



PISE: Automatic Protocol Reverse Engineering

Ron Marcovich, Orna Grumberg and Gabi Nakibly

```
8048b29: 83 c4 f8          add     esp,0xffffffff
8048b2c: 68 c0 97 04 08   push   0x80497c0
8048b31: 50              push   eax
8048b32: e8 f9 04 00 00   call   8049030 <strings_not_equal>
8048b37: 83 c4 10        add     esp,0x10
8048b3a: 85 c0          test   eax,eax
8048b3c: 74 05          je     8048b43 <phase_1+0x23>
8048b3e: e8 b9 09 00 00   call   80494fc <explode_bomb>
8048b43: 89 ec          mov    esp,ebp
8048b45: 5d            pop    ebp
8048b46: c3            ret
8048b47: 90            nop

08048b48 <phase_2>:
8048b48: 55            push   ebp
8048b49: 89 e5          mov    ebp,esp
8048b4b: 83 ec 20      sub    esp,0x20
8048b4e: 56            push   esi
8048b4f: 53            push   ebx
8048b50: 8b 55 08      mov    edx,DWORD PTR [ebp+0x8]
8048b53: 83 c4 10      add    esp,0xffffffff
8048b56: 57            push   edi
8048b5a: 52            push   edx
8048b5b: e8 78 04 00 00   call   8048fd8 <read_six_numbers>
8048b60: 83 c4 10      add    esp,0x10
8048b63: 83 7d e8 01    cmp    DWORD PTR [ebp-0x18],0x1
8048b67: 74 05          je     8048b6e <phase_2+0x26>
8048b69: e8 24 09 00 00   call   80494fc <explode_bomb>
8048b6e: bb 01 00 00 00   mov    ebx,0x1
8048b73: 8d 75 e8      lea   esi,[ebp-0x18]
8048b76: 8d 43 01      lea   eax,[ebx+0x1]
8048b79: 0f af 44 9e fc   imul  eax,DWORD PTR [esi+ebx*4-0x4]
8048b7e: 39 04 9e      cmp    DWORD PTR [esi+ebx*4],eax
8048b81: 74 05          je     8048b88 <phase_2+0x40>
8048b83: e8 74 09 00 00   call   80494fc <explode_bomb>
8048b88: 43            inc    ebx
8048b89: 83 fb 05      cmp    ebx,0x5
8048b8c: 7e e8          jle   8048b76 <phase_2+0x2e>
8048b8e: 8d 65 d8      lea   esp,[ebp-0x28]
8048b91: 5b            pop    ebx
8048b92: 5e            pop    esi
8048b93: 89 ec          mov    esp,ebp
8048b95: 5d            pop    ebp
8048b96: c3            ret
8048b97: 90            nop
```

Introductions



Ron Marcovich



M.Sc. Student



Dr. Gabi Nakibly



Senior Adjunct
Lecturer

Distinguished
Researcher

Formerly at



Prof. Orna Grumberg



Faculty Member

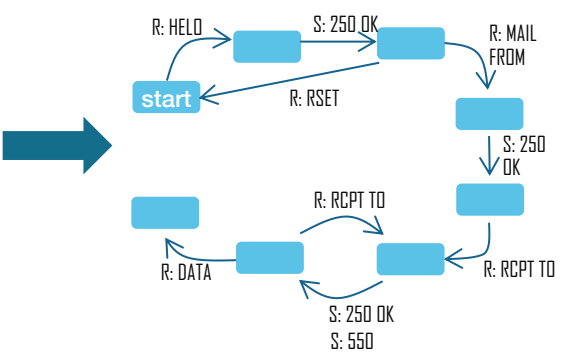
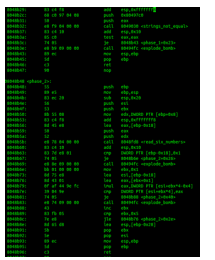
```
cmp ebx,0x...  
jle 8048b76 <phase_2+0x2c>  
lea esp,[ebp-0x28]  
pop ebx  
pop esi  
inc esp,ebp  
pop ebp  
ret  
nop  
  
mov ecx,[ebx+0x1]  
imul eax,DWORD PTR [esi+ebx*4]  
cmp DWORD PTR [ebp-0x18],eax  
jle 8048b88 <phase_2+0x40>  
call 80494fc <explode_bomb>  
inc ebx  
cmp ebx,0x5  
jle 8048b76 <phase_2+0x2c>  
lea esp,[ebp-0x28]  
pop ebx  
pop esi  
  
mov esp,0x10  
mov ebp,esp  
sub esp,0x20  
push esi  
push ebx  
mov edx,DWORD PTR [ebp+0x8]  
add esp,0xffffffff8  
lea eax,[ebp-0x18]  
push eax  
push edx  
call 8048fd8 <read_six_number>  
  
add esp,0xffffffff8  
push eax  
push 0x80497c0  
call 8049030 <strings_not_eq>  
add esp,0x10  
test eax,eax  
je 8048b43 <phase_1+0x23>  
call 80494fc <explode_bomb>  
mov esp,ebp  
pop ebp  
ret
```

Agenda

What is protocol RE?



What is PISE all about?



How PISE does its magic?




```
add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq>
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret

add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq>
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret

add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq>
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret

add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq>
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret
```

Motivation and Background

What is protocol reverse engineering?



What is protocol reverse engineering?

Mail Client

Mail Server

```

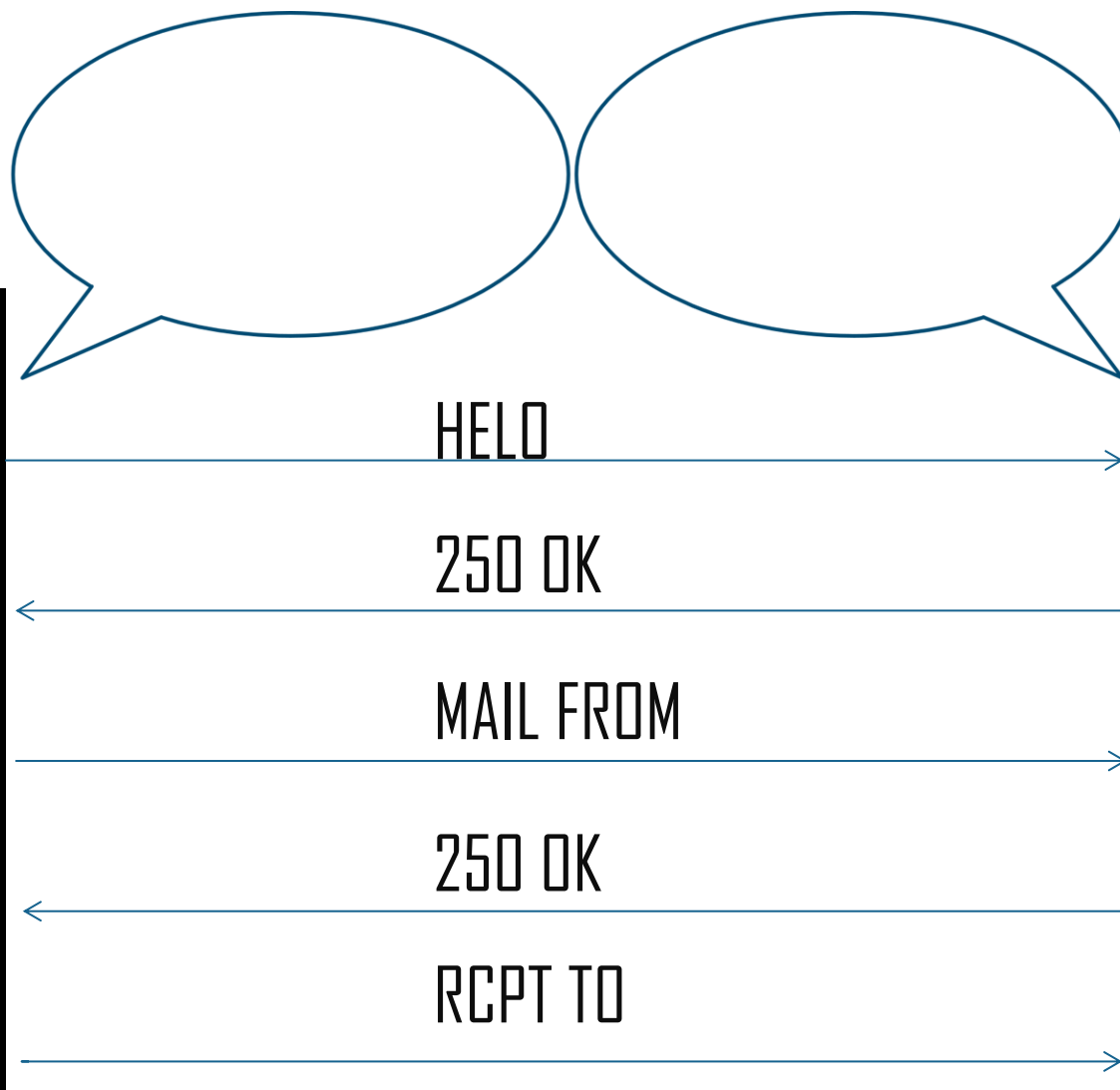
8048b29: 83 c4 f8      add     esp,0xffffffff
8048b2c: 68 c0 97 04 08 push  0x80497c0
8048b31: 50          push  eax
8048b32: e8 f9 04 00 00 call  8049030 <strings_not_equal>
8048b37: 83 c4 10      add     esp,0x10
8048b3a: 85 c0        test   eax,eax
8048b3c: 74 05        je     8048b43 <phase_1+0x23>
8048b3e: e8 b9 09 00 00 call  80494fc <explode_bomb>
8048b43: 89 ec        mov    esp,ebp
8048b45: 5d          pop    ebp
8048b46: c3          ret
8048b47: 90          nop

8048b48 <phase_2>:
8048b48: 55          push  ebp
8048b49: 89 e5        mov    ebp,esp
8048b4b: 83 c4 20      sub    esp,0x20
8048b4e: 56          push  esi
8048b4f: 53          push  ebx
8048b50: 8b 55 08      mov    edx,DWORD PTR [ebp+0x8]
8048b53: 83 c4 f8      add     esp,0xffffffff
8048b56: 8d 45 e8      lea   eax,[ebp-0x18]
8048b59: 50          push  eax
8048b5a: 52          push  edx
8048b5b: e8 78 04 00 00 call  8048fd8 <read_six_numbers>
8048b60: 83 c4 10      add     esp,0x10
8048b63: 83 7d e8 01   cmp    DWORD PTR [ebp-0x18],0x1
8048b67: 74 05        je     8048b6e <phase_2+0x26>
8048b69: e8 8e 09 00 00 call  80494fc <explode_bomb>
8048b6e: bb 01 00 00 00 mov    ebx,0x1
8048b73: 8d 75 e8      lea   esi,[ebp-0x18]
8048b76: 8d 43 01      lea   eax,[ebx+0x1]
8048b79: 0f af 44 9e fc imul  eax,DWORD PTR [esi+ebx*4-0x4]
8048b7e: 39 04 9e      cmp    DWORD PTR [esi+ebx*4],eax
8048b81: 74 05        je     8048b88 <phase_2+0x40>
8048b83: e8 74 09 00 00 call  80494fc <explode_bomb>
8048b88: 43          inc    ebx
8048b89: 83 fb 05      cmp    ebx,0x5
8048b8c: 7e e8        jle   8048b76 <phase_2+0x2e>
8048b8e: 8d 65 d8      lea   esp,[ebp-0x28]
8048b91: 5b          pop    ebx
8048b92: 5e          pop    esi
8048b93: 89 ec        mov    esp,ebp
8048b95: 5d          pop    ebp
8048b96: c3          ret
  
```

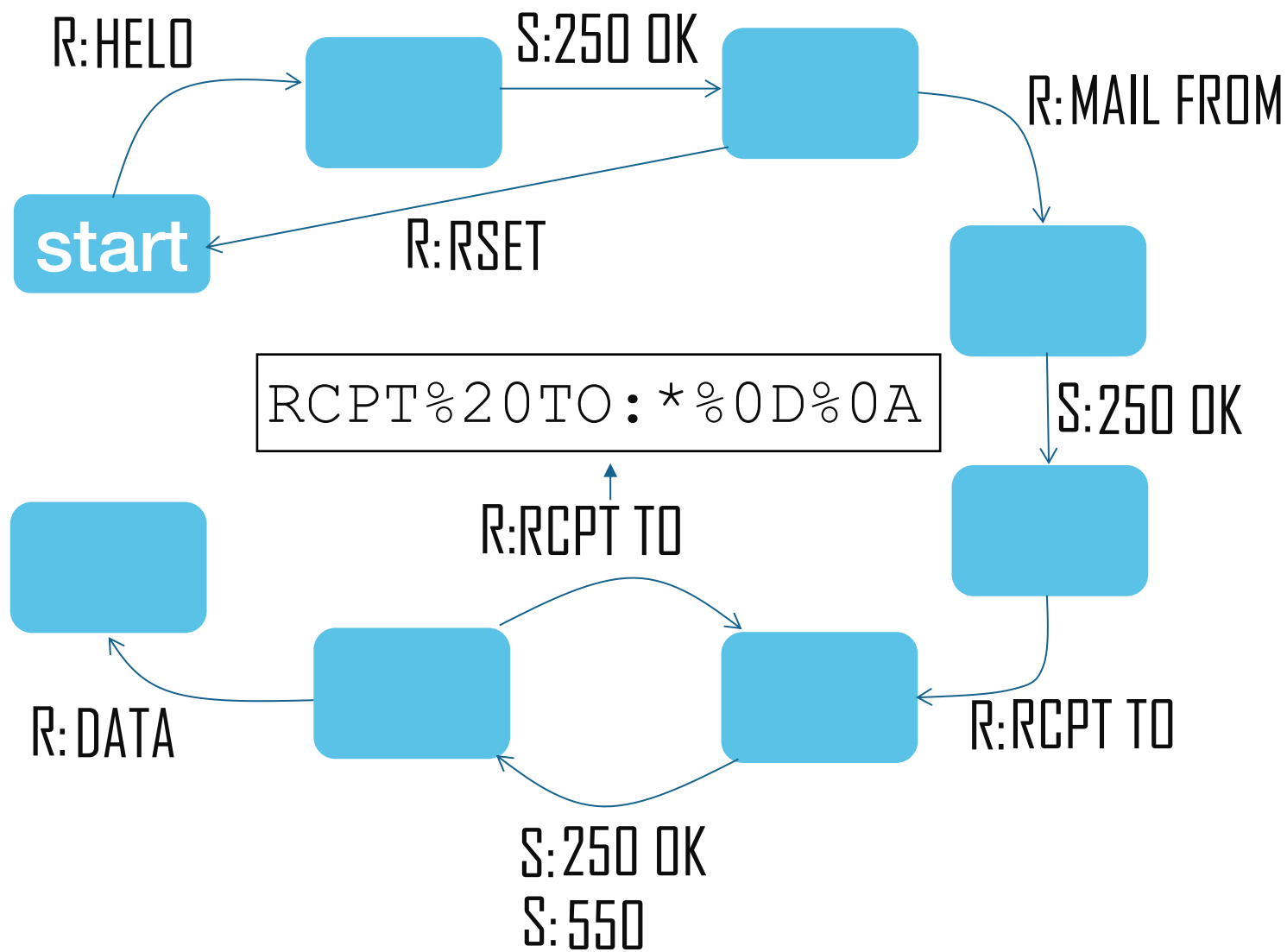
```

8048b29: 83 c4 f8      add     esp,0xffffffff
8048b2c: 68 c0 97 04 08 push  0x80497c0
8048b31: 50          push  eax
8048b32: e8 f9 04 00 00 call  8049030 <strings_not_equal>
8048b37: 83 c4 10      add     esp,0x10
8048b3a: 85 c0        test   eax,eax
8048b3c: 74 05        je     8048b43 <phase_1+0x23>
8048b3e: e8 b9 09 00 00 call  80494fc <explode_bomb>
8048b43: 89 ec        mov    esp,ebp
8048b45: 5d          pop    ebp
8048b46: c3          ret
8048b47: 90          nop

8048b48 <phase_2>:
8048b48: 55          push  ebp
8048b49: 89 e5        mov    ebp,esp
8048b4b: 83 c4 20      sub    esp,0x20
8048b4e: 56          push  esi
8048b4f: 53          push  ebx
8048b50: 8b 55 08      mov    edx,DWORD PTR [ebp+0x8]
8048b53: 83 c4 f8      add     esp,0xffffffff
8048b56: 8d 45 e8      lea   eax,[ebp-0x18]
8048b59: 50          push  eax
8048b5a: 52          push  edx
8048b5b: e8 78 04 00 00 call  8048fd8 <read_six_numbers>
8048b60: 83 c4 10      add     esp,0x10
8048b63: 83 7d e8 01   cmp    DWORD PTR [ebp-0x18],0x1
8048b67: 74 05        je     8048b6e <phase_2+0x26>
8048b69: e8 8e 09 00 00 call  80494fc <explode_bomb>
8048b6e: bb 01 00 00 00 mov    ebx,0x1
8048b73: 8d 75 e8      lea   esi,[ebp-0x18]
8048b76: 8d 43 01      lea   eax,[ebx+0x1]
8048b79: 0f af 44 9e fc imul  eax,DWORD PTR [esi+ebx*4-0x4]
8048b7e: 39 04 9e      cmp    DWORD PTR [esi+ebx*4],eax
8048b81: 74 05        je     8048b88 <phase_2+0x40>
8048b83: e8 74 09 00 00 call  80494fc <explode_bomb>
8048b88: 43          inc    ebx
8048b89: 83 fb 05      cmp    ebx,0x5
8048b8c: 7e e8        jle   8048b76 <phase_2+0x2e>
8048b8e: 8d 65 d8      lea   esp,[ebp-0x28]
8048b91: 5b          pop    ebx
8048b92: 5e          pop    esi
8048b93: 89 ec        mov    esp,ebp
8048b95: 5d          pop    ebp
8048b96: c3          ret
  
```



What is protocol reverse engineering?



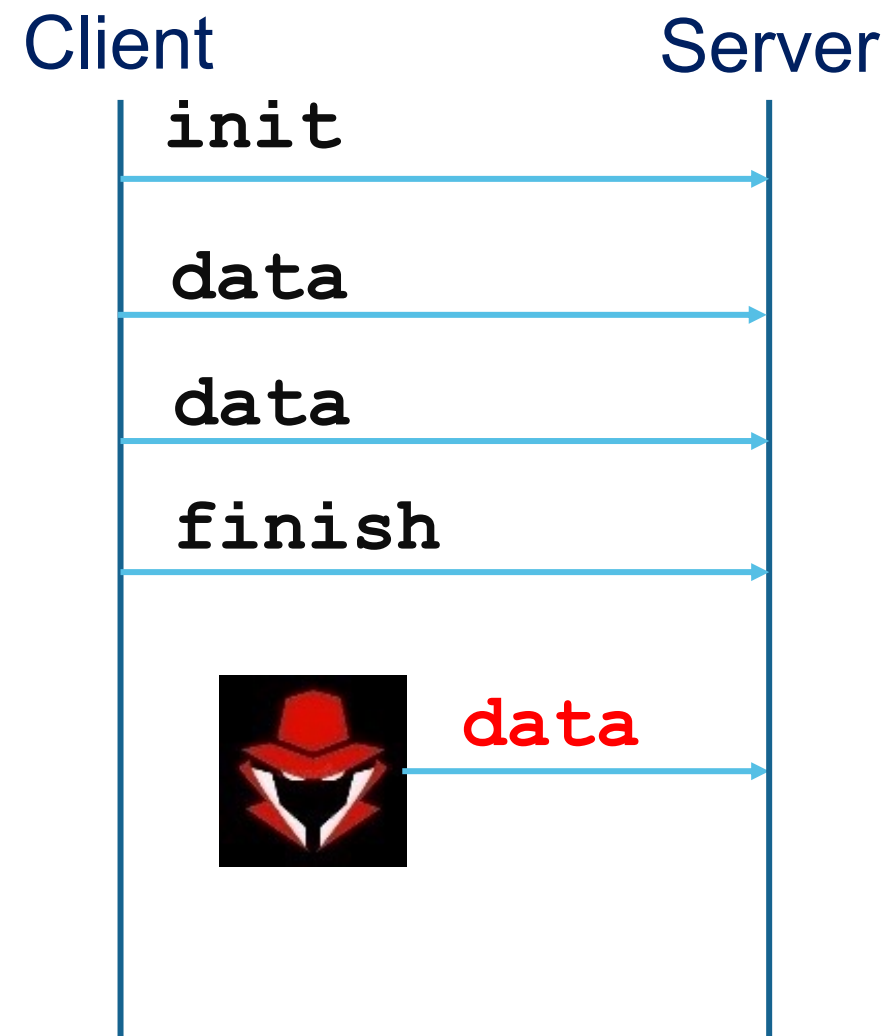
Mail Server

```

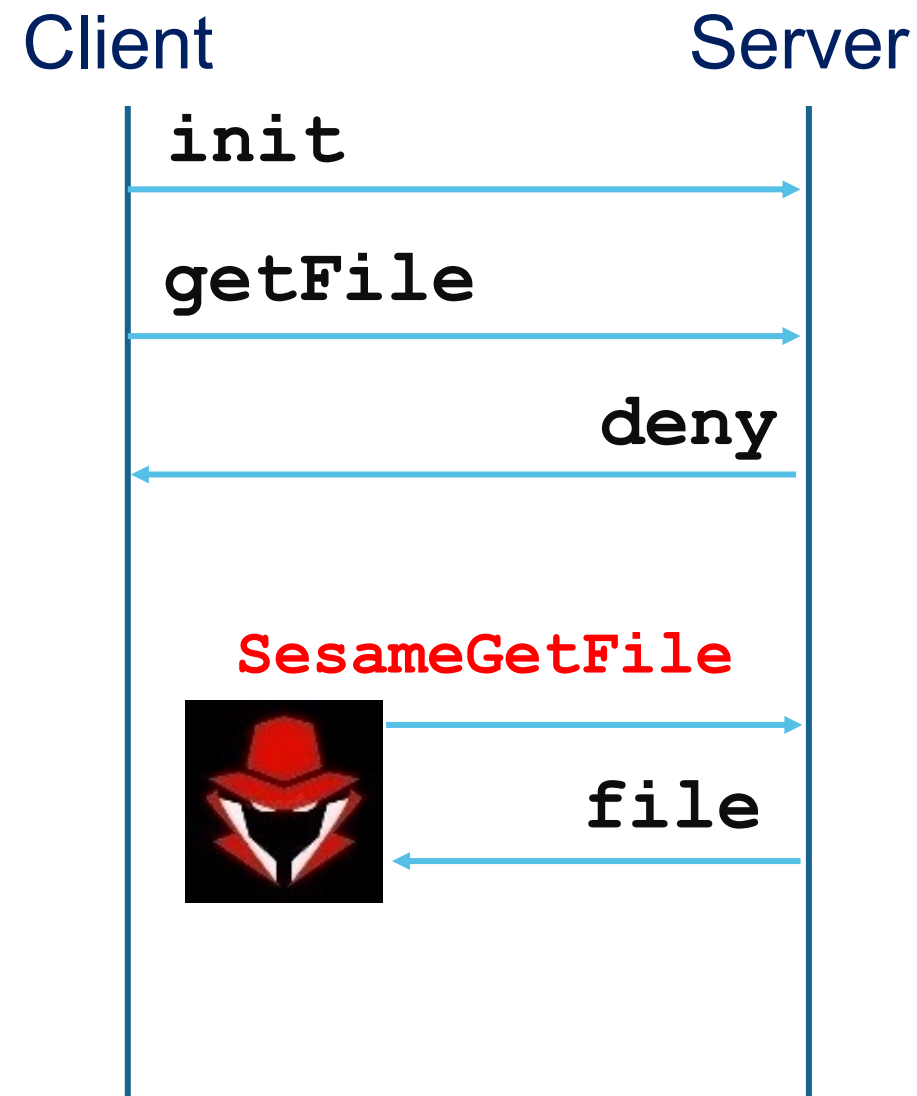
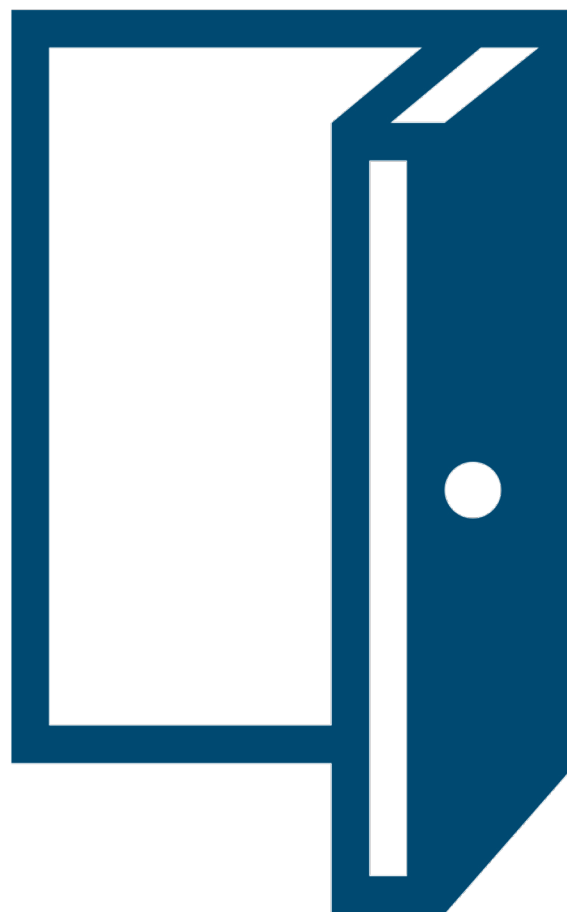
8048b29: 83 c4 f8      add     esp,0xffffffff
8048b2c: 68 c0 97 04 08 push   eax,0x80497c0
8048b31: 50          push   eax
8048b32: e8 f9 04 00 00 call   8049030 <strings_not_equal>
8048b37: 83 c4 10      add     esp,0x10
8048b3a: 85 c0        test   eax,eax
8048b3c: 74 05        je     8048b43 <phase_1+0x23>
8048b3e: e8 b9 09 00 00 call   80494fc <explode_bomb>
8048b43: 89 ec        mov    esp,ebp
8048b45: 5d          pop    ebp
8048b46: c3          ret
8048b47: 90          nop

8048b48 <phase_2>:
8048b48: 55          push   ebp
8048b49: 89 e5        mov    ebp,esp
8048b4b: 83 ec 20     sub    esp,0x20
8048b4e: 56          push   esi
8048b4f: 53          push   ebx
8048b50: 8b 55 08     mov    edx,DWORD PTR [ebp+0x8]
8048b53: 83 c4 f8     add     esp,0xffffffff
8048b56: 8d 45 e8     lea   eax,[ebp-0x18]
8048b59: 50          push   eax
8048b5a: 52          push   edx
8048b5b: e8 78 04 00 00 call   8048fd8 <read_six_numbers>
8048b60: 83 c4 10     add     esp,0x10
8048b63: 83 7d e8 01  cmp    DWORD PTR [ebp-0x18],0x1
8048b67: 74 05        je     8048b6e <phase_2+0x26>
8048b69: e8 8e 09 00 00 call   80494fc <explode_bomb>
8048b6e: bb 01 00 00 00 mov    ebx,0x1
8048b73: 8d 75 e8     lea   esi,[ebp-0x18]
8048b76: 8d 43 01     lea   eax,[ebx+0x1]
8048b79: 0f af 44 9e fc imul  eax,DWORD PTR [esi+ebx*4-0x4]
8048b7e: 39 04 9e     cmp    DWORD PTR [esi+ebx*4],eax
8048b81: 74 05        je     8048b88 <phase_2+0x40>
8048b83: e8 74 09 00 00 call   80494fc <explode_bomb>
8048b88: 43          inc    ebx
8048b89: 83 fb 05     cmp    ebx,0x5
8048b8c: 7e e8        jle   8048b76 <phase_2+0x2e>
8048b8e: 8d 65 d8     lea   esp,[ebp-0x28]
8048b91: 5b          pop    ebx
8048b92: 5e          pop    esi
8048b93: 89 ec        mov    esp,ebp
8048b95: 5d          pop    ebp
8048b96: c3          ret
  
```

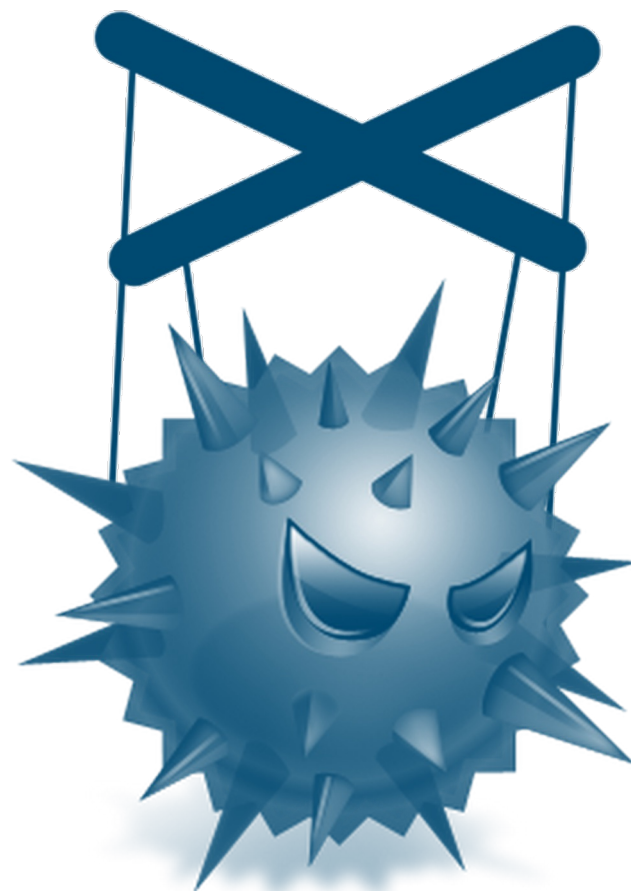
Motivation I – Finding Bugs



Motivation II – Finding backdoors



Motivation III – Analyzing Malware



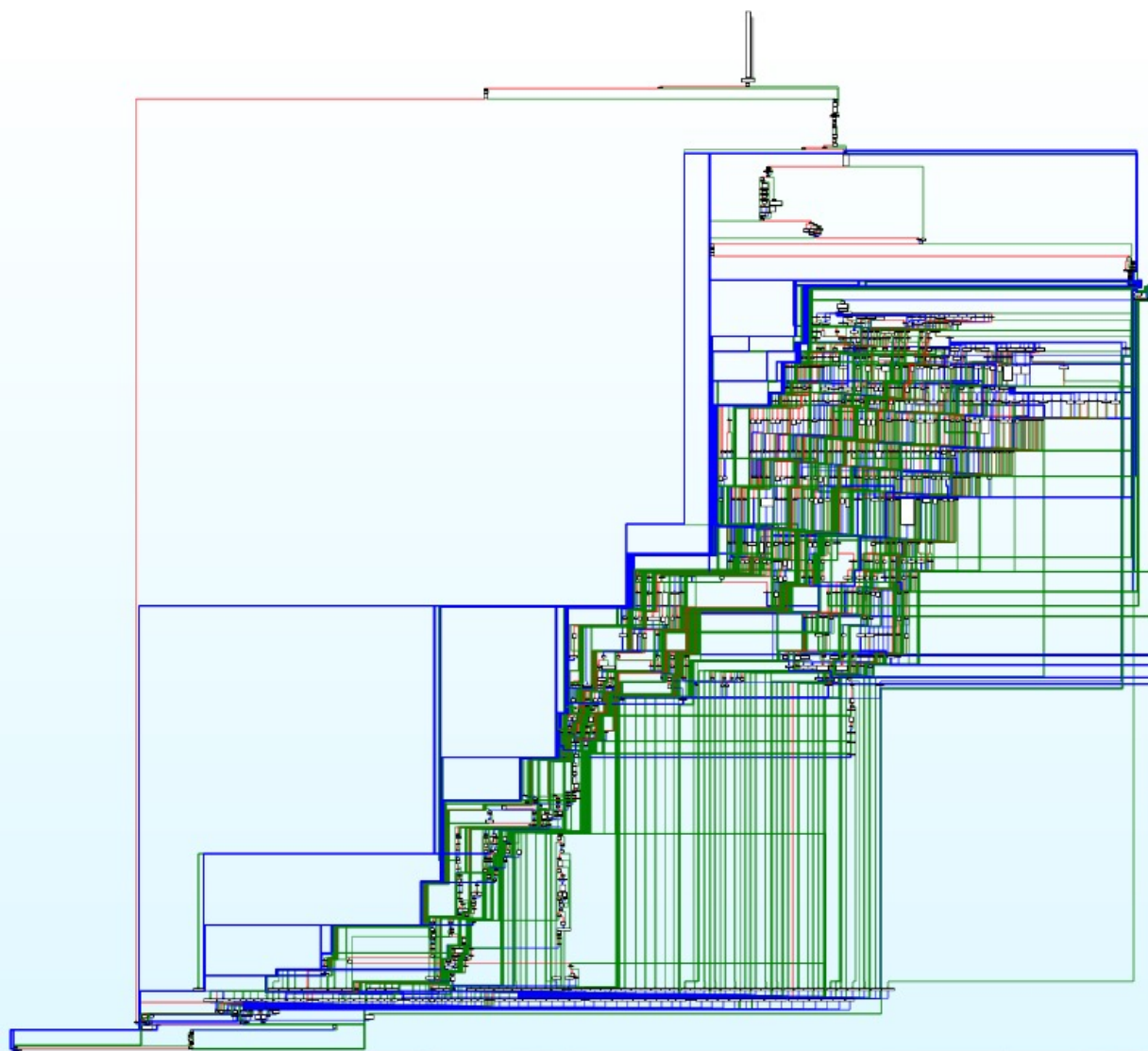
Get info

Send Spam

DoS <url>

Protocol RE is Hard!

It can be days or even weeks!

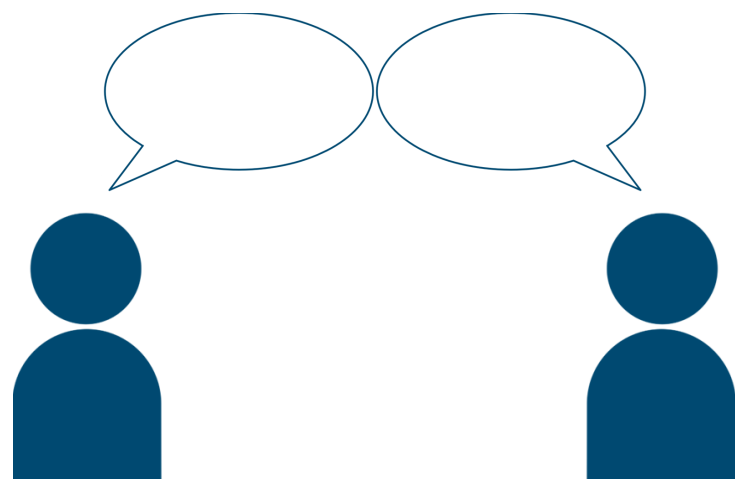


Research Goal



No Assumptions

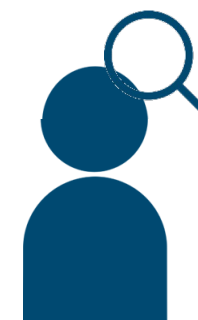
No past traffic captures



No active protocol peer



No source code



```
add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret

add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret

add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret

add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret
```

PISE is action, Examples and Demo

We wanted to get to the real thing

RE: Protocol inference



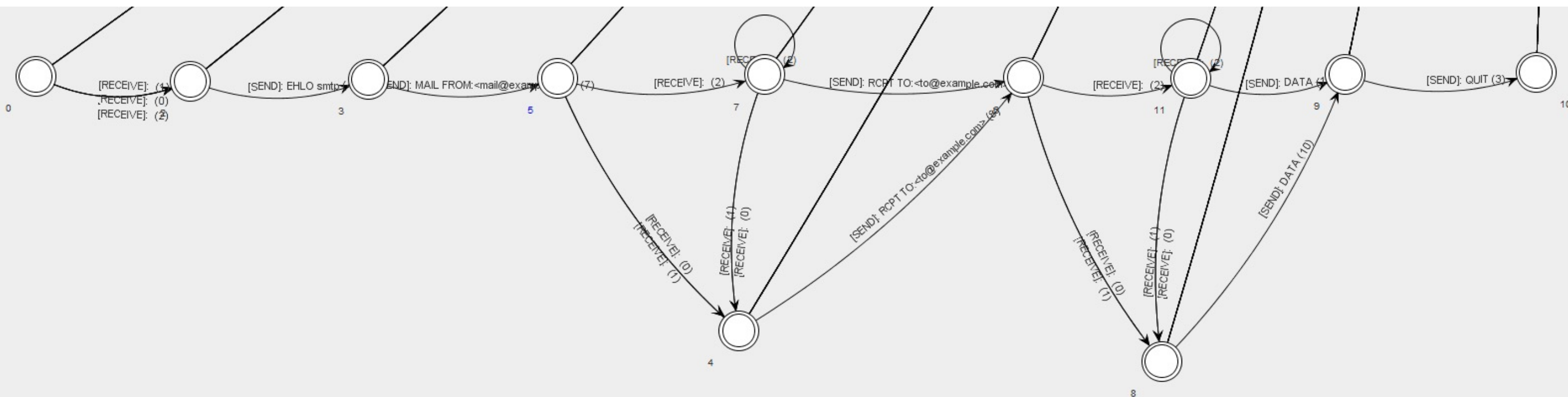
Ron Marcovich
To Gabi Nakibly
Cc Orna Grumberg

Reply Reply All Forward ...

Wed 20/11/2019 19:50

Hi Orna, Gabi,

Another good news! I have changed a couple of things in my algorithm after the meeting today. It now finds a state machine that seems very accurate.



We wanted to get to the real thing

Messages' formats are extracted as well!

SMTP messages



Ron Marcovich

To Gabi Nakibly

MSG ID 0: {RECEIVE} [UNKNOWN] 0x00

MSG ID 2: {SEND} [EHLO smtp] 0x45 0x48 0x4c 0x4f 0x20 0x73 0x6d 0x74 0x70 0x0d 0x0a

MSG ID 3: {RECEIVE} [-] _____ 0x2d

MSG ID 10: {SEND} [MAIL FROM:<mail@example.com>] 0x4d 0x41 0x49 0x4c 0x20 0x46 0x52 0x4f 0x4d 0x3a 0x3c 0x6d 0x61 0x69 0x6c 0x40 0x65

MSG ID 109: {SEND} [RCPT TO:<to@example.com>] 0x52 0x43 0x50 0x54 0x20 0x54 0x4f 0x3a 0x3c 0x74 0x6f 0x40 0x65 0x78 0x61 0x6d 0x70 0x6c

MSG ID 602: {SEND} [DATA] 0x44 0x41 0x54 0x41 0x0d 0x0a

MSG ID 1076: {RECEIVE} [354] 0x33 0x35 0x34

MSG ID 1659: {SEND} [Subject: Subject Line] 0x53 0x75 0x62 0x6a 0x65 0x63 0x74 0x3a 0x20 0x53 0x75 0x62 0x6a 0x65 0x63 0x74 0x20 0x4c 0x69

MSG ID 2119: {SEND} [From: "From Name" <mail@example.com>] 0x46 0x72 0x6f 0x6d 0x3a 0x20 0x22 0x46 0x72 0x6f 0x6d 0x20 0x4e 0x61 0x6d

MSG ID 2304: {SEND} [To: "To Name" <to@example.com>] 0x54 0x6f 0x3a 0x20 0x22 0x54 0x6f 0x20 0x4e 0x61 0x6d 0x65 0x22 0x20 0x3c 0x74 0x

MSG ID 2305: {SEND} [Email Body] 0x45 0x6d 0x61 0x69 0x6c 0x20 0x42 0x6f 0x64 0x79 0x0d 0x0a

MSG ID 2306: {SEND} [.] 0x2e 0x0d 0x0a

MSG ID 2310: {RECEIVE} [250] 0x32 0x35 0x30

MSG ID 2555: {SEND} [QUIT] 0x51 0x55 0x49 0x54 0x0d 0x0a

Then COVID came....

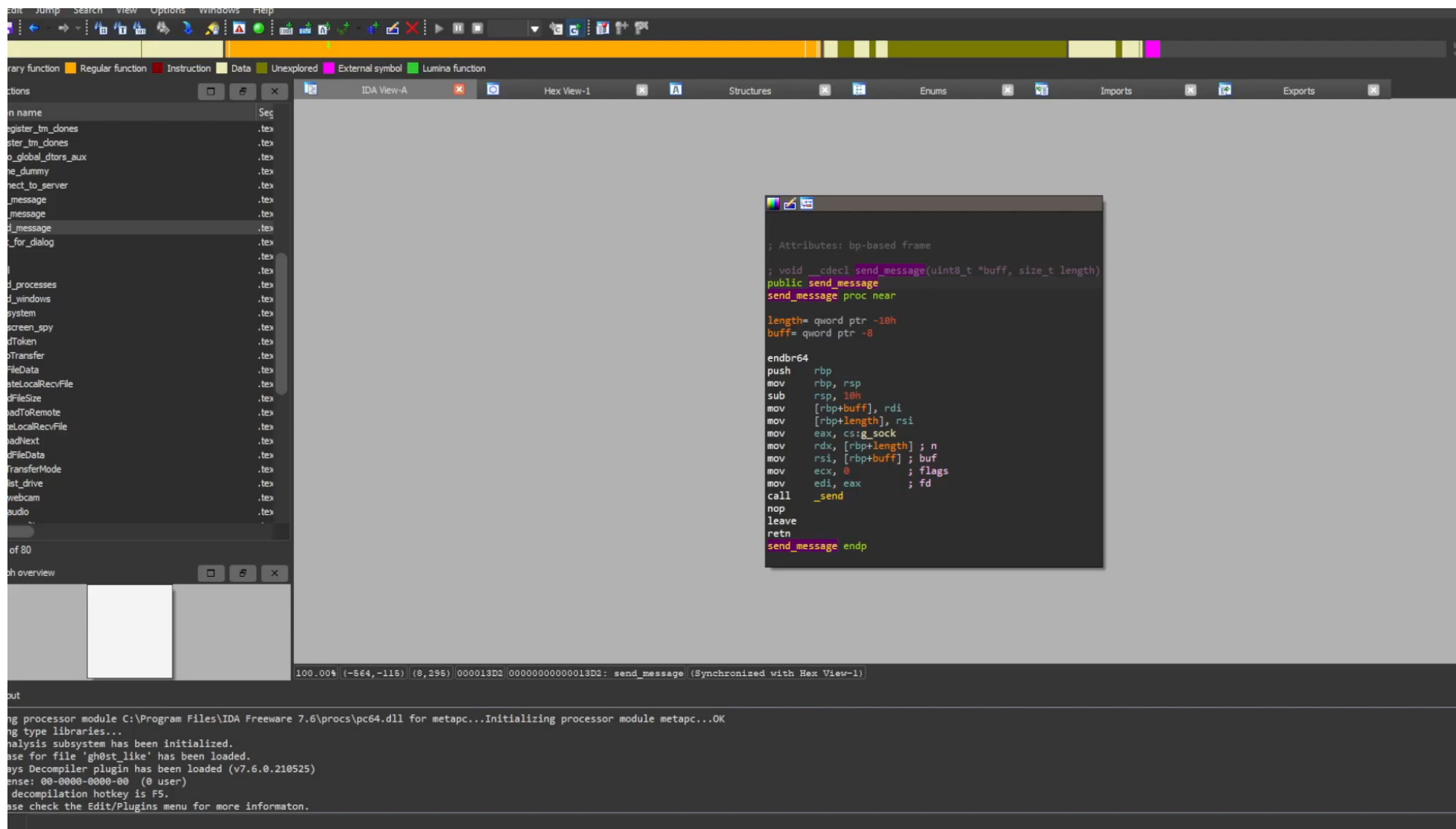
Remember those days when we had no idea what Zoom is?

From: Gabi Nakibly <gabinkbl@gmail.com>
Sent: Tuesday, March 17, 2020 3:26 PM
To: Orna Grumberg <orna@cs.technion.ac.il>
Cc: Ron Marcovich <ron.mar@campus.technion.ac.il>
Subject: Re: meeting tomorrow

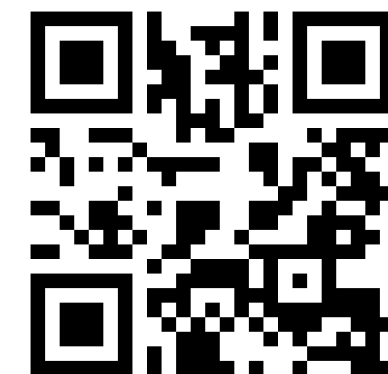


I am OK with Thursday morning. I am not sure what zoom is. Can you send a link?

Then we tried to work with gh0st RAT



~ 2 min



```
add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq>
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret

add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq>
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret

add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq>
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret

add esp,0xffffffff
push 0x80497c0
push eax
call 8049030 <strings_not_eq>
add esp,0x10
test eax,eax
je 8048b43 <phase_1+0x23>
call 80494fc <explode_bomb>
mov esp,ebp
pop ebp
ret
```

Under the Hood



Overview



L* Algorithm



Symbolic Execution

L* Algorithm (Automata Learning)

INFORMATION AND COMPUTATION 75, 87–106 (1987)

Learning Regular Sets from Queries and Counterexamples*

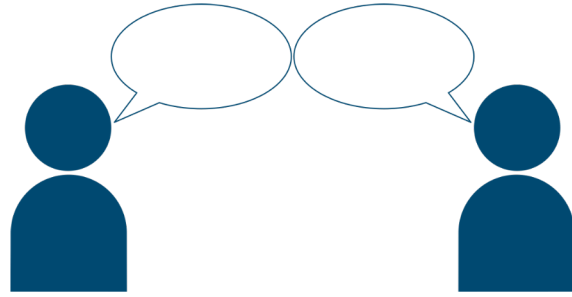
DANA ANGLUIN

Q: Is a given message exchange valid by the

Department of Computer Science, Yale University,
P.O. Box 2158, Yale Station, New Haven, Connecticut 06520

The problem of identifying an unknown regular set from examples of its members and nonmembers is addressed. It is assumed that the regular set is presented by a *minimally adequate Teacher*, which can answer membership queries about the set and can also test a conjecture and indicate whether it is equal to the unknown set and provide a counterexample if not. (A counterexample is a string in the symmetric difference of the correct set and the conjectured set.) A learning algorithm L^* is described that correctly learns any regular set from any minimally adequate Teacher in time polynomial in the number of states of the minimum dfa for the set and the maximum length of any counterexample provided by the Teacher. It is shown that in a stochastic setting the ability of the Teacher to test conjectures may be replaced by a random sampling oracle, $EX()$. A polynomial-time learning

L* Algorithm



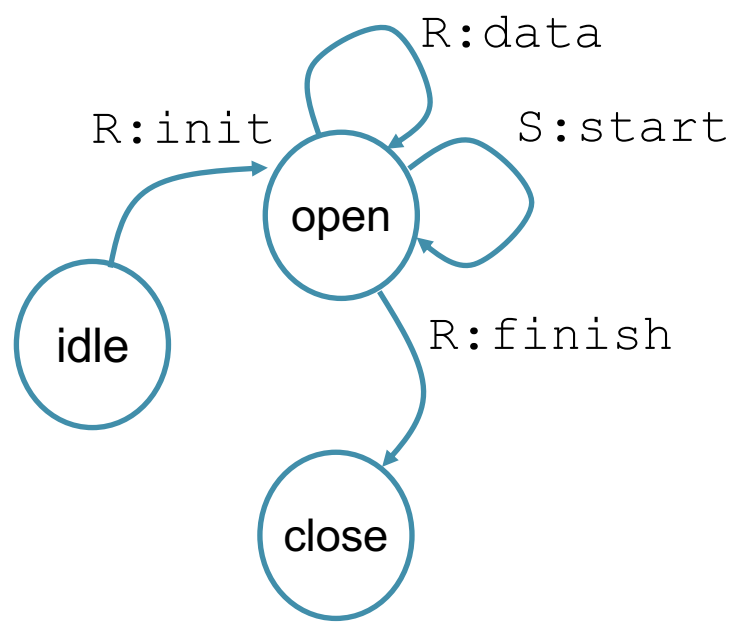
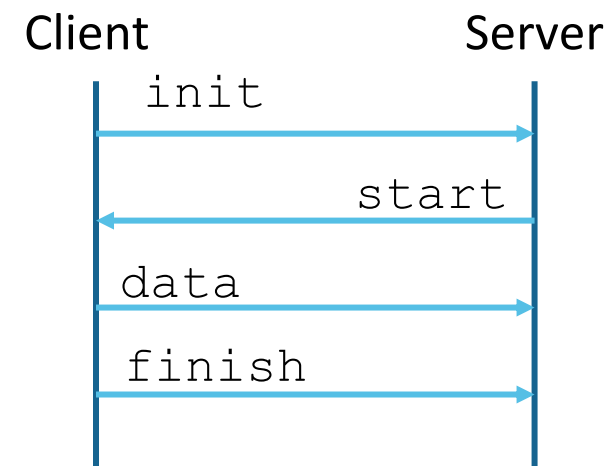
Q: Is this conversation valid?

{ S: Hi, R: Hello, S: How are you?, R: I am fine. } ✓

{ S: Hi, R: Hello, S: How are you?, R: Tuesday } ✗

L* Algorithm

{R:init, S:start} ✓
 {R:init, R:init} ✗

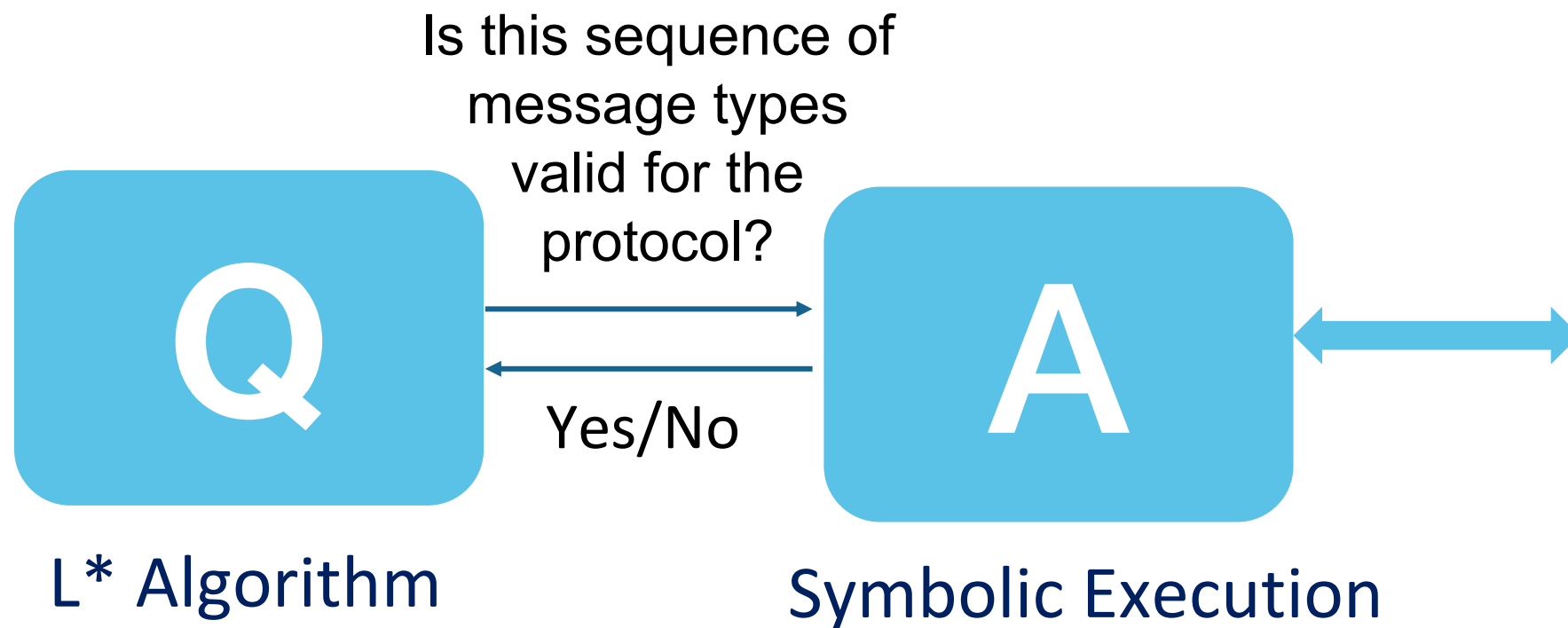


But there is a problem!

We do not know what are the protocol's message types!!

Let's assume for now we do know the message types.

Answering Membership queries

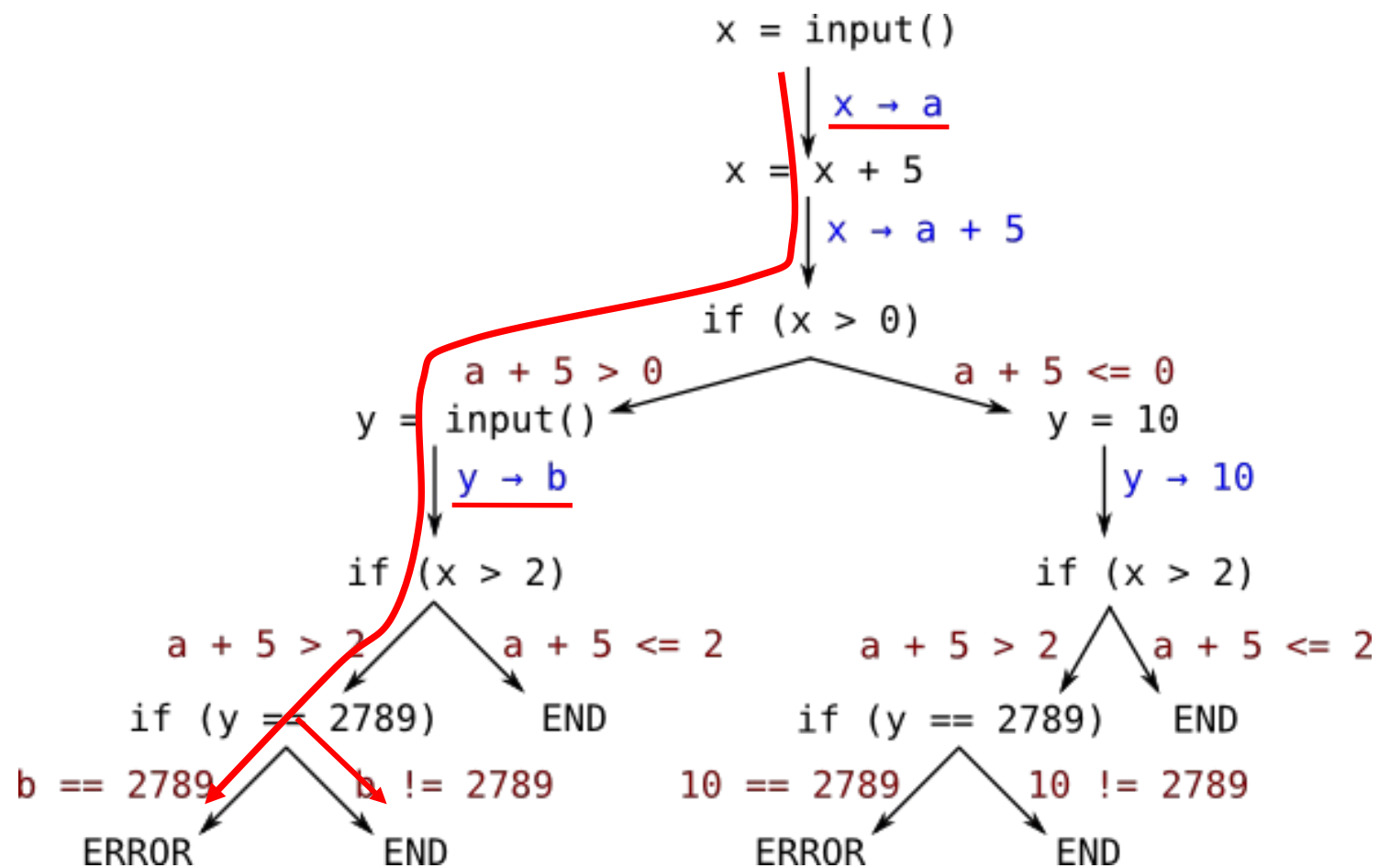


```
0048b29: 83 c4 f8      add esp,0xffffffff
0048b2c: 68 c0 97 04 08 push 0x80497c0
0048b31: 50          push eax
0048b32: e8 f9 04 00 00 call 8049030 <strings_not_equal>
0048b37: 83 c4 10      add esp,0x10
0048b3a: 85 c0        test eax,eax
0048b3c: 74 05        je 8048b43 <phase_1+0x23>
0048b3e: e8 b9 09 00 00 call 80494fc <explode_bomb>
0048b43: 89 ec        mov esp,ebp
0048b45: 5d          pop ebp
0048b46: c3          ret
0048b47: 90          nop

0048b48 <phase_2>:
0048b48: 55          push ebp
0048b49: 89 e5        mov ebp,esp
0048b4b: 83 ec 20      sub esp,0x20
0048b4e: 56          push esi
0048b4f: 53          push ebx
0048b50: 8b 55 08      mov edx,DWORD PTR [ebp+0x8]
0048b53: 83 c4 f8      add esp,0xffffffff
0048b56: 8d 45 e8      lea eax,[ebp-0x18]
0048b59: 50          push eax
0048b5a: 52          push edx
0048b5b: e8 78 04 00 00 call 8048fd8 <read_six_numbers>
0048b60: 83 c4 10      add esp,0x10
0048b63: 83 7d e8 01   cmp DWORD PTR [ebp-0x18],0x1
0048b67: 74 05        je 8048b6e <phase_2+0x26>
0048b69: e8 8e 09 00 00 call 80494fc <explode_bomb>
0048b6e: bb 01 00 00 00 mov ebx,0x1
0048b73: 8d 75 e8      lea esi,[ebp-0x18]
0048b76: 8d 43 01      lea eax,[ebx+0x1]
0048b79: 0f af 44 9e fc imul eax,DWORD PTR [esi+ebx*4-0x4]
0048b7e: 39 04 9e      cmp DWORD PTR [esi+ebx*4],eax
0048b81: 74 05        je 8048b88 <phase_2+0x48>
0048b83: e8 74 09 00 00 call 80494fc <explode_bomb>
0048b88: 43          inc ebx
0048b89: 83 fb 05      cmp ebx,0x5
0048b8c: 7e e8        jle 8048b76 <phase_2+0x2e>
0048b8e: 8d 65 d8      lea esp,[ebp-0x28]
0048b91: 5b          pop ebx
0048b92: 5e          pop esi
0048b93: 89 ec        mov esp,ebp
0048b95: 5d          pop ebp
0048b96: c3          ret
```

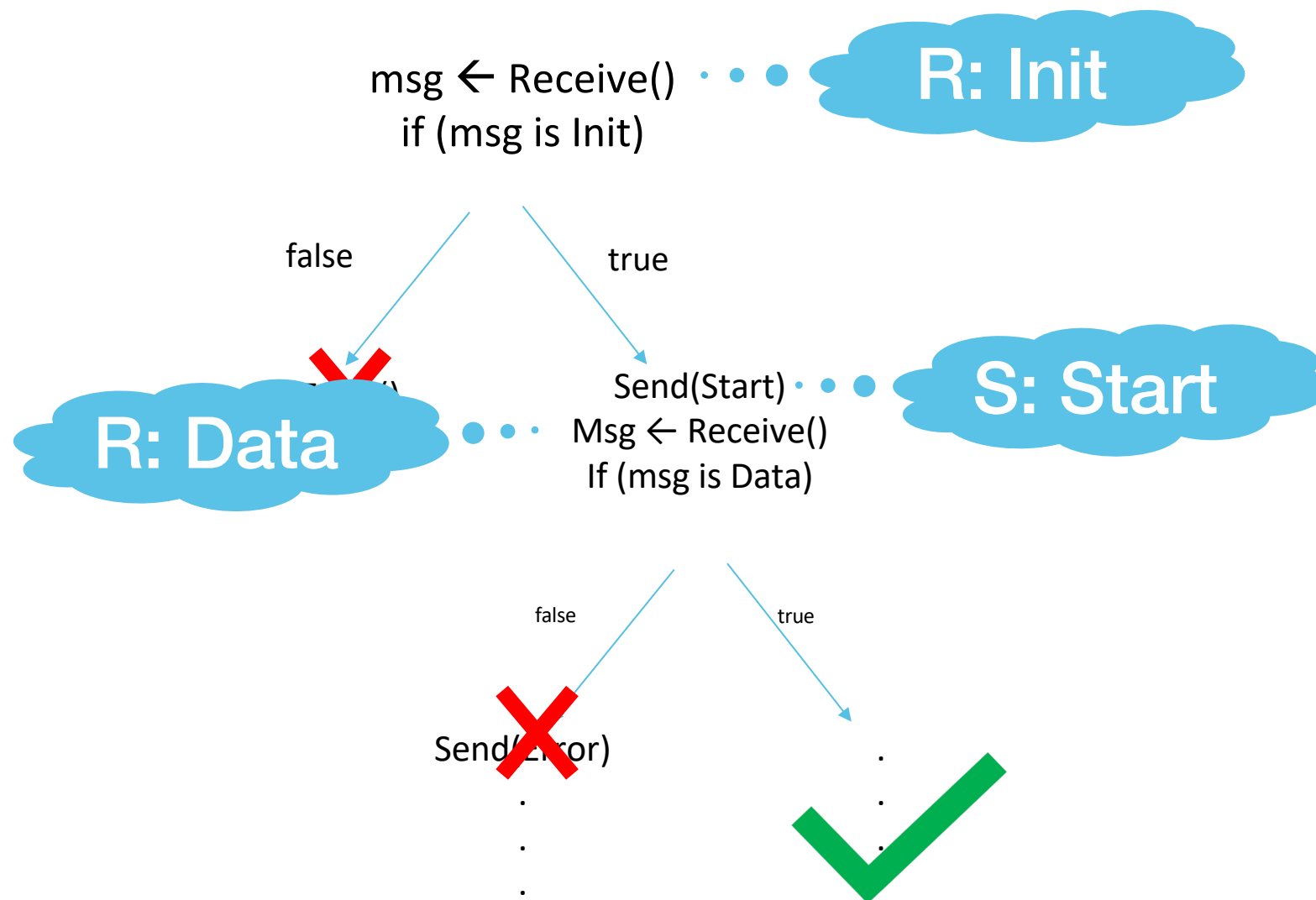
Symbolic Execution

$a > 3$, $b = 2789$



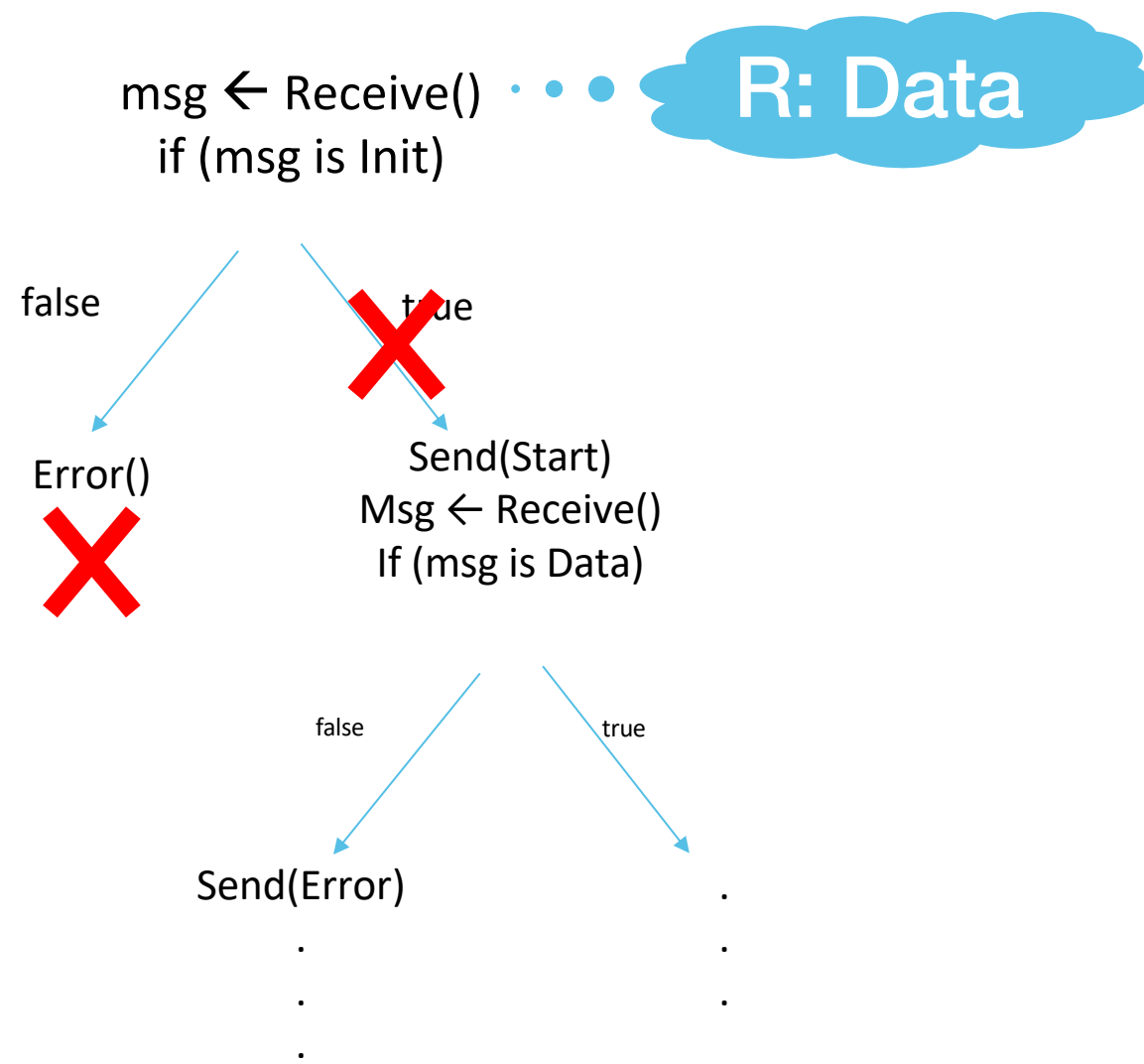
Answering Membership queries

Is {R: Init, S: Start, R: Data} valid for the protocol?



Answering Membership queries

Is {R: Data} valid for the protocol?



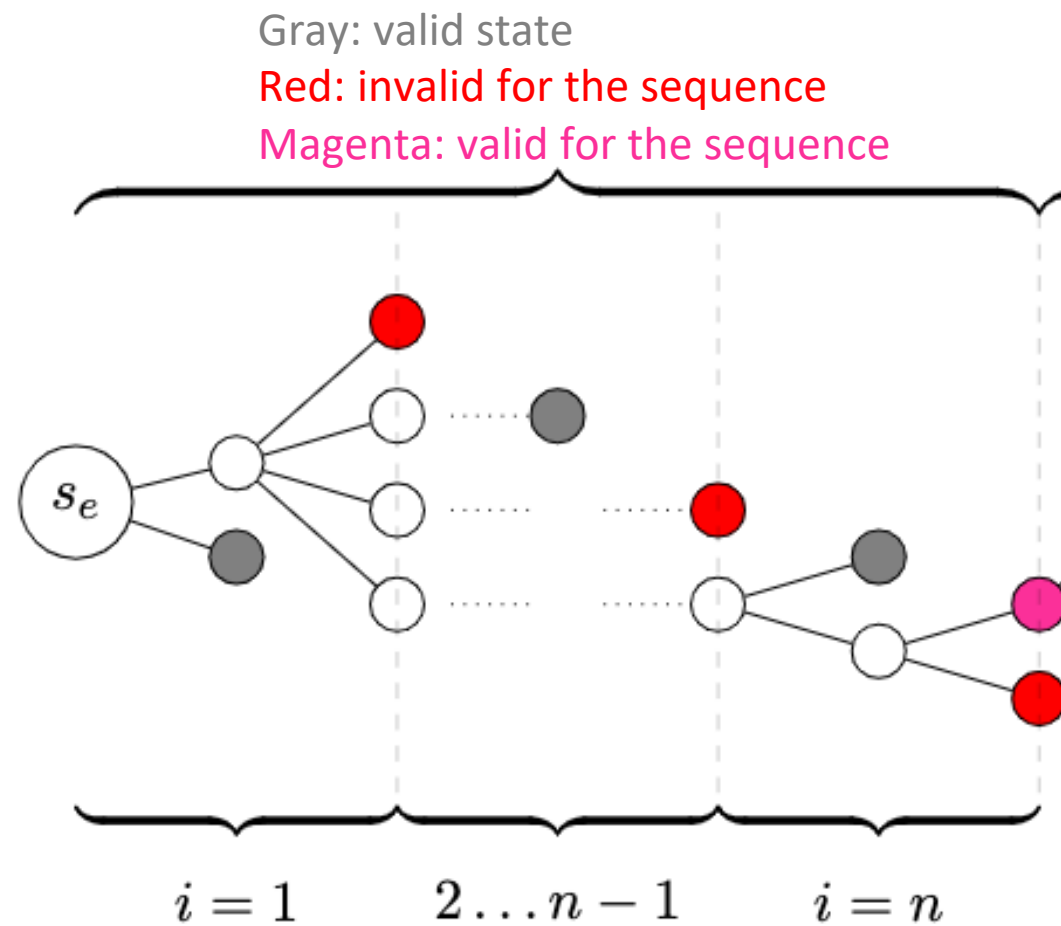
Answering Membership queries

- Let $M = \{M_1, \dots, M_n\}$
- Whenever send/receive procedures are called for the i -th time, append M_i as constraint
- After n {send/receive}s, if there are feasible executions – then the sequence M is valid

Send/receive
a message
(i -th time)



Constraint to
 M_i



How to identify a send or receive?

- Intercept calls to send and receive procedures

```

; smtp_status_code __fastcall smtp_write(smtp *const smtp, const char *
smtp_write      proc near          ; CODE XREF: smtp_puts+31↓p

len              = qword ptr -38h
buf              = qword ptr -30h
smtp             = qword ptr -28h
bytes_to_send    = qword ptr -18h
bytes_sent       = qword ptr -10h
buf_offset       = qword ptr -8

endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 40h
mov     [rbp+smtp], rdi
mov     [rbp+buf], rsi
mov     [rbp+len], rdx
mov     rdx, [rbp+buf] ; str
mov     rax, [rbp+smtp]
lea     rsi, aClient ; "Client"
mov     rdi, rax ; smtp
call    smtp_puts_dbg
mov     rax, [rbp+len]
mov     [rbp+bytes_to_send], rax
mov     rax, [rbp+buf]
mov     [rbp+buf_offset], rax
jmp     short loc_402F2B

; -----
loc_402EAC:          ; CODE XREF: smtp_write+C5↓j
mov     eax, 80000000h
cmp     [rbp+bytes_to_send], rax
jb     short loc_402ECA

```

```

; str_getdelim_retcode __cdecl smtp_getline(smtp *const smtp)
smtp_getline    proc near          ; CODE XREF: smtp_read_and_parse_code+26↓p
; smtp_initiate_handshake+2F↓p

smtp            = qword ptr -18h
rc              = dword ptr -4

endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     [rbp+smtp], rdi
call    __errno_location
mov     dword ptr [rax], 0
mov     esi, 1 ; size
mov     edi, 8 ; nmemb
call    _calloc
mov     rdx, rax
mov     rax, [rbp+smtp]
mov     [rax+20h], rdx
mov     rax, [rbp+smtp]
mov     rcx, [rax+20h]
mov     rax, [rbp+smtp]
mov     eax, [rax+4]
mov     edx, 8 ; size
mov     rsi, rcx ; buff
mov     edi, eax ; sock
call    smtp_read_aux
mov     [rbp+rc], eax

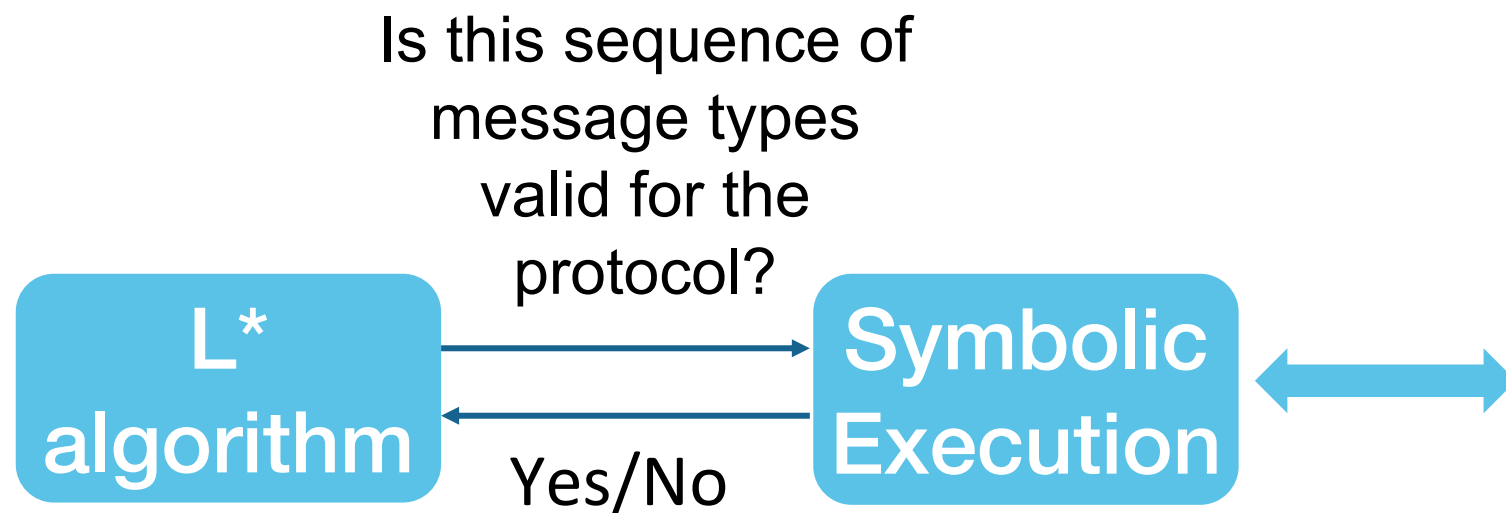
```

Discovering message types

As said, we do not know in advance the protocol's message types.

We update membership queries to discover it little by little.

Extend L* to handle new message types



If yes, here are message types that can follow the sequence.

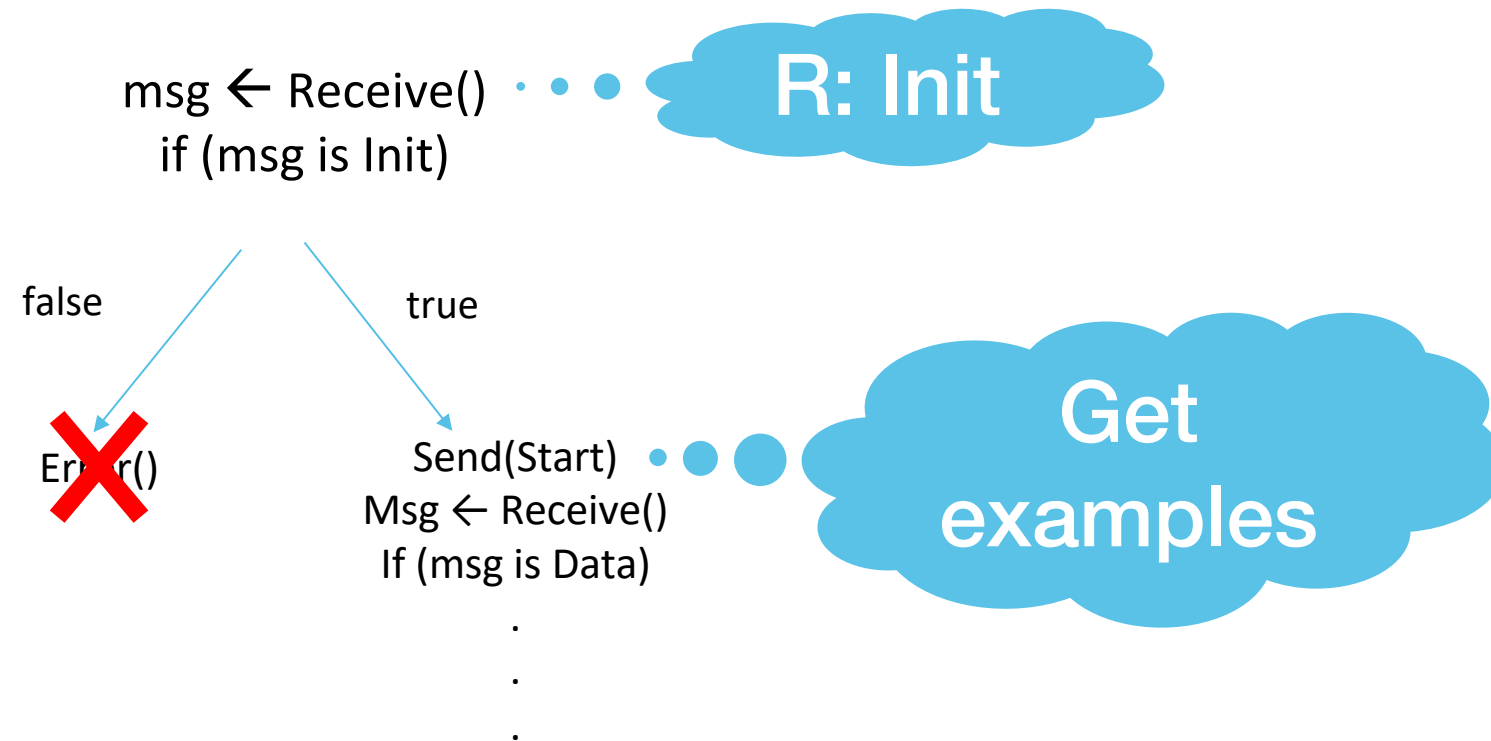
```

8048b29: 83 c4 f8      add     esp,0xffffffff
8048b2c: 68 c0 97 04 08 push   0x80497c0
8048b31: 50          push   eax
8048b32: e8 f9 04 00 00 call   8049030 <strings_not_equal>
8048b37: 83 c4 10      add     esp,0x10
8048b3a: 85 c0        test   eax,eax
8048b3c: 74 05        je     8048b43 <phase_1+0x23>
8048b3e: e8 b9 09 00 00 call   80494fc <explode_bomb>
8048b43: 89 ec        mov     esp,ebp
8048b45: 5d          pop     ebp
8048b46: c3          ret
8048b47: 90          nop

8048b48: <phase_2>:
8048b48: 55          push   ebp
8048b49: 89 e5        mov     ebp,esp
8048b4b: 83 ec 20      sub     esp,0x20
8048b4e: 56          push   esi
8048b4f: 53          push   ebx
8048b50: 8b 55 08      mov     edx,DWORD PTR [ebp+0x8]
8048b53: 83 c4 f8      add     esp,0xffffffff
8048b56: 8d 45 e8      lea    eax,[ebp-0x18]
8048b59: 50          push   eax
8048b5a: 52          push   edx
8048b5b: e8 78 04 00 00 call   8048fd8 <read_six_numbers>
8048b60: 83 c4 10      add     esp,0x10
8048b63: 83 7d e8 01   cmp    DWORD PTR [ebp-0x18],0x1
8048b67: 74 05        je     8048b6e <phase_2+0x26>
8048b69: e8 0e 09 00 00 call   80494fc <explode_bomb>
8048b6e: bb 01 00 00 00 mov     ebx,0x1
8048b73: 8d 75 e8      lea    esi,[ebp-0x18]
8048b76: 8d 43 01      lea    eax,[ebx+0x1]
8048b79: 0f af 44 9e fc imul   eax,DWORD PTR [esi+ebx*4],eax
8048b7e: 39 04 9e      cmp    DWORD PTR [esi+ebx*4],eax
8048b81: 74 05        je     8048b88 <phase_2+0x40>
8048b83: e8 74 09 00 00 call   80494fc <explode_bomb>
8048b88: 43          inc    ebx
8048b89: 83 fb 05      cmp    ebx,0x5
8048b8c: 7e e8        jle   8048b76 <phase_2+0x2e>
8048b8e: 8d 65 d8      lea    esp,[ebp-0x28]
8048b91: 5b          pop     ebx
8048b92: 5e          pop     esi
8048b93: 89 ec        mov     esp,ebp
8048b95: 5d          pop     ebp
8048b96: c3          ret
  
```

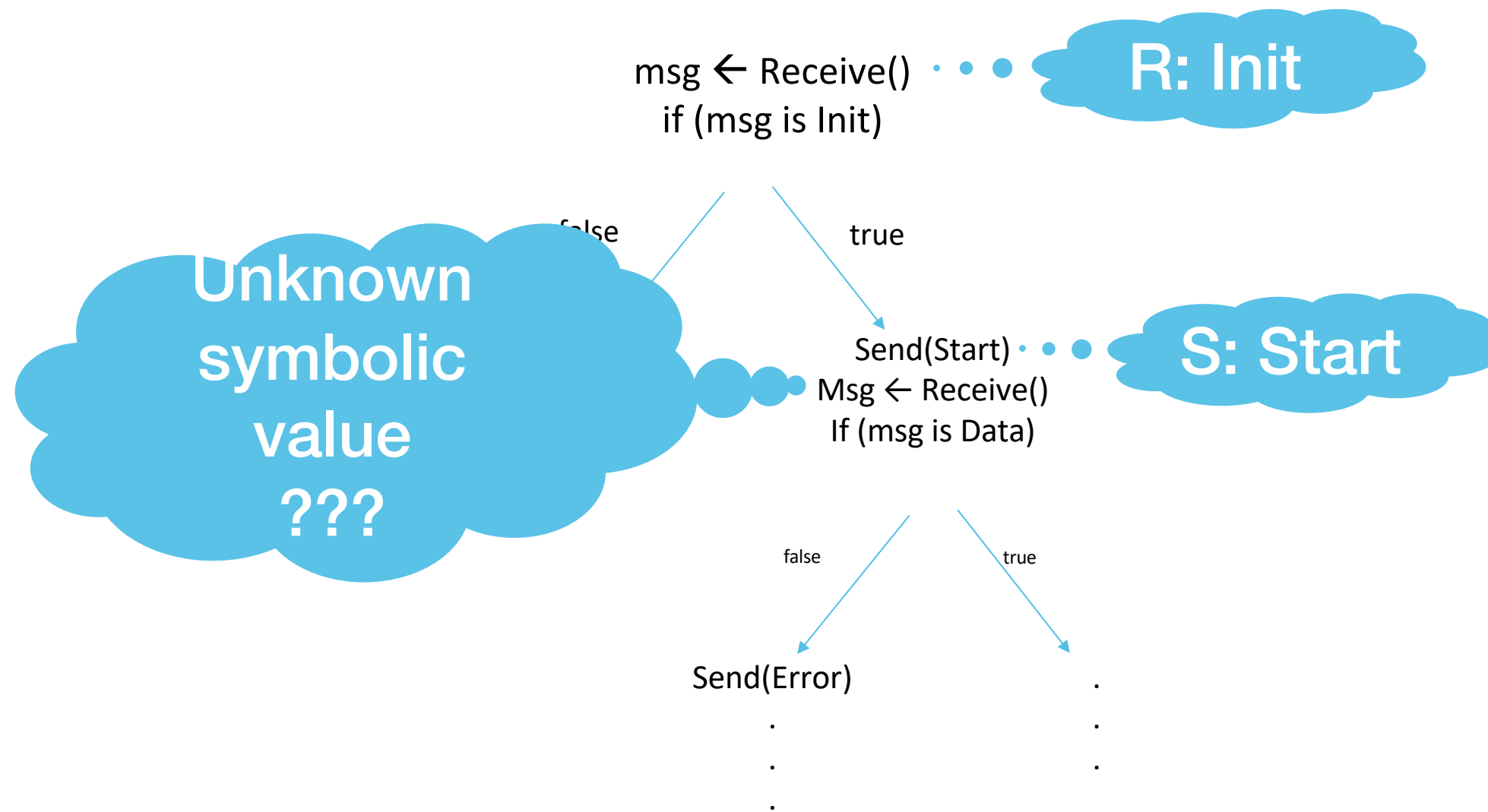

Probing for following message types

What message types can follow {R: Init}?

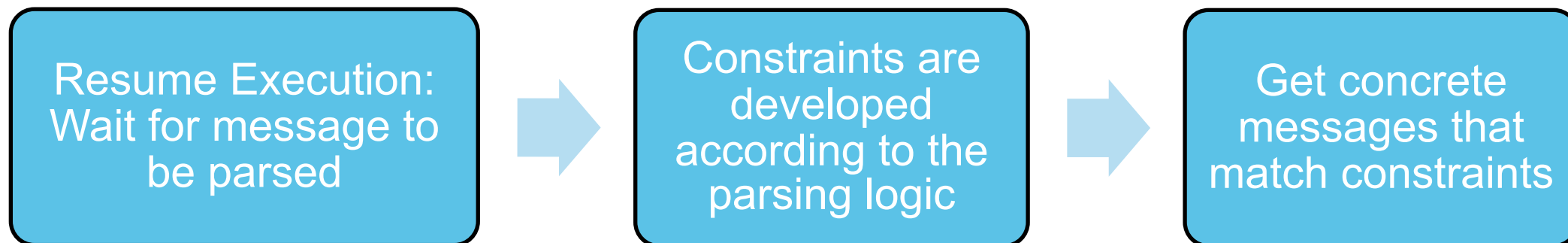


Probing for following message types

What message types can follow {R: Init, S: Start}?



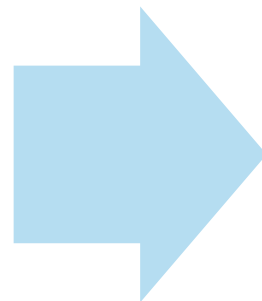
Probing for following message types



```
msg ← receive()  
if (msg begins with 'data') {  
    // Constraint: msg begins with 'Data' ✓  
}  
else {  
    // I can't parse this message, error  
}
```

Concrete messages → Message type

Example
Messages

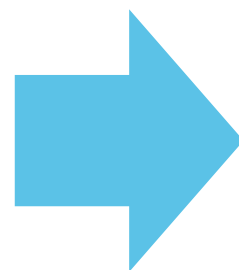


Find features of
message type

RCPT TO: email1@blabla.com

RCPT TO: email2@lalala.com

RCPT TO: email3@nana.com



RCPT%20TO: *%0D%0A

Tying it all together

Is this sequence of message types valid for the protocol?

L* algorithm

Yes/No

Symbolic Execution

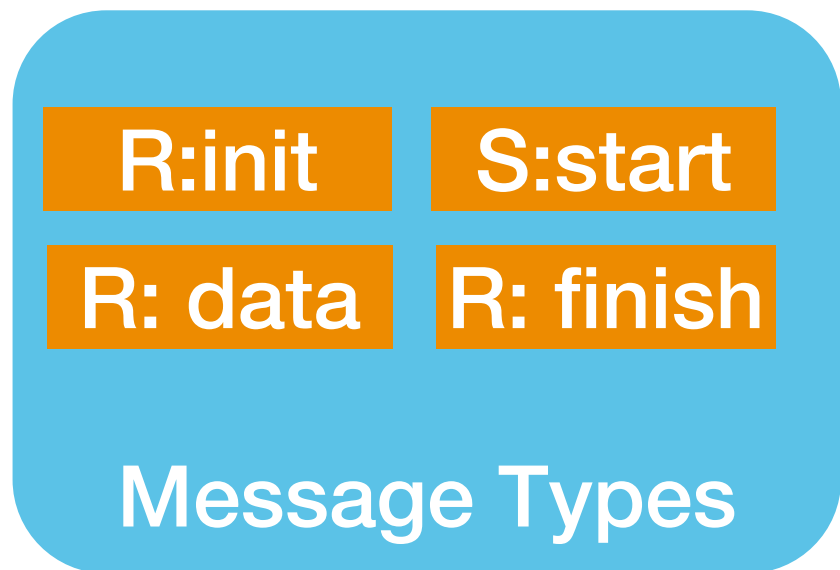
If yes, here are message types that can follow the sequence.

```

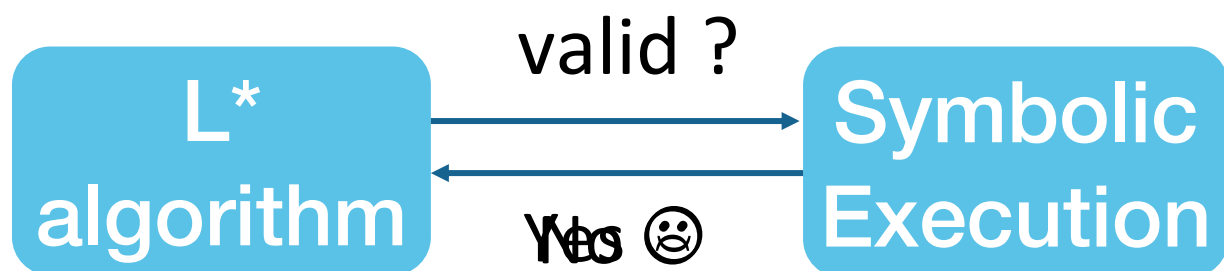
804bb29: 83 c4 f8      add     esp,0xffffffff
804bb2c: 68 c0 97 04 08 push  0x80497c0
804bb31: 50           push  eax
804bb32: e8 f9 04 00 00 call  8049030 <strings_not_equal>
804bb37: 83 c4 10      add     esp,0x10
804bb3a: 85 c0        test   eax,eax
804bb3c: 74 05        je     8048b43 <phase_1+0x23>
804bb3e: e8 b9 09 00 00 call  80494fc <explode_bomb>
804bb43: 89 ec        mov   esp,ebp
804bb45: 5d          pop   ebp
804bb46: c3          ret
804bb47: 90          nop

804bb48 <phase_2>:
804bb48: 55          push  ebp
804bb49: 89 e5        mov   ebp,esp
804bb4b: 83 ec 20      sub   esp,0x20
804bb4e: 56          push  esi
804bb4f: 53          push  ebx
804bb50: 8b 55 08      mov   edx,DWORD PTR [ebp+0x8]
804bb53: 83 c4 f8      add   esp,0xffffffff
804bb56: 8d 45 e8      lea  eax,[ebp-0x18]
804bb59: 50          push  eax
804bb5a: 52          push  edx
804bb5b: e8 78 04 00 00 call  8048fd8 <read_six_numbers>
804bb60: 83 c4 10      add   esp,0x10
804bb63: 83 7d e8 01   cmp   DWORD PTR [ebp-0x18],0x1
804bb67: 74 05        je     8048b6e <phase_2+0x26>
804bb69: e8 8e 09 00 00 call  80494fc <explode_bomb>
804bb6e: bb 01 00 00 00 mov   ebx,0x1
804bb73: 8d 75 e8      lea  esi,[ebp-0x18]
804bb76: 8d 43 01      lea  eax,[ebp+0x1]
804bb79: 0f af 44 9e fc imul  eax,DWORD PTR [esi+ebx*4-0x4]
804bb7e: 39 04 9e      cmp   DWORD PTR [esi+ebx*4],eax
804bb81: 74 05        je     8048b88 <phase_2+0x40>
804bb83: e8 74 09 00 00 call  80494fc <explode_bomb>
804bb88: 43          inc   ebx
804bb89: 83 fb 05      cmp   ebx,0x5
804bb8c: 7e e8        jle  8048b76 <phase_2+0x2e>
804bb8e: 8d 65 d8      lea  esp,[ebp-0x28]
804bb91: 5b          pop   ebx
804bb92: 5e          pop   esi
804bb93: 89 ec        mov   esp,ebp
804bb95: 5d          pop   ebp
804bb96: c3          ret
  
```

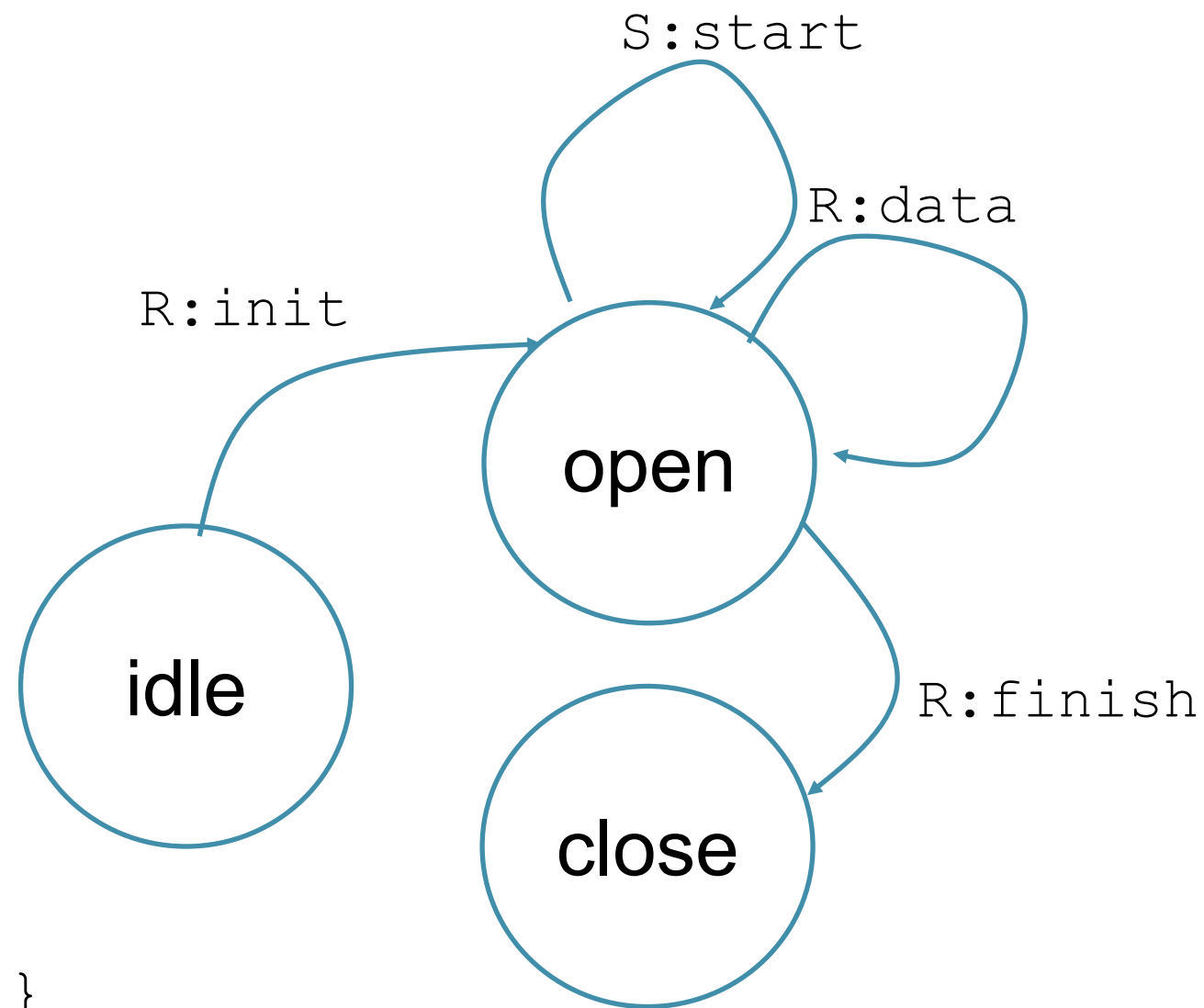
An illustrative example



$M = \{ R:init, R: data, R: finish, S:start \}$



$m_{next} = \{ S:start, R: data, R: finish \}$



```

cmp     ebx,0x1
jle     8048b76 <phase_2+0x2e>
lea     esp,[ebp-0x28]
pop     ebx
pop     esi
mov     esp,ebp
pop     ebp
ret     nop

lea     ecx,[ebx+0x1]
imul   eax,DWORD PTR [esi+ebx*4]
cmp     DWORD PTR [ebp-0x18],0x1
jle     8048b6e <phase_2+0x26>
call   80494fc <explode_bomb>
inc     ebx
cmp     ebx,0x5
jle     8048b76 <phase_2+0x2e>
lea     esp,[ebp-0x28]
pop     ebx
pop     esi

esp,0x10
mov     ebp,esp
sub     esp,0x20
push   eax
push   esi
call   8049030 <strings_not_eq
add     esp,0x10
test   eax,eax
je      8048b43 <phase_1+0x23>
call   80494fc <explode_bomb>
mov     esp,ebp
pop     ebp
ret

esp,0x10
mov     ebp,esp
sub     esp,0x20
push   eax
push   esi
call   8049030 <strings_not_eq
add     esp,0x10
test   eax,eax
je      8048b43 <phase_1+0x23>
call   80494fc <explode_bomb>
mov     esp,ebp
pop     ebp
ret

eax,DWORD PTR [esi+ebx*4]
push   eax
push   edx
call   8048fd8 <read_six_number>

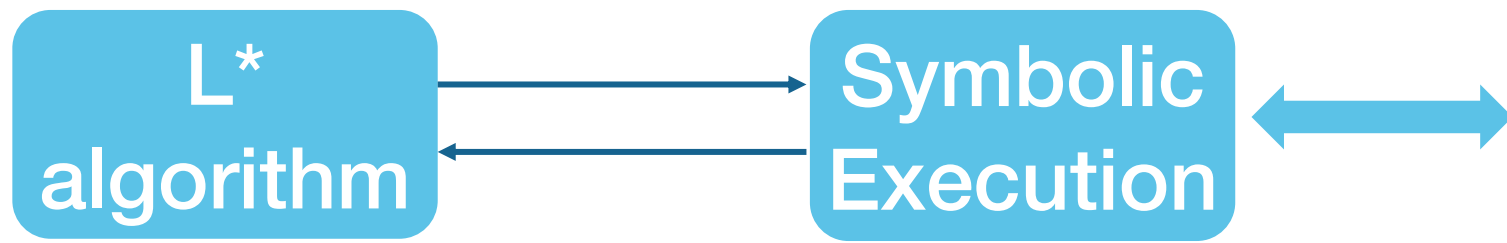
```

Caveats

PISE is as good or as bad as the symbolic tool it uses.

Currently, PISE supports only **angr**. 🤪

- Trouble supporting multiple threads.
- Does not fully support windows API



```

8048b29: 83 c4 f8      add     esp,0xffffffff
8048b2c: 68 c0 97 04 08 push  0x80497c0
8048b31: 50          push   eax
8048b32: e8 f9 04 00 00 call  8049030 <strings_not_equal>
8048b37: 83 c4 10      add     esp,0x10
8048b3a: 85 c0        test   eax,eax
8048b3c: 74 05        je      8048b43 <phase_1+0x23>
8048b3e: e8 b9 09 00 00 call  80494fc <explode_bomb>
8048b43: 89 ec        mov     esp,ebp
8048b45: 5d          pop     ebp
8048b46: c3          ret
8048b47: 90          nop

8048b48 <phase_2>:
8048b48: 55          push   ebp
8048b49: 89 e5        mov     ebp,esp
8048b4b: 83 ec 20     sub     esp,0x20
8048b4e: 56          push   esi
8048b4f: 53          push   ebx
8048b50: 8b 55 08     mov     edx,DWORD PTR [ebp+0x8]
8048b53: 83 c4 f8     add     esp,0xffffffff
8048b56: 8d 45 e8     lea    eax,[ebp-0x18]
8048b59: 50          push   eax
8048b5a: 52          push   edx
8048b5b: e8 78 04 00 00 call  8048fd8 <read_six_numbers>
8048b60: 83 c4 10     add     esp,0x10
8048b63: 83 7d e8 01  cmp     DWORD PTR [ebp-0x18],0x1
8048b67: 74 05        je      8048b6e <phase_2+0x26>
8048b69: e8 8e 09 00 00 call  80494fc <explode_bomb>
8048b6e: bb 01 00 00 00 mov     ebx,0x1
8048b73: 8d 75 e8     lea    esi,[ebp-0x18]
8048b76: 8d 43 01     lea    eax,[ebx+0x1]
8048b79: 0f af 44 9e fc imul   eax,DWORD PTR [esi+ebx*4]
8048b7e: 39 04 9e     cmp     DWORD PTR [esi+ebx*4],eax
8048b81: 74 05        je      8048b88 <phase_2+0x40>
8048b83: e8 74 09 00 00 call  80494fc <explode_bomb>
8048b88: 43          inc     ebx
8048b89: 83 fb 05     cmp     ebx,0x5
8048b8c: 7e e8        jle    8048b76 <phase_2+0x2e>
8048b8e: 8d 65 d8     lea    esp,[ebp-0x28]
8048b91: 5b          pop     ebx
8048b92: 5e          pop     esi
8048b93: 89 ec        mov     esp,ebp
8048b95: 5d          pop     ebp
8048b96: c3          ret

```

Summary

```

cmp     ebx,0x5
jle    8048b76 <phase_2+0x2e>
lea    esp,[ebp-0x28]
pop    ebx
pop    esi
mov    esp,ebp
pop    ebp
ret

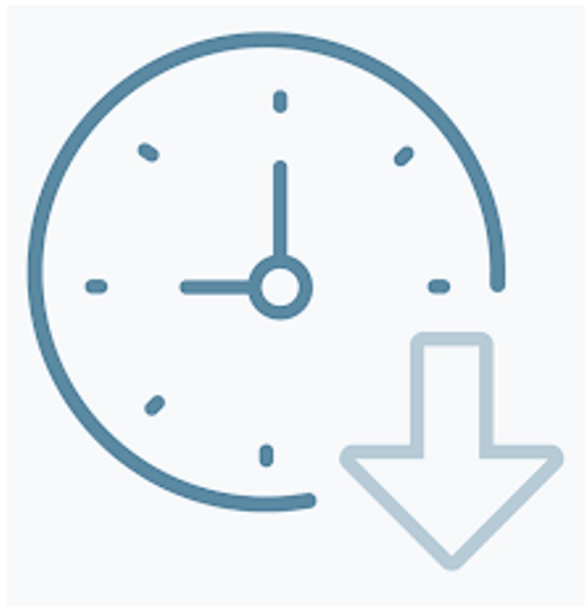
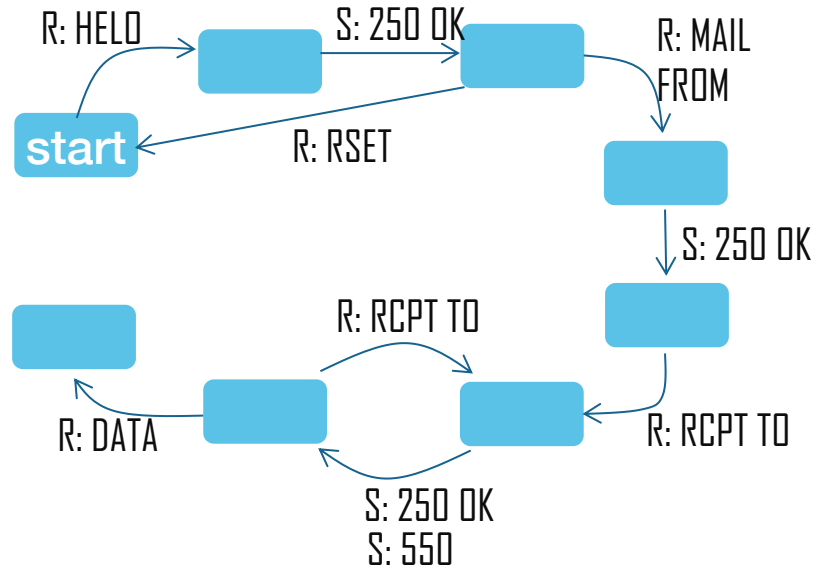
lea    ebx,[ebx+0x1]
imul   eax,DWORD PTR [esi+ebx*4],eax
add    esp,0x10
mov    ebp,esp
push   ebx
push   esi
push   ebx
call   8049030 <strings_not_eq>
add    esp,0x10
test   eax,eax
je     8048b43 <phase_1+0x23>
call   80494fc <explode_bomb>
mov    esp,ebp
pop    ebp
ret

mov    ebx,0x5
cmp    ebx,0x5
jle    8048b76 <phase_2+0x2e>
lea    esp,[ebp-0x28]
pop    ebx
pop    esi
ret
  
```

```

8048b27 83 c4 fb      add     esp,0xffffffff
8048b2c 86 c9 97 04 00 push  0x0097c0
8048b31 59           push   eax
8048b32 49 f9 04 00 00 call   8049030 <strings_not_eq>
8048b37 83 c4 10      add     esp,0x10
8048b3a 85 c9        test   eax,eax
8048b3c 74 05        je     8048b43 <phase_1+0x23>
8048b3e 48 09 09 00 00 call   80494fc <explode_bomb>
8048b43 89 ec        mov    esp,ebp
8048b45 5d           pop    ebp
8048b47 9b         ret

8048b48 <phase_2>:
8048b48 55         push   ebp
8048b49 89 e5        mov    ebp,esp
8048b4b 83 ec 20      sub    esp,0x20
8048b4e 56         push   esi
8048b4f 53         push   ebx
8048b50 8b 55 08      mov    edx,DWORD PTR [ebp+0x8]
8048b53 83 c4 f8      add    esp,0xffffffff
8048b56 8d 45 e8      lea   eax,[ebp-0x18]
8048b59 58         push   eax
8048b5a 52         push   edx
8048b5b 48 78 04 00 00 call   8048fd8 <read_six_numbers>
8048b60 83 c4 10      add    esp,0x10
8048b62 83 76 e8 01 01 cmp    DWORD PTR [ebp-0x18],0x1
8048b67 74 05        je     8048b6e <phase_2+0x26>
8048b69 48 0e 09 00 00 call   80494fc <explode_bomb>
8048b6e 8b 01 00 00 00 mov    ebx,0x1
8048b71 8d 75 e8      lea   esi,[ebp-0x18]
8048b74 8b 43 01      lea   eax,[ebx+0x1]
8048b77 8f e4 7e fc 01 imul  eax,DWORD PTR [esi+ebx*4],eax
8048b7c 39 04 9e      cmp    DWORD PTR [esi+ebx*4],eax
8048b81 74 05        je     8048b88 <phase_2+0x40>
8048b83 48 74 09 00 00 call   80494fc <explode_bomb>
8048b88 43         inc    ebx
8048b89 83 fb 05      cmp    ebx,0x5
8048b8c 7e e8        jle   8048b76 <phase_2+0x2e>
8048b90 8d 65 08      lea   esp,[ebp+0xc]
8048b93 5b         pop    ebx
8048b94 5e         pop    esi
8048b95 89 ec        mov    esp,ebp
8048b97 5d         pop    ebp
8048b99 83         ret
  
```



<https://github.com/ron4548/PISE>


```
cmp ebx,0x...  
jle 8048b76 <phase_2+0x2e>  
lea esp,[ebp-0x28]  
pop ebx  
pop esi  
inc esp,ebp  
pop ebp  
ret  
nop  
  
mov ecx,[ebx+0x1]  
imul eax,DWORD PTR [esi+ebx*4]  
cmp DWORD PTR [ebp-0x18],eax  
je 8048b88 <phase_2+0x40>  
call 80494fc <explode_bomb>  
inc ebx  
cmp ebx,0x5  
jle 8048b76 <phase_2+0x2e>  
lea esp,[ebp-0x28]  
pop ebx  
pop esi  
  
mov esp,0x10  
push esp  
mov ebp,esp  
sub esp,0x20  
push esi  
push ebx  
mov edx,DWORD PTR [ebp+0x8]  
add esp,0xffffffff  
lea eax,[ebp-0x18]  
push eax  
push edx  
call 8048fd8 <read_six_number>  
  
add esp,0xffffffff  
push 0x80497c0  
push eax  
call 8049030 <strings_not_eq>  
add esp,0x10  
test eax,eax  
je 8048b43 <phase_1+0x23>  
call 80494fc <explode_bomb>  
mov esp,ebp  
pop ebp  
ret
```

Questions



ron.mar@campus.technion.ac.il

gabin@radware.com

<https://github.com/ron4548/PISE>