CS 268: Lecture 19 (Malware)

Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

(Based on slides from Vern Paxson and Stefan Savage)

Motivation

- Internet currently used for important services
  - Financial transactions, medical records
- Could be used in the future for *critical* services
  - 911, surgical operations, energy system control, transportation system control
- Networks more open than ever before
  - Global, ubiquitous Internet, wireless
- Malicious Users
  - Selfish users: want more network resources than you
  - Malicious users: would hurt you even if it doesn't get them more network resources

Network Security Problems

- Host Compromise
  - Attacker gains control of a host
- Denial-of-Service
  - Attacker prevents legitimate users from gaining service
- Attack can be both
  - E.g., host compromise that provides resources for denial-of-service

Host Compromise

- One of earliest major Internet security incidents
  - Internet Worm (1988): compromised almost every BSD-derived machine on Internet
- Today: estimated that a single worm could compromise 10M hosts in < 5 min
- Attacker gains control of a host
  - Read data
  - Erase data
  - Compromise another host
  - Launch denial-of-service attacks on another host

Definitions

- Worm
  - Replicates itself
  - Usually relies on stack overflow attack
- Virus
  - Program that attaches itself to another (usually trusted) program
- Trojan horse
  - Program that allows a hacker a back way
  - Usually relies on user exploitation
- Botnet
  - A collection of programs running autonomously and controlled remotely
  - Can be used to spread out worms, mounting DDoS attacks

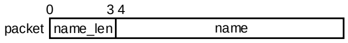
Host Compromise: Stack Overflow

- Typical code has many bugs because those bugs are not triggered by common input
- Network code is vulnerable because it accepts input from the network
- Network code that runs with high privileges (i.e., as root) is especially dangerous
  - E.g., web server

Example

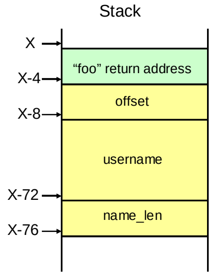
- What is wrong here?

```
// Copy a variable length user name from a packet
#define MAXNAMELEN 64
int offset = OFFSET_USERNAME;
char username[MAXNAMELEN];
int name_len;

name_len = packet[offset];
memcpy(&username, packet[offset + 1], name_len);
```



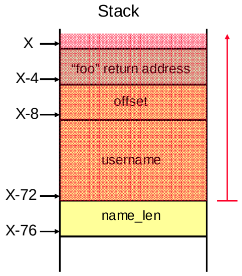
Example

```
void foo(packet) {
  #define MAXNAMELEN 64
  int offset = OFFSET_USERNAME;
  char username[MAXNAMELEN];
  int name_len;

  name_len = packet[offset];
  memcpy(&username,
         packet[offset + 1],name_len);
  …
}
```



Example

```
void foo(packet) {
  #define MAXNAMELEN 64
  int offset = OFFSET_USERNAME;
  char username[MAXNAMELEN];
  int name_len;

  name_len = packet[offset];
  memcpy(&username,
         packet[offset + 1],name_len);
  …
}
```



Effect of Stack Overflow

- Write into part of the stack or heap
  - Write arbitrary code to part of memory
  - Cause program execution to jump to arbitrary code
- Worm
  - Probes host for vulnerable software
  - Sends bogus input
  - Attacker can do anything that the privileges of the buggy program allows
    - Launches copy of itself on compromised host
  - Spread at exponential rate
  - 10M hosts in < 5 minutes

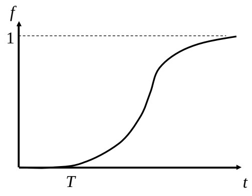
Outline

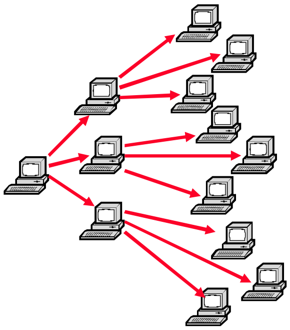
- ➢ Worm propagation
- Threat detection – content sifting

Worm Spreading

$f = (e^{K(t-T)} - 1) / (1 + e^{K(t-T)})$

- f – fraction of hosts infected
- K – rate at which one host can compromise others
- T – start time of the attack





Worm Examples

- Morris worm (1988)
- Code Red (2001)
- MS Slammer (January 2003)
- MS Blaster (August 2003)

Morris Worm (1988)

- Infect multiple types of machines (Sun 3 and VAX)
  - Spread using a Sendmail bug
- Attack multiple security holes including
  - Buffer overflow in fingerd
  - Debugging routines in Sendmail
  - Password cracking
- Intend to be benign but it had a bug
  - Fixed chance the worm wouldn't quit when reinfecting a machine → number of worm on a host built up rendering the machine unusable

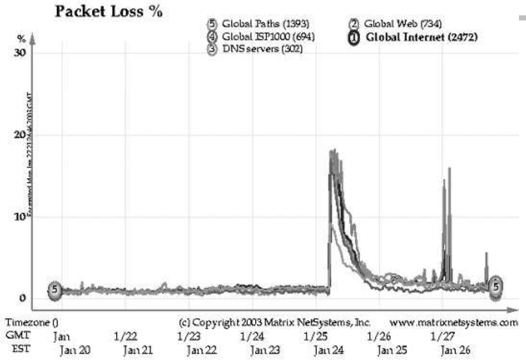
Code Red Worm (2001)

- Attempts to connect to TCP port 80 on a randomly chosen host
- If successful, the attacking host sends a crafted HTTP GET request to the victim, attempting to exploit a buffer overflow
- Worm "bug": all copies of the worm use the same random generator to scan new hosts
  - DoS attack on those hosts
  - Slow to infect new hosts
- 2nd generation of Code Red fixed the bug!
  - It spread much faster

MS SQL Slammer (January 2003)

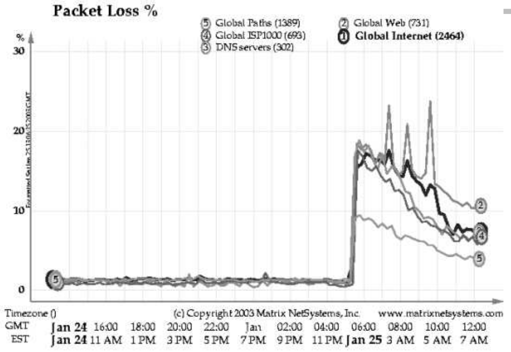
- Uses UDP port 1434 to exploit a buffer overflow in MS SQL server
- Effect
  - Generate massive amounts of network packets
  - Brought down as many as 5 of the 13 internet root name servers
- Others
  - The worm only spreads as an in-memory process: it never writes itself to the hard drive
    - Solution: close UDP port on fairewall and reboot

MS SQL Slammer (January 2003)



(From http://www.f-secure.com/v-descs/mssqlm.shtml)

MS SQL Slammer (January 2003)



(From http://www.f-secure.com/v-descs/mssqlm.shtml)

MS Blaster (August 2003)

- Exploit a buffer overflow vulnerability of the RPC (Remote Procedure Call) service
- Scan a random IP range to look for vulnerable systems on TCP port 135
- Open TCP port 4444, which could allow an attacker to execute commands on the system
- DoS windowsupdate.com on certain versions of Windows

Hall of Shame

- Software that have had many stack overflow bugs:
  - BIND (most popular DNS server)
  - RPC (Remote Procedure Call, used for NFS)
    - NFS (Network File System), widely used at UCB
  - Sendmail (most popular UNIX mail delivery software)
  - IIS (Windows web server)
  - SNMP (Simple Network Management Protocol, used to manage routers and other network devices)

Spreading faster—distributed coordination (*Warhol* worms)

- Idea 1: *reduce redundant scanning*.
  - Construct permutation of address space.
  - Each new worm instance starts at random point
  - Worm instance that “encounters” another instance re-randomizes
- Idea 2: *reduce slow startup phase*.
  - Construct a “hit-list” of vulnerable servers in advance
  - Then: for 1M vulnerable hosts, 10K hit-list, 100 scans/worm/sec, 1 sec to infect → 99% infection in 5 minutes.

Spreading still faster — *Flash* worms

- Idea: use an *Internet-sized hit list*.
  - Initial copy of the worm has the entire hit list
  - Each generation, infects n from the list, gives each 1/n of list
  - Need to engineer for locality, failure & redundancy.
  - But: n = 10 requires, 7 generations to infect 10^7 hosts → tens of seconds.

How can we defend against Internet-scale worms?

- Time scales rule out human intervention → Need automated detectors, response (And perhaps honeypots to confuse scanning?)
- Very hard research question!
- And it’s only half of the problem . . .

Contagion worms

- Suppose you have two exploits: Es (Web server) and Ec (Web client)
- You infect a server (or client) with Es (Ec)
- Then you . . . wait (Perhaps you bait, e.g., host porn)
- When vulnerable client arrives, infect it
- You send over *both* Es and Ec
- As client happens to visit other vulnerable servers) infects

Contagion worms (cont'd)

- No change in communication patterns, other than slightly larger-than-usual transfers
- How do you detect this?
- How bad can it be?

Outline

- Worm propagation
- ➢ Threat detection – content sifting

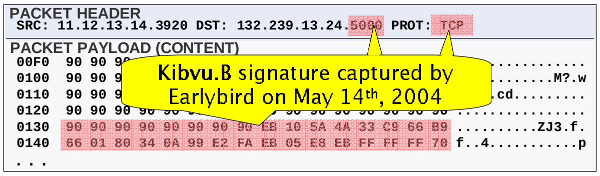
Threat Detection

- Both defense and deterrence are predicated on getting good ***intelligence***
  - Need to detect, characterize and analyze new malware threats
  - Need to be do it quickly across a very large number of events
- Classes of monitors
  - Network-based
  - Endpoint-based
- Monitoring environments
  - In-situ: real activity as it happens
    - Network/host IDS
  - Ex-situ: "canary in the coal mine"
    - HoneyNets/Honeypots

(Stefan Savage, UCSD *)

Worm Signature Inference

- Challenge: need to automatically ***learn*** a content "signature" for each new worm – in less than a second!
- Approach: Monitor network and look for strings common to traffic with *worm-like* behavior
- Signatures can then be used for content filtering



PACKET HEADER

```
SRC: 11.12.13.14.3920 DST: 132.239.13.24.5000 PROT: TCP
```

PACKET PAYLOAD (CONTENT)

```
00F0  90 90 90                                          ...........
0100  90 90 9                                           .......M?.w
0110  90 90 9                                           .cd........
0120  90 90 90                                          ................
0130  90 90 90 90 90 90 90 90 EB 10 5A 4A 33 C9 66 B9 ..........ZJ3.f.
0140  66 01 80 34 0A 99 E2 FA EB 05 E8 EB FF FF FF 70 f..4...........p
. . .
```

Kibvu.B signature captured by Earlybird on May 14th, 2004

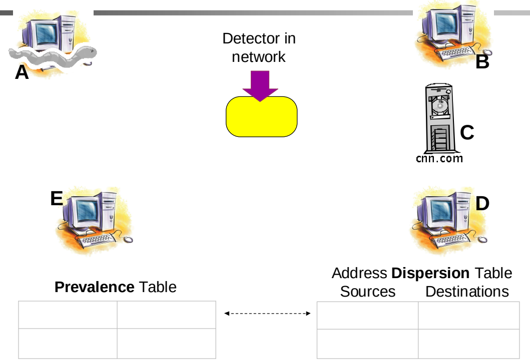
(Stefan Savage, UCSD *)

Content sifting

- Assume there exists some (relatively) unique invariant bitstring *W* across all instances of a particular worm
- Two consequences
  - **Content Prevalence**: *W* will be more common in traffic than other bitstrings of the same length
  - **Address Dispersion**: the set of packets containing *W* will address a disproportionate number of distinct sources and destinations
- *Content sifting*: find *W*'s with high content prevalence and high address dispersion and drop that traffic

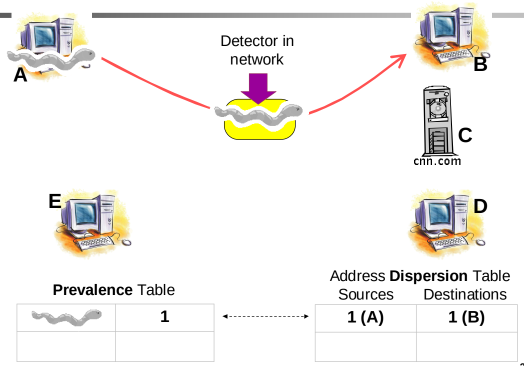
(Stefan Savage, UCSD *)

The basic algorithm



(Stefan Savage, UCSD *)

The basic algorithm



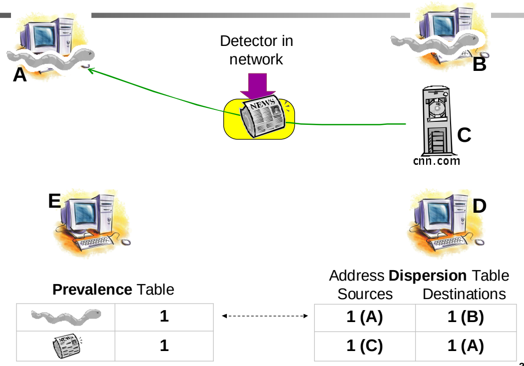
Prevalence Table

	1

Address **Dispersion** Table

Sources	Destinations
1 (A)	1 (B)

The basic algorithm



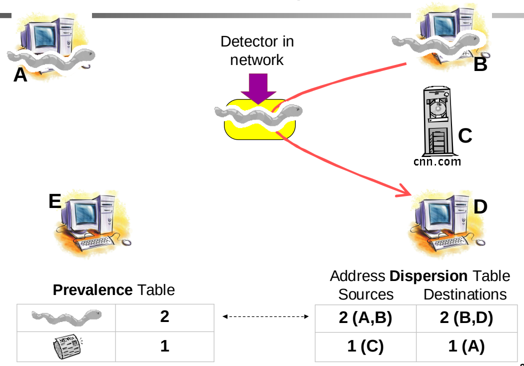
Prevalence Table

	1
	1

Address **Dispersion** Table

Sources	Destinations
1 (A)	1 (B)
1 (C)	1 (A)

The basic algorithm



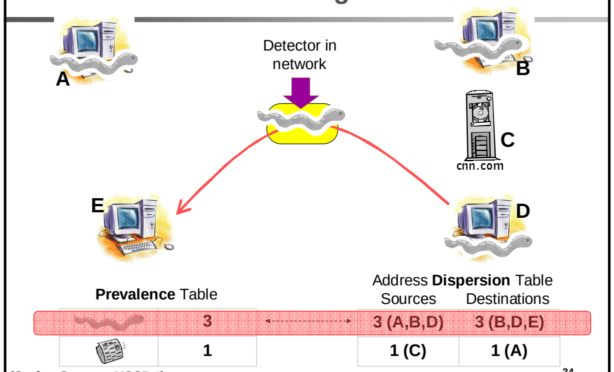
Prevalence Table

	2
	1

Address **Dispersion** Table

Sources	Destinations
2 (A,B)	2 (B,D)
1 (C)	1 (A)

The basic algorithm



Prevalence Table

	3
	1

Address **Dispersion** Table

Sources	Destinations
3 (A,B,D)	3 (B,D,E)
1 (C)	1 (A)

Challenges

- **Computation**
  - To support a 1Gbps line rate we have 12us to process each packet, at 10Gbps 1.2us, at 40Gbps…
    - Dominated by memory references; state expensive
  - Content sifting requires looking at ***every*** byte in a packet
- **State**
  - On a fully-loaded 1Gbps link a naïve implementation can easily consume 100MB/sec for table
  - Computation/memory duality: on high-speed (ASIC) implementation, latency requirements may limit state to on-chip SRAM

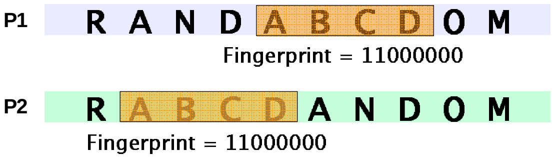
Which substrings to index?

- **Approach 1: Index all substrings**
  - Way too many substrings → too much computation → too much state
- **Approach 2: Index whole packet**
  - Very fast but trivially evadable (e.g., Witty, Email Viruses)
- **Approach 3: Index all contiguous substrings of a fixed length ‘S’**
  - Can capture all signatures of length ‘S’ and larger

A B C D E F G H I J K

How to represent substrings?

- Store **hash** instead of literal to reduce state
- **Incremental hash** to reduce computation
- **Rabin fingerprint** is one such efficient incremental hash function [Rabin81,Manber94]
  - One multiplication, addition and mask per byte

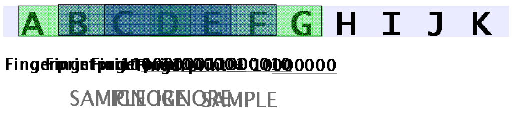


How to subsample?

- **Approach 1: sample packets**
  - If we chose 1 in N, detection will be slowed by N
- **Approach 2: sample at particular byte offsets**
  - Susceptible to simple evasion attacks
  - No guarantee that we will sample same sub-string in every packet
- **Approach 3: sample based on the hash of the substring**

Value sampling [Manber '94]

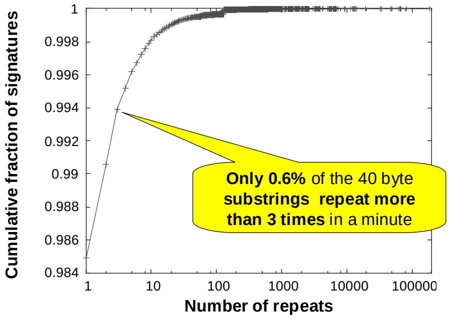
- Sample hash if last 'N' bits of the hash are equal to the value 'V'
  - The number of bits 'N' can be dynamically set
  - The value 'V' can be randomized for resiliency



- P_{track} → Probability of selecting **at least one** substring of length *S* in a *L* byte invariant
  - For 1/64 sampling (last 6 bits equal to 0), and 40 byte substrings

    P_{track} = 99.64% for a 400 byte invariant

Observation: High-prevalence strings are rare



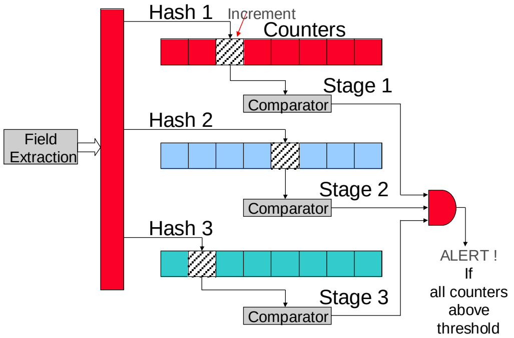
Only 0.6% of the 40 byte **substrings repeat more than 3 times** in a minute

---

Efficient high-pass filters for content

- Only want to keep state for prevalent substrings
- Chicken vs egg: how to count strings without maintaining state for them?
- **Multi Stage Filters:** randomized technique for counting "heavy hitter" network flows with low state and few false positives [Estan02]
  - Instead of using flow id, use **content hash**
    - Rabin Fingerprints with Mandber's Value sampling
  - **Three orders of magnitude** memory savings

Finding "heavy hitters" via Multistage Filters

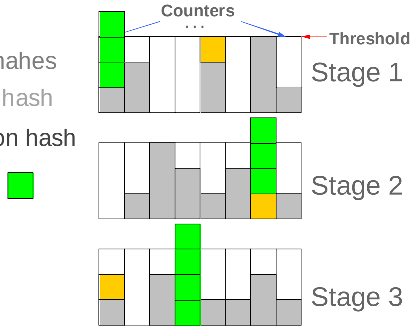


Multistage filters in action

Grey = other hahes

Yellow = rare hash

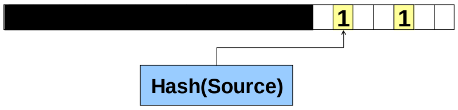
Green = common hash



Observation:
High *address dispersion* is rare too

- Naïve implementation might maintain a list of sources (or destinations) for each string hash
- But dispersion **only** matters if its *over* threshold
  - Approximate counting may suffice
  - **Trades accuracy for state in data structure**
- **Scalable Bitmap Counters**
  - Similar to multi-resolution bitmaps [Estan03]
  - Reduce memory by 5x for modest accuracy error

Scalable Bitmap Counters



- Hash : based on Source (or Destination)
- Sample : keep only a sample of the bitmap
- Estimate : scale up sampled count
- Adapt : periodically increase scaling factor

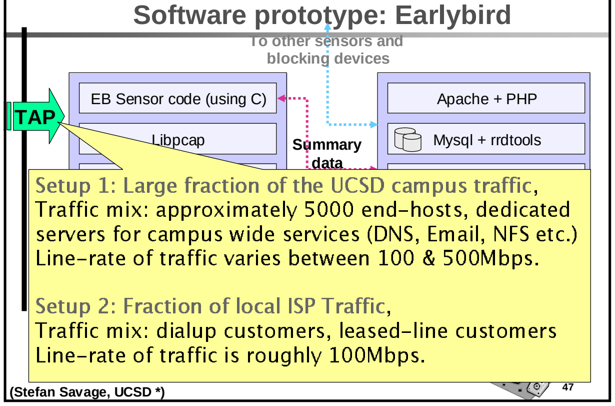
$$\text{Error Factor} = 2/(2^{numBitmaps}-1)$$

- With 3, 32-bit bitmaps, error factor = 28.5%

Content sifting summary

- Index fixed-length substrings using incremental hashes
- Subsample hashes as function of hash value
- Multi-stage filters to filter out uncommon strings
- Scalable bitmaps to tell if number of distinct addresses per hash crosses threshold
- **Now** its fast enough to implement

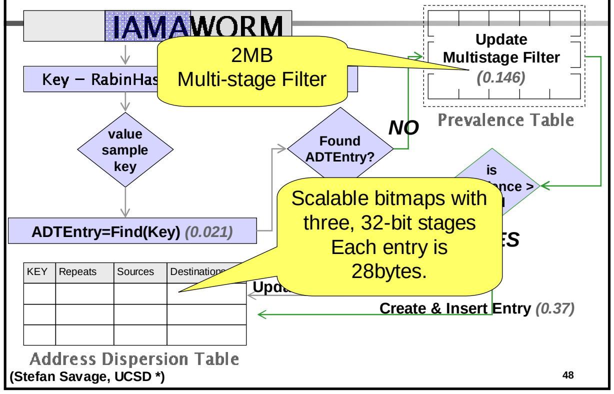
Software prototype: Earlybird



Setup 1: Large fraction of the UCSD campus traffic,
Traffic mix: approximately 5000 end-hosts, dedicated servers for campus wide services (DNS, Email, NFS etc.)
Line-rate of traffic varies between 100 & 500Mbps.

Setup 2: Fraction of local ISP Traffic,
Traffic mix: dialup customers, leased-line customers
Line-rate of traffic is roughly 100Mbps.

Content Sifting in Earlybird



2MB
Multi-stage Filter

Scalable bitmaps with three, 32-bit stages
Each entry is 28bytes.

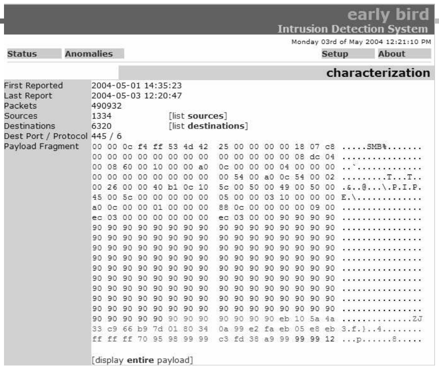
Content sifting overhead

- Mean per-byte processing cost
  - **0.409 microseconds**, without value sampling
  - **0.042 microseconds**, with 1/64 value sampling (~60 microseconds for a 1500 byte packet, can keep up with 200Mbps)
- Additional overhead in per-byte processing cost for flow-state maintenance (if enabled):
  - **0.042 microseconds**

Experience

- Quite *good.*
  - Detected and automatically generated signatures for **every** known worm outbreak over eight months
  - ***Can*** produce a precise signature for a new worm in a *fraction* of a second
  - Software implementation keeps up with 200Mbps
- **Known worms detected**:
  - Code Red, Nimda, WebDav, Slammer, Opaserv, …
- **Unknown worms (with no public signatures) detected**:
  - MsBlaster, Bagle, Sasser, Kibvu, …

Sasser



Monday 03rd of May 2004 12:21:10 PM

characterization

First Reported 2004-05-01 14:35:23
Last Report 2004-05-03 12:20:47
Packets 490932
Sources 1334 [list **sources**]
Destinations 6320 [list **destinations**]
Dest Port / Protocol 445 / 6
Payload Fragment

```
00 00 0c f4 ff 53 4d 42  25 00 00 00 00 18 07 c8 .....SMB%.......
00 00 00 00 00 00 00 00  00 00 00 00 00 08 dc 04 ................
00 08 60 00 10 00 00 a0  0c 00 00 00 04 00 00 00 ..`.............
00 00 00 00 00 00 00 00  00 54 00 a0 0c 54 00 02 .........T...T..
00 26 00 00 40 b1 0c 10  5c 00 50 00 49 00 50 00 .&..@...\.P.I.P.
45 00 5c 00 00 00 00 00  05 00 00 03 10 00 00 00 E.\.............
a0 0c 00 00 01 00 00 00  88 0c 00 00 00 00 09 00 ................
ec 03 00 00 00 00 00 00  ec 03 00 00 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90  90 90 90 90 eb 10 5a 4a ..............ZJ
33 c9 66 b9 7d 01 80 34  0a 99 e2 fa eb 05 e8 eb 3.f.}..4........
ff ff ff 70 95 98 99 99  c3 fd 38 a9 99 99 99 12 ...p......8.....
```

[display **entire** payload]

False Negatives

- Easy to prove presence, impossible to prove absence
- **Live evaluation**: over 8 months detected every worm outbreak reported on popular security mailing lists
- **Offline evaluation**: several traffic traces run against both Earlybird and Snort IDS (w/all worm-related signatures)
  - Worms not detected by Snort, but detected by Earlybird
  - The converse never true

False Positives

- **Common protocol headers**
  - Mainly HTTP and SMTP headers
  - Distributed (P2P) system protocol headers
  - **Procedural whitelist**
    - Small number of popular protocols
- **Non-worm epidemic Activity**
  - **SPAM**
  - BitTorrent

```
GNUTELLA.CONNECT
/0.6..X-Max-TTL:
.3..X-Dynamic-Qu
erying:.0.1..X-V
ersion:.4.0.4..X
-Query-Routing:.
0.1..User-Agent:
.LimeWire/4.0.6.
.Vendor-Message:
.0.1..X-Ultrapee
r-Query-Routing:
```