

# Outwitting the Witty Worm – Exploiting Underlying Structure for Detailed Reconstruction of an Internet-Scale Event

---

**Abhishek Kumar**

Georgia Institute of Technology  
akumar@cc.gatech.edu

**Vern Paxson**

ICSI  
vern@icir.org

**Nicholas Weaver**

ICSI  
nweaver@icsi.berkeley.edu

# Overview

---

- Network Telescopes
- Introduction to the Witty worm
- Linear Congruential Pseudo Random Number Generator (LC-PRNG)
- Breaking the State of the witty PRNG
- Measurements
- Estimating the seed of the PRNG
- More measurements
- Reconstruction of the Origin

## Network Telescopes

---

- Idea: monitor a cross-section of Internet address space to measure network traffic involving wide range of addresses
  - Backscatter from DOS floods
  - Attackers probing blindly
  - Random scanning from worms
- UCSD and UWisc's passive telescopes: 1/256 of entire Internet.
- Previous uses: Aggregate analysis of worms, their scan-rates, etc.
- For a worm outbreak, what can we learn from such measurements?
  - Beyond just which hosts were infected and how quickly

## NGC6543: A planetary nebula, also known as the Cat's Eye.

---

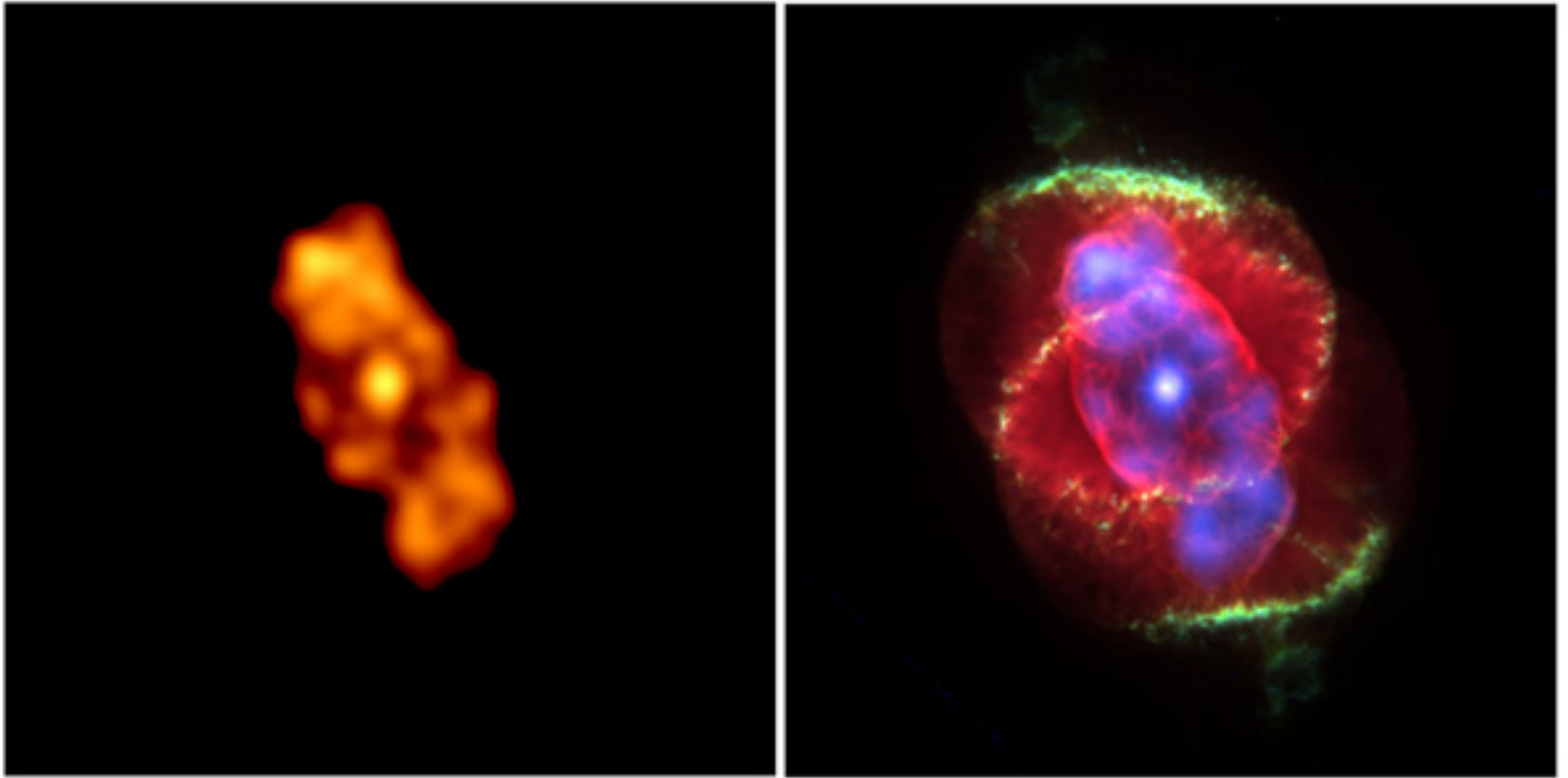


Figure 1: Chandra X-ray Observatory Center (<http://chandra.harvard.edu>).

## The Witty worm

---

- Released March 19, 2004.
- Exploited flaw in the passive analysis of Internet Security Systems products
- Worm fit in a single Internet “packet”
  - Stateless: When scanning, worm could “fire and forget”
- Vulnerable pop. (12K) attained in 75 minutes.
- Payload: slowly corrupt random disk blocks.
- Flaw had been announced the previous day.
- Written by a Pro.

## What does this worm do?

---

1. Seed the PRNG using system time.
2. Send 20,000 Copies of self to random destinations.
3. Open a physical disk chosen randomly between 0 & 7.
4. If success:
  5. Overwrite a randomly chosen block on this disk.
  6. Goto line 1.
7. Else:
  8. Goto line 2.

## Generating random numbers

---

The linear congruential generator (LCG) was proposed by D.H.Lehmer in 1948.

$$X_{i+1} = X_i * a + b \text{ mod } m$$

$$a = 214013$$

$$b = 2531011$$

$$m = 2^{32}$$

## Generating random numbers.

---

The linear congruential generator (LCG) was proposed by D.H.Lehmer in 1948.

$$X_{i+1} = X_i * a + b \bmod m$$

$$a = 214013$$

$$b = 2531011$$

$$m = 2^{32}$$

*Anyone who considers arithmetical  
methods of producing random  
digits is, of course, in a state of sin.  
- John Von Neumann (1951)*



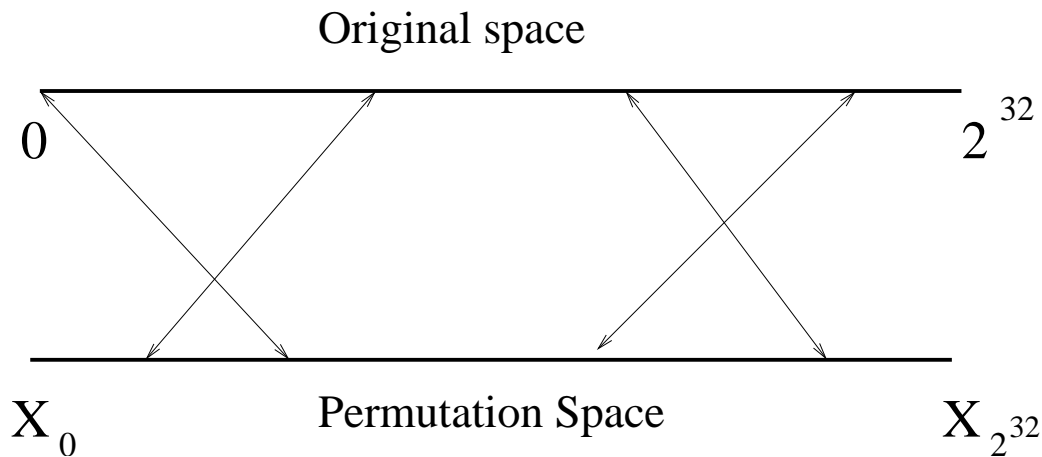
## Properties of the Linear Congruential PRNG

---

$$X_{i+1} = X_i * a + b \bmod m$$

$$a = 214013, b = 2531011, m = 2^{32}$$

- With the above values of  $a, b, m$ , the LC PRNG generates a permutation of all the integers in the range  $[0, 2^{32} - 1]$ .



## Pseudocode of Witty

---

```
 $a = 214013, b = 2531011;$   
rand(){  $X = X * a + b;$     return  $X;$ }  
srand(seed){  $X = seed;$ }  
main(){  
1.  srand(get_tick_count());  
2.  for( $i=0; i<20,000; i++$ )  
3.     $dest\_ip \leftarrow rand()_{[0...15]} || rand()_{[0...15]}$ ;  
4.     $dest\_port \leftarrow rand()_{[0...15]}$ ;  
5.     $packet\_size \leftarrow 768 + rand()_{[0...8]}$ ;  
6.     $packet\_contents \leftarrow top\ of\ stack;$   
7.    sendto();  
8.    if(open(physicaldisk,  $rand()_{[13...15]}$ ))  
9.      write( $rand()_{0...14} || 0x4e20$ );  
10.   goto 1;  
11.  else goto 2;}
```

## Breaking the State of the witty PRNG

---

- Recall that each packet contains bits from 4 consecutive random numbers.

3.  $dest\_ip \leftarrow rand()_{[0...15]} || rand()_{[0...15]}$ ;

4.  $dest\_port \leftarrow rand()_{[0...15]}$ ;

5.  $packet\_size \leftarrow 768 + rand()_{[0...8]}$ ;

- If the first call to  $rand()$  returned  $X_i$ , then:

$$dest\_ip = X_{i,[0...15]} || X_{i+1,[0...15]}$$
$$dest\_port = X_{i+2,[0...15]}$$

- Solve for  $X_i$  using,  $X_{i+1} = X_i * a + b \bmod m$  and  $X_{i+2} = X_{i+1} * a + b \bmod m$ .

- Simple brute force, trying all  $2^{16}$  possibilities, works.

- Single Witty packet suffices to extract infectee's complete PRNG state! Think of this as a sequence number.

## What can we measure from this inferred PRNG state ?

---

- Say the state of two consecutive packets, seen from the same source is  $X_i$  and  $X_j$  respectively.
- Compute  $j - i$  by starting from  $X_i$  and cranking forward till  $X_j$ .
- The total number of packets sent in this period is  $(j - i)/4$ .
- The *sendto()* system call in windows blocks untill adequate buffer space is available.
- Thus access bandwidth is given by:

$$(j - i)/4 * (avg\_packet\_size)/(T_j - T_i)$$

- Note: works even in the presence of very heavy packet loss

## Inferred access bandwidth of the Infectee

---

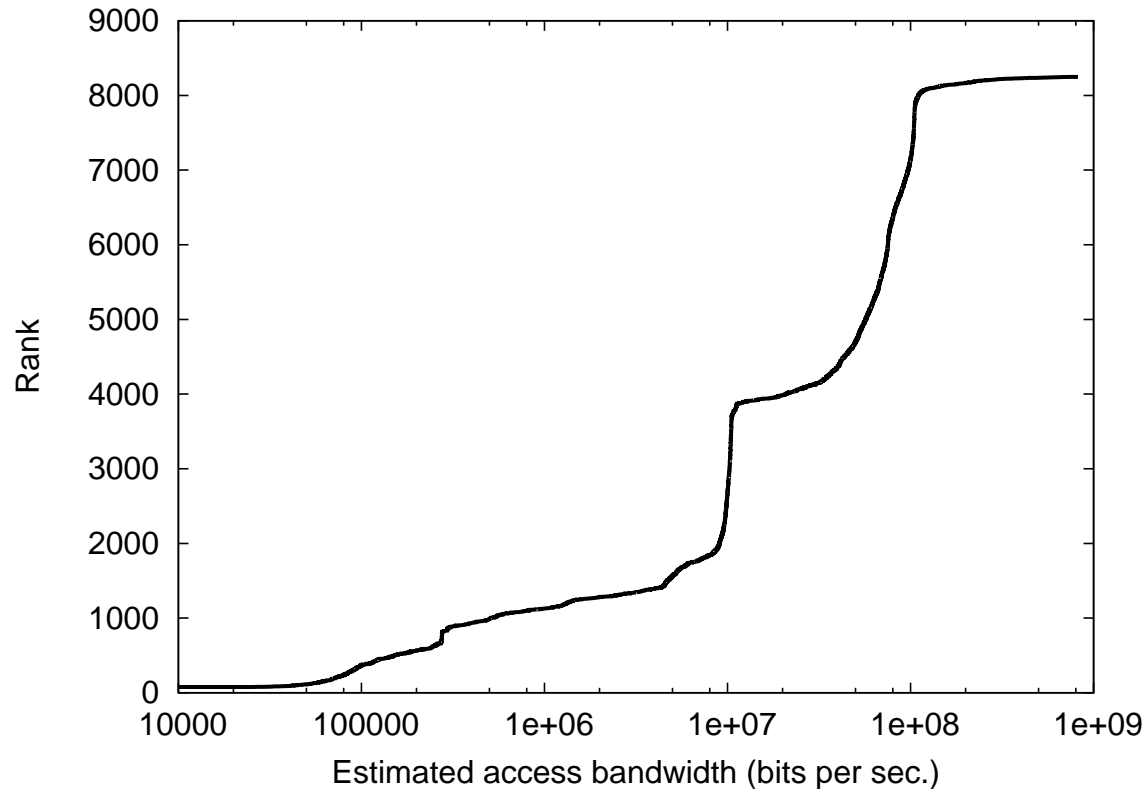


Figure 2: CDF of access bandwidth of witty infectees

## Comparison with “Effective” bandwidth

---

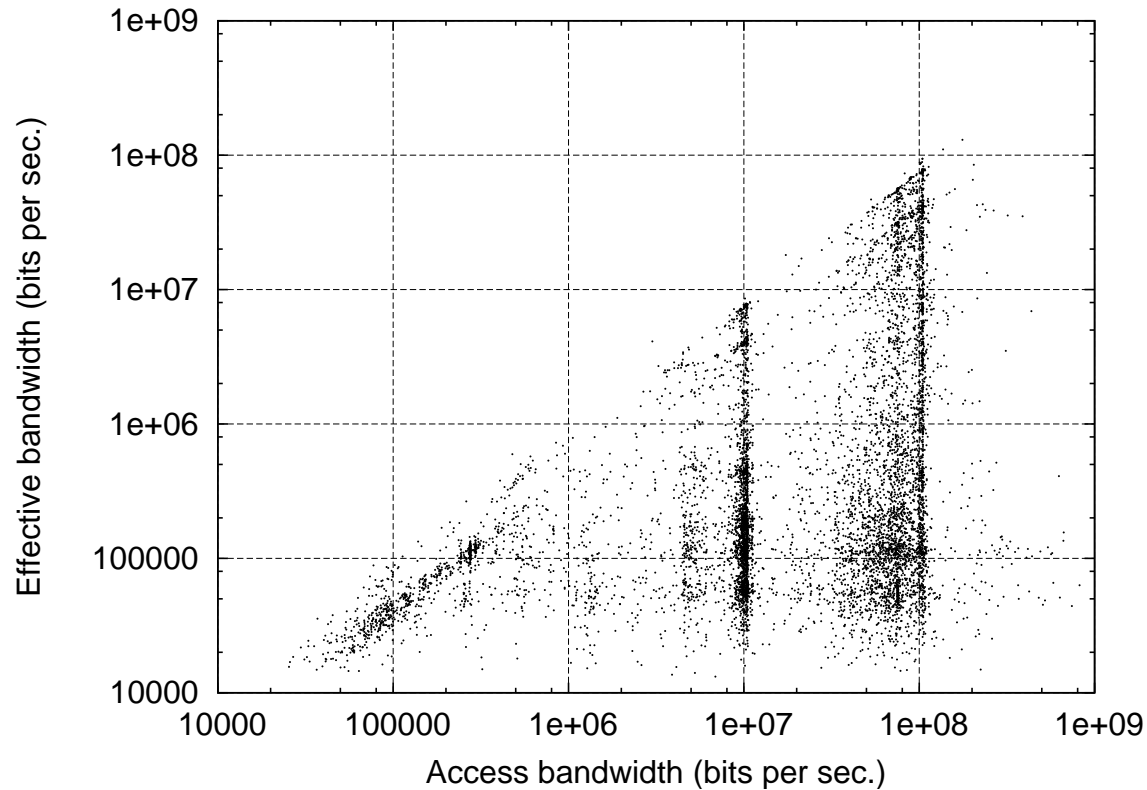


Figure 3: Scatterplot of effective bandwidth of witty infectees as computed by the two methods

## Estimating the seed of Witty's PRNG

---

- Consecutive packets from the same source that are “too far” from each other indicate reseeds of the PRNG.
- The function *rand()* is called exactly 4 times in each run of the inner loop.

## Pseudocode of Witty

---

```
rand(){       $X = X * a + b$ ;      return  $X$ ;}
srand(seed){       $X = seed$ ;}
main(){
1.  srand(get_tick_count());
2.  for( $i=0; i < 20,000; i++$ )
3.       $dest\_ip \leftarrow rand()_{[0...15]} || rand()_{[0...15]}$ ;
4.       $dest\_port \leftarrow rand()_{[0...15]}$ ;
5.       $packet\_size \leftarrow 768 + rand()_{[0...8]}$ ;
6.       $packet\_contents \leftarrow top\ of\ stack$ ;
7.      sendto();
8.  if(open(physicaldisk,  $rand()_{[13...15]}$ ))
9.      write( $rand()_{0...14} || 0x4e20$ );
10.    goto 1;
11.  else goto 2;}
```



## Estimating the seed of Witty's PRNG

---

- Consecutive packets from the same source that are “too far” from each other indicate reseeds of the PRNG.
- The function *rand()* is called exactly 4 times in each run of the inner loop.
- Thus two packets from the same group of 20,000 are spaced  $4k$  steps apart.
- If two packets are spaced  $4k + 1$  steps apart, they straddle a failed disk write.

## A geometric view of the PRNG

---

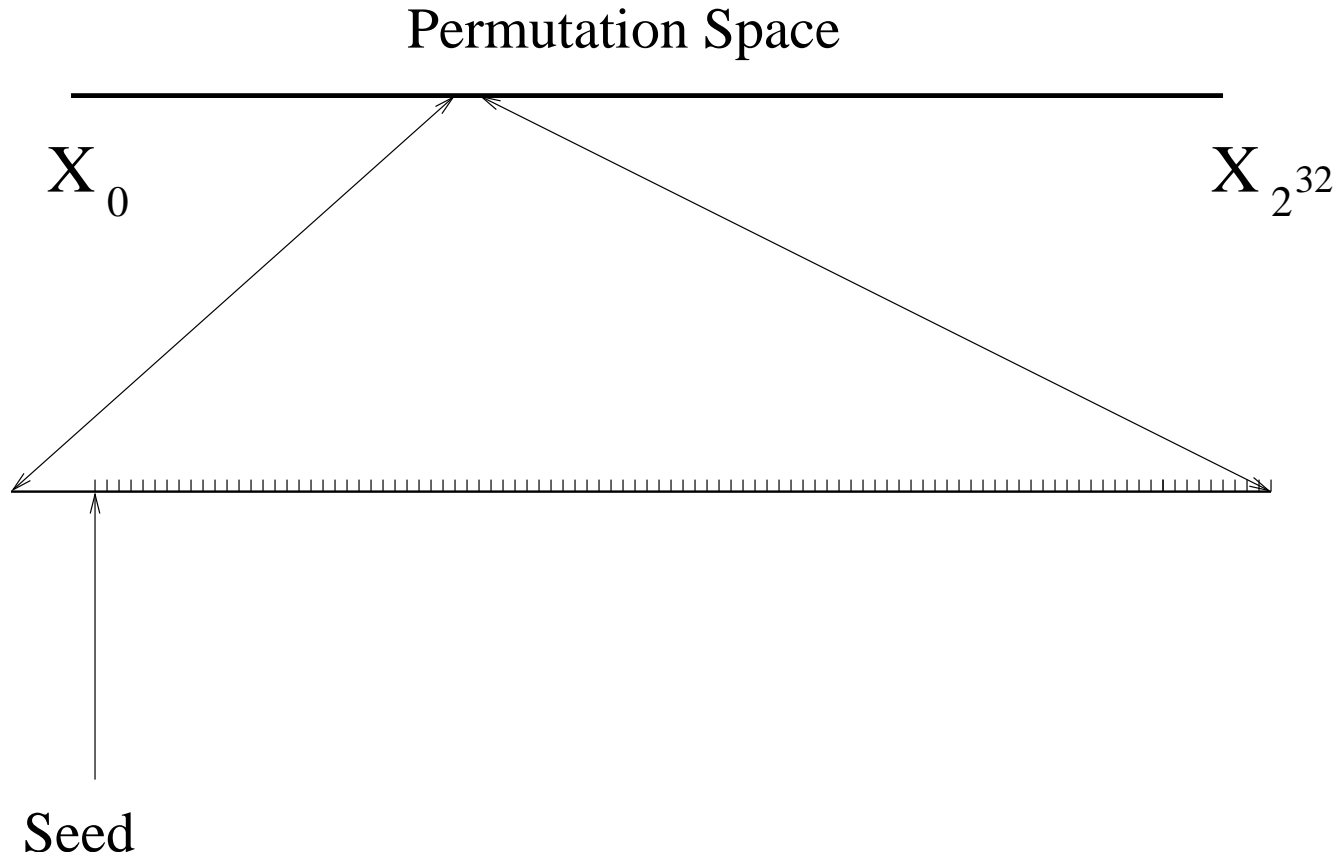


Figure 4: In the permutation space, reseeds take us to a random location on the line.

## A geometric view of the PRNG

---

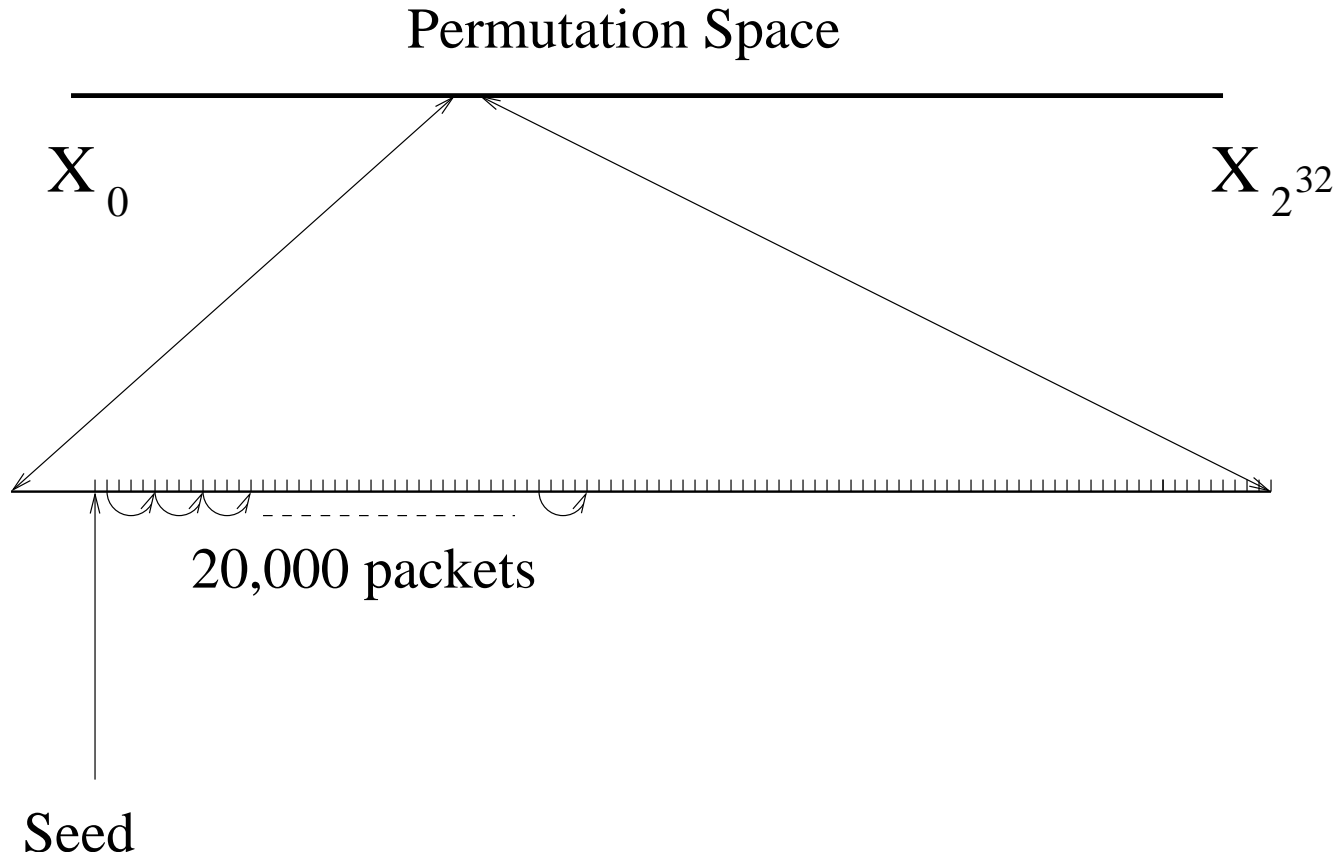


Figure 5: Each of the next 20,000 packets uses 4 consecutive random numbers.

## A geometric view of the PRNG

---

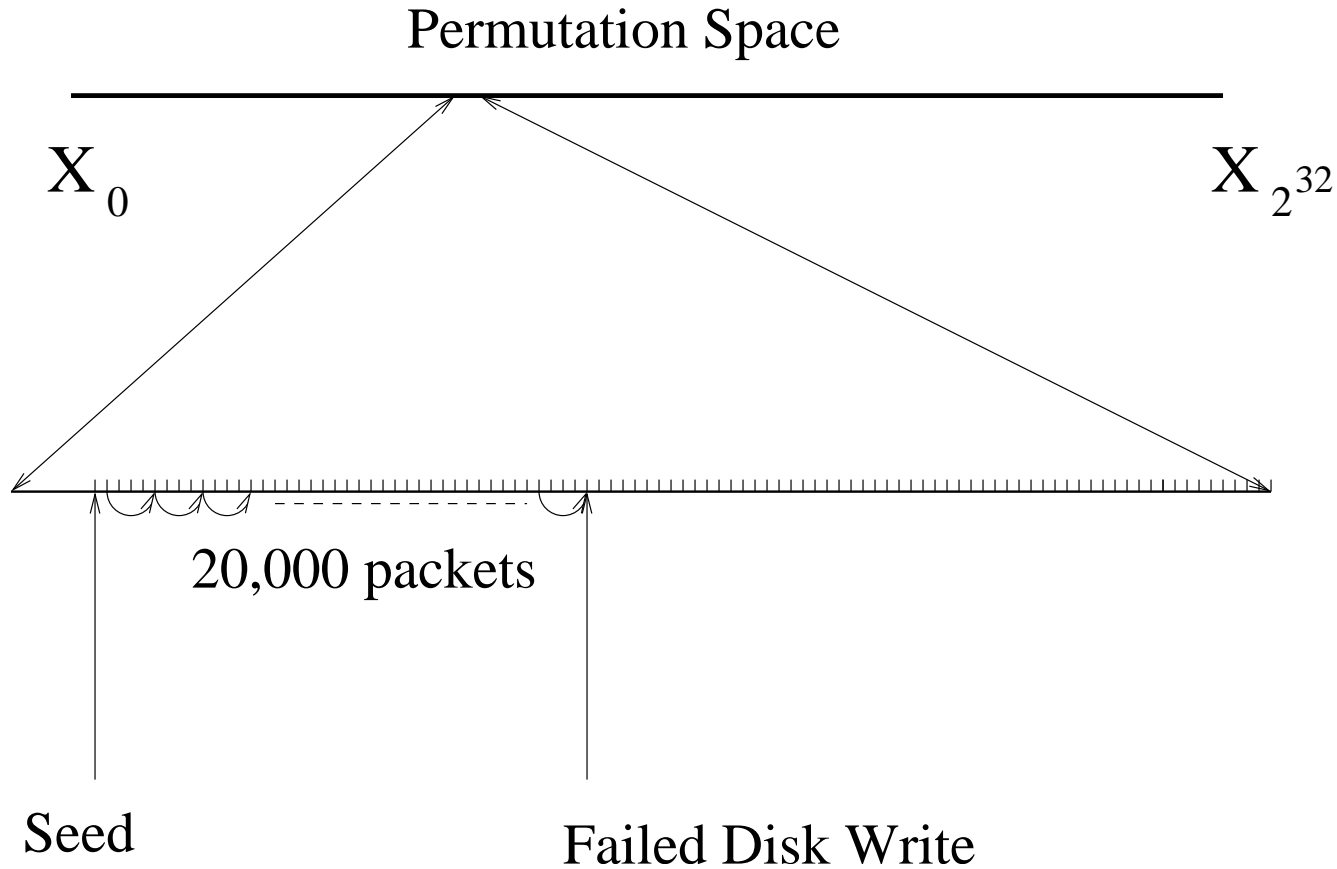


Figure 6: A failed disk write uses one random number.

## A geometric view of the PRNG

---

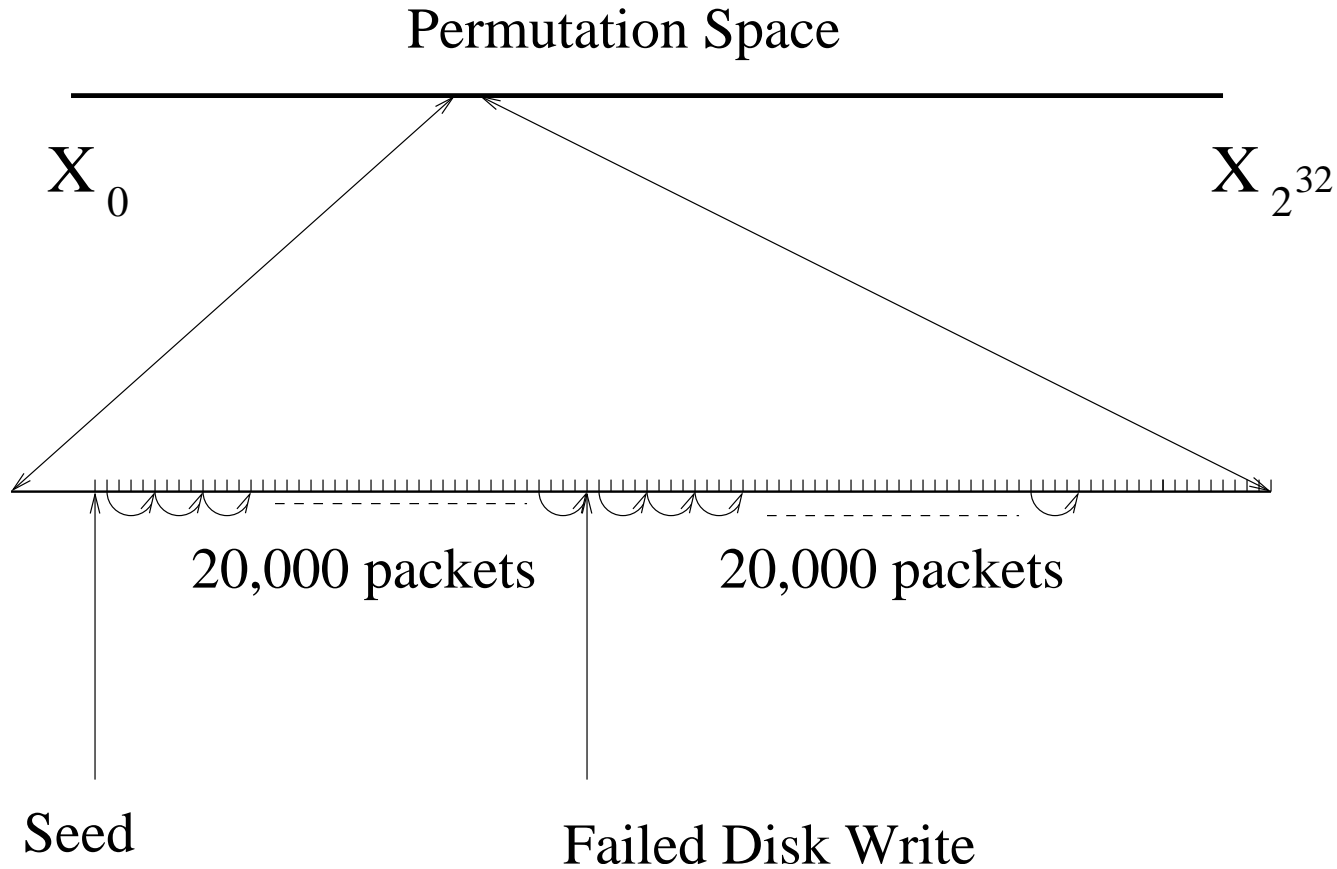


Figure 7: The next 20,000 packets have a “phase-shift” by one. .

## View at the Telescope

---

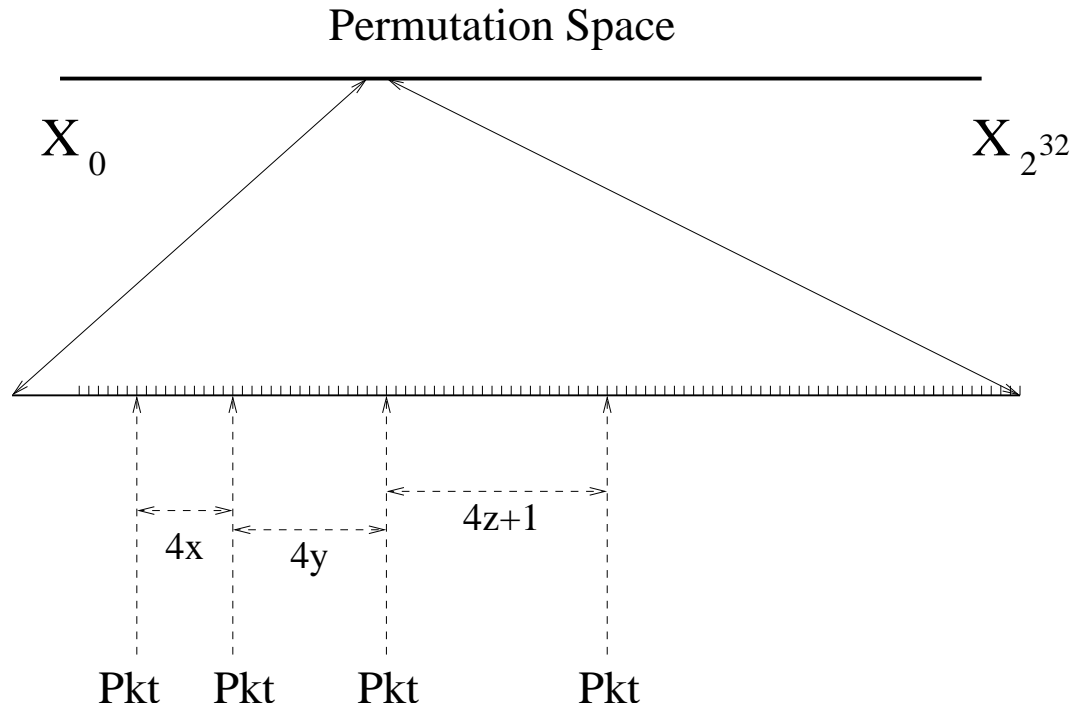


Figure 8: The telescope sees only a subset of all packets, often spaced by multiples of 4.

## View at the Telescope

---

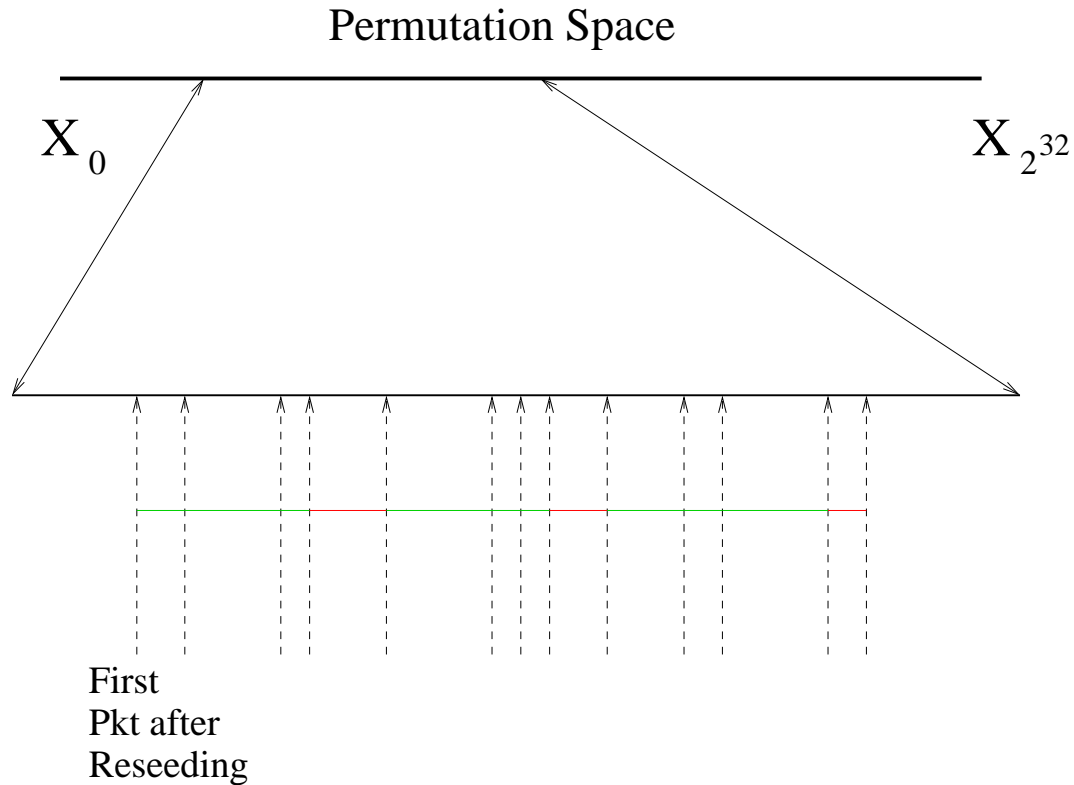


Figure 9: Packets spaced by multiples of 4 in green, packets spaced by multiples of 4 plus one in red.

## View at the Telescope

---

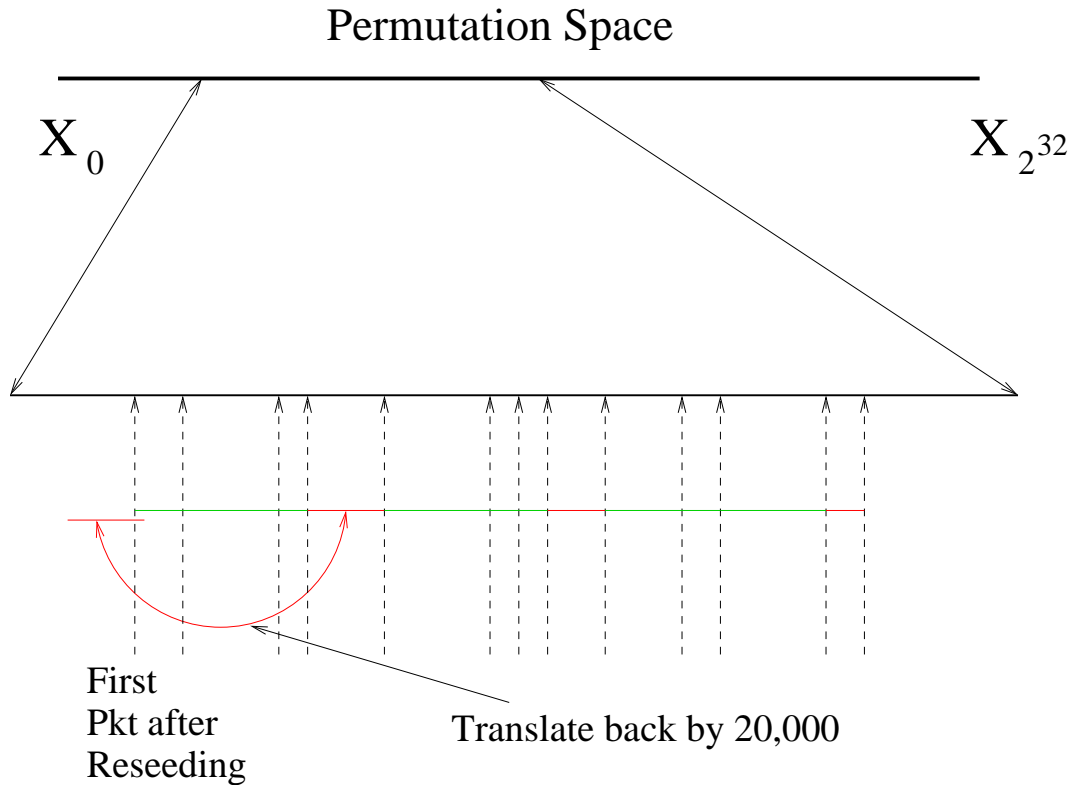


Figure 10: A failed disk write must succeed a reseeding by exactly  $20,000 * 4$ .



## View at the Telescope

---

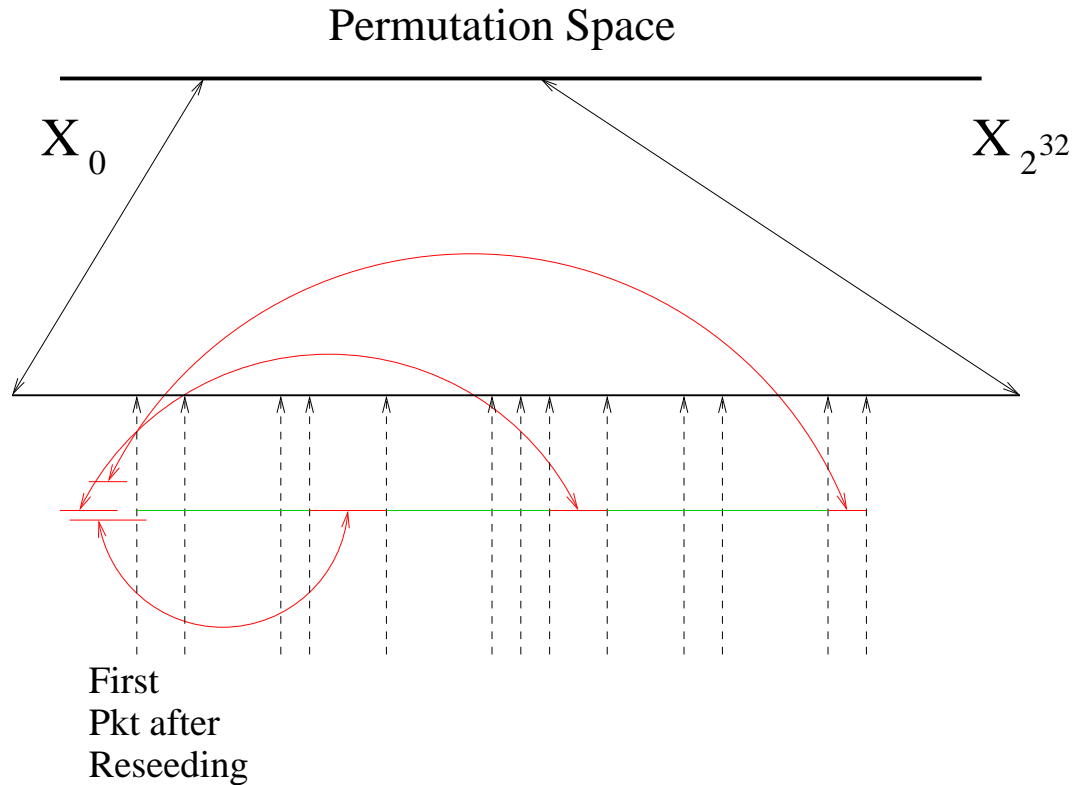
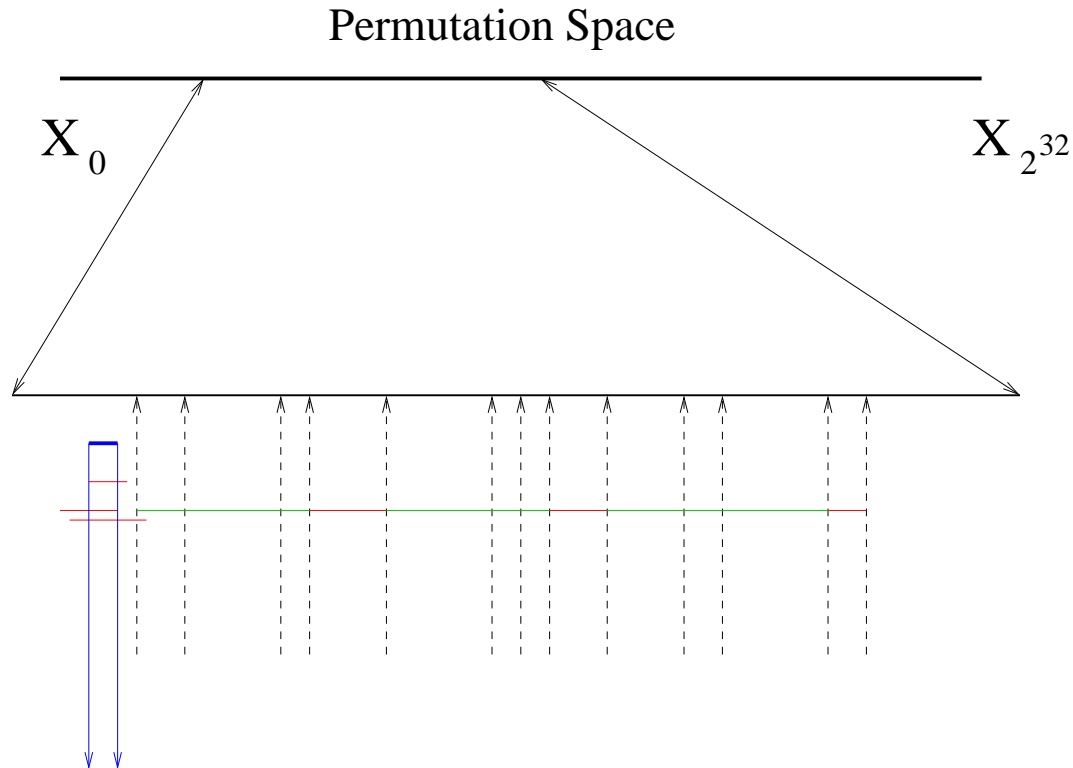


Figure 11: A failed disk write must succeed a reseeding by exactly  $20,000 * 4$ .

## View at the Telescope

---



Range where the seed must lie.

Figure 12: This allows us to narrow down the range of possible reseeding.

# View at the Telescope

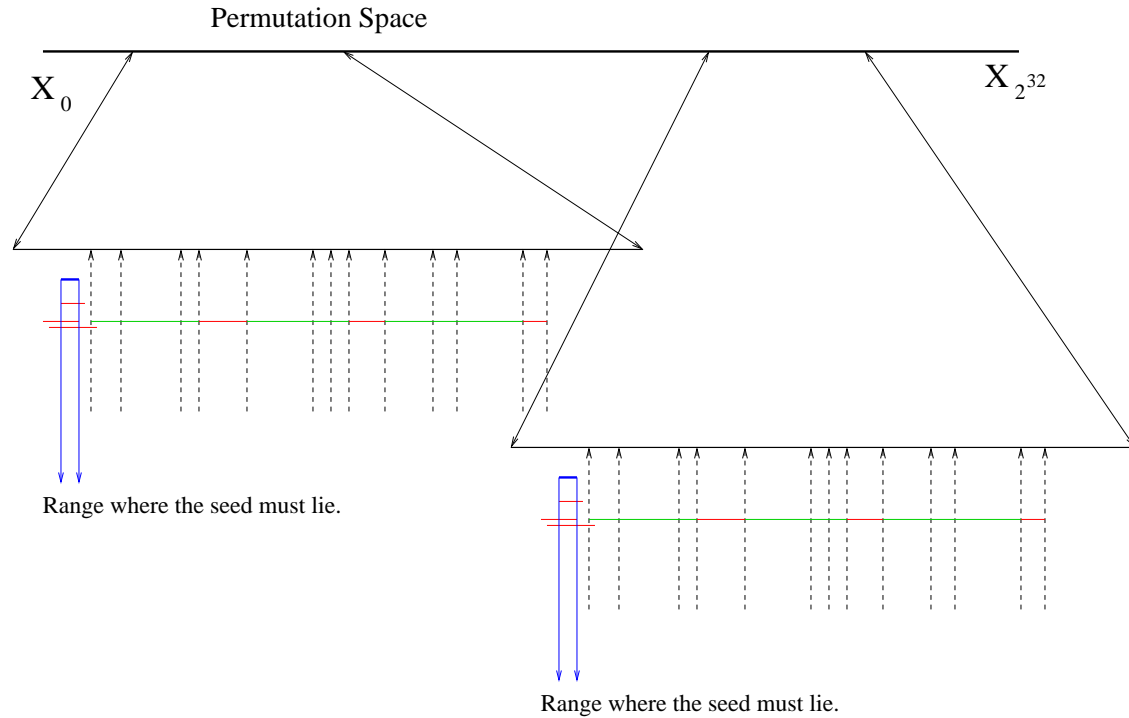
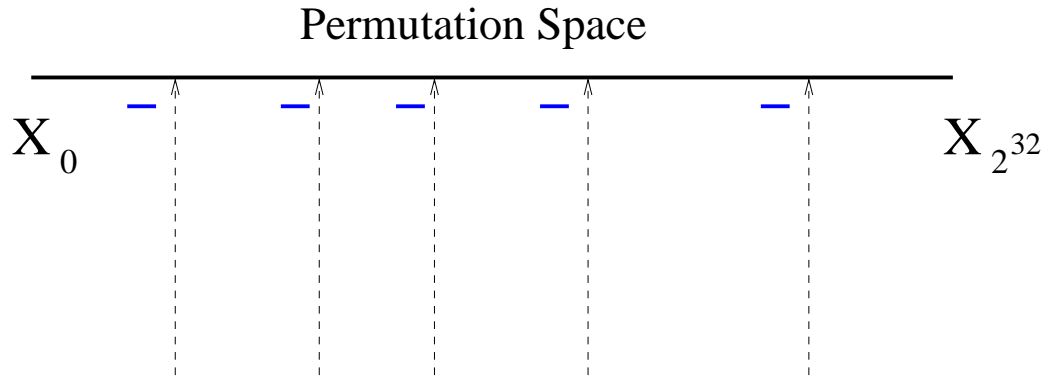


Figure 13: Repeat this for each possible reseeding.

## View at the Telescope

---



Packets unrelated to predecessors

Figure 14: Potential seeds in blue for each reseeding event.

# Identifying the seeds

---

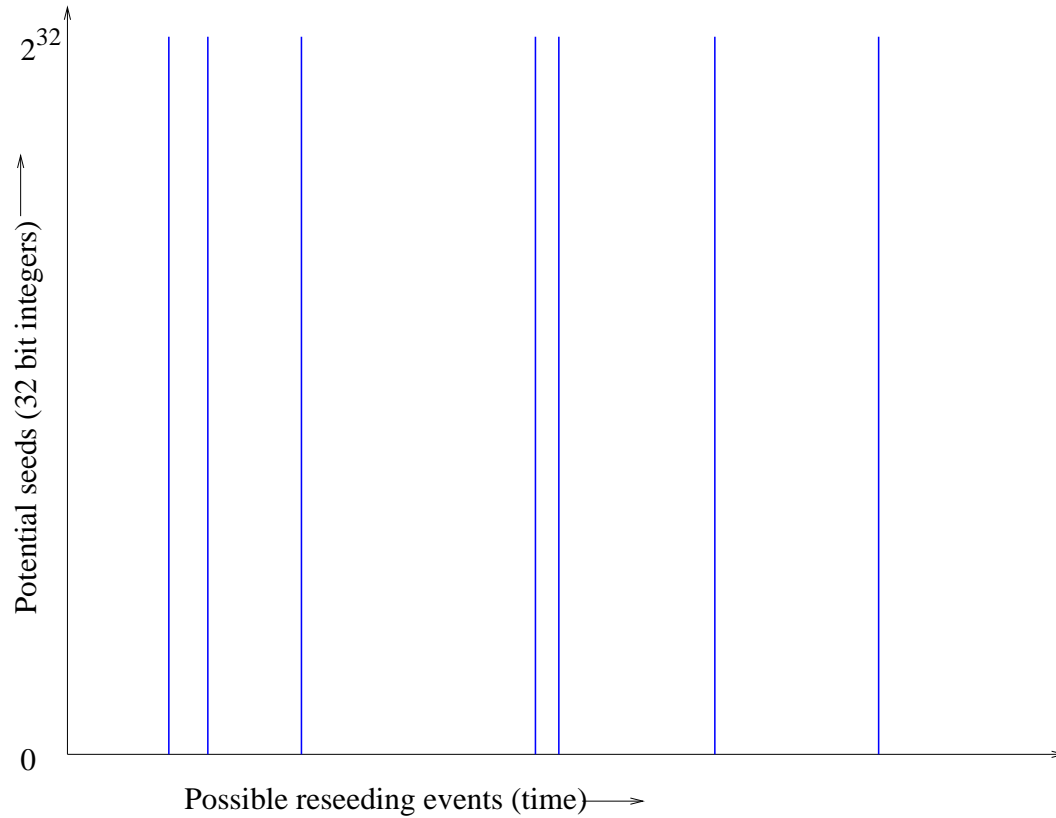


Figure 15: Potential seeds in blue for each reseeding event.

# Identifying the seeds

---

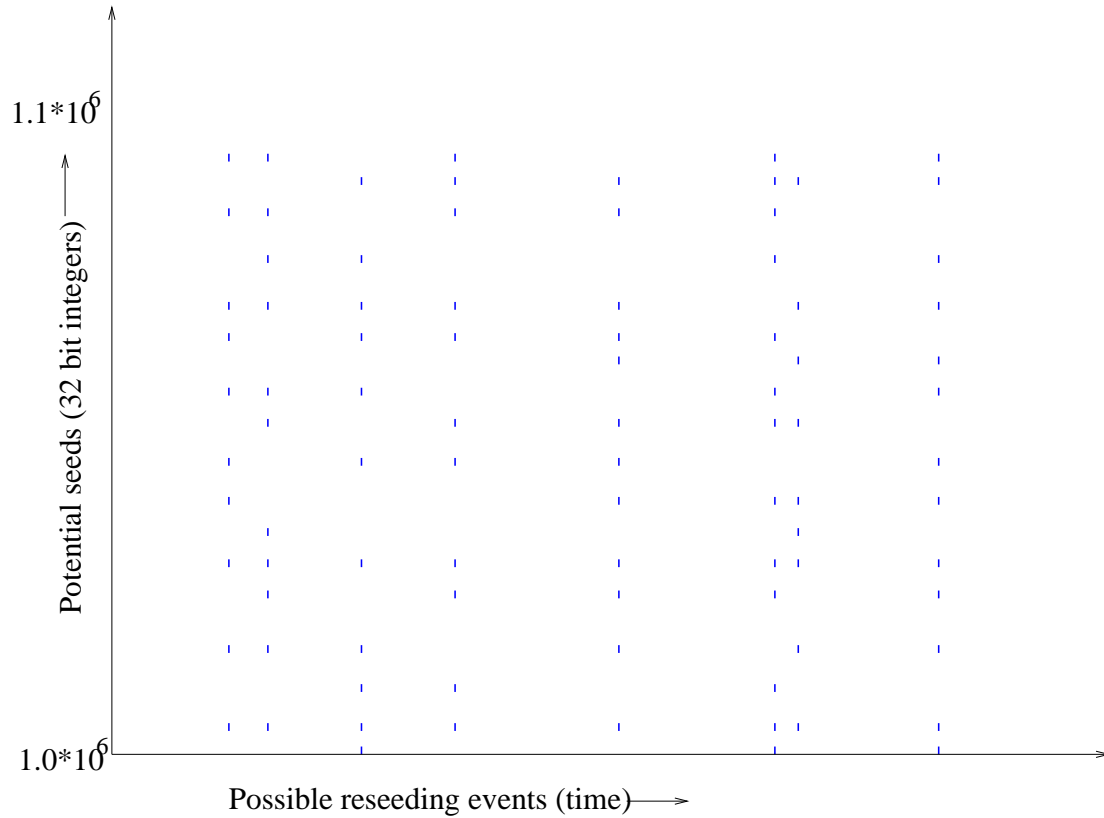


Figure 16: Potential seeds in blue for each reseeding event.

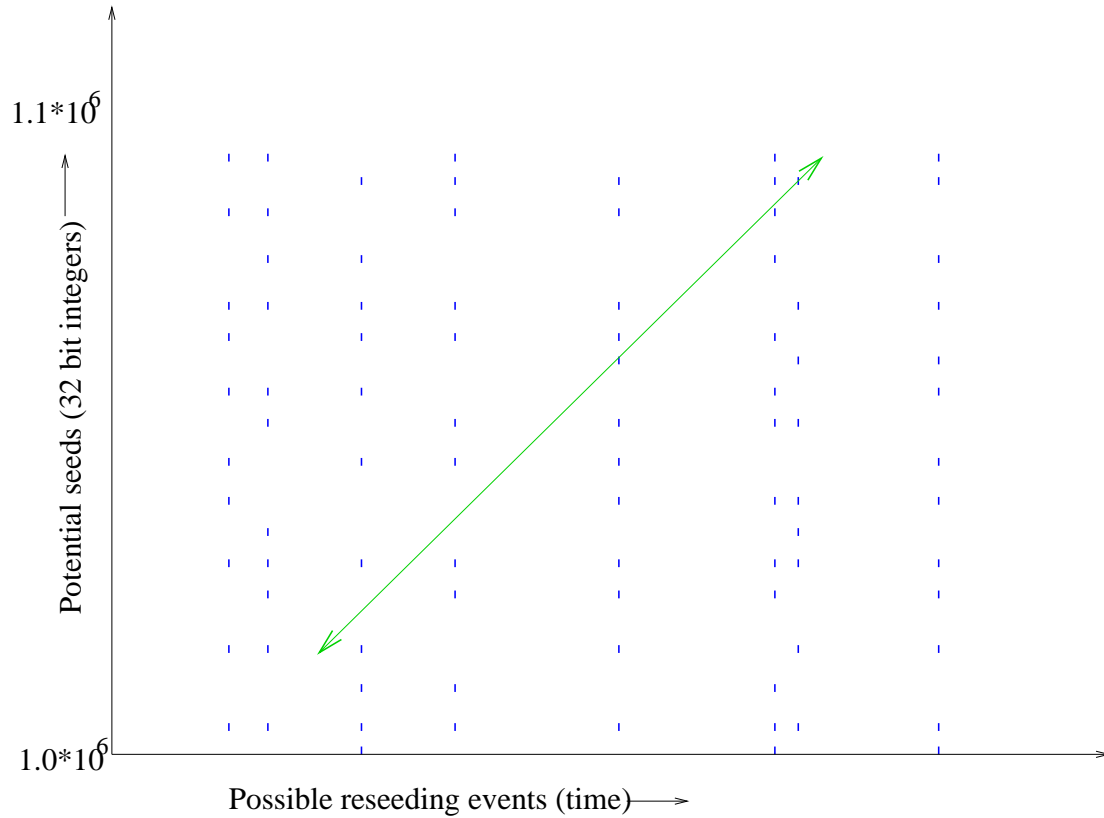
## Pseudocode of Witty

---

```
a=214013,b=2531011;
rand(){     $X = X * a + b$ ;    return  $X$ ;}
srand(seed){     $X = seed$ ;}
main(){
1.  srand(get_tick_count());
2.  for(i=0;i<20,000;i++)
3.       $dest\_ip \leftarrow rand()_{[0...15]} || rand()_{[0...15]}$ ;
4.       $dest\_port \leftarrow rand()_{[0...15]}$ ;
5.       $packet\_size \leftarrow 768 + rand()_{[0...8]}$ ;
6.       $packet\_contents \leftarrow top\ of\ stack$ ;
7.      sendto();
8.      if(open(physicaldisk,  $rand()_{[13...15]}$ ))
9.          write( $rand()_{0...14} || 0x4e20$ );
10.     goto 1;
11.  else goto 2;}
```

# A little Computational Geometry

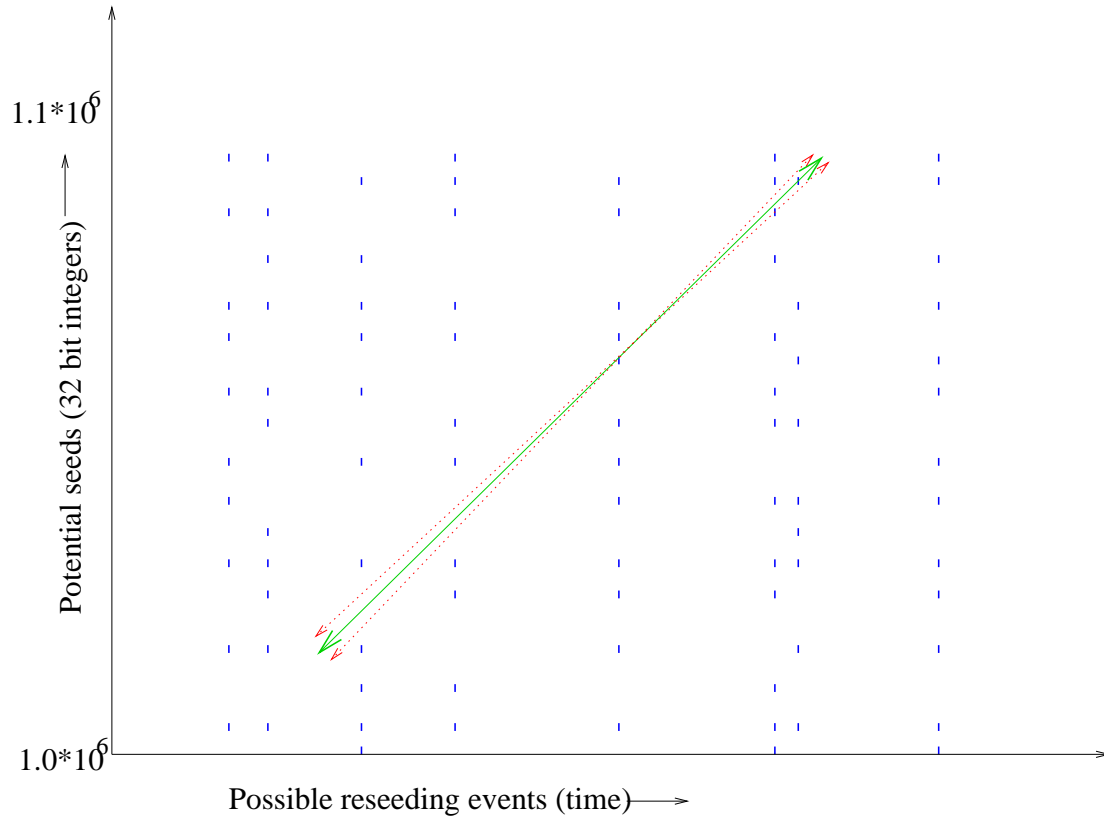
---





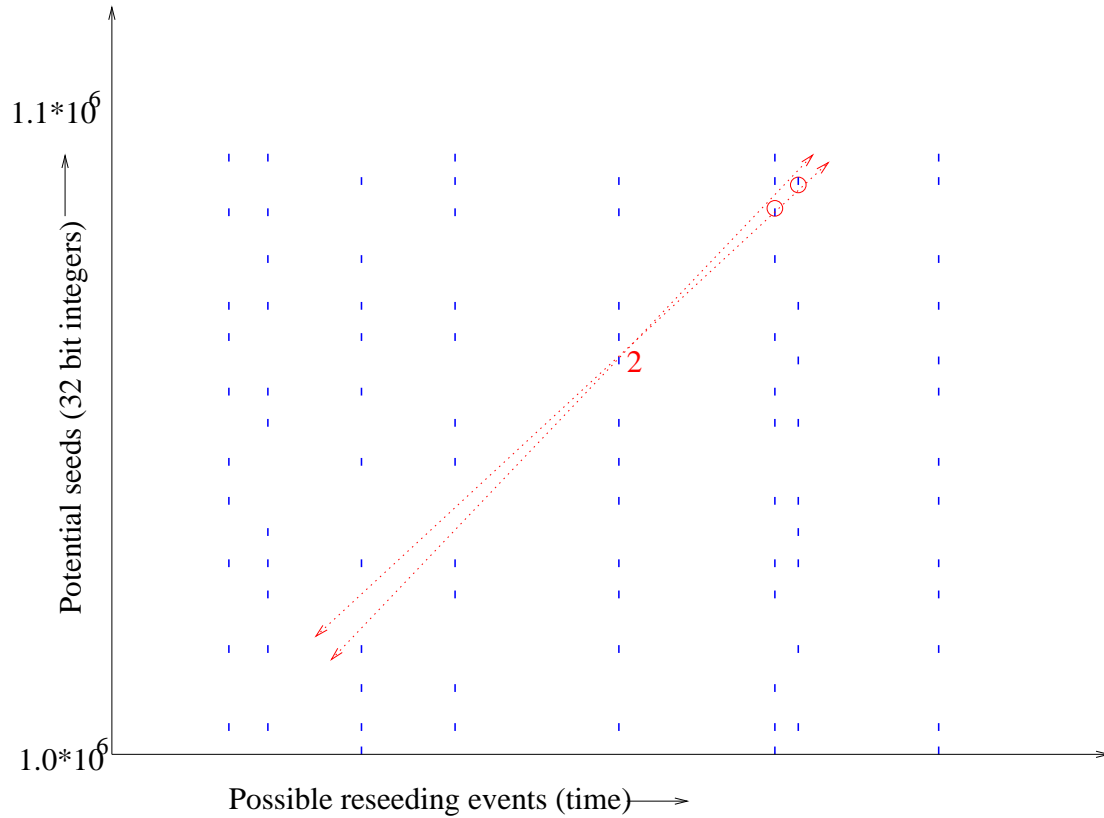
# A little more Computational Geometry

---



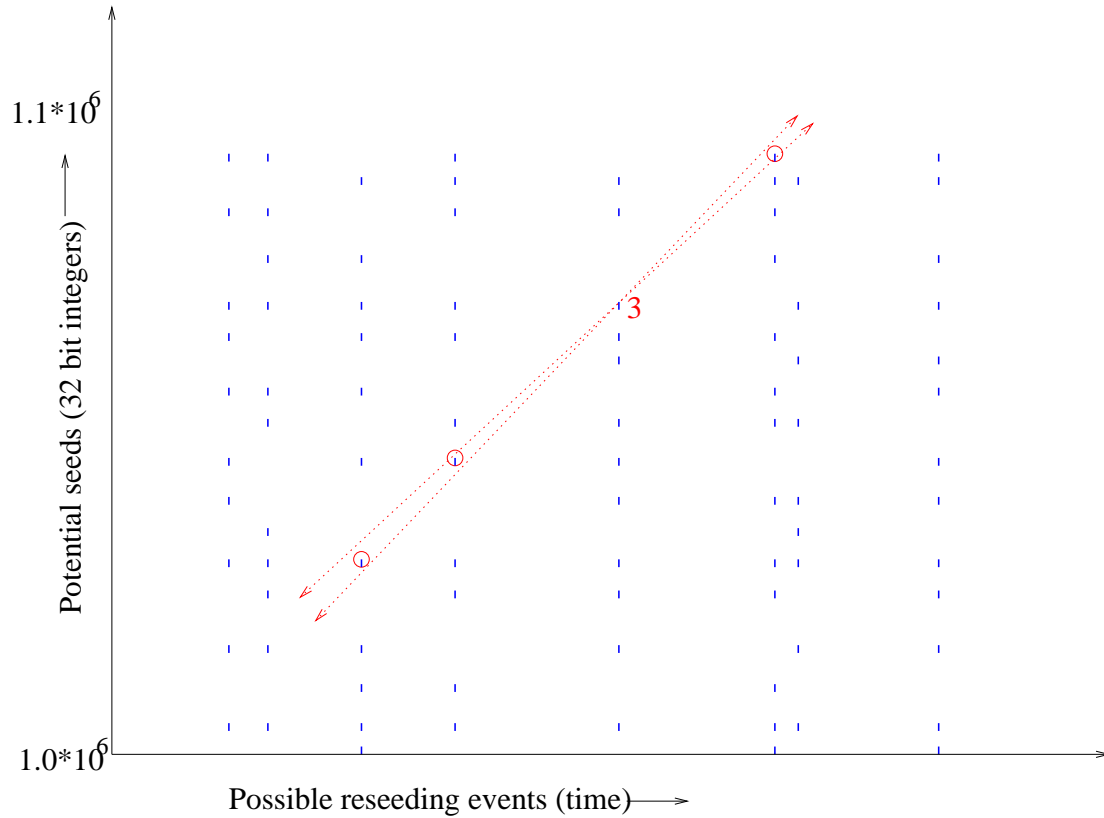
# A bit more Computational Geometry

---



# Some more Computational Geometry

---



# Almost done with Computational Geometry

---

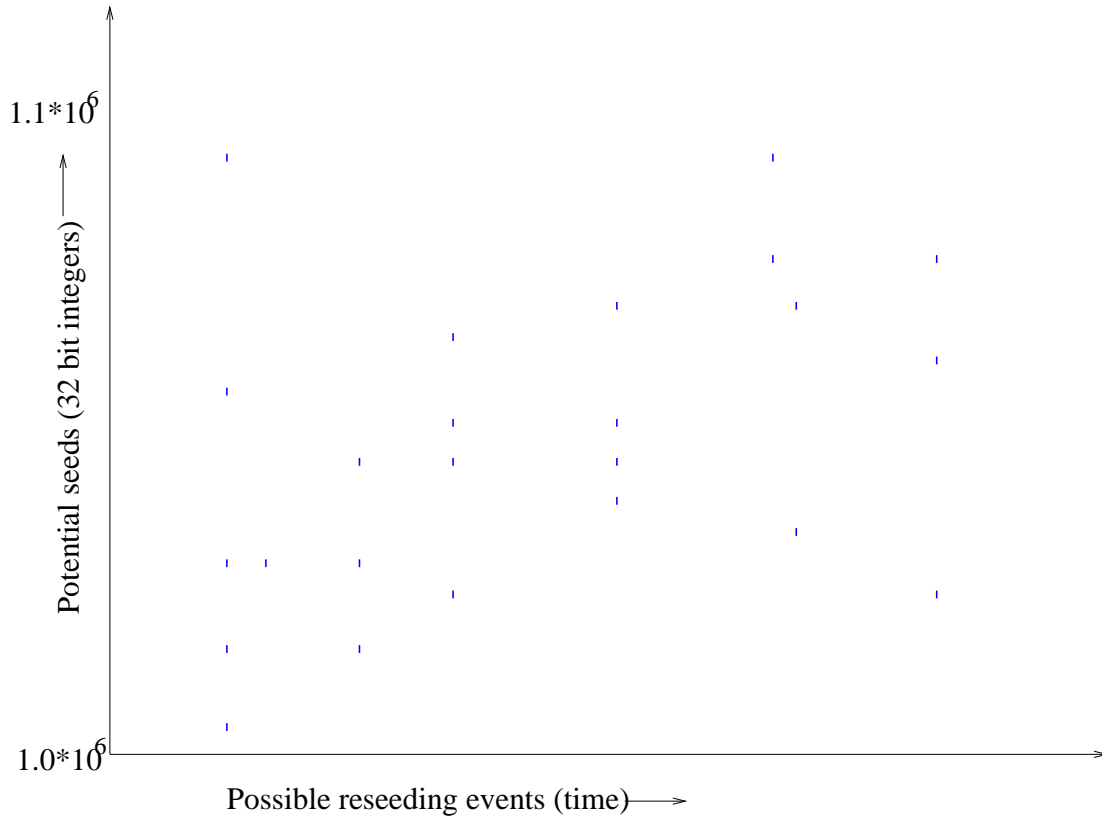


Figure 17: Most potential seeds are eliminated in the first iteration.

# Geometric Computation

---

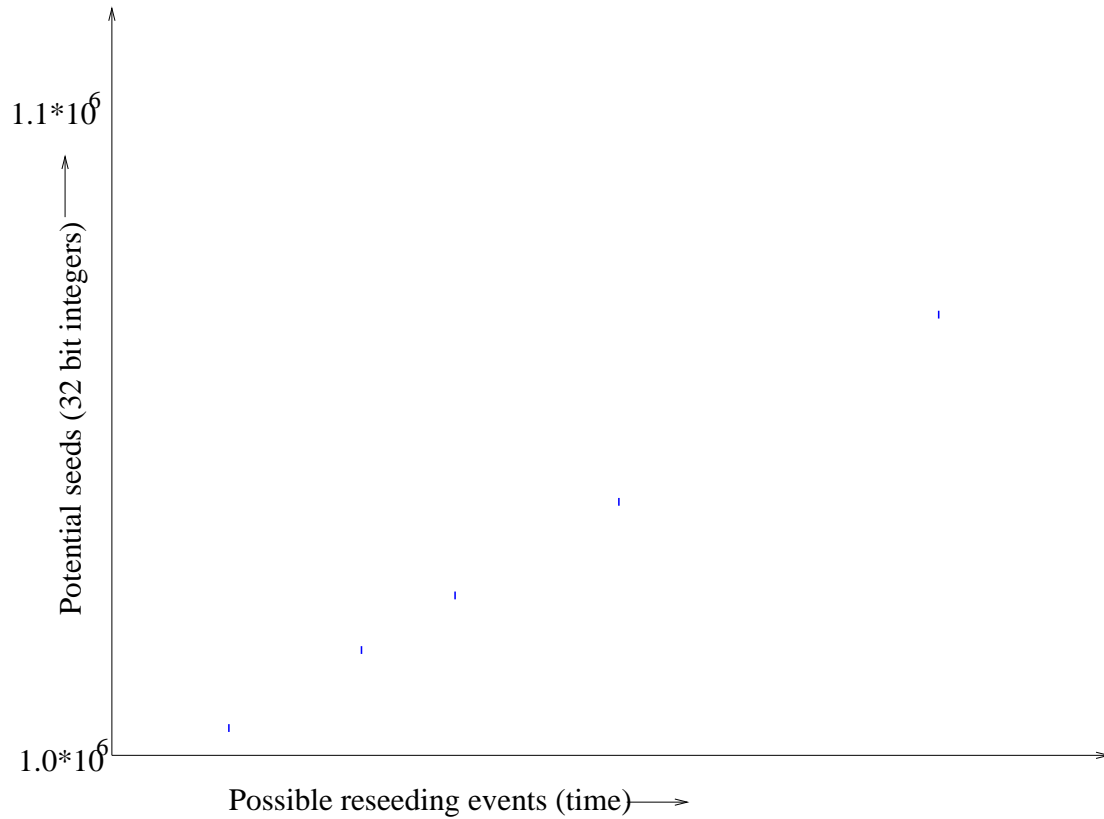


Figure 18: Subsequent iterations clean up the rest of the crud to leave us with a line.

## Uptime of Witty Infectees

---

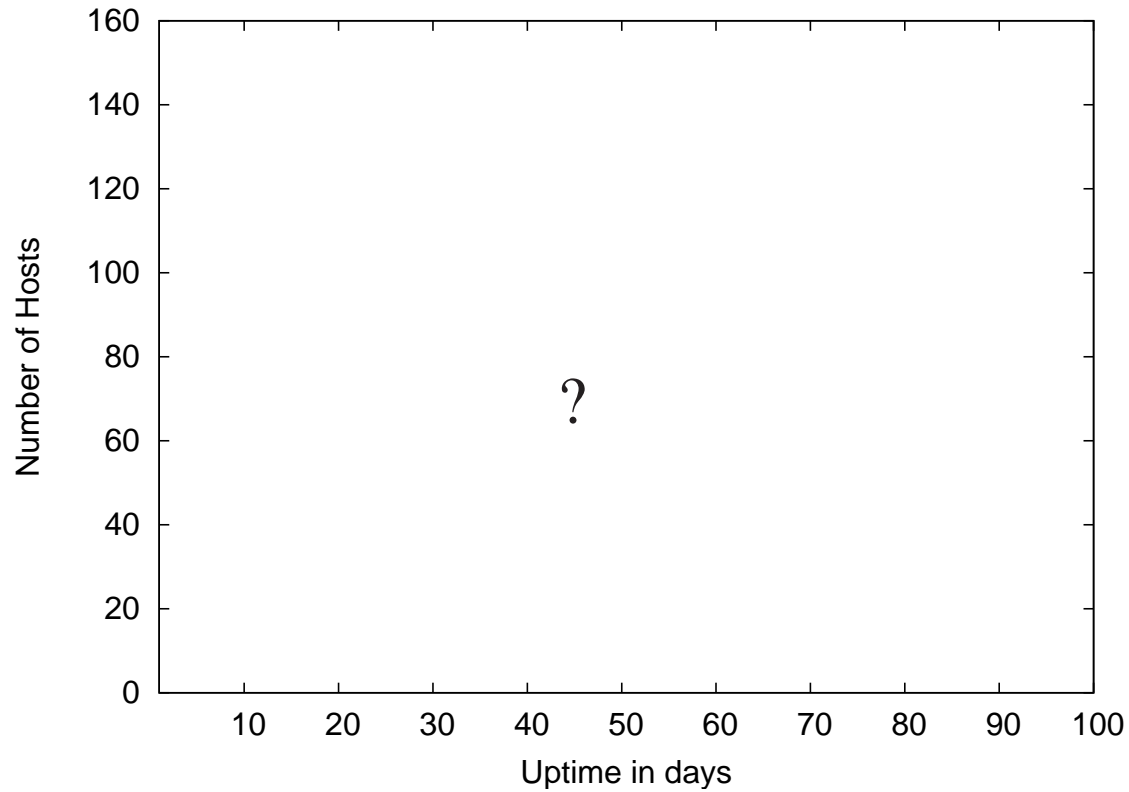


Figure 19: No. of machines vs. Uptime in days.

## Uptime of 750 Witty Infectees

---

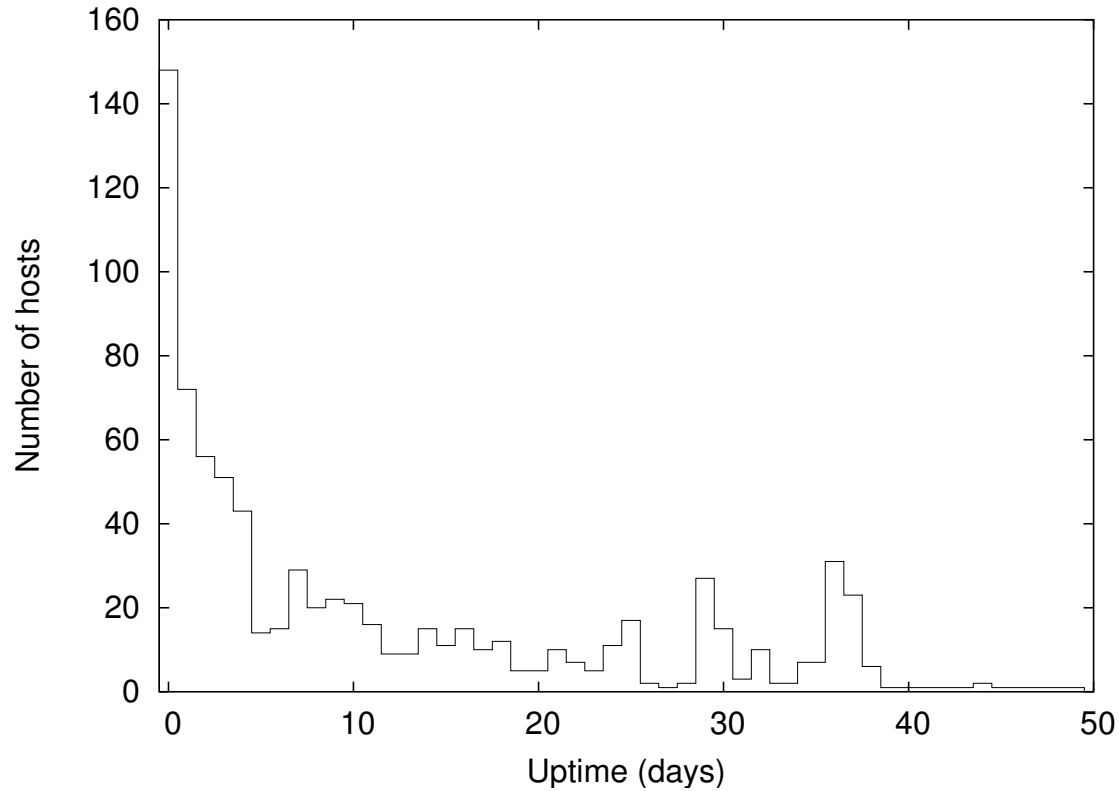


Figure 20: No. of machines vs. Uptime in days.

## Number of physical disks on the infectee.

---

- After reseeding, witty generates 20,000 packets using 4 random numbers for each.
- Then it generates a random number to pick a disk to write to.



## Pseudocode of Witty

---

```
rand(){       $X = X * a + b;$       return  $X;$ }
srand(seed){       $X = seed;$ }
main(){
1.  srand(get_tick_count());
2.  for( $i=0; i < 20,000; i++$ )
3.       $dest\_ip \leftarrow rand()_{[0...15]} || rand()_{[0...15]}$ ;
4.       $dest\_port \leftarrow rand()_{[0...15]}$ ;
5.       $packet\_size \leftarrow 768 + rand()_{[0...8]}$ ;
6.       $packet\_contents \leftarrow top\ of\ stack;$ 
7.      sendto();
8.  if(open(physicaldisk,  $rand()_{[13...15]}$ ))
9.      write( $rand()_{0...14} || 0x4e20$ );
10.  goto 1;
11. else goto 2;}
```

## Number of physical disks on the infectee.

---

- After reseeding, witty generates 20,000 packets using 4 random numbers for each.
- Then it generates a random number to pick a disk to write to.
- We can pick the exact 3 bits used to identify the physical disk in the following code segment.

8. `if(open(physicaldisk, rand()[13...15]))`

Number of Disks	1	2	3	4	5	6	7
Number of Infectees	52	32	12	2	2	0	0

Table 1: Disk counts of 100 infectees.

## The complete list of packets sent by the infectee.

---

- Actually, since we know the seed, we can compute the exact sequence of packets generated.
- Thus we know the *dest\_ip*, *dest\_port* and the payload size of each packet.
- The first packet seen from any infectee indicates the infection time of the infectee.
- Look up the list of packets sent immediately before that to identify the infector.
- Ability to construct the infection graph !

## Time between scan by other infectees and observed infection.

---

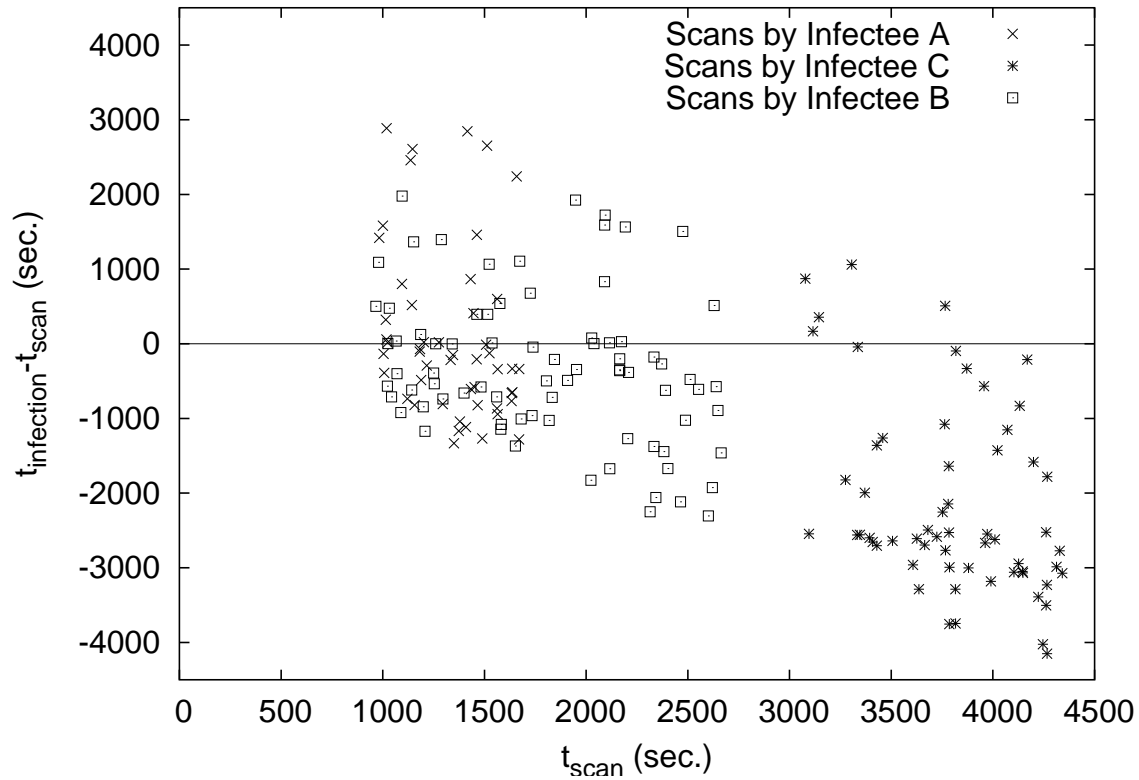


Figure 21: Scans from infectees, targeted to other victims

## Time between scan by other infectees and observed infection.

---

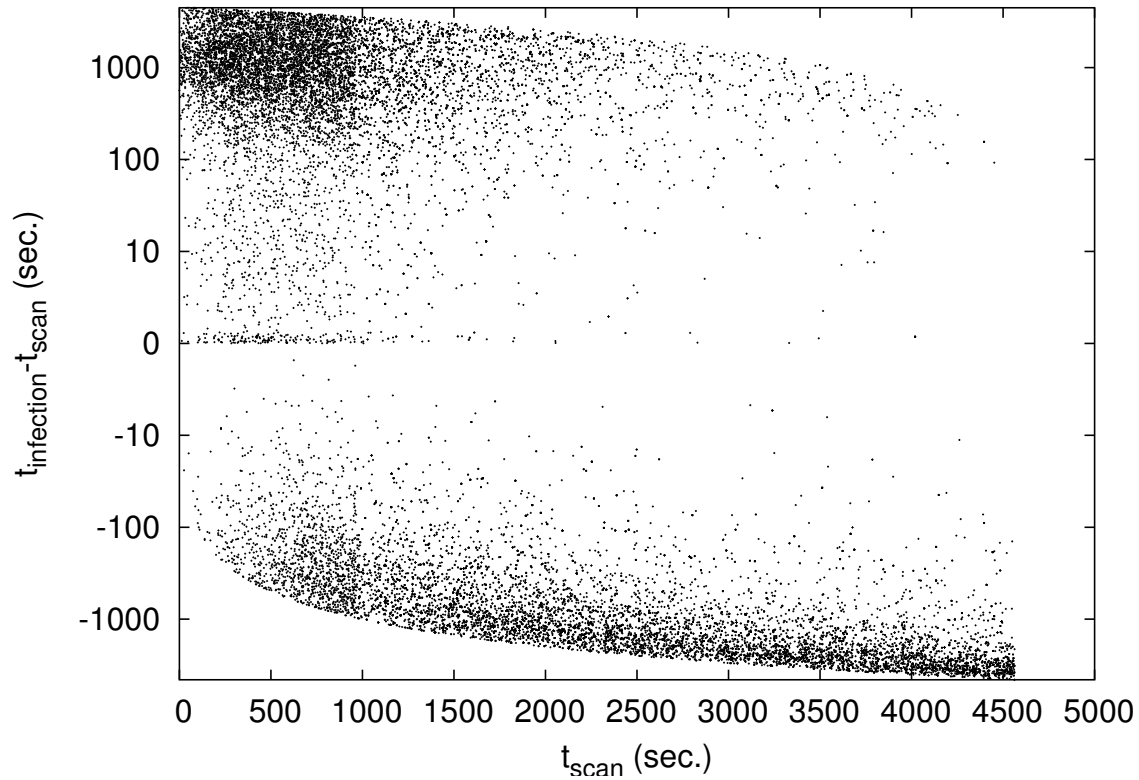


Figure 22: Scans from infectees, targeted to other victims

## Infection attempts that were too late, too early, or just right.

---

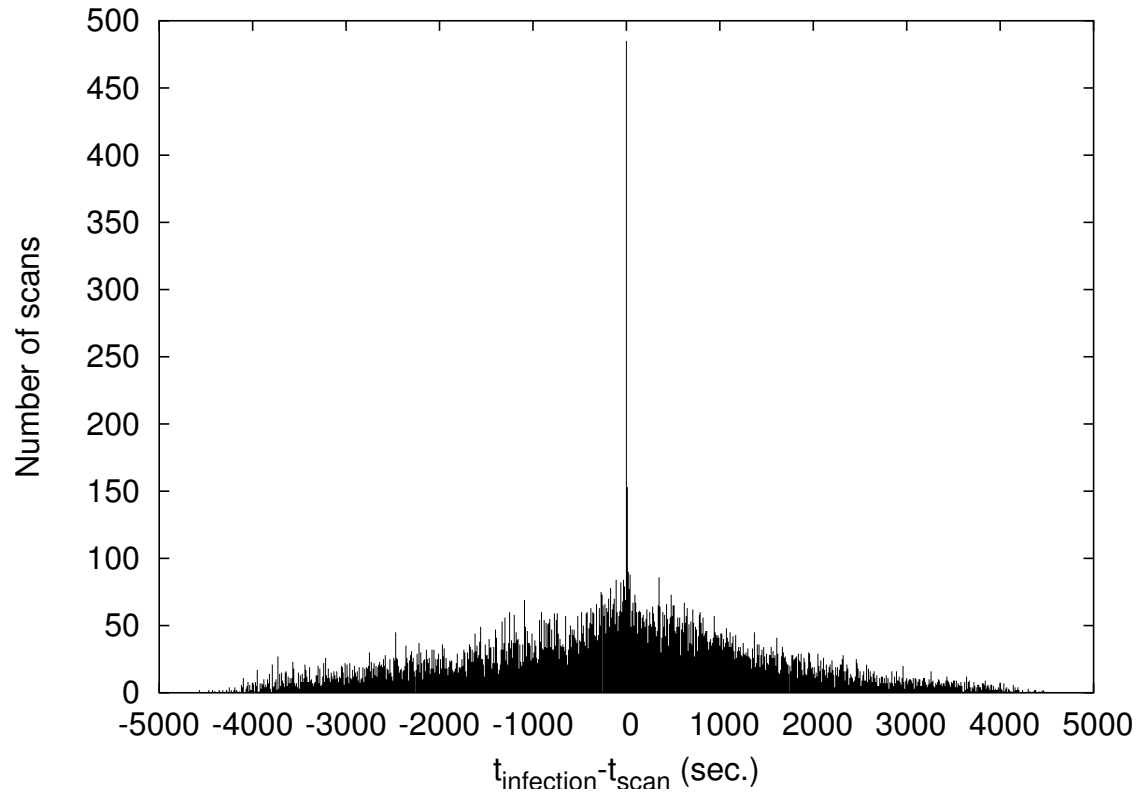


Figure 23: Number of scans in 10 second buckets

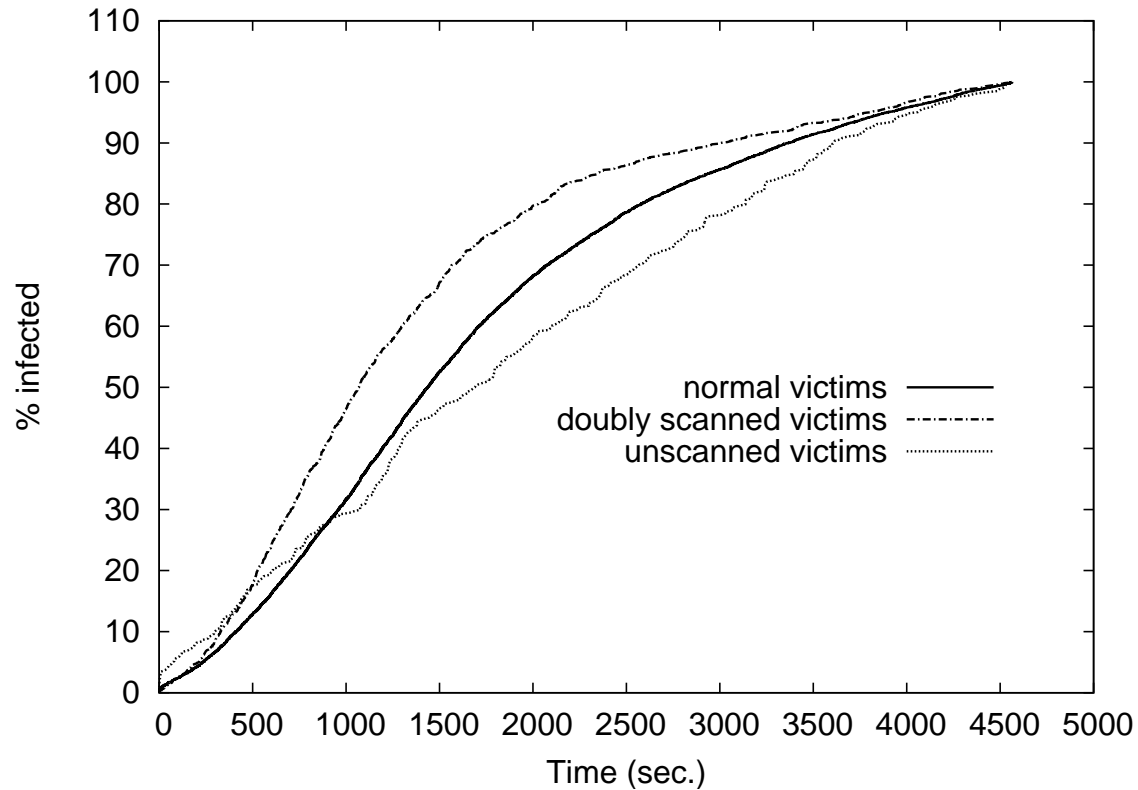
## Witty is incomplete

---

- Recall that LC PRNG generates a complete orbit over a permutation of  $0..2^{32}-1$ .
- But: Witty author didn't use all 32 bits of single PRNG value  
3.  $dest\_ip \leftarrow rand()_{[0...15]} || rand()_{[0...15]}$ ;
  - Knuth recommends top bits as having better pseudo-random properties
- But: This does not generate a complete orbit!
  - Misses 10% of the address space
  - Visits 10% of the addresses (exactly) twice
- So, were 10% of the potential infectees protected?

## Time when infectee was seen at telescope

---





## Recall: Uptime of 750 Witty Infectees

---

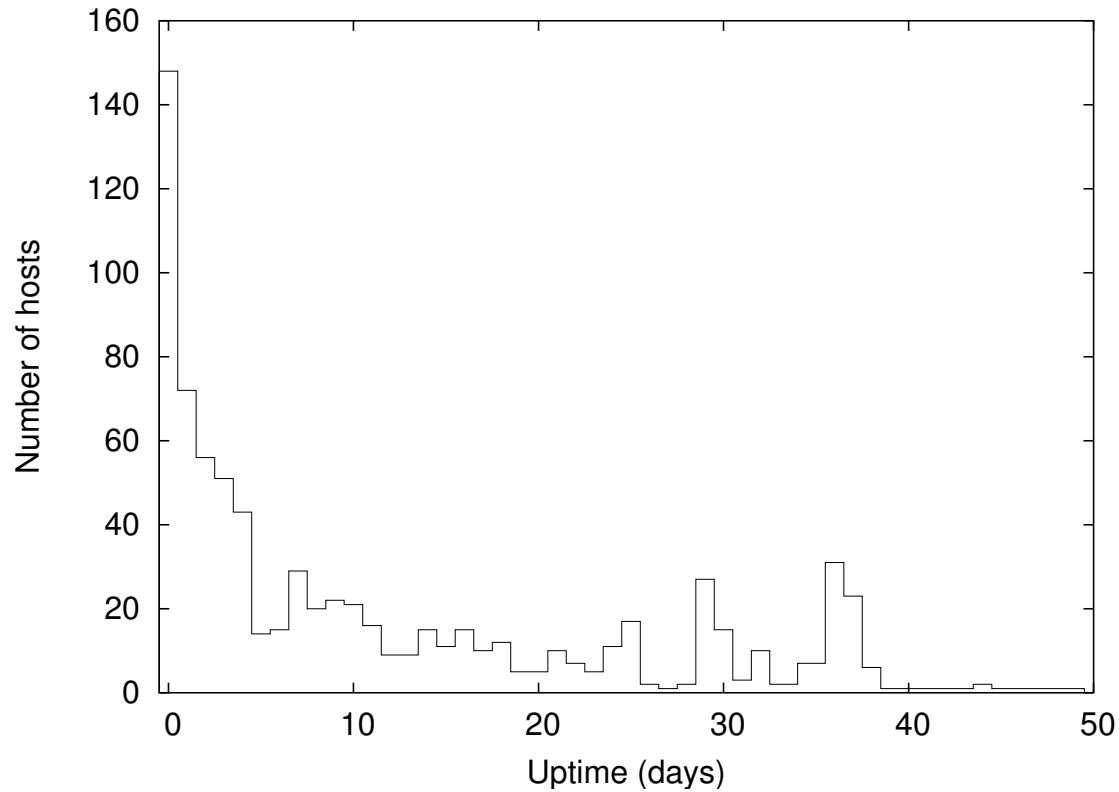


Figure 24: No. of machines vs. Uptime in days.

## Witty started with a “Hit List”

---

- Initial infectees exhibit super-exponential growth → they weren't found by random scanning
- Attacker knew of ISS security software installation at military site ? ISS insider? (or ex-insider)
- Fits with very rapid development of worm after public vulnerability disclosure

## Are All The Worms In Fact Executing Witty?

---

- Answer: No.
- There is one “infectee” that probes addresses not on the orbit.
- This “special” sender was sending packets with *dest\_ip* of the form *a.b.a.b*.
- Each probe contains Witty contagion, but lacks randomized payload size. Shows up very near beginning of trace.
- Patient Zero - machine attacker used to launch Witty. (Really, Patient Negative One.)
  - European retail ISP.
  - Information passed along to Law Enforcement.

## More Results in the paper and tech report

---

- Comparison of access bandwidth measurements across the telescopes
- Telescope saturation
- Divergence in telescope observation
- Inference of the topology of a distant subnet
- Report available at: <http://www.cc.gatech.edu/~akumar/witty.html>
- Or Google: Outwitting worm

## Summary of Witty Telescope Analysis

---

- Understanding a measurement's underlying structure adds enormous analytic power

## Summary of Witty Telescope Analysis

---

- Understanding a measurement's underlying structure adds enormous analytic power
- Cuts both ways: Anonymization would make such studies harder

## Summary of Witty Telescope Analysis

---

- Understanding a measurement's underlying structure adds enormous analytic power
- Cuts both ways: Anonymization would make such studies harder
- I will never look at “random” number generation the same way

## Summary of Witty Telescope Analysis

---

- Understanding a measurement's underlying structure adds enormous analytic power
- Cuts both ways: Anonymization would make such studies harder
- I will never look at “random” number generation the same way
- With enough effort, worm “attribution” can be possible



## Summary of Witty Telescope Analysis

---

- Understanding a measurement's underlying structure adds enormous analytic power
- Cuts both ways: Anonymization would make such studies harder
- I will never look at “random” number generation the same way
- With enough effort, worm “attribution” can be possible
- Applications to worm/worm-defense simulation, calibration of network telescope fidelity, large-scale measurement of Internet path properties, inferring firewall policies: **Opportunistic Measurement**

## Acknowledgments

---

- NSF grants: Collaborative Cybertrust NSF-0433702, ITR/ANI-0205519, NRT-0335290, and ANI-0238315
- Colleen Shannon and David Moore at CAIDA
- Paul Barford and Vinod Yegneswaran at the University of Wisconsin
- Travel funds from CCIED and GTISC
- Support for the Witty Worm Dataset and the UCSD Network Telescope are provided by Cisco Systems, Limelight Networks, the US Dept. Homeland Security, the National Science Foundation, and CAIDA, DARPA, Digital Envoy, and CAIDA Members.

## Random Numbers

---

- PRNGs are difficult to implement for script kiddies
- Worm authors get it wrong all the time
- Even Witty's author made a subtle error in using the random numbers
- The sources of entropy are limited