload of a new ICMP echo request packet, and sent back to the reflector. This never takes longer than 80 microseconds.

When a volleyblock has not been seen for more than $3rtt$, a recovery is initiated. Two error recovery schemes were implemented: parity and replication. In either case, the lost volley block is recovered from a set of other volley blocks.

When using the parity scheme, we break each storage block into two volley blocks and generate a third volley block which is the byte-wise XOR of the first two. Any two of these volley blocks can be XOR'ed together to get obtain the third volley block. When, a timeout (erasure) is detected, we use a volley block-sized buffer for reconstruction. First the reconstruction buffer is zeroed. When one of the needed blocks arrives, we XOR it into

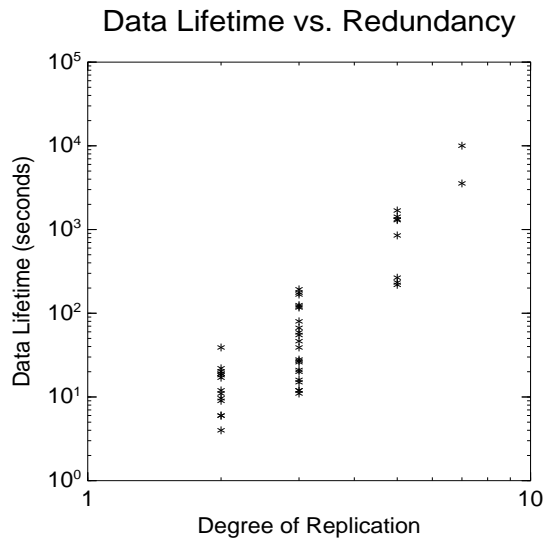


Fig. 3. Blocks were injected into the system with redundancy factors 2, 3, 5, 7, and 8. The mean rtt was 250 msec. The lifetime of the data was observed across several trials. Data loss was not observed for redundancy factors greater than 7.

the reconstruction buffer. When the second needed block arrives, we XOR it into the reconstruction buffer. At that point the reconstruction buffer contains the lost volley block. Then the lost volley packet is reconstructed and injected back into the network channel.

When replication is being used, the recovery process is similar to parity, but even simpler. Instead of waiting until we see all of the other volley packets that constitute the degraded storage block, we only need to wait until we see any one of the other blocks.

The storage overhead for volleybot has not been optimized at all and is currently 21 bytes per volleyblock. For 1024-byte volleyblocks this is 2.3%. We expect that this overhead can be significantly reduced without affecting any other aspects of the system. When the parity scheme is being used, a volley block-sized reconstruction buffer is also needed. When using replication, this buffer is not needed.

V. RESULTS AND DISCUSSION

A. Lifetime of Data

Measurements of the lifetime of the data stored in parasitic channels indicate some redundancy is needed in order to obtain practical data lifetimes. For redundancy factors above 8 we were not able to observe any data loss. However, if the Internet link of the volleybot node were to be inter-

rupted for more than a few hundred milliseconds, no amount of error correction coding could keep the ICMP-based volleys alive. To make a parasitic storage system robust against interruptions in connectivity and to expand the storage capacity of the system, the designer would want to use what Wojciech and Zalewski refer to as class B storage[6], exploiting the willingness of remote systems to use their hard drives to store data without requiring any authorization. This suggests a sort of memory hierarchy, using class A storage as a cache for the class B storage. This technique would be of use for a workload consisting mainly of reads. Writes are problematic since a write-back cache would not be able to guarantee that data would not be lost, while a write-through policy would perform poorly do to the higher latency in class B storage channels. The main source of class B storage in current Internet protocols is SMTP.

B. Preventing Parasitic Storage

Parasitic storage is a threat to current Internet-connected systems. The widespread exploitation of this latent storage capacity would overload victim systems. Current systems cannot deny parasitic storage without also forgoing important functionality.

Abstractly, the solution is to employ strict accounting. If everything that is provided is paid for, exploitation of latent resources cannot take place.

A technical solution might be to modify common protocols so they don't echo data back to the sender without positive authentication first. Another technical solution is to modify common protocols to force the initiator (the parasite) to store one token block for each volley block. If the parasite has to store this token, it is a penalty that offsets any gain that the attacker might have obtained. If there is no net gain, there is no motivation. When the conditions that invite resource exploitation cease, the exploitation ceases.

We favor this second technical solution because it is reasonably easy to implement. It is vaguely related to TCP cookies.

To illustrate the way a protocol can be modified to resist parasitic storage, we present anti-parasitic modifications to ICMP. Specifically, we show how the ECHO REQUEST/REPLY functionality in ICMP can be maintained without providing any net gain for a user who wishes to obtain free storage.

Conventional ICMP [5] serves a number of diagnostic purposes. Among those is ping, which

is intended as a simple test for IP network functionality. An ICMP ECHO_REQUEST packet is sent from host A to host B. Host B sends an ICMP ECHO_REPLY packet to host A, which includes any payload data that host A included in the ECHO_REQUEST. The purpose of allowing payload data is to support diagnosis of problems that depend on packet size. The purpose of allowing arbitrary payload data, and echoing it back, is to support diagnosis of data-dependent link problems. Under normal circumstances, host A receives its payload data after some delay, the round trip time, which can range from less than 1 millisecond for hosts on the same LAN, to 500 milliseconds for hosts on opposite sides of the Internet. In practice, the payload can be up to 1024 bytes in length with no other constraints. Sending an ECHO_REQUEST packet with a 1024 byte payload to a host 500 milliseconds away results in 512 byte-seconds of parasitic storage.

Our anti-parasitic ping protocol works as follows.

1. A sends B an ECHO_REQUEST with n bytes of payload data, D.
2. B sends A an ECHO_REPLY with n bytes of random data, R.
3. A sends B another ECHO_REQUEST.
4. B sends A an ECHO_REPLY with n bytes of payload, which is D xor R.

If A has kept R between step 2 and step 4, then A can easily recover D. This supports the network testing purpose of ICMP/ping. However, if A discarded R, then A cannot recover D from the message it gets from B in step 4. So A cannot achieve a net gain of storage, since for every block of data that A is storing on the network, A is forced to store one random block locally. Thus the parasitic storage effect is canceled.

VI. CONCLUSIONS

We have outlined a vulnerability in most Internet-connected systems. An analytical framework for studying parasitic storage is presented. An implementation was built and studied, with reliability results shown. Finally, we have presented a direction toward deterring parasitic storage by canceling the gains that its user would achieve.

REFERENCES

- [1] Albert-Laszlo Barabasi, Vincent W. Freeh, Hawoong Jeong, and Jay B. Brockman. Letters to nature. parasitic computing., 2001.
- [2] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Series in Telecommunications. Wiley, 1991.

- [3] Erik Riedel, Mahesh Kallahalla, and Ram Swaminathan. A framework for evaluating storage system security. In *Proc. of 1st Conference on File and Storage Technologies (FAST)*, 2002.
- [4] Gregory R. Ganger, Pradeep K. Khosla, Mehmet Bakkaloglu, Michael W. Bigrigg, Garth R. Goodson, Semih Oguz, Vijay Pandurangan, Craig A. N. Soules, John D. Strunk, and Jay J. Wylie. Survivable storage systems. In *DARPA Information Survivability Conference and Exposition*, pages 184–195, Anaheim, CA, 2001.
- [5] J. Postel. Internet Control Message Protocol. RFC 792 (Standard), September 1981. Updated by RFC 950.
- [6] Wojciech Purczynski and Michal Zalewski. Juggling with packets. Available at http://isec.pl/papers/juggling_with_packets.txt.
- [7] R. Anderson, R. Needham, and A. Shamir. The steganographic file system. In *Information Hiding: Second International Workshop, IH'98. Proceedings*, volume 1525 of *LNCS*, page 73, 1998.
- [8] Rubberhose. User Documentation. Available at <http://iq.org/~proff/rubberhose.org/>.
- [9] Truecrypt. User Documentation. Available at <http://www.truecrypt.org/docs/>.
- [10] John von Neumann. The first draft of a report on the EDVAC. Technical report, University of Pennsylvania, 1945. Available at <http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf>.
- [11] Michal Zalewski. *Silence on the Wire: A Field Guide to Passive Reconnaissance and Indirect Attacks*. No Starch Press, San Francisco, CA, USA, 2005.