Worms: Attacks and Defense (II)

Dawn Song

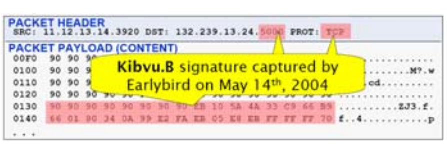
dawnsong@cs.berkeley.edu

Review

- **Worms**
  - **Self-propagating**
  - **How does worm propagate?**
  - **Worm modeling & measurement**
- **Today: defenses**

Identifying Worm Patterns

- **Monitor network and look for strings common to traffic with worm-like behavior**
  - **EarlyBird**
  - **Signatures can then be used for content filtering**



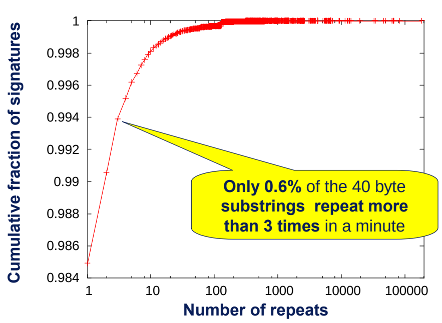
Slide: S Savage

Content sifting

- **Assume there exists some (relatively) unique invariant bitstring *W* across all instances of a particular worm (*true today, not tomorrow...*)**
- **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**

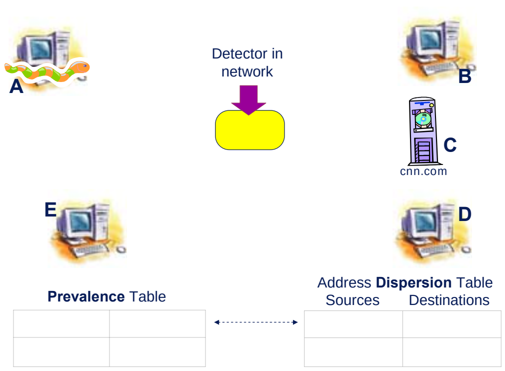
Slide: S Savage

Observation: High-prevalence strings are rare



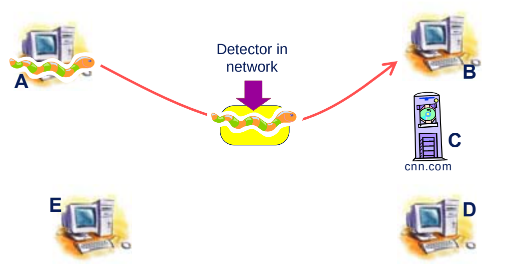
(Stefan Savage, UCSD *)

The basic algorithm



(Stefan Savage, UCSD *)

The basic algorithm



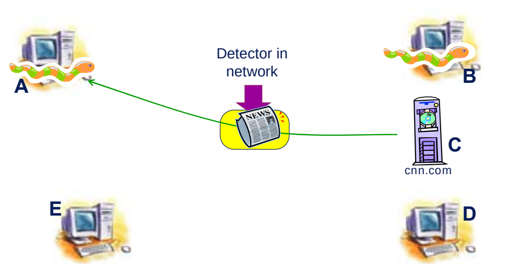
Prevalence Table

	1

Address **Dispersion** Table

Sources	Destinations
1 (A)	1 (B)

The basic algorithm



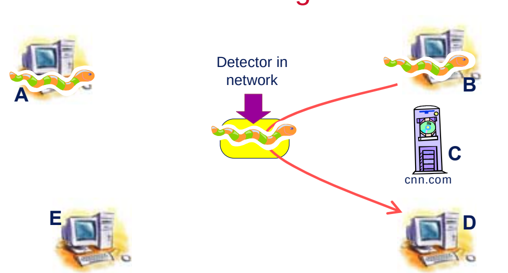
Prevalence Table

	1
	1

Address **Dispersion** Table

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

The basic algorithm



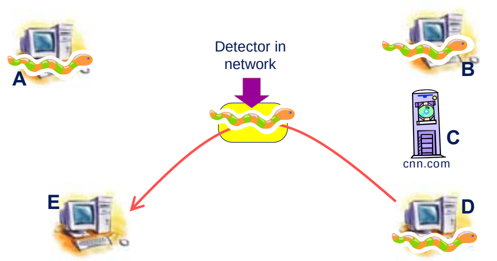
Prevalence Table

	2
	1

Address **Dispersion** Table

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

The basic algorithm



Prevalence Table		Address Dispersion Table: Sources	Destinations
	3	3 (A,B,D)	3 (B,D,E)
	1	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**

P1 R A N D **A B C D** O M

Fingerprint = 11000000

P2 R **A B C D** A N D O M

Fingerprint = 11000000

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

Solution

- **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**

- **This is fast enough to implement**
  - **Netsift bought by Cisco**

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

(Stefan Savage, UCSD *)

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:
```

(Stefan Savage, UCSD *)

Other Disadvantages

- Insufficient for polymorphic worms & unseen variants
- What kinds of invariants can it discover?
  - Depending on the classes of functions learned
  - What other functions may be of interest to learn?
- No guarantee of signature quality
  - How to evaluate signature quality?
- Susceptible to adversarial learning
  - Attackers crafting malicious samples
  - How?
- Purely bit-pattern syntactic approach, so no semantic understanding of vulnerability
  - Only generating exploit-signatures

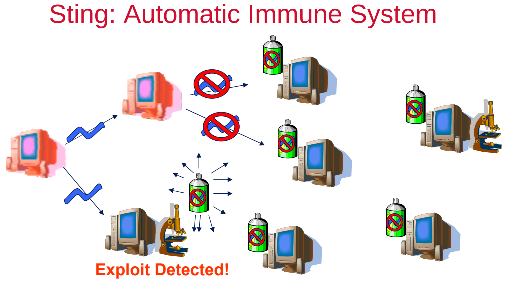
Another Approach

- **Semantic-based detection & defense**

Administravia

- **Group effort for project**

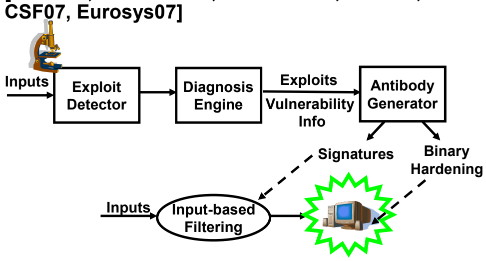
Sting: Automatic Immune System



- **Requirements:**
  - **Fast**
  - **No false positives**
  - **Handle morphing attacks**
  - **Work directly on binary (for commodity software, legacy code)**

Solution: Semantic-based

- **Focus on the root cause (the vulnerability)**
- **Detect exploits, diagnose, generate antibodies**
  - **[NDSS05, IEEE S&P05, IEEE S&P06, RAID06, NDSS06, CSF07, Eurosys07]**



Exploit Detection

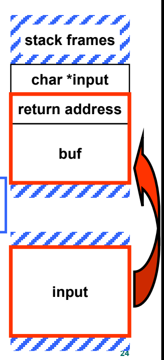
- **Question:**
  Will given network inputs exploit new vulnerability?
- **Use binary instrumentation to detect safety violations**
  - **E.g., dynamic taint analysis**
- **Advantages:**
  - **Semantic-based: focus on root cause of attack**
    - **In contrast to behavior-based detection**
  - **Detects wide spectrum of overwrite attacks**
    - **Higher coverage than previous techniques**
  - **Supports causality analysis**
  - **No false positives (with verification), low false negatives**

HTTP-like Example

```
int check_http( char *input ) {
  char buf[8];
  if (strncmp(input, “get”,3) != 0 &&
      strncmp(input, “put”,3) != 0 )
    return -1;
  if (input[3] != ‘/‘ ) return -1;
  strncpy( buf, input, 4);
  int i = 4;
  while ( input[i] != ‘\n’)
    { buf[i] = input[i];
      i++; }
  return i;
}
```

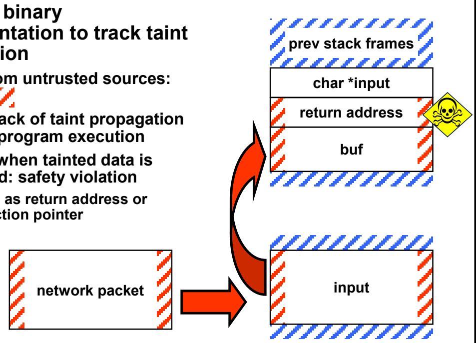
mov %al,(%edx,%ecx,1)
%edx is EA of buf, %ecx is i

Vulnerability condition: $i \geq 8$



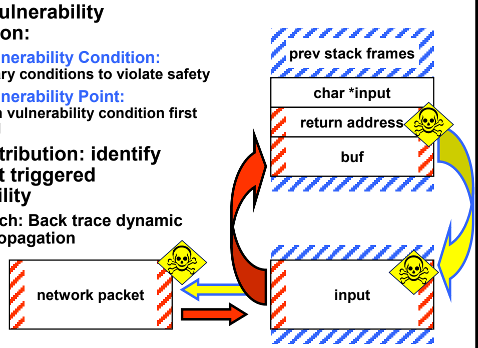
Dynamic Taint Analysis

- **Dynamic binary instrumentation to track taint propagation**
  - **Data from untrusted sources: tainted**
  - **Keep track of taint propagation during program execution**
  - **Detect when tainted data is misused: safety violation**
    - » ***e.g.*, as return address or function pointer**



Automatic Diagnosis

- **Extract vulnerability information:**
  - **The Vulnerability Condition:** **Necessary conditions to violate safety**
  - **The Vulnerability Point:** **Location vulnerability condition first satisfied**
- **Attack attribution: identify input that triggered vulnerability**
  - **Approach: Back trace dynamic taint propagation**



Automatic Vulnerability Signature Generation

- **Instead of bit patterns, use root cause**
  - **Generating signatures based on vulnerability**
- **As exploits morph, they need to trigger vulnerability**
- **So, vulnerability puts constraints on exploits**
- **Problem reduction:**
  - **Signature generation = constraints on inputs that trigger vulnerability**
- **Symbolic execution**
  - **A very useful concept, we'll see more of it later in class**
- **Soundness guaranteed (no false positives)**

Conclusion

- **Worms**
  - What is a worm?
  - How does it propagate?
  - How to measure it?
- **Detection & Defense**
  - Traffic monitoring based detection & defense
  - Semantic-based detection & defense