Detecting Polymorphic Cyberattacks

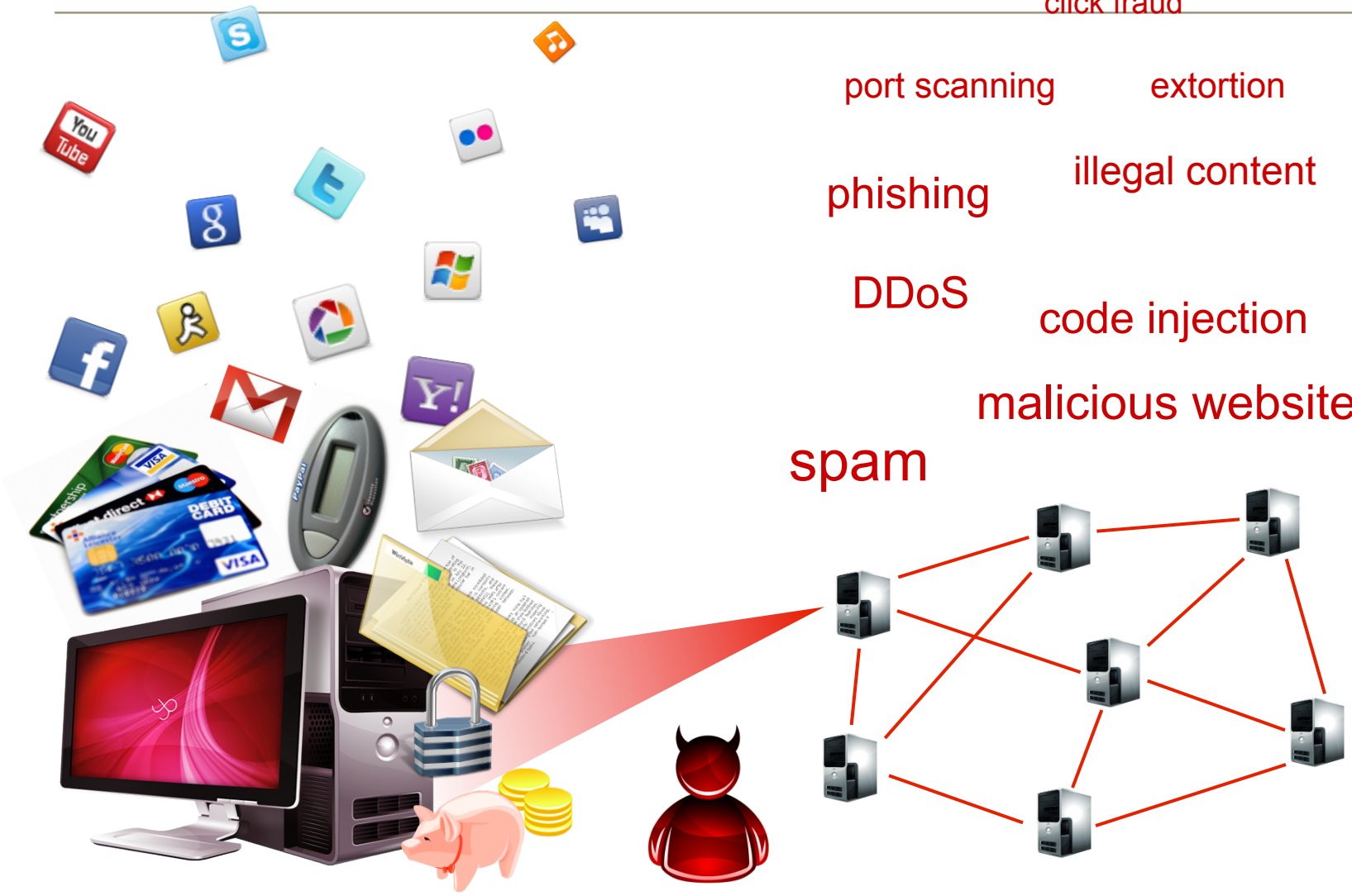
Evangelos Markatos
FORTH-ICS

work done with
Michalis Polychronakis
FORTH and Columbia U

Outline

- Introduction to the problem: shell code attacks – buffer overflows
- Polymorphic attacks (self modifying shell-code)
- Network-level Emulation (NEMU)
- Findings from real-world deployment
- Conclusion

Attackers need compromised computers



click fraud

port scanning

extortion

phishing

illegal content

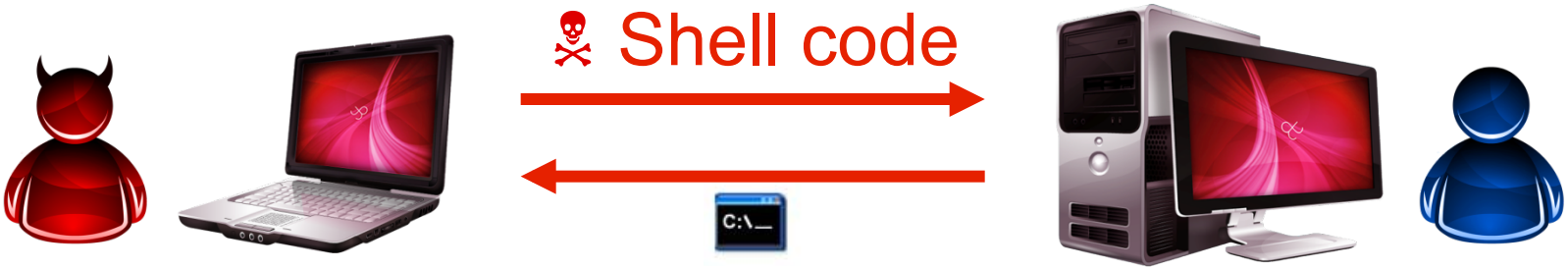
DDoS

code injection

malicious websites

spam

Code Injection Attacks



Remote Code-injection Attacks

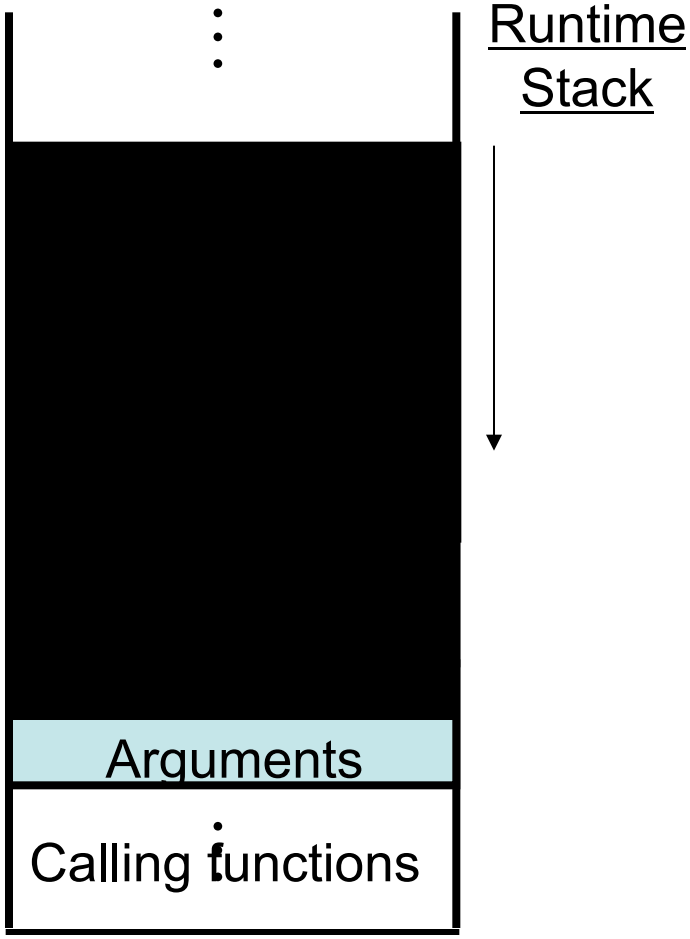
- Code-injection attacks persist
  - Among the most common methods for remote system compromise
  - e.g., Conficker (MS08-067)
- Mechanics
  1. Send malicious request to network service
  2. Divert the execution flow of the vulnerable process
     - **Buffer Overflow**
       - (Stack/heap/integer overflow, 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\x00
```

What is a buffer overflow?

```
main(){
f(10) ;
ret_addr:  printf("End of program\n"); }
```

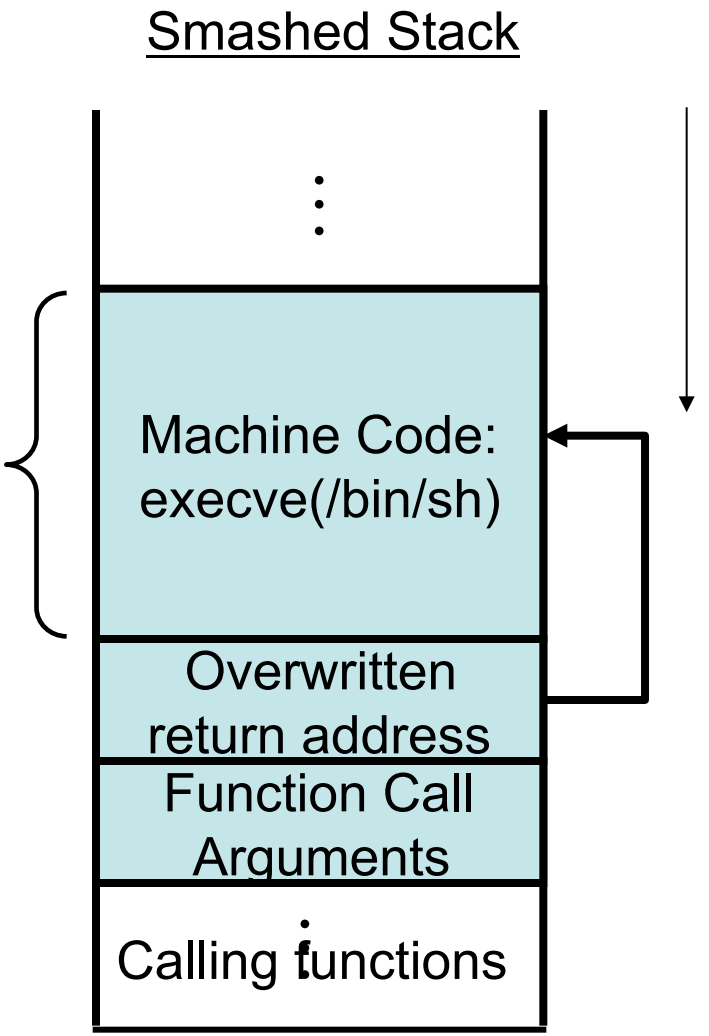
```
void  f ( int x )
{
char buffer[10] ;
scanf("%s", &buffer) ;
// other code
}
```



What if the input data is longer than 10 bytes?

What is a buffer overflow?

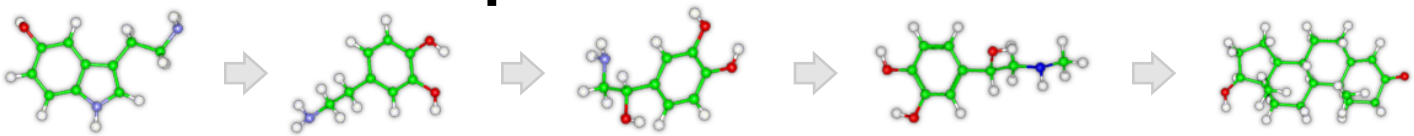
- Buffer overflow
- Attacker puts code
  - i.e. execve(/bin/sh)
  - In buffer[10]
- And transfers control to it
- Via the return address



To make matters worse…

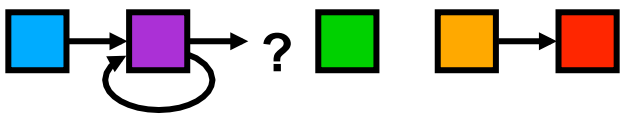
- **Problem:** obfuscated polymorphic shellcode can be highly evasive
  - Each attack instance looks different from each other

    Difficult to fingerprint

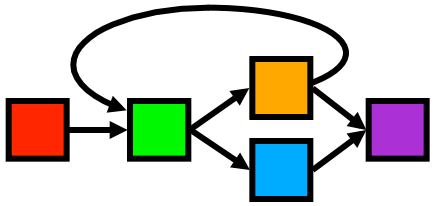


  - Self-modifying code can hide the real malicious code

    Difficult to statically analyze



Observed CFG

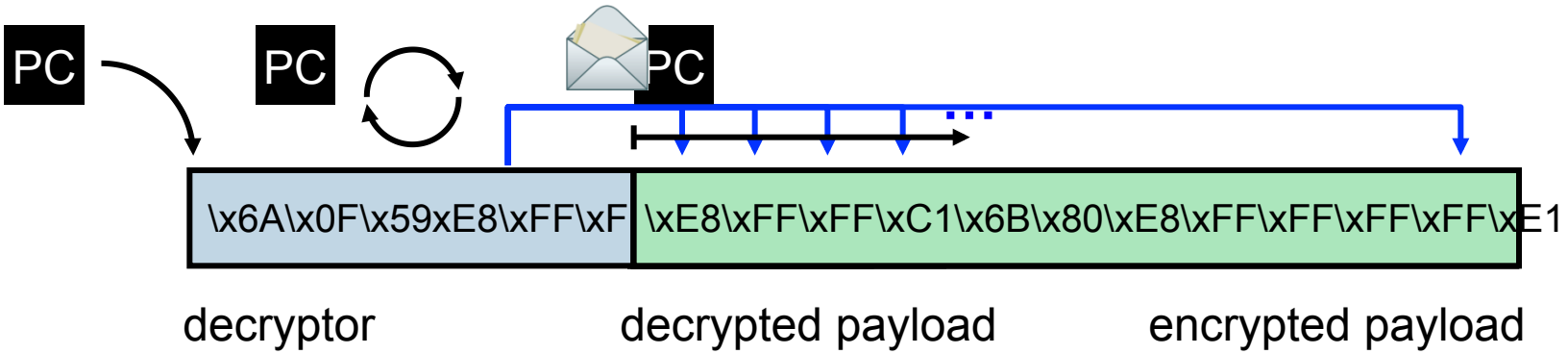


Real CFG

Our solution: Network-level Emulation

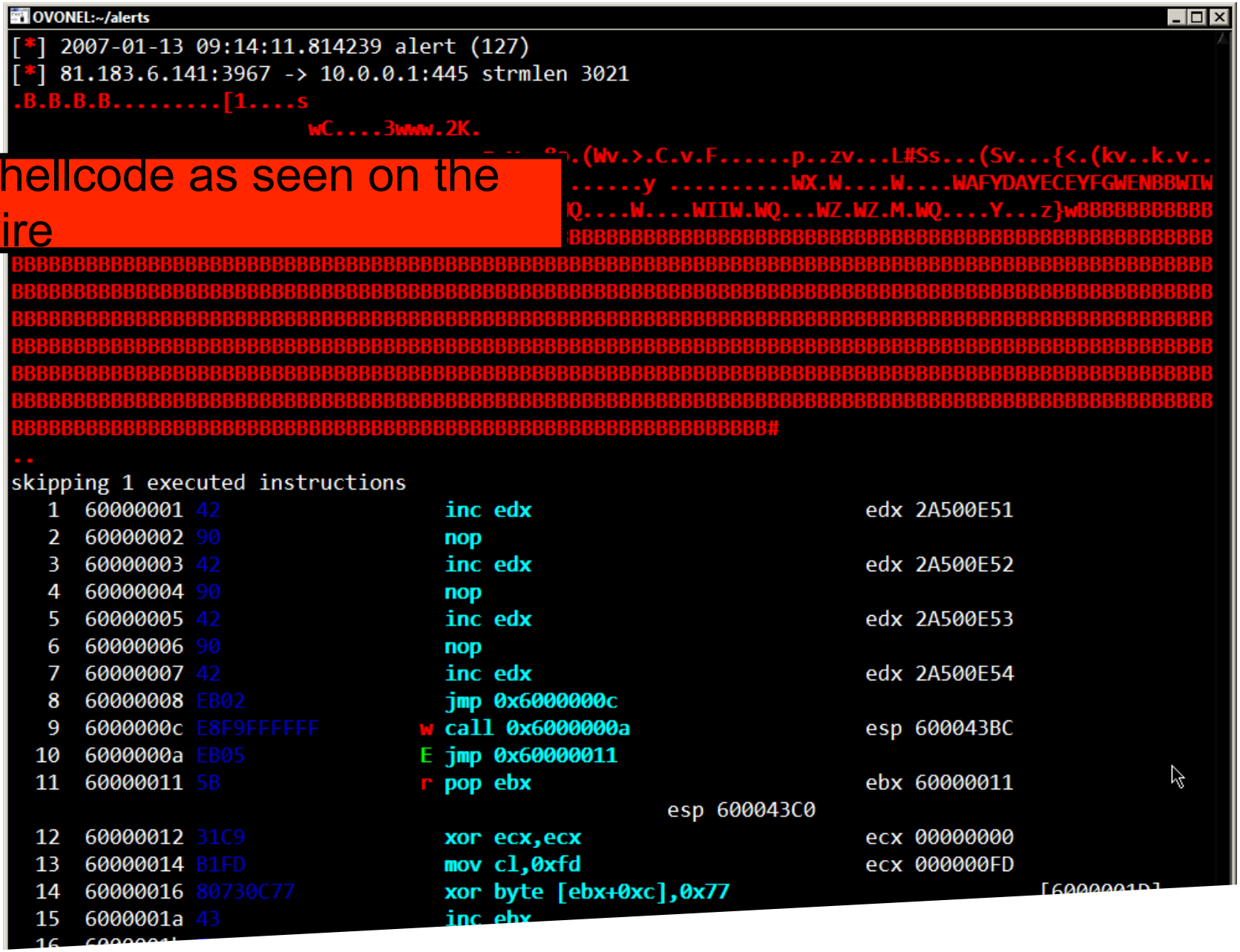
- **Main idea:** execute each network request as if it were executable code
  - Resilience to code obfuscation
- Identify the inherent execution behavior of polymorphic shellcode
  - Focus on the decryption process
  - Generic, independent of the exploit/vulnerability/OS

Polymorphic Shellcode



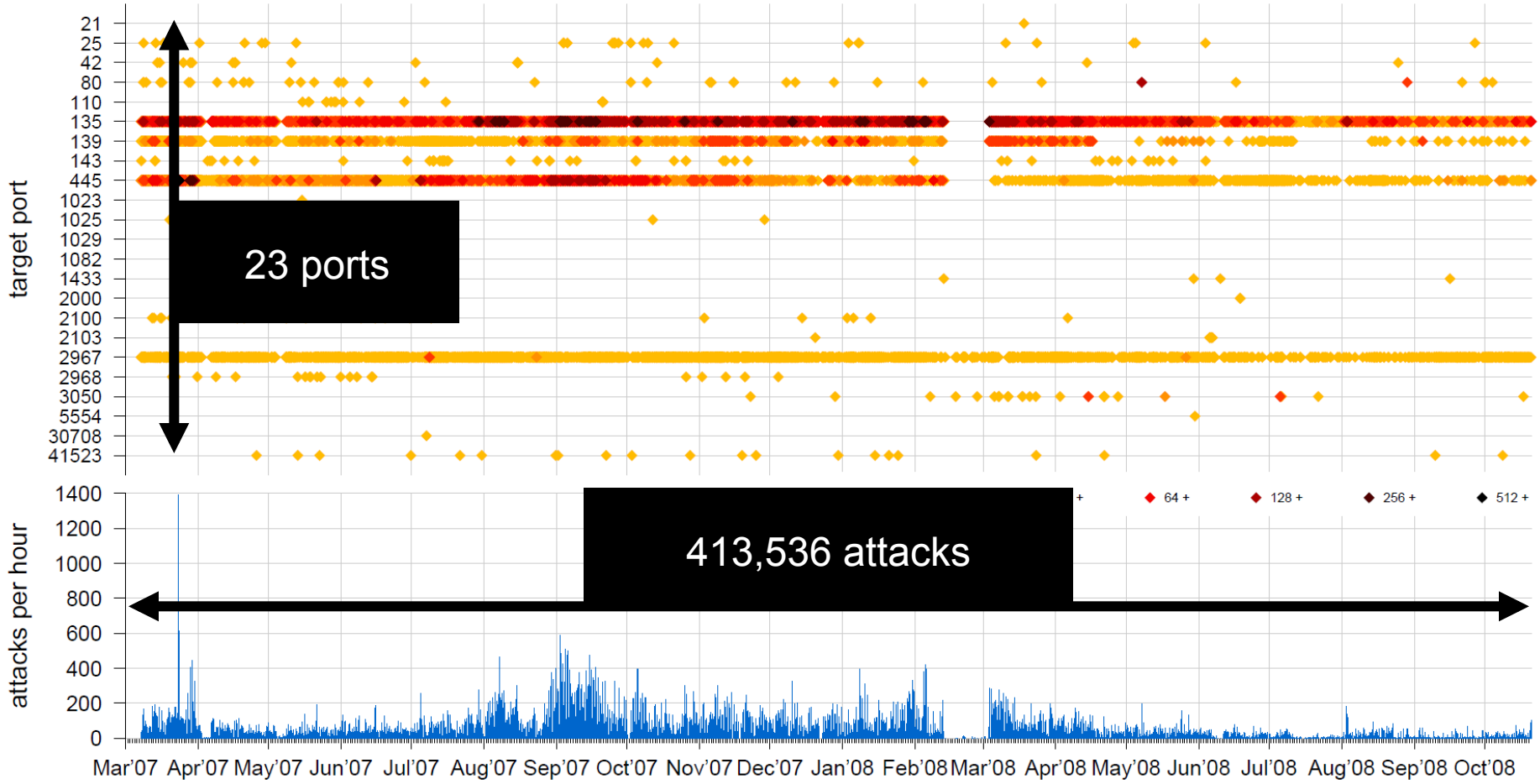
- Self-decrypting code
  - The actual shellcode is not revealed until runtime
- Shellcode “packing” has become essential
  - IDS Evasion
  - Avoidance of restricted bytes in the attack vector

Shellcode as seen on the wire



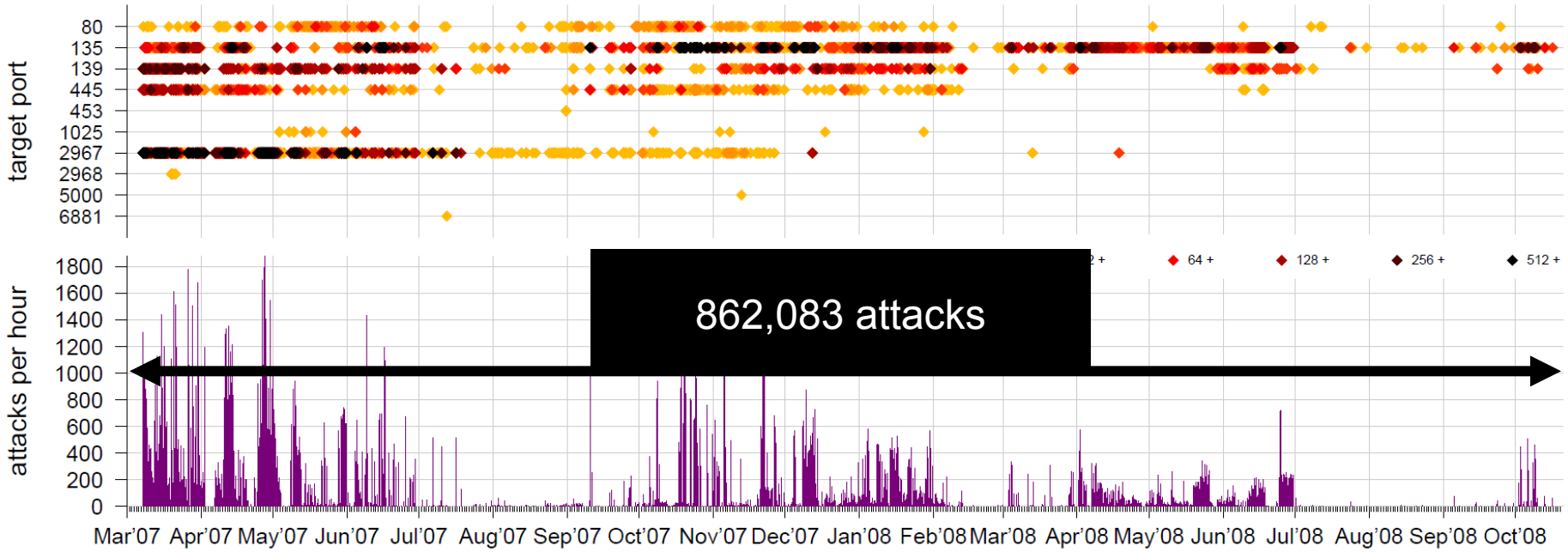
Actual decrypted payload

Overall Activity: External Attacks



Overall Activity: Internal Attacks

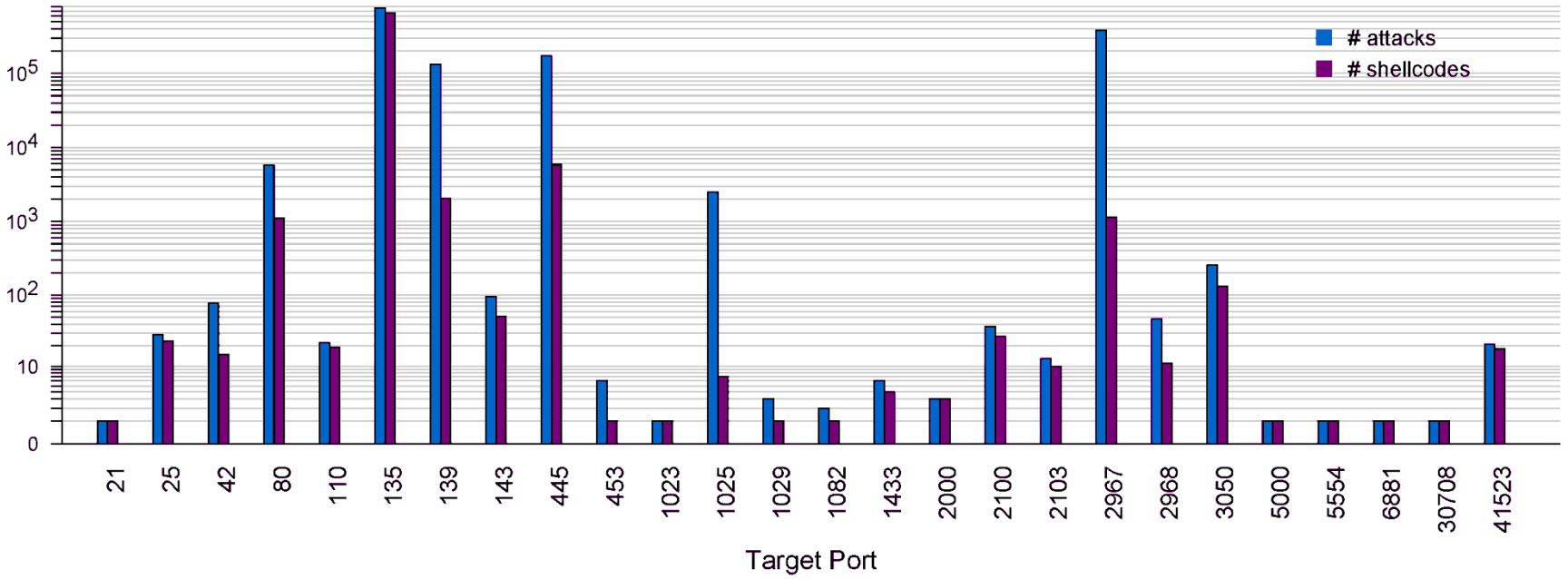
- Large attack volume due to infected hosts
  - Against hosts inside and outside the organization



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 Diversity



- In most cases, the number of unique shellcodes as seen on the wire is comparable to the number of attacks
  - Polymorphism
  - Variable fields in the initial shellcode

Summary

- Pattern matching/static analysis not enough
  - Highly polymorphic and self-modifying code
- Network-level emulation
  - Detects self-modifying polymorphic shellcode
- Remote code-injection attacks are still a threat
  - Increasing sophistication
- Attackers have also turned their attention to less widely used services and third-party applications

Detecting Polymorphic Cyberattacks

Evangelos Markatos
FORTH-ICS

work done with
Michalis Polychronakis
FORTH and Columbia U