

Passive Monitoring for Security-Related Applications

Michalis Polychronakis
mikepo@ics.forth.gr

Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)
Crete, Greece

TERENA Networking Conference, 16 May 2006

Outline

- Introduction
- Network Intrusion Detection Systems
- Content sifting
- Sled detection
- Network-level Emulation
- Summary

Passive Network Monitoring

- Examine the network traffic as it passes by
 - Packet capture (tcpdump), NetFlow, ...
- Non-intrusive: invisible on the network
 - vs. active monitoring, e.g., ping
- Many applications
 - Performance Measurements
 - Network trouble-shooting
 - Network planning
 - Traffic characterization
 - ...
 - **Security** ← focus of this talk

```
15:07:16.609603 IP 139.91.70.46.631 > 139.91.70.255.631: UDP, length 122
15:07:16.821924 IP 139.91.171.116.1049 > 239.255.255.250.1900: UDP, length 325
15:07:16.821980 IP 139.91.171.116.1049 > 239.255.255.250.1900: UDP, length 325
15:07:16.822297 IP 139.91.70.148.8008 > 239.255.255.250.1900: UDP, length 101
15:07:16.822370 IP 139.91.70.26.8008 > 239.255.255.250.1900: UDP, length 101
15:07:16.825070 IP 139.91.70.254 > 224.0.0.13: PIMv2, Assert, length: 28
15:07:16.826708 IP 139.91.70.253 > 224.0.0.13: PIMv2, Assert, length: 28
15:07:16.869700 endnode-hello endnode vers 2 eco 0 ueco 0 src 1.10 blksize 149:
rtr 0.0 hello 10 data 2
15:07:16.929894 IP 139.91.171.116.1049 > 239.255.255.250.1900: UDP, length 325
15:07:17.040099 IP 139.91.171.116.1049 > 239.255.255.250.1900: UDP, length 361
15:07:17.119970 IP 139.91.70.254.1985 > 224.0.0.2.1985: HSRPv0-hello 20: state:
tandby group=70 addr=139.91.70.80
15:07:17.149897 IP 139.91.171.116.1049 > 239.255.255.250.1900: UDP, length 361
15:07:17.259974 IP 139.91.171.116.1049 > 239.255.255.250.1900: UDP, length 429
15:07:17.284411 802.lid config 2000.00:d0:00:dc:50:45.2105 root 2000.00:d0:00:d
50:45 pathcost 0 age 0 max 20 hello 2 fdelay 15
15:07:17.369924 IP 139.91.171.116.1049 > 239.255.255.250.1900: UDP, length 429
15:07:17.696390 endnode-hello endnode vers 2 eco 0 ueco 0 src 1.10 blksize 149:
rtr 0.0 hello 10 data 2
15:07:18.764737 IP 139.91.70.253 > 224.0.0.13: PIMv2, Assert, length: 28
15:07:18.963784 IP 139.91.70.253.1985 > 224.0.0.2.1985: HSRPv0-hello 20: state:
ctive group=70 addr=139.91.70.80
15:07:18.988021 IP 139.91.70.254 > 224.0.0.10: EIGRP Hello, length: 40
15:07:18.999754 IP 139.91.70.253 > 224.0.0.10: EIGRP Hello, length: 40
15:07:19.291410 802.lid config 2000.00:d0:00:dc:50:45.2105 root 2000.00:d0:00:d
50:45 pathcost 0 age 0 max 20 hello 2 fdelay 15
15:07:19.351836 00:d0:d3:36:6f:54 > 01:00:0c:dd:dd:sap aa ui/C
15:07:19.923630 endnode-hello endnode vers 2 eco 0 ueco 0 src 1.10 blksize 149:
rtr 0.0 hello 10 data 2
15:07:20.004023 IP 139.91.70.254.1985 > 224.0.0.2.1985: HSRPv0-hello 20: state:
tandby group=70 addr=139.91.70.80
15:07:20.821598 IP 139.91.70.148.8008 > 239.255.255.250.1900: UDP, length 101
15:07:21.292518 802.lid config 2000.00:d0:00:dc:50:45.2105 root 2000.00:d0:00:d
50:45 pathcost 0 age 0 max 20 hello 2 fdelay 15
15:07:21.609511 IP 139.91.70.46.631 > 139.91.70.255.631: UDP, length 153
15:07:21.883722 IP 139.91.70.253.1985 > 224.0.0.2.1985: HSRPv0-hello 20: state:
ctive group=70 addr=139.91.70.80
15:07:22.129438 IP 139.91.70.46.41988 > 139.91.70.255.111: UDP, length 112
15:07:22.864093 IP 139.91.70.254.1985 > 224.0.0.2.1985: HSRPv0-hello 20: state:
tandby group=70 addr=139.91.70.80
15:07:23.293656 802.lid config 2000.00:d0:00:dc:50:45.2105 root 2000.00:d0:00:d
50:45 pathcost 0 age 0 max 20 hello 2 fdelay 15
15:07:23.440208 IP 139.91.70.254 > 224.0.0.10: EIGRP Hello, length: 40
15:07:23.671846 IP 139.91.70.253 > 224.0.0.10: EIGRP Hello, length: 40
15:07:24.009474 IP 139.91.70.46.631 > 139.91.70.255.631: UDP, length 117
15:07:24.594258 arp who-has 139.91.70.181 tell 139.91.70.254
15:07:24.755842 IP 139.91.70.253.1985 > 224.0.0.2.1985: HSRPv0-hello 20: state:
ctive group=70 addr=139.91.70.80
15:07:25.294625 802.lid config 2000.00:d0:00:dc:50:45.2105 root 2000.00:d0:00:d
50:45 pathcost 0 age 0 max 20 hello 2 fdelay 15
15:07:25.609338 IP 139.91.70.46.631 > 139.91.70.255.631: UDP, length 138
15:07:25.864144 IP 139.91.70.254.1985 > 224.0.0.2.1985: HSRPv0-hello 20: state:
tandby group=70 addr=139.91.70.80
15:07:26.139315 IP 139.91.70.46.41988 > 139.91.70.255.111: UDP, length 112
15:07:26.869271 endnode-hello endnode vers 2 eco 0 ueco 0 src 1.10 blksize 149:
rtr 0.0 hello 10 data 2
15:07:27.295746 802.lid config 2000.00:d0:00:dc:50:45.2105 root 2000.00:d0:00:d
50:45 pathcost 0 age 0 max 20 hello 2 fdelay 15
15:07:27.695642 endnode-hello endnode vers 2 eco 0 ueco 0 src 1.10 blksize 149:
rtr 0.0 hello 10 data 2
15:07:27.743866 IP 139.91.70.253.1985 > 224.0.0.2.1985: HSRPv0-hello 20: state:
ctive group=70 addr=139.91.70.80
15:07:28.067904 IP 139.91.70.253 > 224.0.0.10: EIGRP Hello, length: 40
15:07:28.264320 IP 139.91.70.254 > 224.0.0.10: EIGRP Hello, length: 40
```

Passive Network Monitoring for Security

→ How is passive monitoring used for security?

- Inspect network traffic
e.g., traffic to open services
- Try to identify malicious activity

→ Common operations

- Packet sniffing
- Filtering (e.g., “tcp and port 80”)
- IP defragmentation
- TCP stream reassembly (not always)

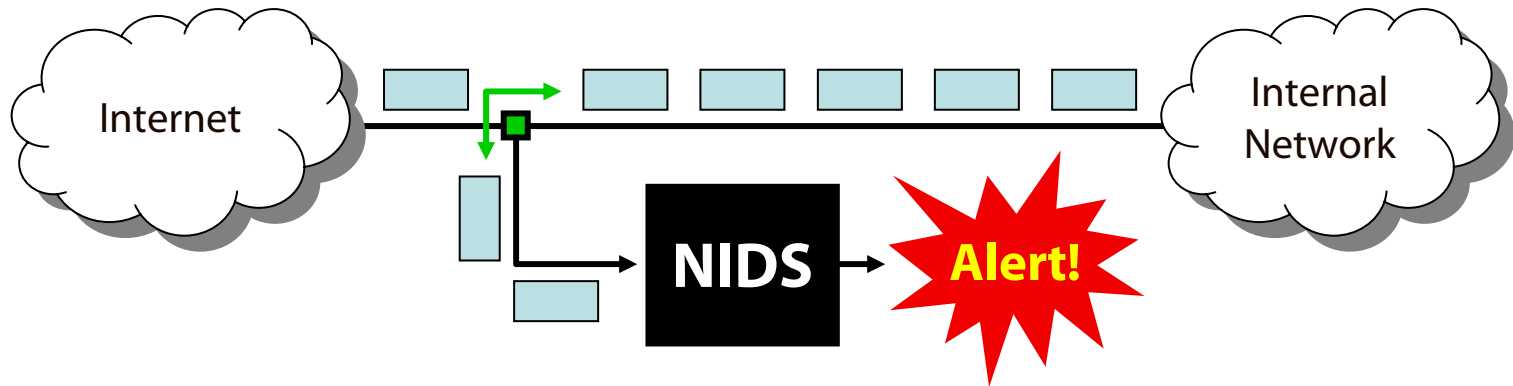


What is Intrusion Detection?

- Computers are vulnerable to cyberattacks
 - Break-ins (exploits)
 - Worms (self-replicating malicious programs)
 - **Not user-dependent (like viruses)!**
- Network Intrusion Detection Systems
 - Provide early warning of cyberattacks
 - Even for poorly administered systems like home computers



Network Intrusion Detection Systems



- A NIDS is a passive monitoring system that
 - Observes network traffic and detects suspicious behavior
 - Logs suspect packets and generates alerts / blocks packets (IPS)
 - Protects an entire network segment
- Signature-based or Anomaly-based
 - Snort is the most popular signature-based NIDS



What is a Signature?

- An attack description
 - As seen at Layer 4 or below
- Witty worm Snort signature example:

action *protocol* *source/destination* *content*

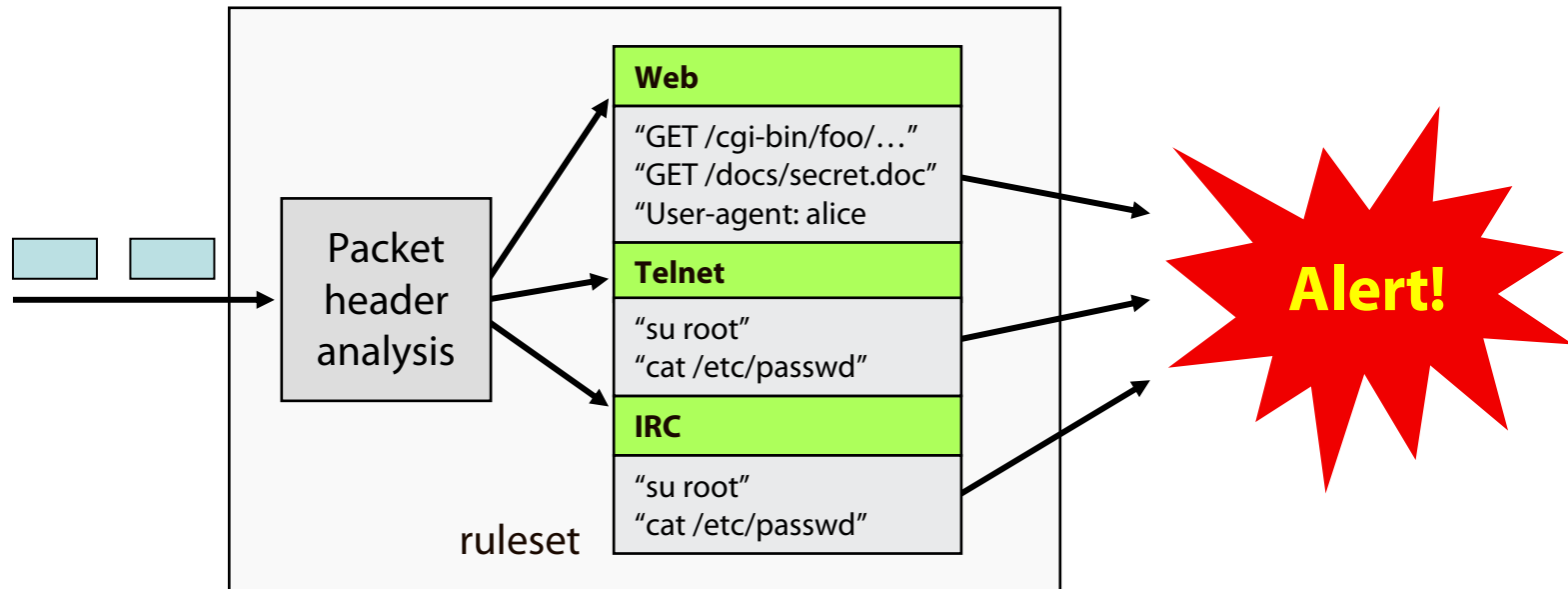
↓ ↙ ↘ ↓

alert **udp** **any 4000 -> 193.92.123.0/24 any** (msg:"ISS
PAM/Witty Worm Shellcode"; **content:"|65 74 51 68 73 6f 63
6b 54 53|"**; **depth:246**; sid:1000078; rev:1;)

Signature Matching

```
Shell - Konsole <2>
=====
05/13-16:46:08.570308  [**] [1:0:0] ISS PAM/Witty Worm Shellcode [**] [Priority: 0]
05/13-16:46:10.571009  0:4:75:AD:3E:E1 -> 0:C:6E:F3:98:3E type:0x800 len:0x42B
139.91.70.31 4000 -> 139.91.70.40 322 UDP TTL:64 TOS:0x0 ID:55882 IpLen:20 DgmLen:1053
Len: 1025
45 00 04 01 D3 B4 00 00 71 11 DD A9 DB 9A 9C A1 E.....q.....
41 AD DA A4 0F A0 C4 24 03 ED DD 38 05 00 00 00 A.....$.8....
00 00 00 12 02 00 00 00 00 00 00 00 00 00 00 .....
00 02 2C 00 05 00 00 00 00 00 00 00 6E 00 00 00 .,.....n....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
41 02 05 00 00 00 00 00 00 00 DE 03 00 00 00 00 00 A.....
00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 01 00 00 .....
01 00 00 1E 02 20 20 20 20 20 20 20 28 5E 2E 5E ..... (^.^
29 20 20 20 20 20 20 69 6E 73 65 72 74 20 77 69 ) insert wi
74 74 79 20 6D 65 73 73 61 67 65 20 68 65 72 65 tty message here
2E 20 20 20 20 20 20 28 5E 2E 5E 29 20 20 20 20 . (^.^)
20 20 20 89 E7 8B 7F 14 83 C7 08 81 C4 E8 FD FF .....
FF 31 C9 66 B9 33 32 51 68 77 73 32 5F 54 3E FF .1.f.32Qhws2_T>.
15 9C 40 0D 5E 89 C3 31 C9 66 B9 65 74 51 68 73 ..@.^..1.f.etQhs
6F 63 6B 54 53 3E FF 15 98 40 0D 5E 6A 11 6A 02 ockTS>...@.^j.j.
6A 02 FF D0 89 C6 31 C9 51 68 62 69 6E 64 54 53 j.....1.QhbindTS
3E FF 15 98 40 0D 5E 31 C9 51 51 51 81 E9 FE FF >...@.^1.QQQ...
F0 5F 51 89 E1 6A 10 51 56 FF D0 31 C9 66 B9 74 . Q..j.QV..1.f.t
```

Signature-based NIDS

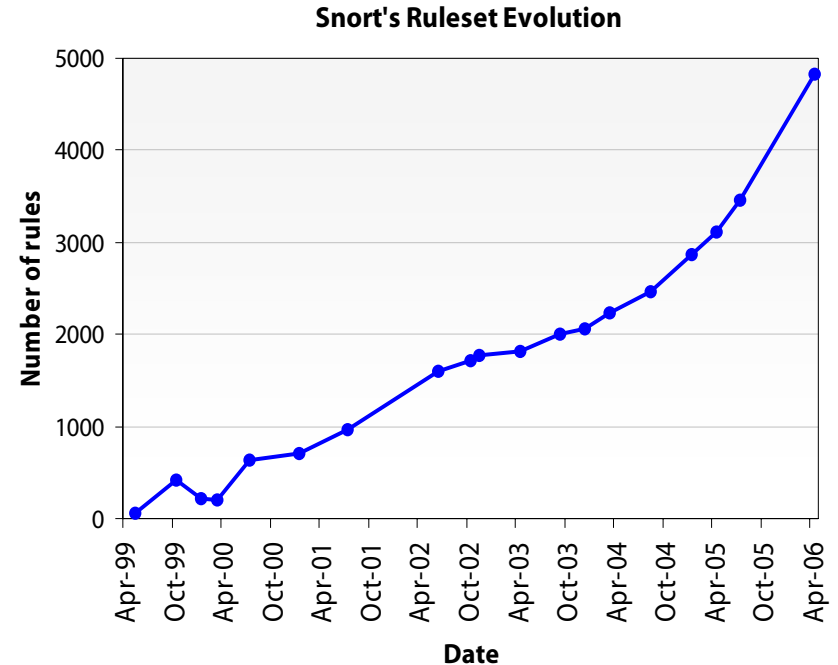


→ For each packet

- Check if it's header matches any signature group
- Then check if it's payload matches any signature

Detection is Expensive

- Gap between
 - link speeds
 - detection capacity of IDS sensors
- Pattern matching is expensive
 - 31%-81% of NIDS processing time
- The gap is increasing
 - Growing number of signatures
- What can we do?
 - **Push functionality to hardware**
 - **Improve pattern matching algorithms**
 - Antonatos et al. Piranha: Fast and Memory-efficient Pattern Matching for Intrusion Detection. IFIP-SEC'05
 - Anagnostakis et al. E2xB: A domain-specific string matching algorithm for intrusion detection. IFIP-SEC'03
 - Markatos et al. ExB: Exclusion-based signature matching for intrusion detection. CCN'02
 - **Splitting**
 - Xinidis et al. *An Active Traffic Splitter Architecture for Intrusion Detection and Prevention*. IEEE Transactions on Dependable and Secure Computing, 2006

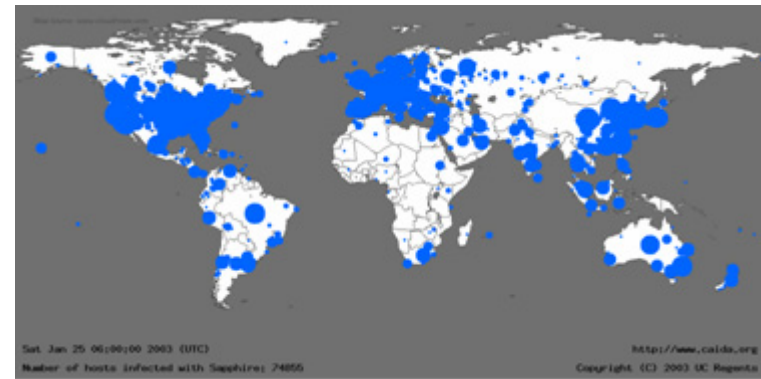


Worms

- Self-replicating programs
 - Spread by exploiting vulnerabilities in widely-used services
- NIDS alone are not effective in the face of worms
 - Reaction time must be in the order of minutes!
 - Must take humans out of the loop

**Slammer, 30 min after its release:
75.000+ infected hosts**

- **Need for automated response**
 - Automated signature generation



Automated Signature Generation

- Main idea: *look for similar traffic*
 - Frequently appearing fixed-length strings in packet payloads are an indication of a new worm outbreak
- “Content sifting”
 - Introduced by *Earlybird* [Singh '04], *Autograph* [Kim '04]
- Enables fingerprinting of previously unknown worms

Akritidis et al. *Efficient Content-Based Fingerprinting of Zero-Day Worms*. ICC'05

Common Properties of Known Worms (So Far)

→ Payload repetition

- Worm attacks seem to contain mostly the same contents

→ Diversity of destinations

- Worms spread → packets are destined to many hosts

→ Spread by clients

- Hosts that *initiate* a connection
- Worm attacks are carried out over requests, not replies

→ Small size

- An exceptionally large size would hinder propagation

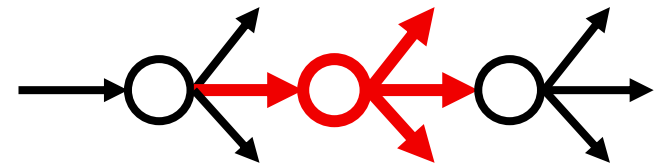
Common Properties of Known Worms (So Far)

→ Regions without NULL characters

- Worms typically exploit buffer overflow vulnerabilities
- The attack vector must not contain the string terminating character

→ Contagion

- Worm connections are chained
- Once infected, a host will propagate the worm
- We can correlate inbound and outbound traffic



→ Diversity of sources

- As the worm spreads, attacks are sent from more and more sources

→ Scanning

- Worms to this day use some form of random scanning to find targets

Putting it all together...

→ **Packet payload repetition**

- Find frequently repeated substrings
- Sent from clients to servers
- Targeting several destinations
- Are at least 200-300 bytes long
- Are within the first few kilobytes of flows

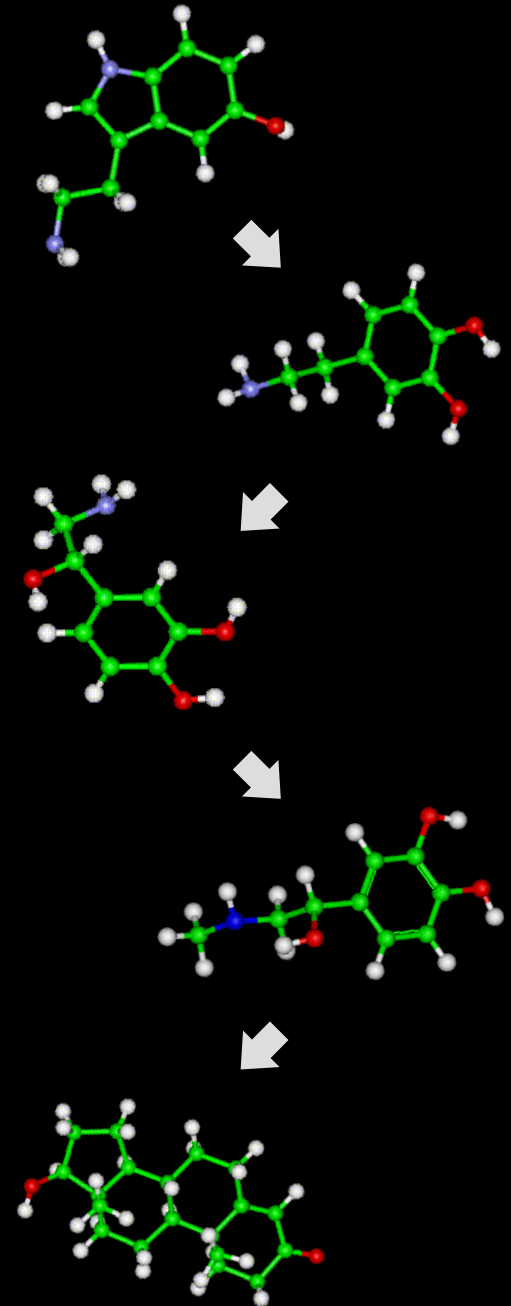
then...

→ **This is a very suspicious substring**

- Probably belongs to a worm

Polymorphism

- Attackers evolve...
 - Invent techniques to evade detection
- Polymorphism effectively evades signature-based detection
 - Known since early 90's from the virus scene
- Each attack instance is a **different mutation**
 - Extremely difficult to fingerprint even using regular expressions
- **Content-based fingerprinting becomes infeasible**
 - *Need for alternative detection methods*

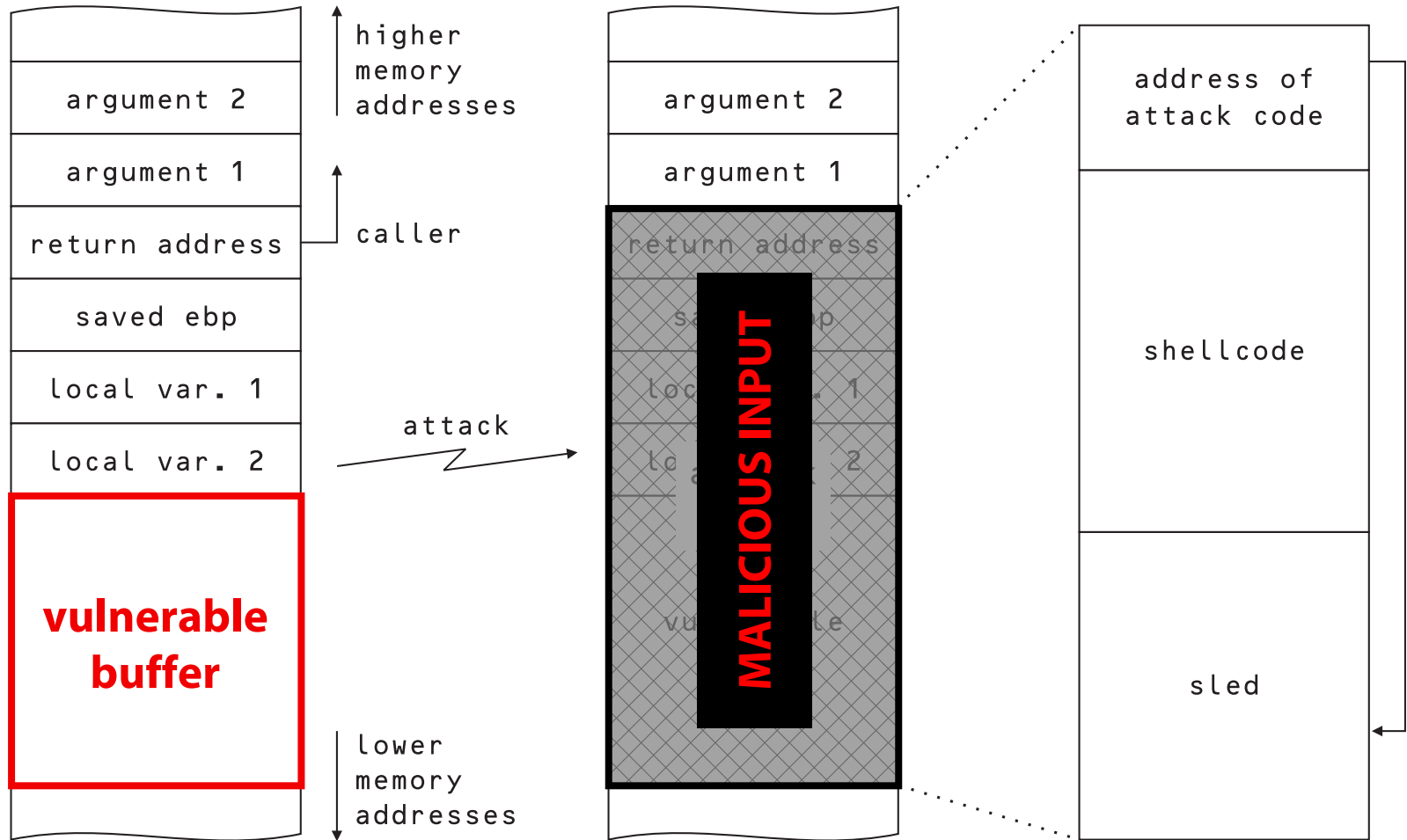


Buffer Overflow

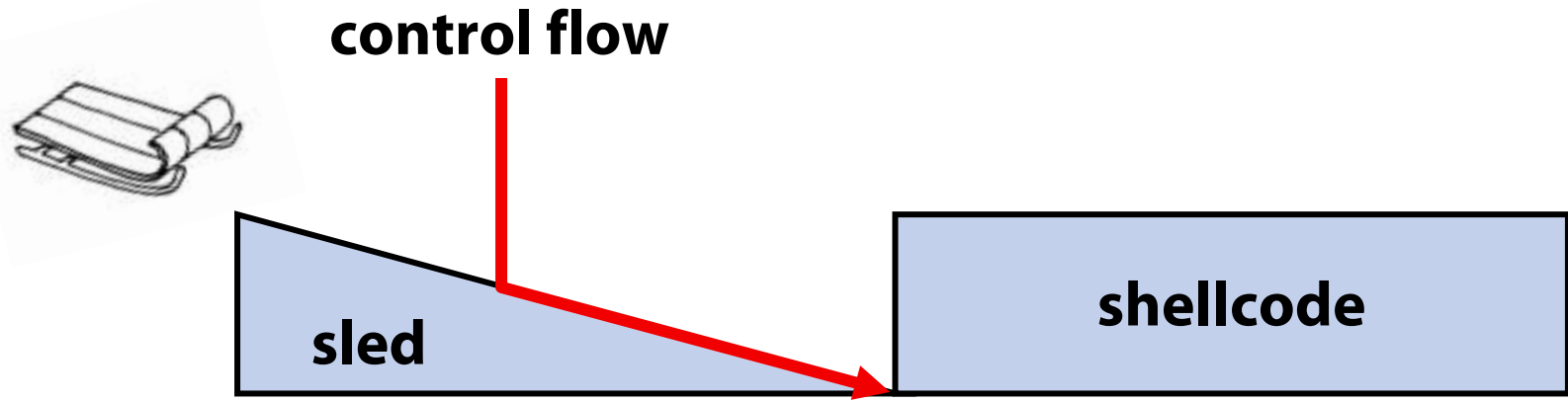
- Most common vulnerability: enables arbitrary code execution on a remote host
 - Have been a major security concern for a long time
- Tremendous exploitation opportunities in combination with the highly interconnected Internet environment
- Main means of worm propagation
 - For all known worms so far



Exploitation Example: Stack Smashing



The Sled



- The runtime location of the vulnerable buffer is only approximately known in advance
- The attacker can overcome this by appending to the malicious code a sequence of NOP instructions
- The flow of control can land in any position within the sled and then *slide* towards the shellcode by executing NOPs

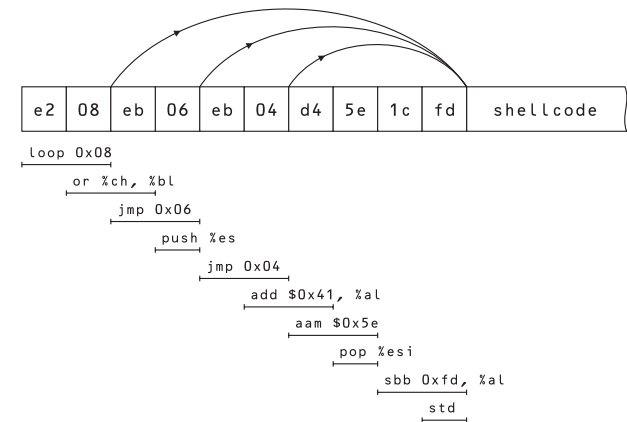
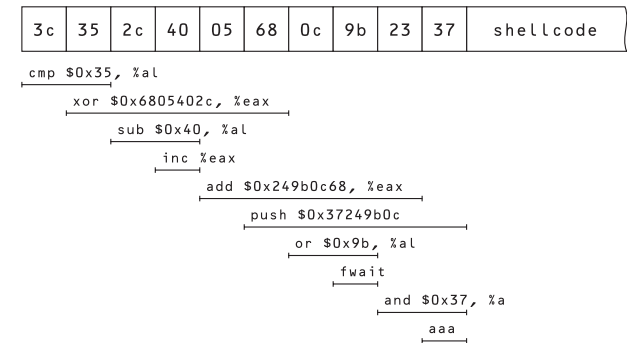
STRIDE: Obfuscated Sled Detection

→ Apply *static binary code analysis* to network traffic for identifying the sled component

- Improvement to the pioneer *Abstract Payload Execution* method [Kruegel '02]

→ Main idea: **disassemble** incoming requests

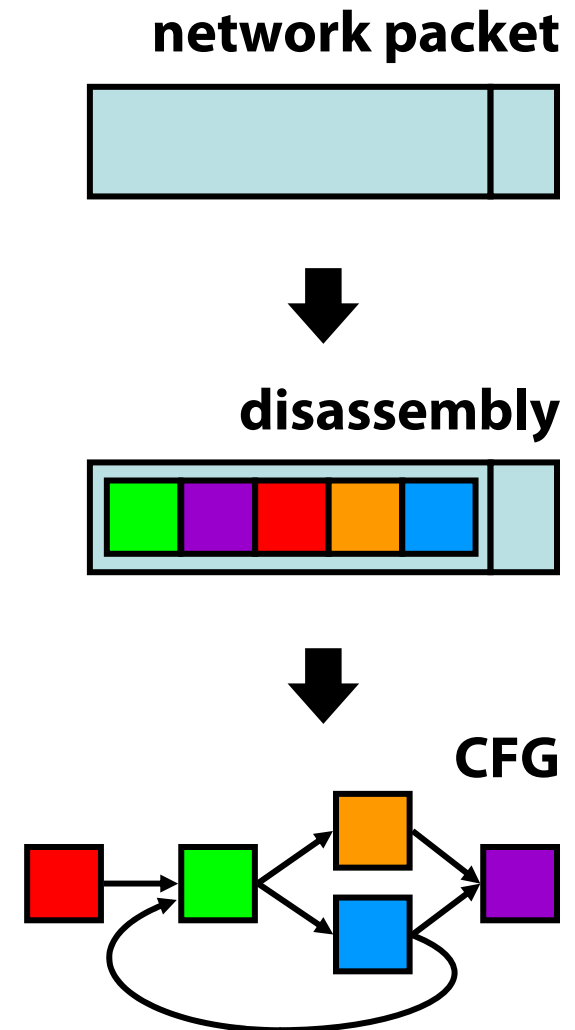
- Find long instruction sequences
- Containing only instructions that can be used as NOPs (“valid instructions”)
- Starting from *multiple consecutive* positions of the stream
- *Then this is an indication of a sled*



Akritidis et al. *STRIDE: Polymorphic Sled Detection through Instruction Sequence Analysis*. IFIP-SEC'05

Polymorphic Shellcode Detection

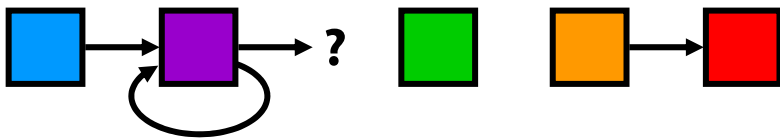
- The sled component can be omitted through careful engineering
 - Most windows exploits do not use a sled
- Need to identify the polymorphic shellcode itself
 - Difficult to discriminate between benign binary data and actual exploit code
- Several new *shellcode detection* methods based on static analysis
 - **Disassembly**
 - **Control flow graph extraction**



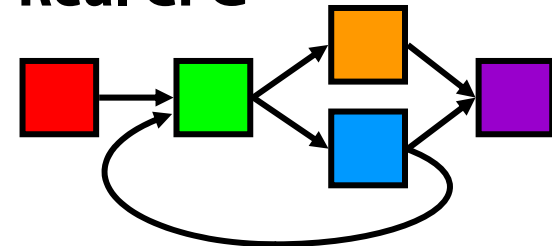
Static Analysis is not Enough

- Attackers can specially craft the shellcode to hinder disassembly and CFG extraction
- **Self-modifying code** can evade static analysis based detection methods
 - Overwrites its own instructions on-the-fly with new ones
 - The malicious code that will eventually be executed does not appear in the network packet
 - The real Control Flow Graph can be known only at runtime

Observed CFG



Real CFG

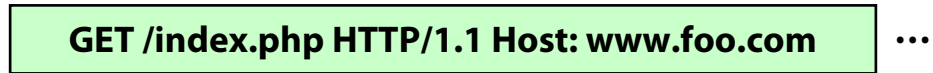
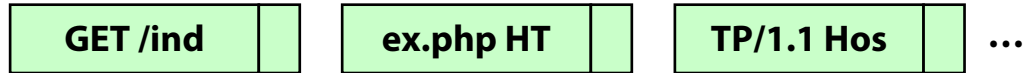


Network-Level Emulation

- **Motivation:** Highly obfuscated code will not reveal its actual form until it is executed
- **Main idea:** execute incoming network traffic as if it was executable code
 - Execution behavior differs for normal data and real executable code
- **Goal:** identify the specific behavior inherent in polymorphic shellcodes

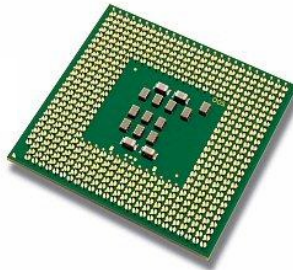
Polychronakis et al. *Network-level Polymorphic Shellcode Detection using Emulation*. DIMVA'06

Network-Level Emulation



G
E
T
/
index.p
hp HT
T
P

1.
1
...



```
inc edi  
inc ebp  
push esp  
and [edi],ch  
imul ebp,[esi+0x64],dword 0x702e7865  
push dword 0x54482070  
push esp  
push eax  
das  
xor [esi],ebp  
xor [eax],esp  
...
```

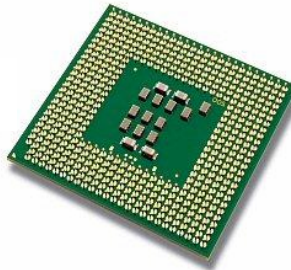


benign request

Network-Level Emulation



6A07
59
E8FFFFFFF
FFC1
5E
80460AEO
304C0E0B
E2FA
...



```
push byte +0x7f
pop ecx
call 0x7
inc ecx
pop esi
add [esi+0xa],0xe0
xor [esi+ecx+0xb],c1
loop 0xe
xor [esi+ecx+0xb],c1
loop 0xe
xor [esi+ecx+0xb],c1
...
```



X malicious request!

Summary

- Passive network monitoring is indispensable for a large class of security applications
 - Not covered in this presentation: firewalls, DDoS attack prevention, SPAM, phishing, network telescopes, ...
- Emerging applications process the captured traffic in new ways
 - Beyond filtering and pattern matching:
 - Static analysis
 - Code emulation

Challenges

- **Attack – defenses coevolution**
 - Need for more accurate and robust detection methods
 - Means more CPU cycles per packet
- **Increasing link speeds**
 - 1 Gbps, 10 Gbps, 40 Gbps, ...
 - Means less CPU (FPGA) cycles per packet
- **Encrypted traffic**
 - SSL, VPN, compressed HTTP, ...

Passive Monitoring for Security-Related Applications

thank you!

Michalis Polychronakis
mikepo@ics.forth.gr

Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)
Crete, Greece