black hat USA 2022

PISE: Automatic Protocol Reverse Engineering

Ron Marcovich, Orna Grumberg and Gabi Nakibly

black hat USA 2022

Introductions

Ron Marcovich

TECHNION Israel Institute of Technology

M.Sc. Student

Dr. Gabi Nakibly

TECHNION Israel Institute of Technology

Senior Adjunct Lecturer

radware

Distinguished Researcher

Formerly at

RAFAEL SMART AND TO THE POINT

Prof. Orna Grumberg

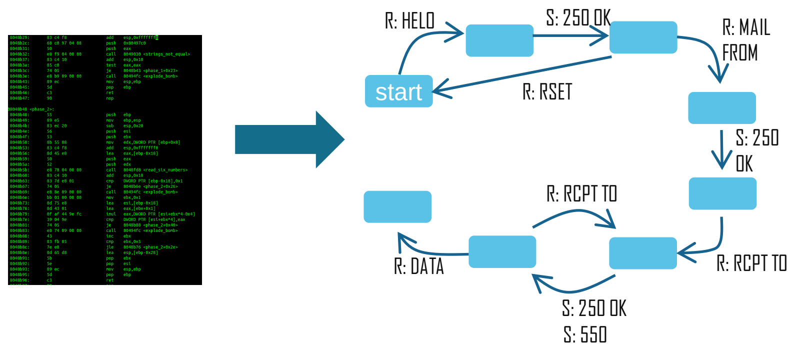
TECHNION Israel Institute of Technology

Faculty Member

Agenda

What is protocol RE?

What is PISE all about?



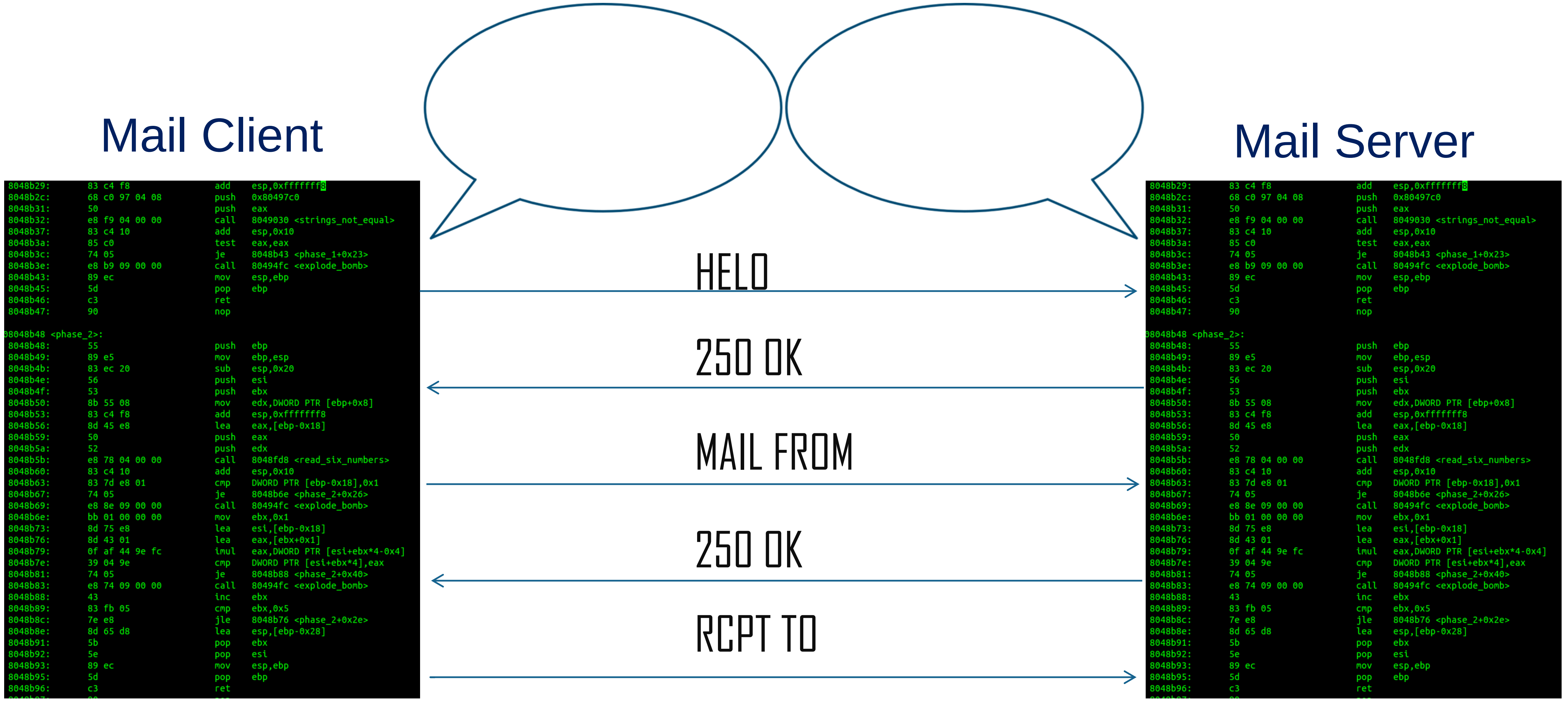
How PISE does its magic?

Motivation and Background

What is protocol reverse engineering?

Hi
How are you?
Hello
Tuesday
I am fine.

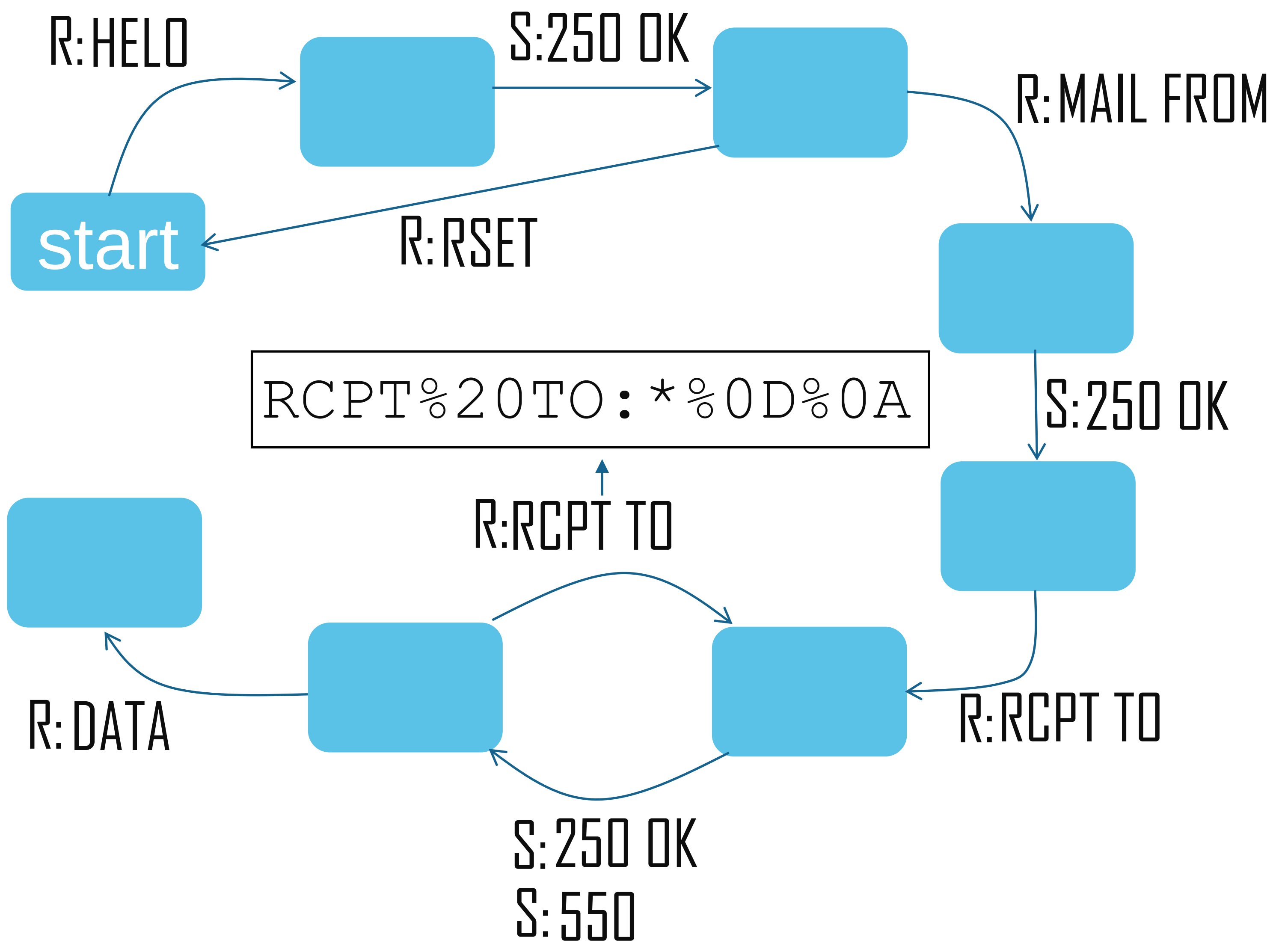
What is protocol reverse engineering?



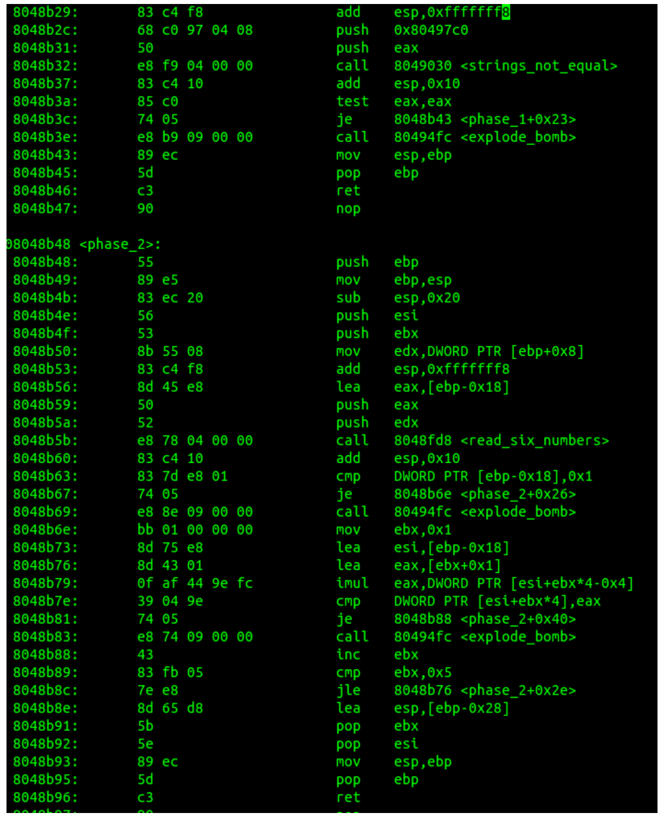
```
 8048b29:       83 c4 f8                add    esp,0xfffffff8
 8048b2c:       68 c0 97 04 08          push   0x80497c0
 8048b31:       50                      push   eax
 8048b32:       e8 f9 04 00 00          call   8049030 <strings_not_equal>
 8048b37:       83 c4 10                add    esp,0x10
 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 f8                add    esp,0xfffffff8
 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,0xfffffff8
 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 f8                add    esp,0xfffffff8
 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?

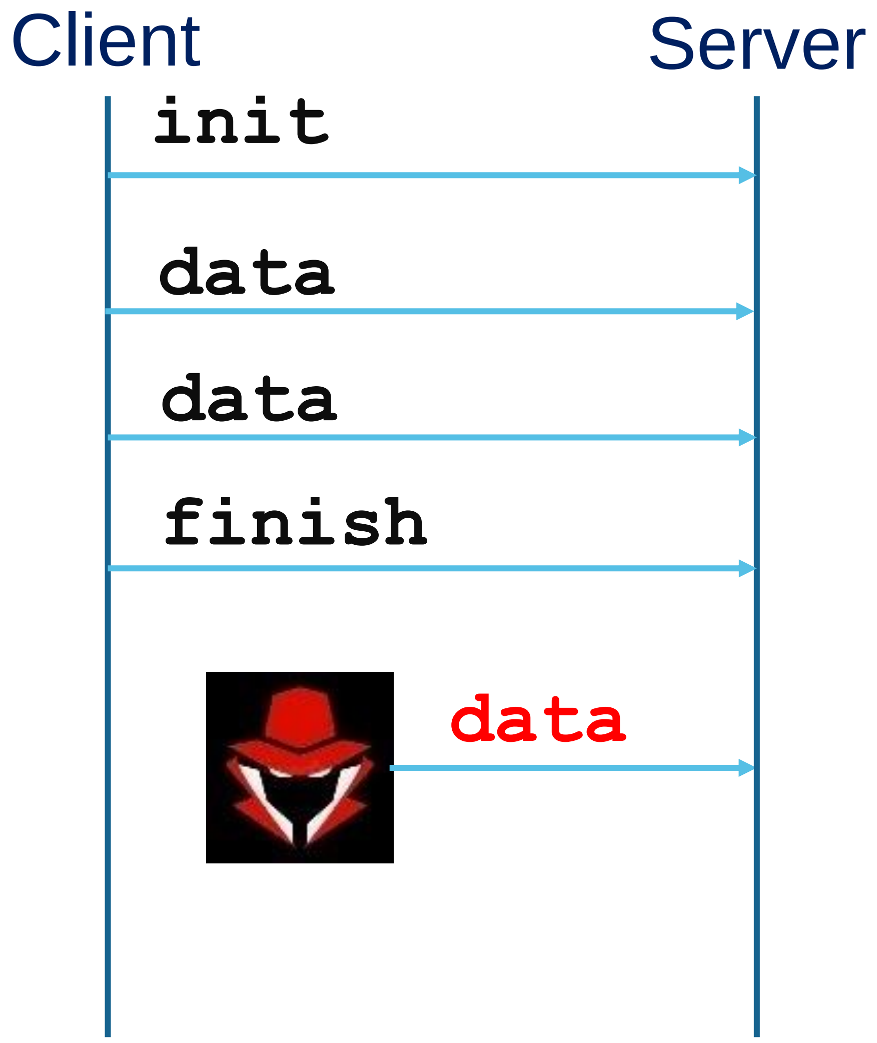


Mail Server

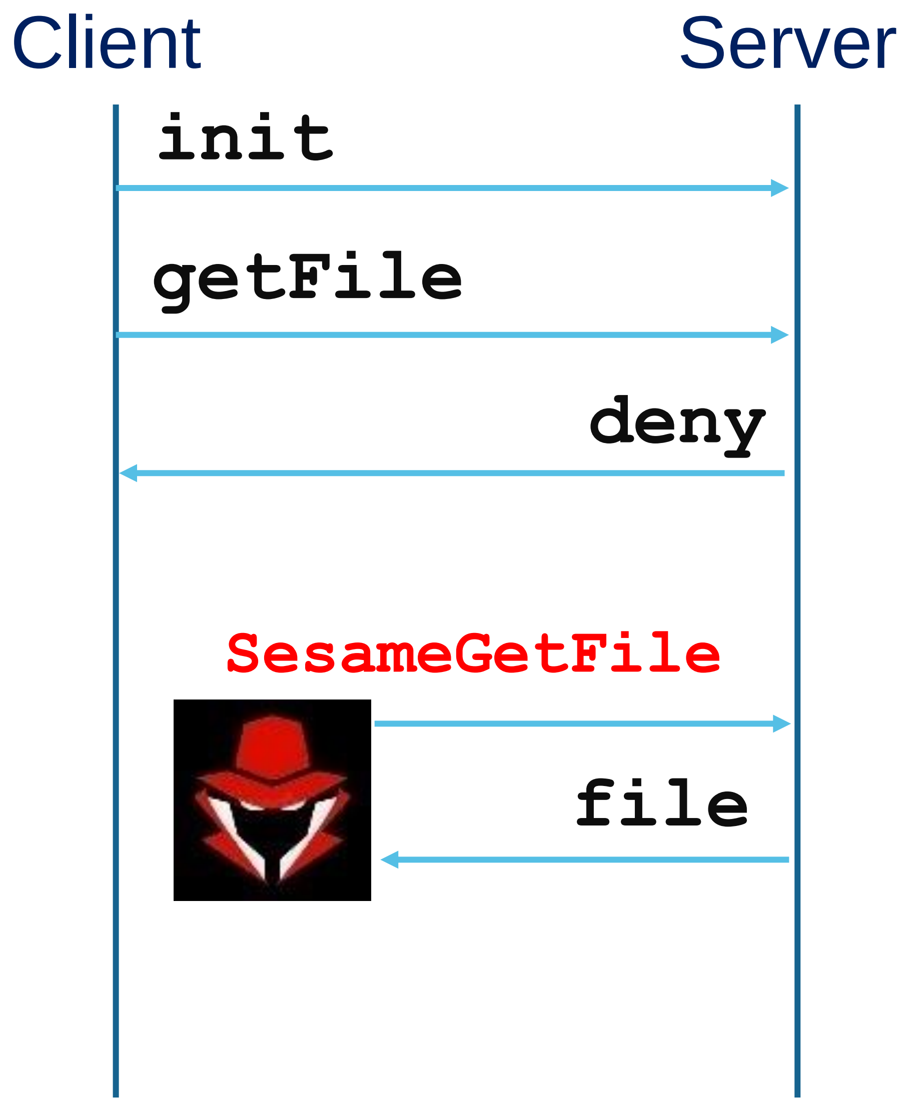


```
 8048b29:       83 c4 f8                add    esp,0xfffffff8
 8048b2c:       68 c0 97 04 08          push   0x80497c0
 8048b31:       50                      push   eax
 8048b32:       e8 f9 04 00 00          call   8049030 <strings_not_equal>
 8048b37:       83 c4 10                add    esp,0x10
 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 f8                add    esp,0xfffffff8
 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
```

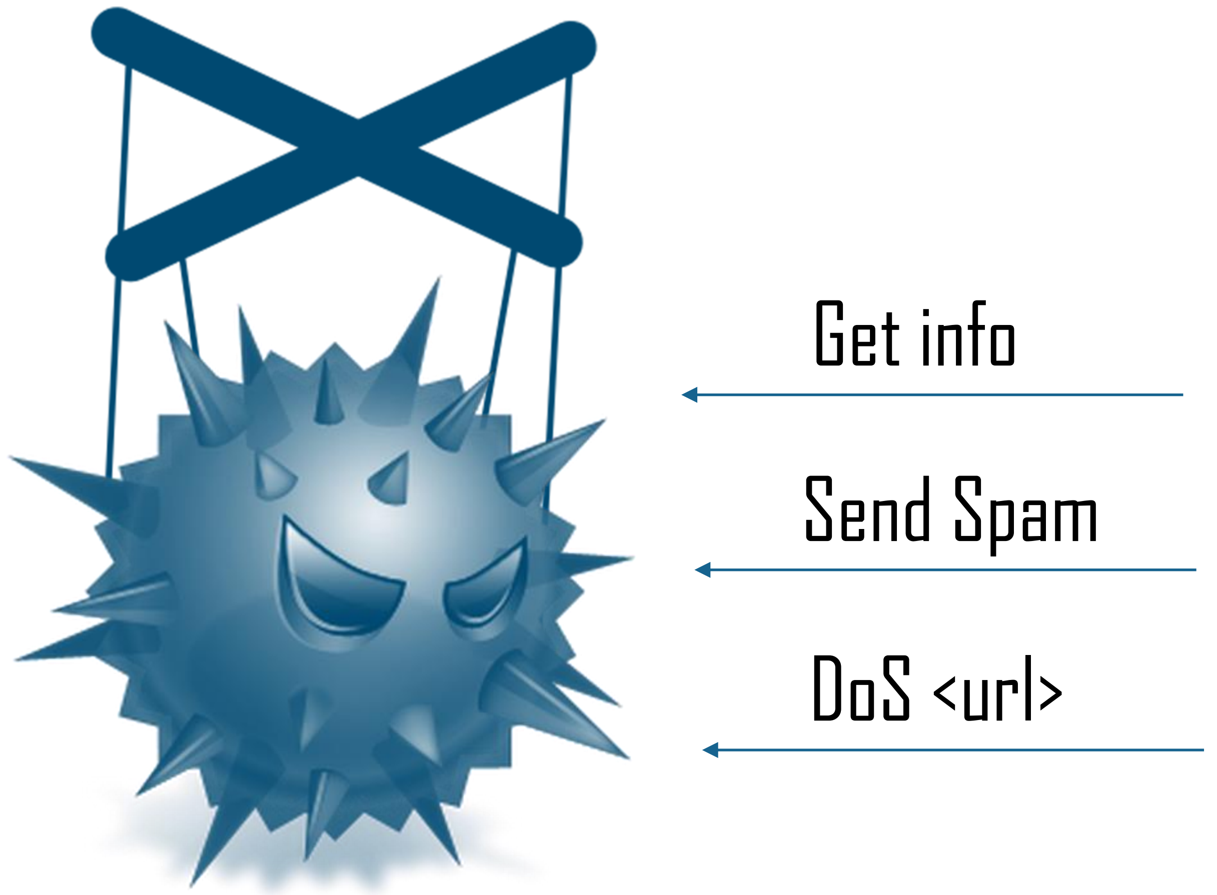
Motivation I – Finding Bugs



Motivation II – Finding backdoors

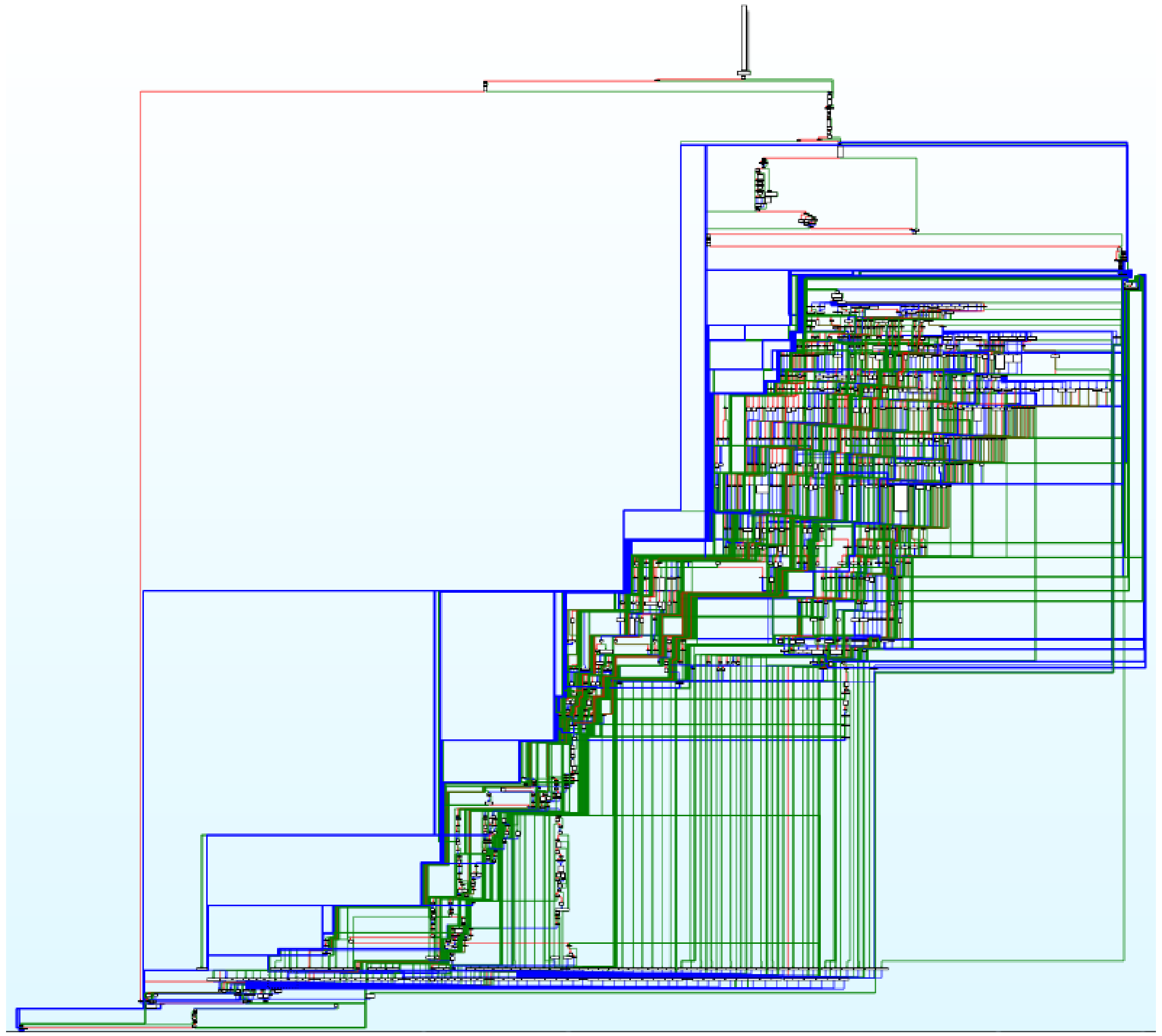


Motivation III – Analyzing Malware

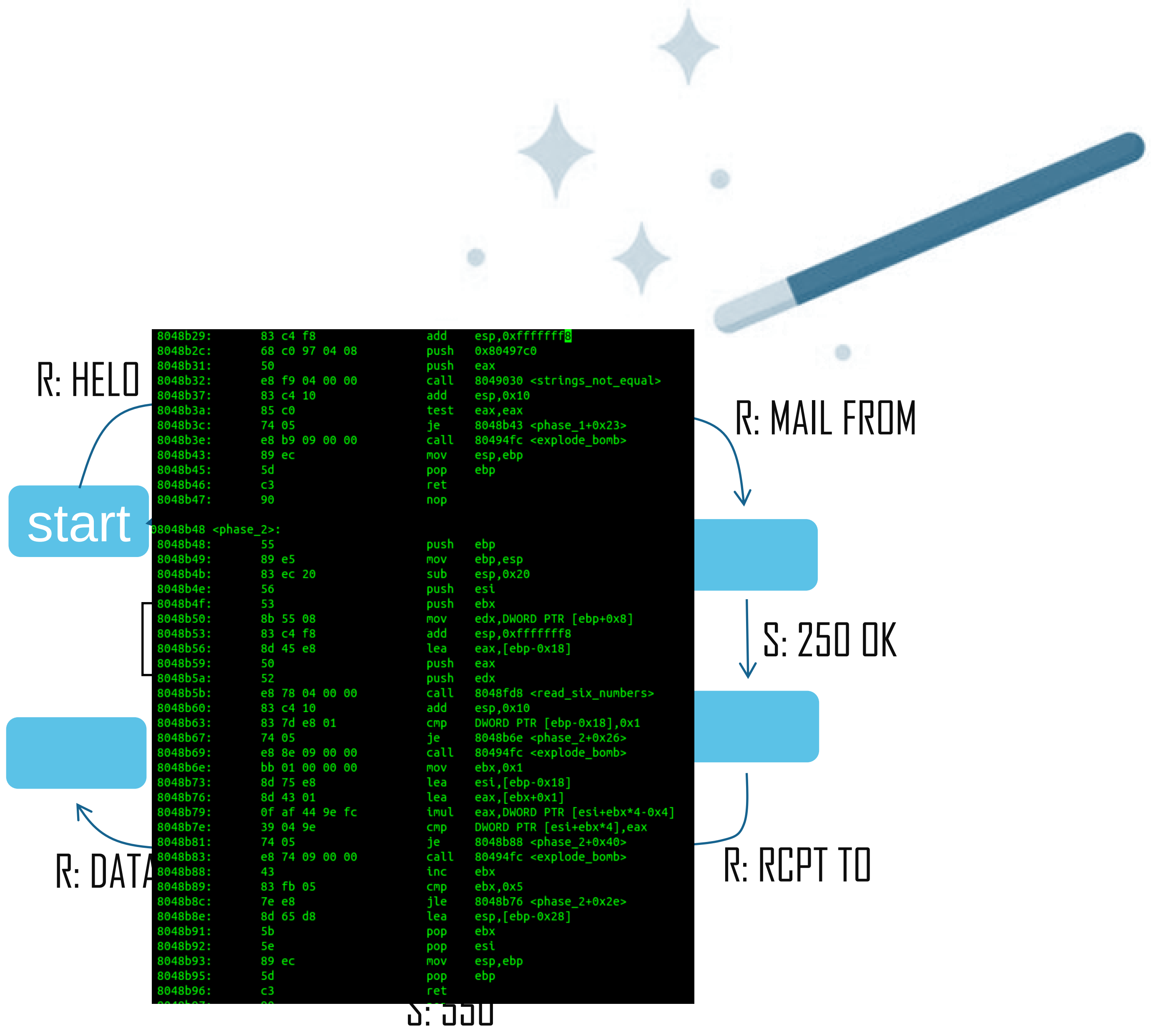


Protocol RE is Hard!

It can be days or even weeks!



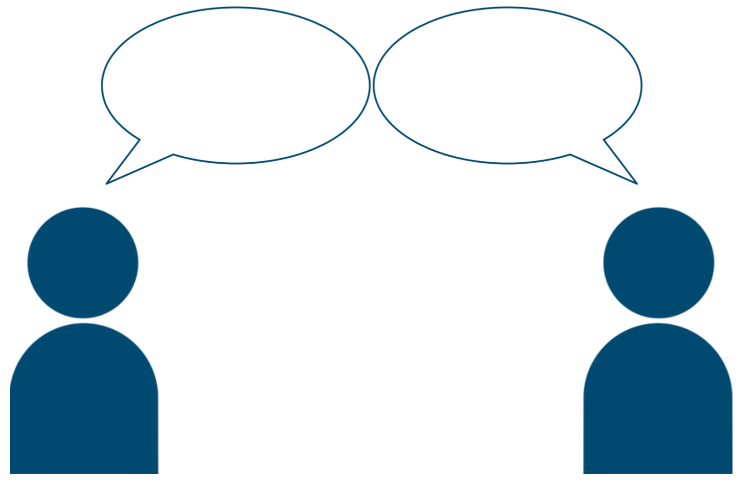
Research Goal



```
8048b29:       83 c4 f8                add    esp,0xfffffff8
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 f8                add    esp,0xfffffff8
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
```

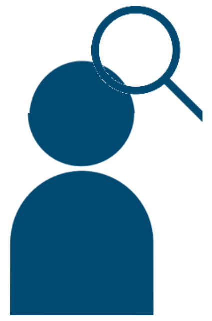
No Assumptions

No past traffic captures



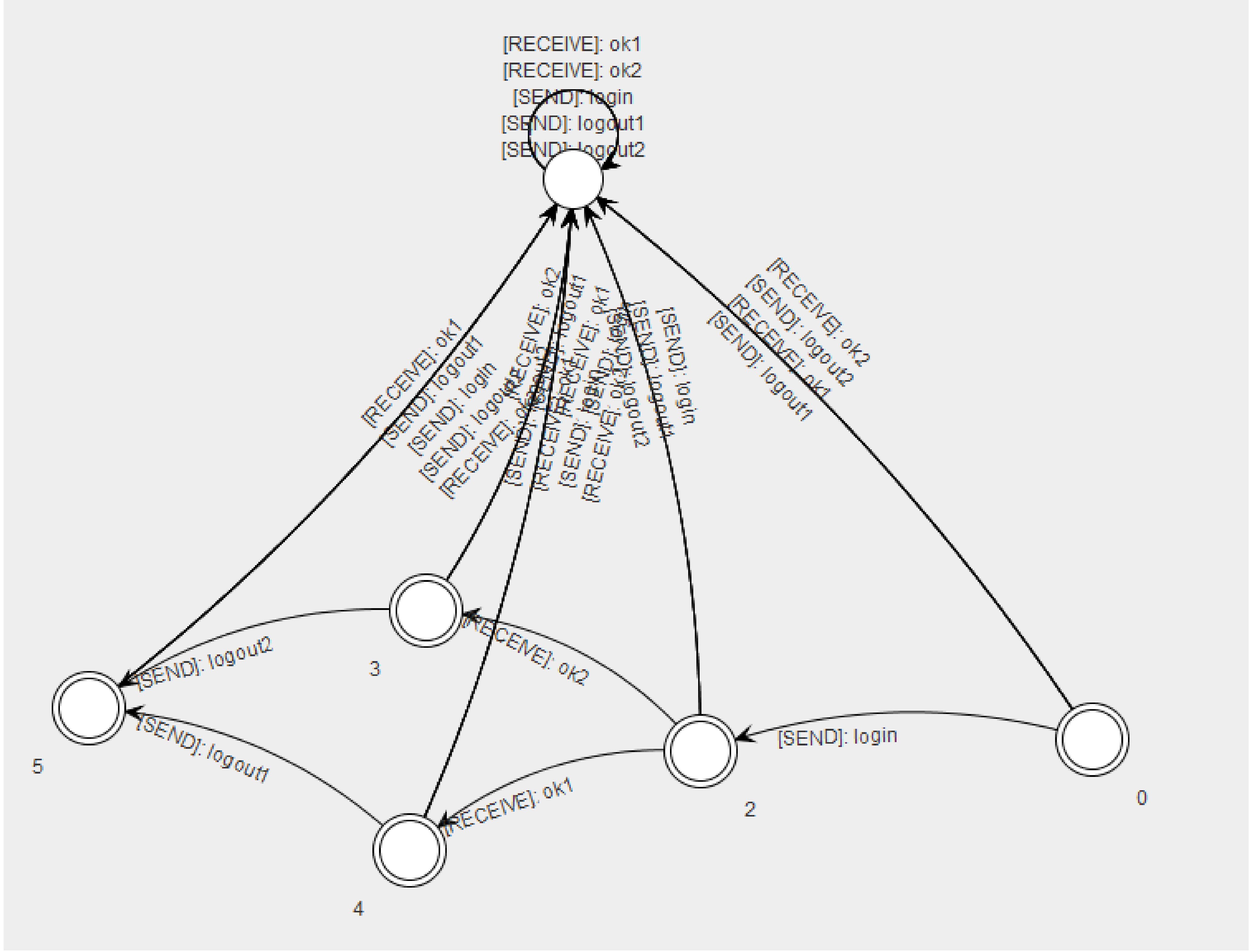
No active protocol peer

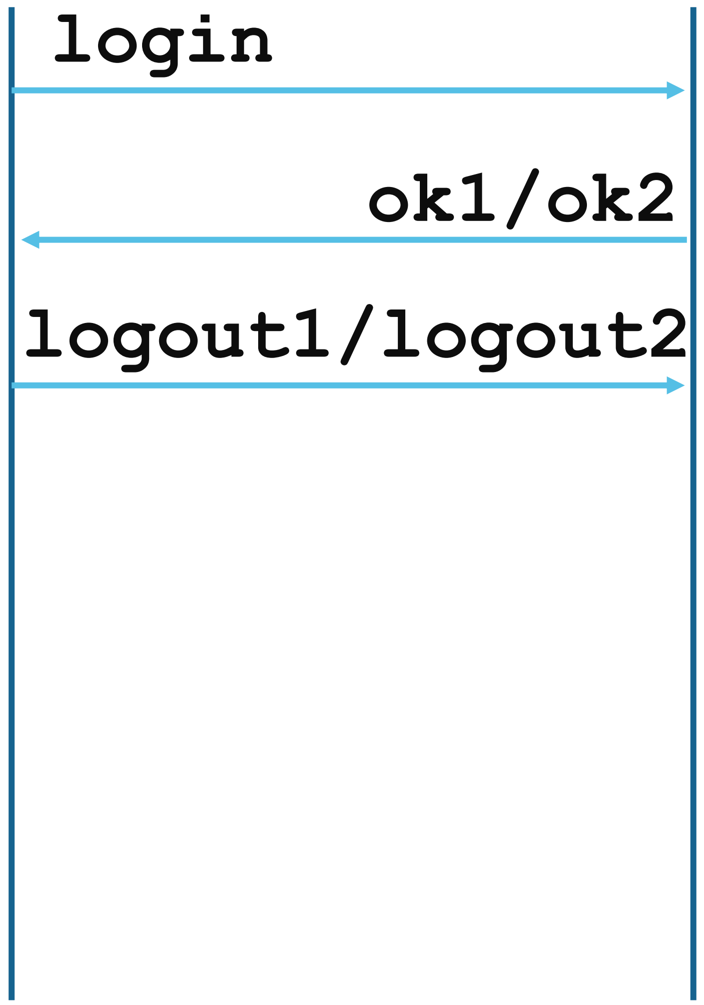
No source code



PISE is action, Examples and Demo

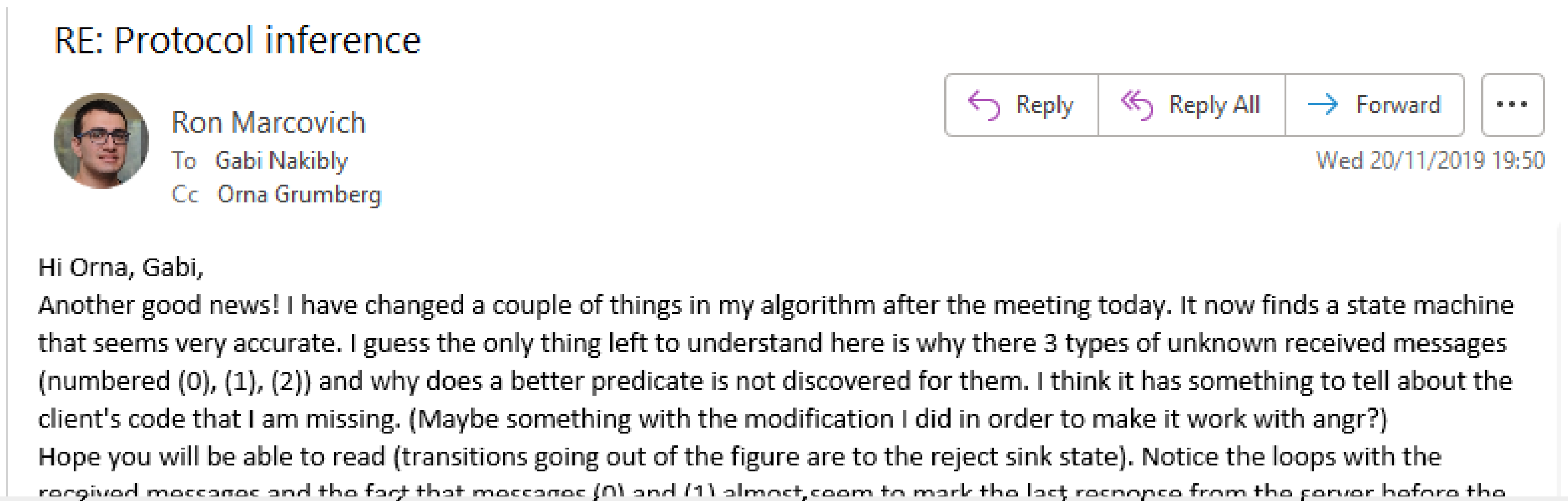
y example


[RECEIVE]: ok1
[RECEIVE]: ok2
[SEND]: login
[SEND]: logout1
[SEND]: logout2
[RECEIVE]: ok1
[SEND]: logout1
[SEND]: login
[SEND]: logout2
[RECEIVE]: ok2
[RECEIVE]: ok2
[SEND]: logout2
[RECEIVE]: ok1
[SEND]: logout1
[SEND]: login
[SEND]: logout1
[SEND]: logout2
[SEND]: logout2
[RECEIVE]: ok2
[SEND]: logout1
[RECEIVE]: ok1
[SEND]: login
[SEND]: logout2
[SEND]: logout1
[RECEIVE]: ok2
[RECEIVE]: ok1
[SEND]: login
3
5
4
2
0


login
ok1/ok2
logout1/logout2

We wanted to get to the real thing

SMTP client

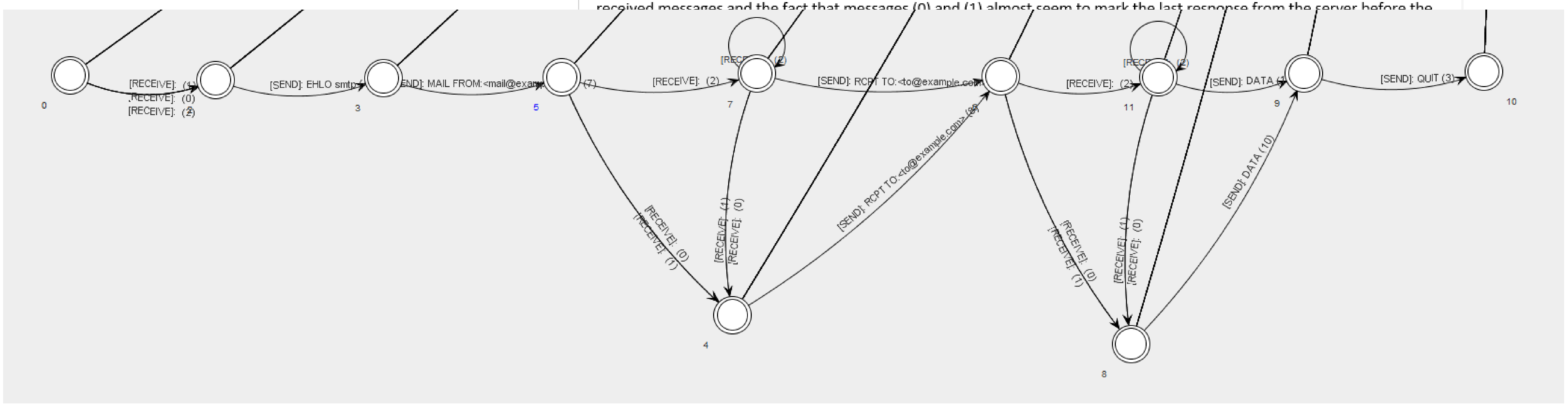


RE: Protocol inference

Ron Marcovich
To Gabi Nakibly
Cc Orna Grumberg

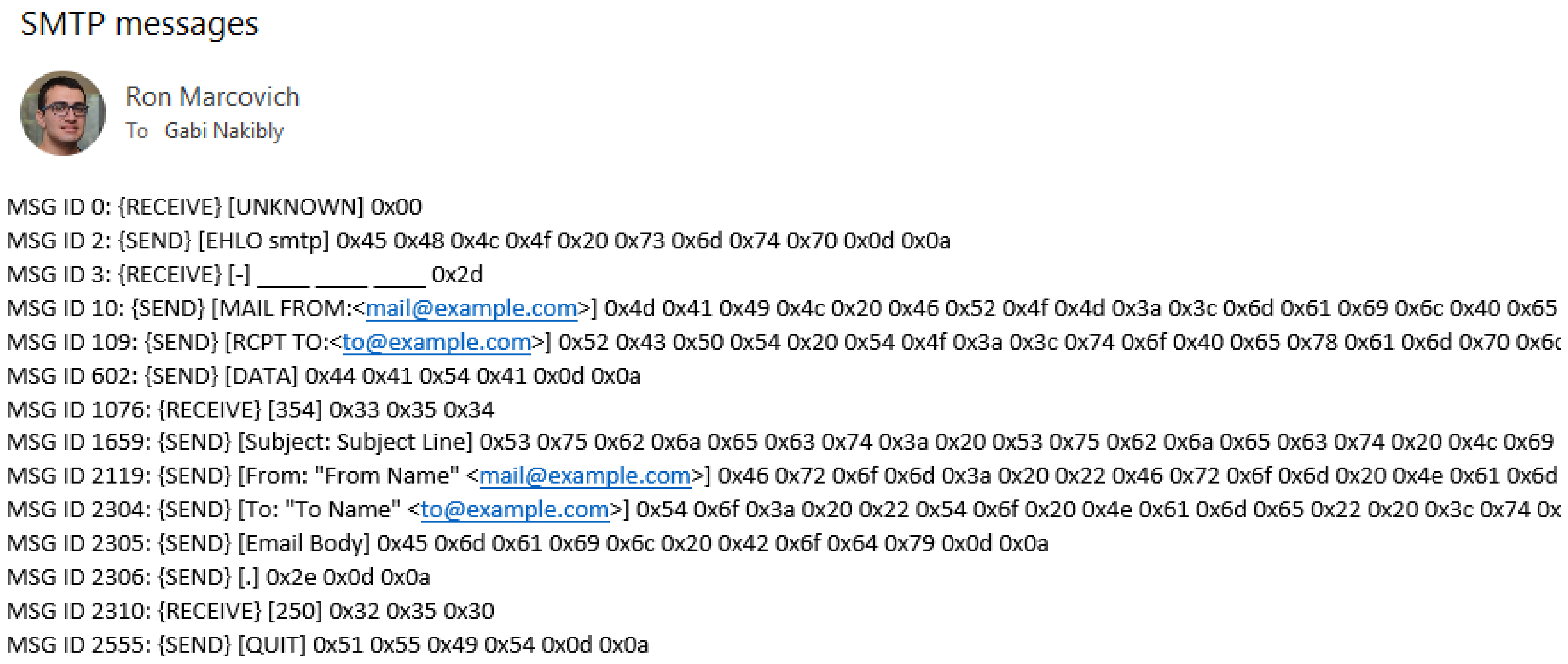
Wed 20/11/2019 19:50

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. I guess the only thing left to understand here is why there 3 types of unknown received messages (numbered (0), (1), (2)) and why does a better predicate is not discovered for them. I think it has something to tell about the client's code that I am missing. (Maybe something with the modification I did in order to make it work with angr?)
Hope you will be able to read (transitions going out of the figure are to the reject sink state). Notice the loops with the received messages and the fact that messages (0) and (1) almost seem to mark the last response from the server before the



We wanted to get to the real thing

Messages' formats are extracted as well!



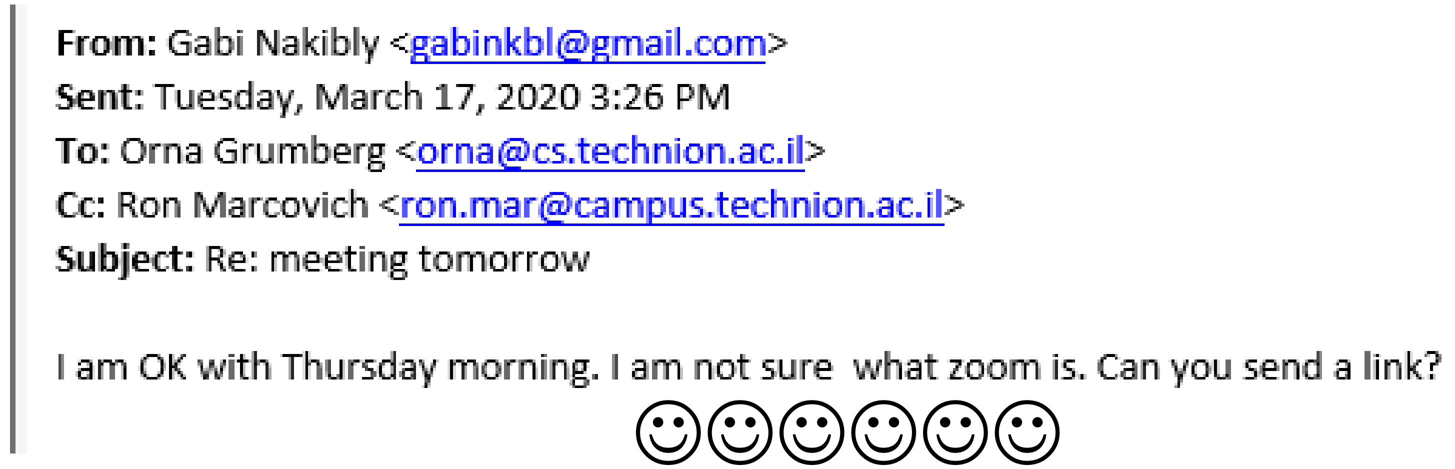
SMTP messages

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 0x6
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?

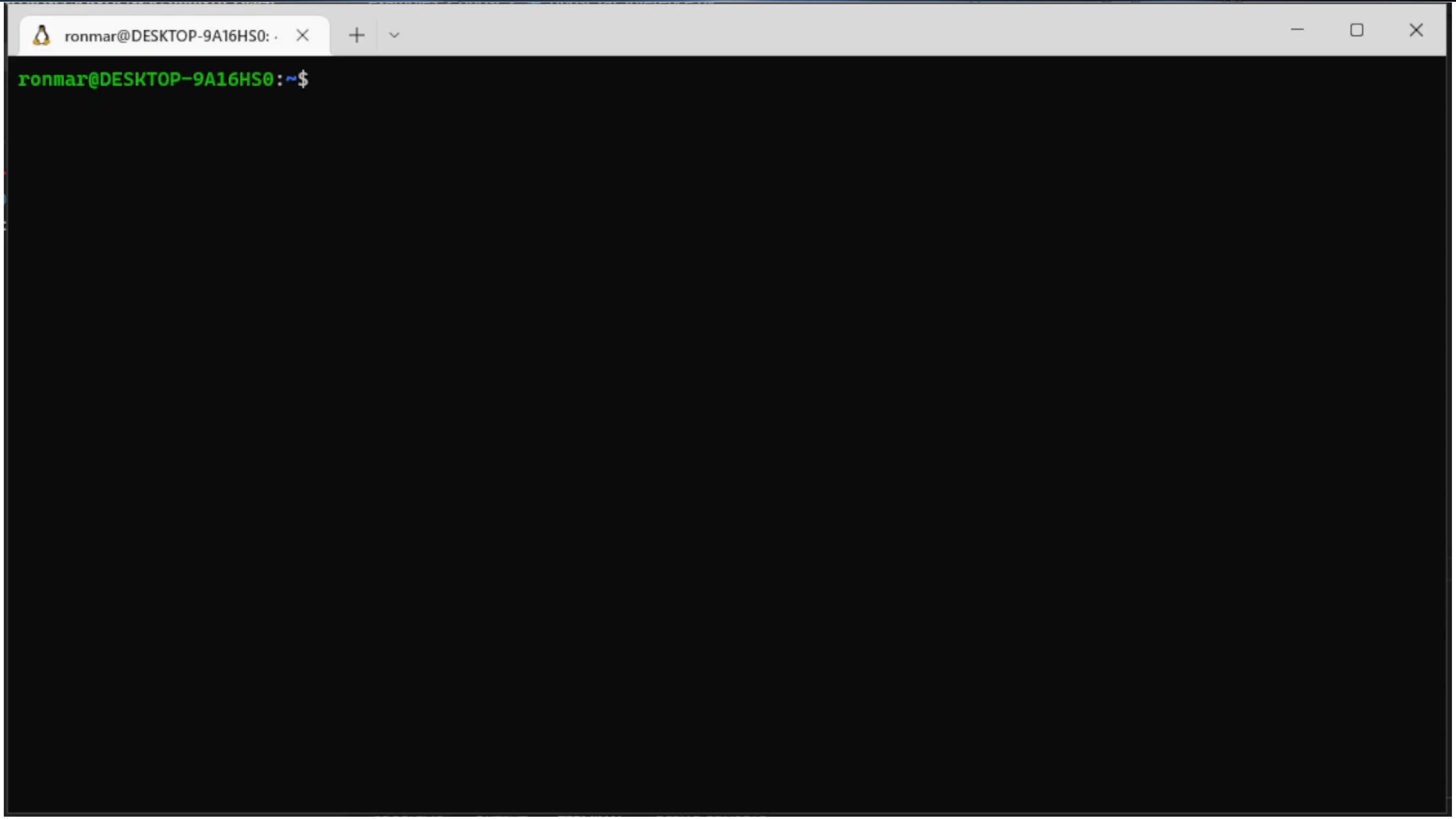


From: Gabi Nakibly <gabinkbl@gmail.com>
Sent: Tuesday, March 17, 2020 3:26 PM
To: Orna Grumberg <orna@cs.technion.ac.il>
Cc: Ron Marcovich <ron.mar@campus.technion.ac.il>
Subject: Re: meeting tomorrow

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


ronmar@DESKTOP-9A16HS0: ·

```
ronmar@DESKTOP-9A16HS0:~$
```

Under the Hood

black hat USA 2022

Overview

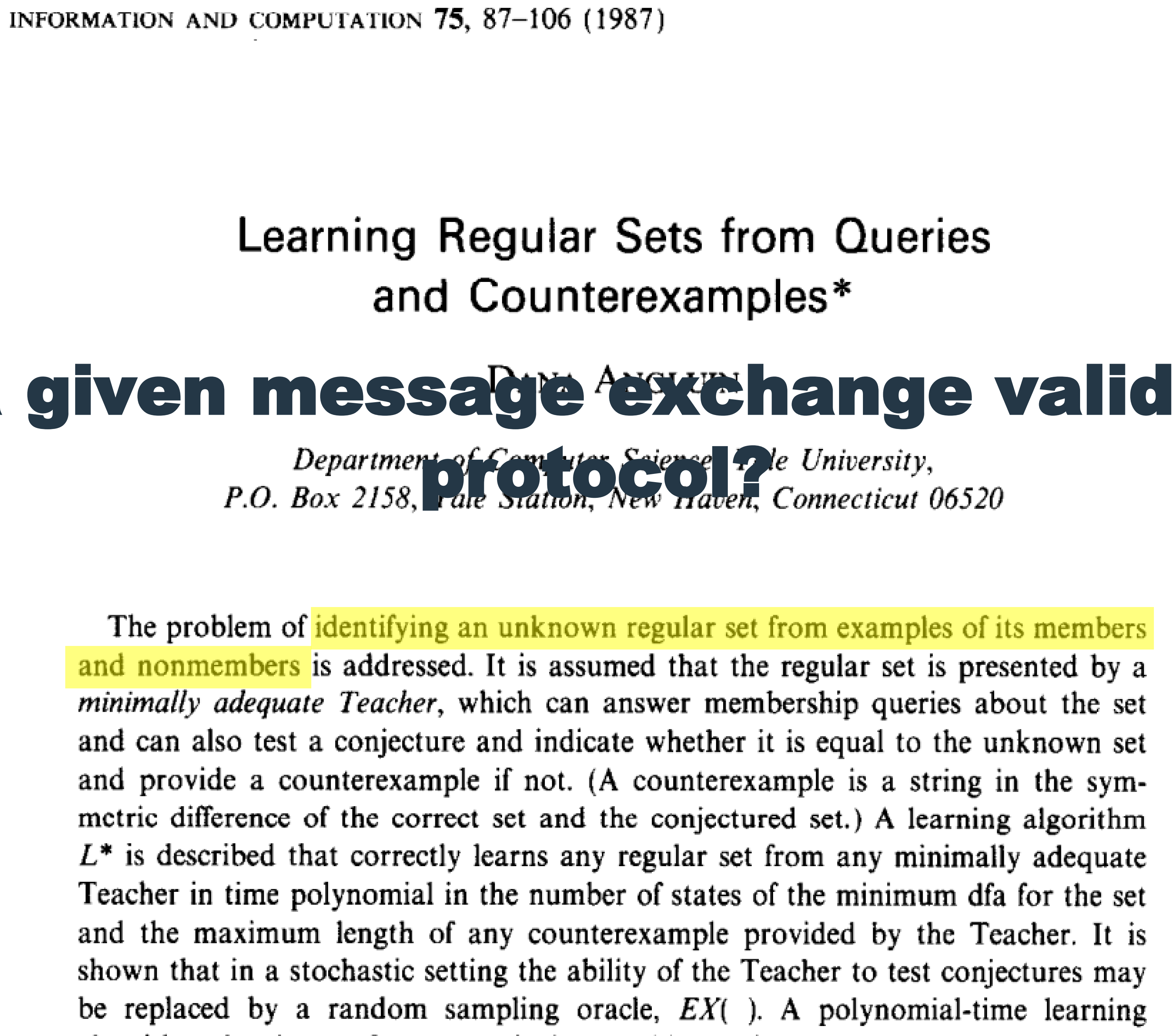
Q

L* Algorithm

A

Symbolic Execution

L* Algorithm (Automata Learning)



INFORMATION AND COMPUTATION 75, 87–106 (1987)

Learning Regular Sets from Queries and Counterexamples*

DANA ANGLUIN

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

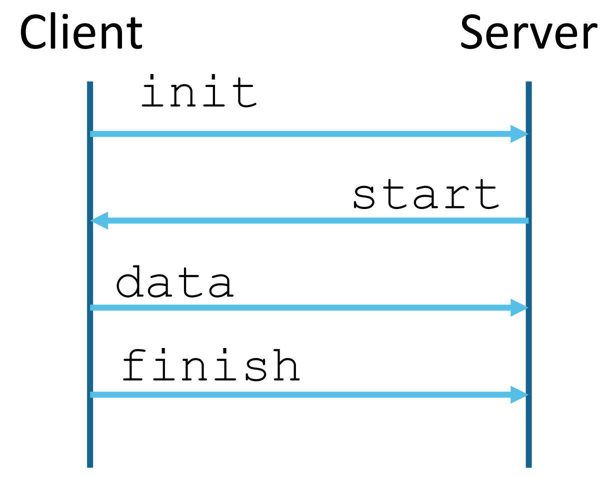
The problem of identifying an unknown regular set from examples of its members and nonmembers is addressed. It is assumed that the regular set is presented by a *minimally adequate Teacher*, which can answer membership queries about the set and can also test a conjecture and indicate whether it is equal 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

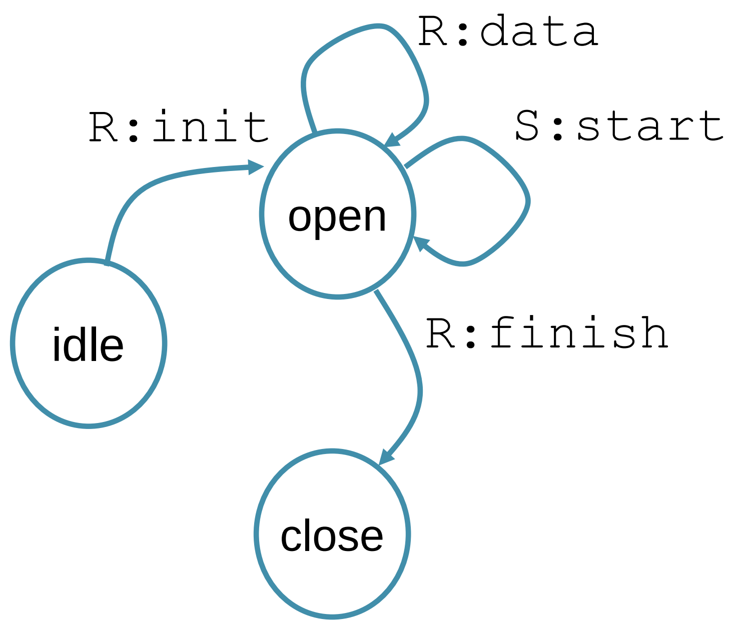
Q: Is a given message exchange valid by the protocol?

Using L* Algorithm

{R:init, S:start} ✓

{R:init, R:init} ✗



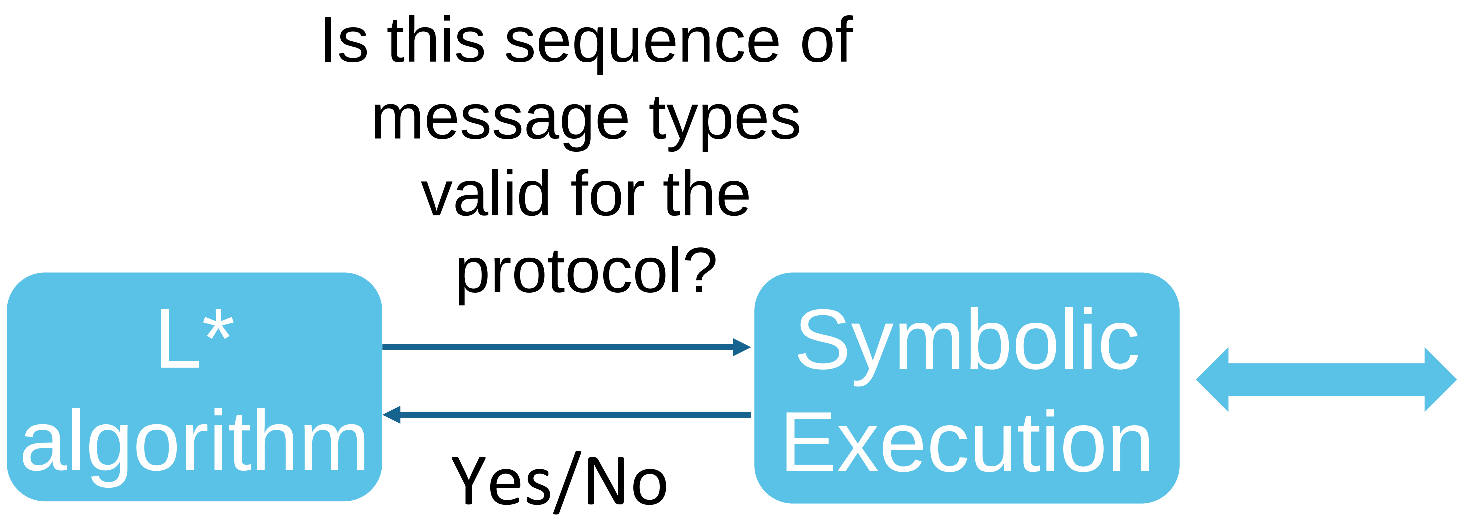


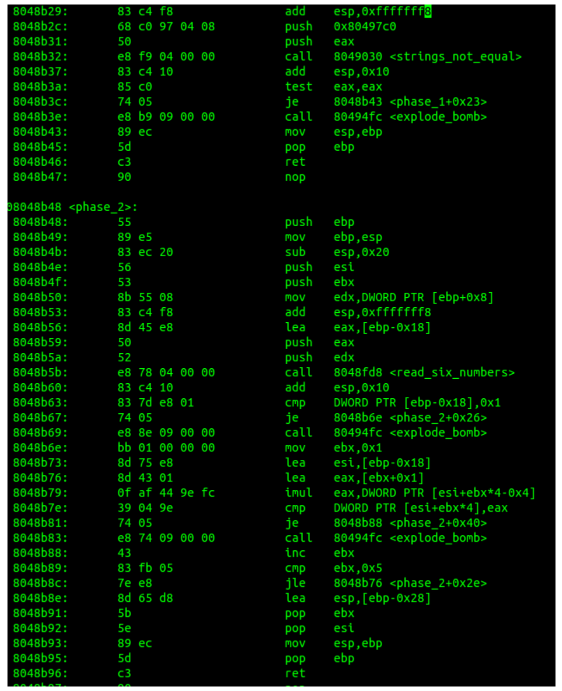
But there is a problem!

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

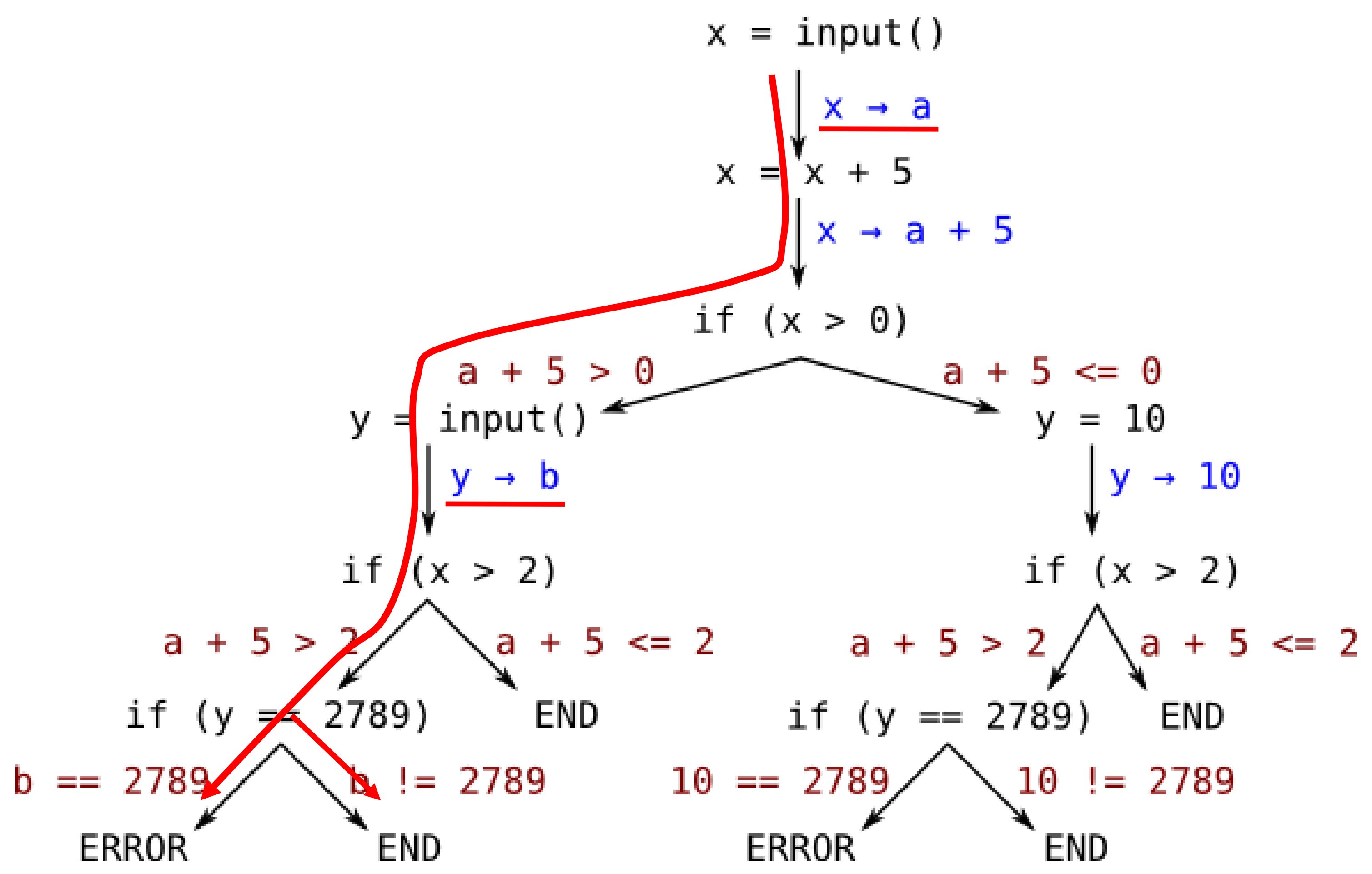




```
 8048b29:       83 c4 f8                add    esp,0xfffffff8
 8048b2c:       68 c0 97 04 08          push   0x80497c0
 8048b31:       50                      push   eax
 8048b32:       e8 f9 04 00 00          call   8049030 <strings_not_equal>
 8048b37:       83 c4 10                add    esp,0x10
 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 f8                add    esp,0xfffffff8
 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
```

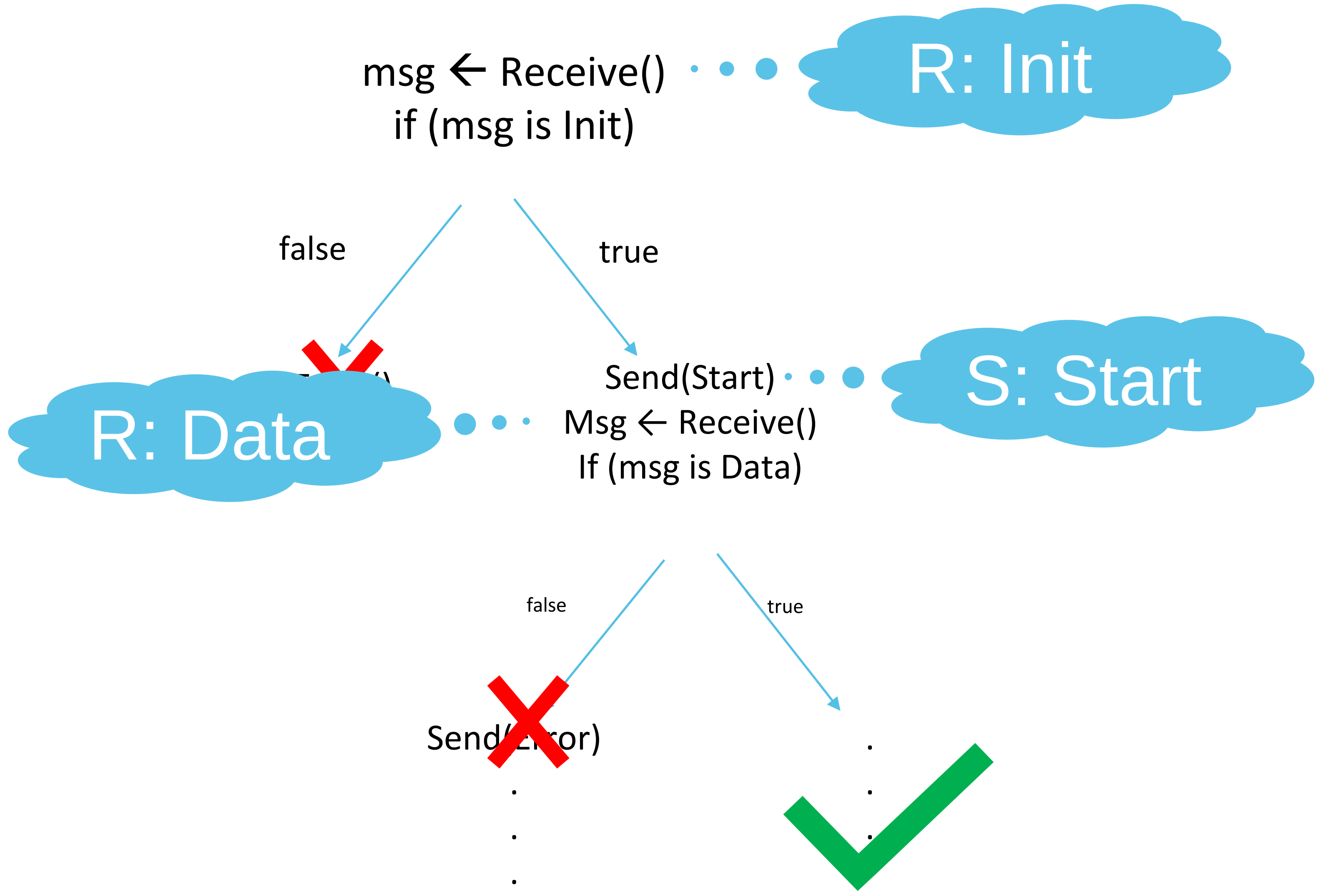
Symbolic Execution

a > 3 ,b = 2789



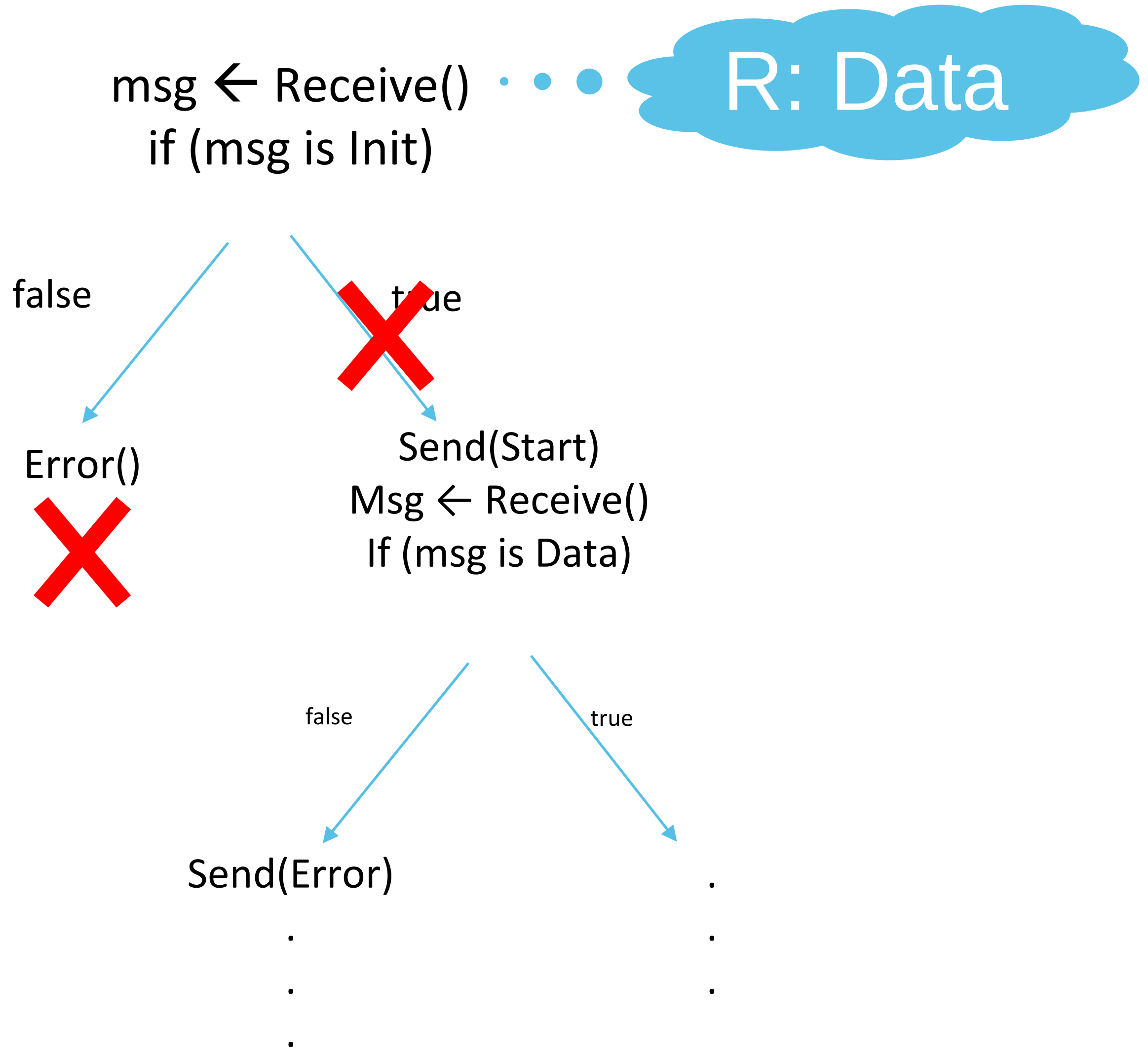
Answering Membership queries

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



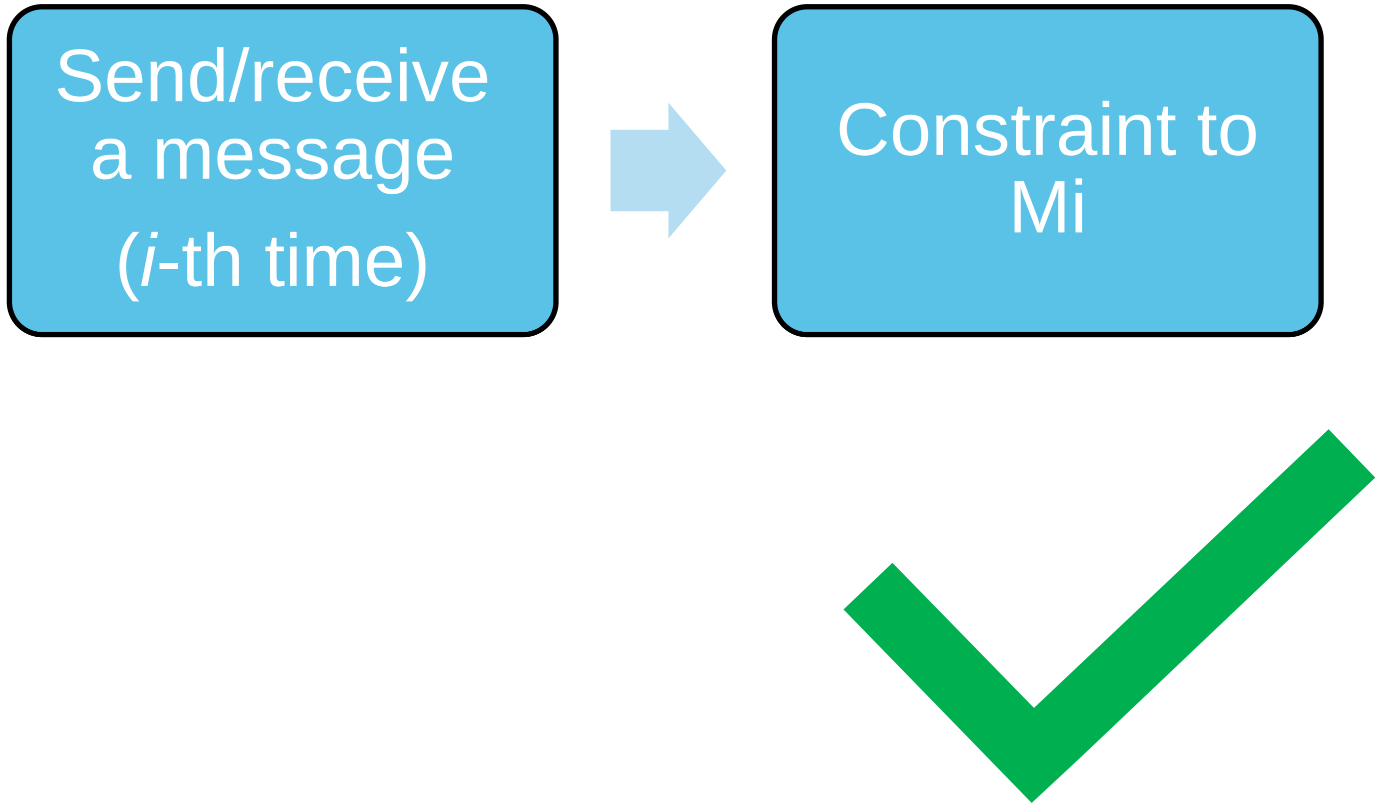
Answering Membership queries

Is {R: Data} valid for the protocol?

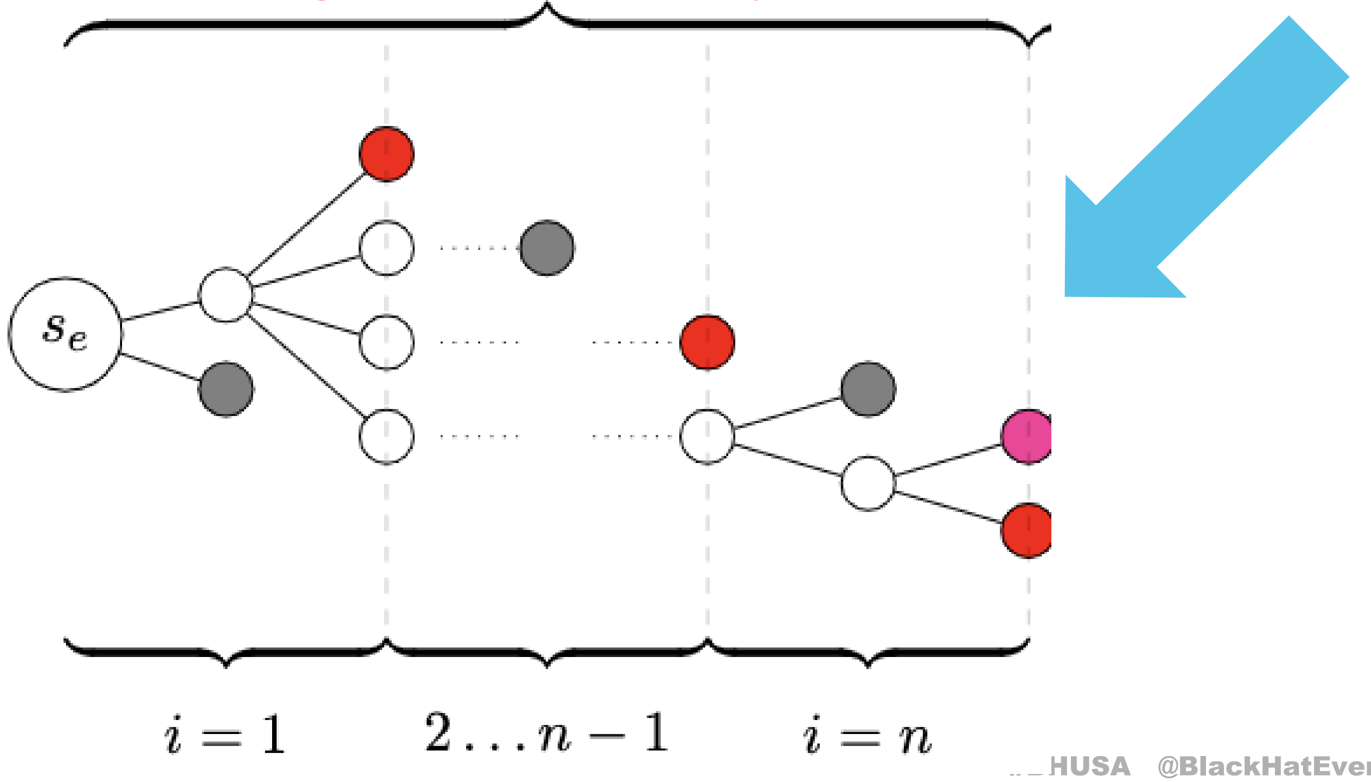


Answering Membership queries

- Let M = {M1, .., Mn}
- Whenever send/receive procedures are called for the *i*-th time, append a predicate that identifies Mi, as constraint
- After n {send/receive}s, if there are feasible executions – then the sequence M is valid

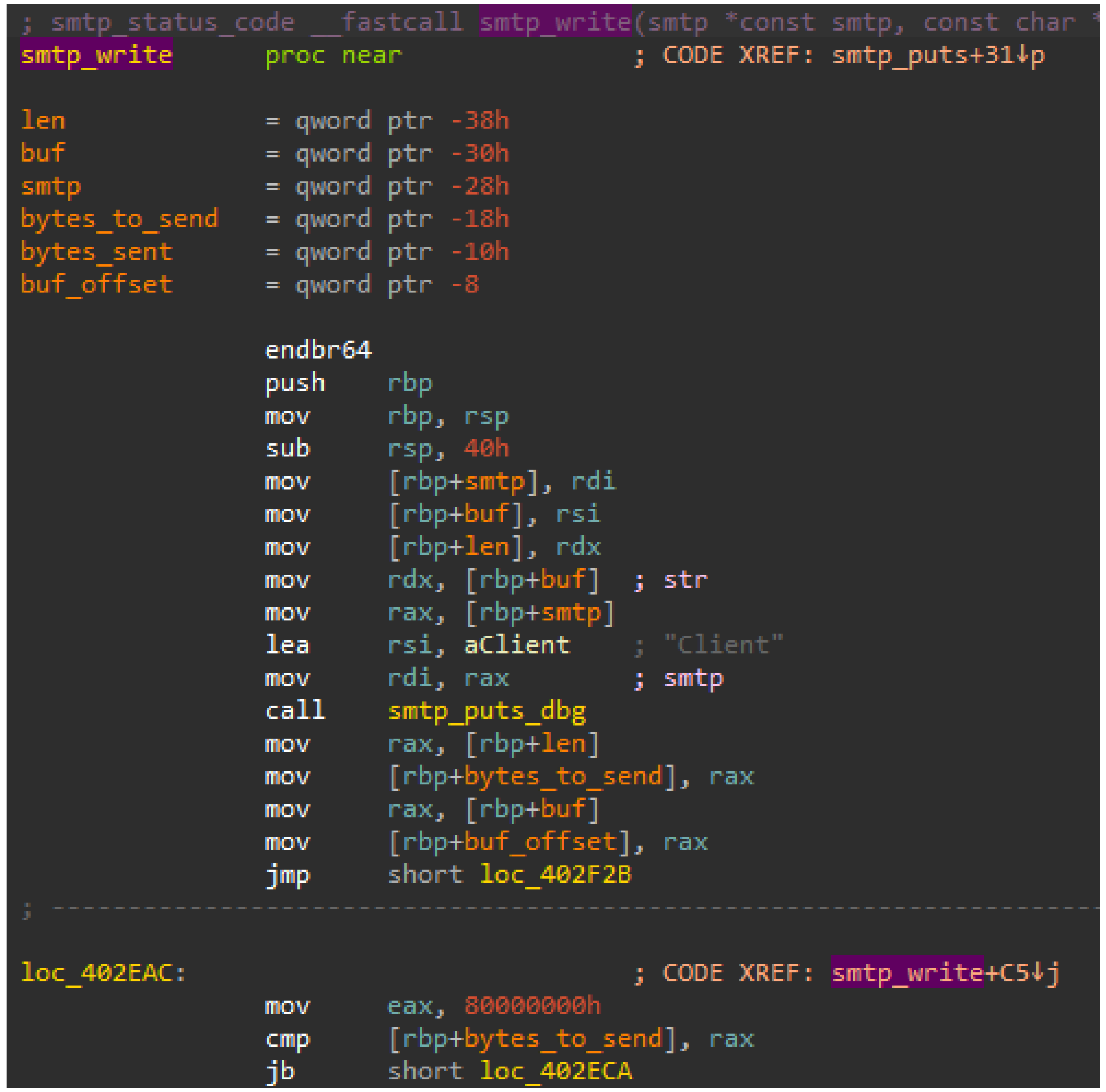


Gray: valid state
Red: invalid for the sequence
Magenta: valid for the sequence

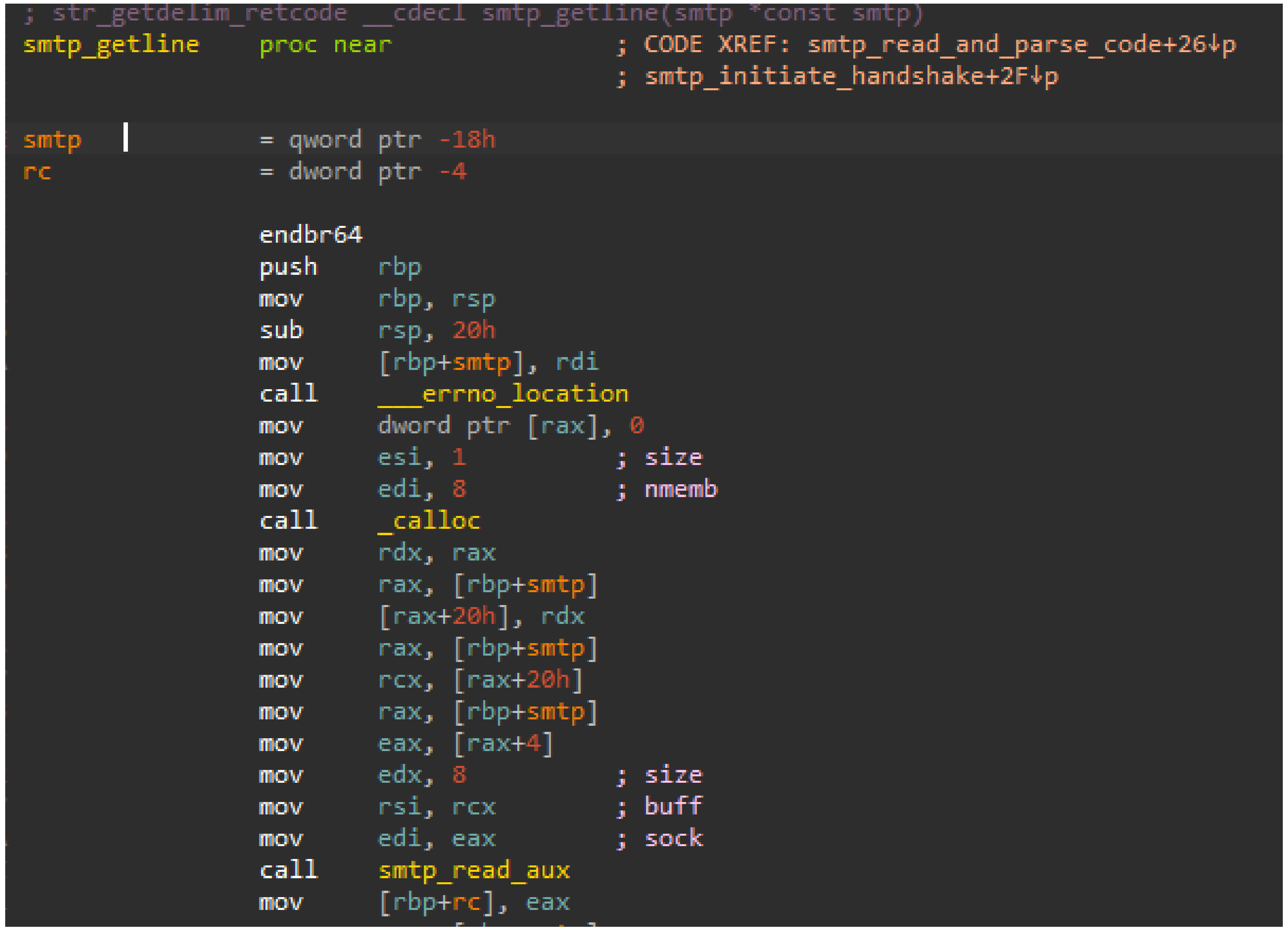


How to identify a send or receive?

- 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
```

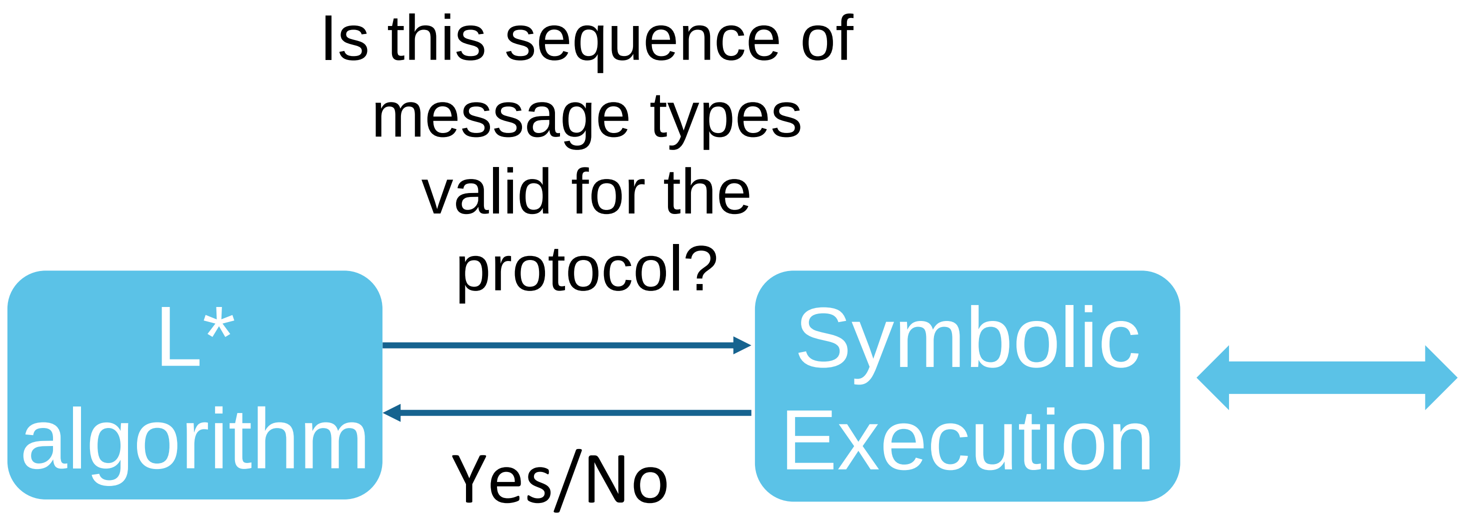


```
; str_getdelim_retcode __cdecl smtp_getline(smtp *const smtp)
smtp_getline    proc near               ; CODE XREF: smtp_read_and_parse_code+26↓p
                                        ; smtp_initiate_handshake+2F↓p

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

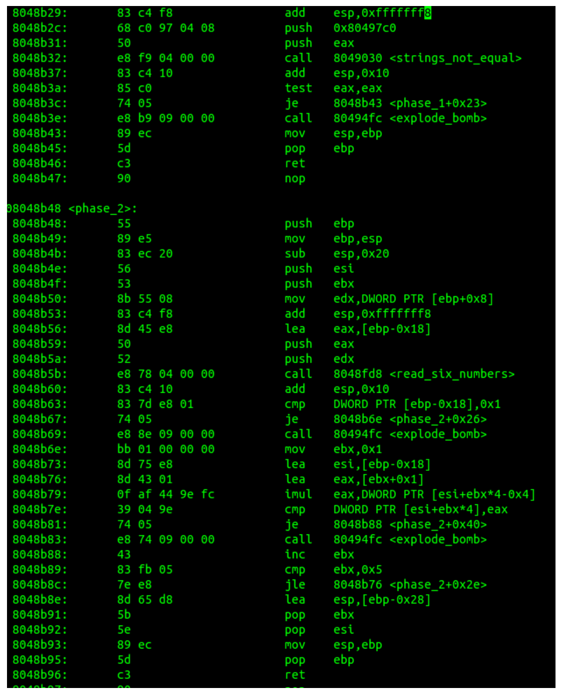
As said, we do not know is advance the protocol’s message types.

We utilize update membership queries to discover it little by little.



Extend L* to handle the new message type

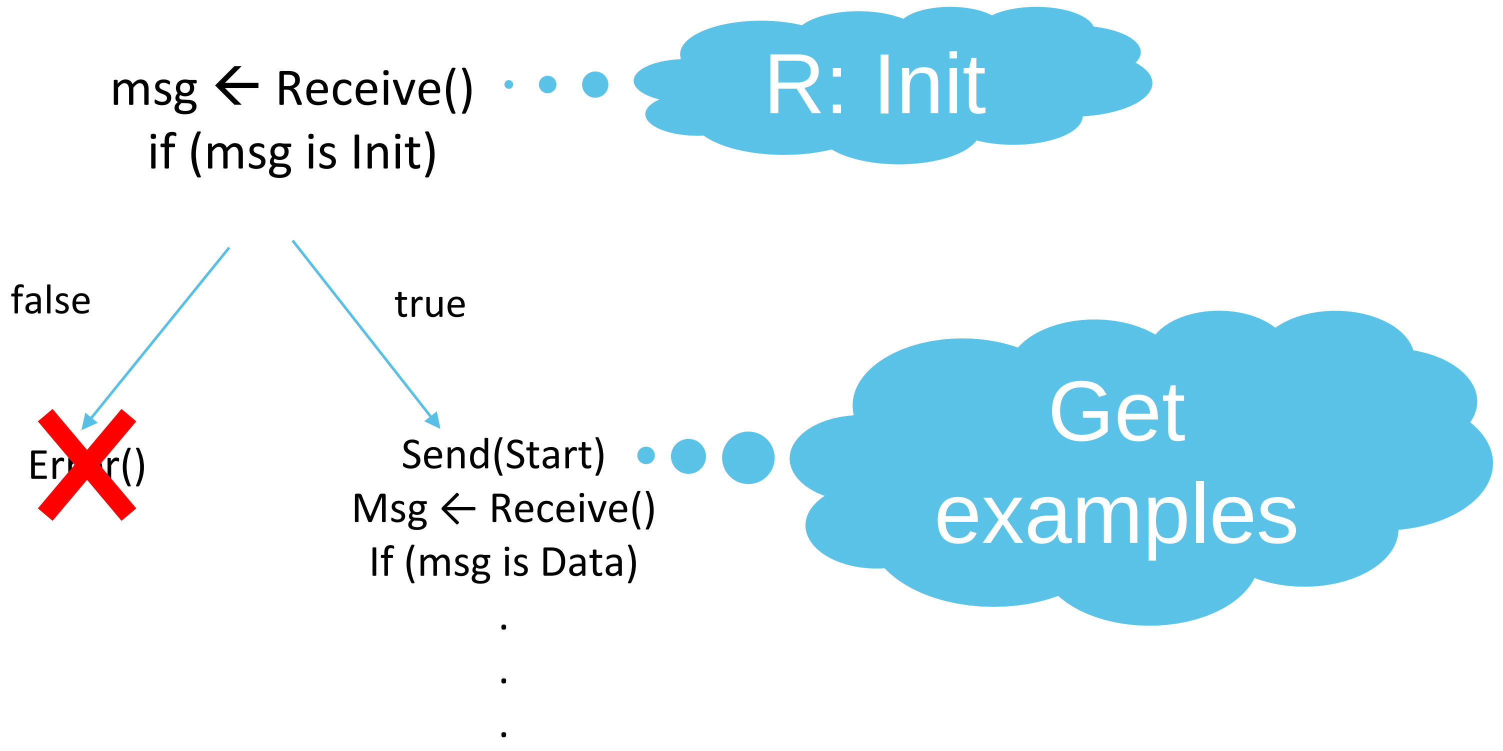
If yes, here is a message type that can follow the sequence.



```
 8048b29:       83 c4 f8                add    esp,0xfffffff8
 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 f8                add    esp,0xfffffff8
 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
```

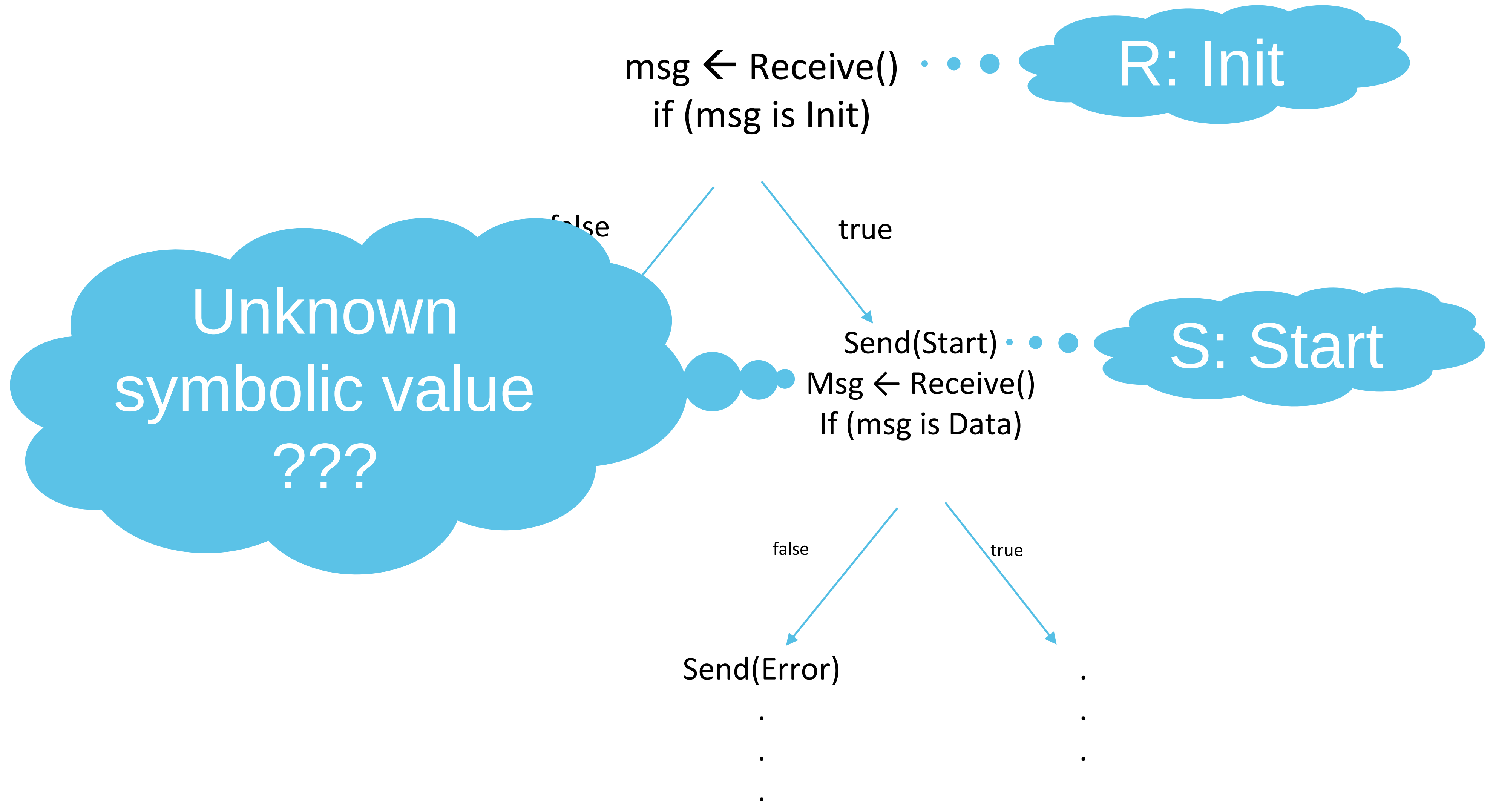
Probing for following message types

What message types can follow {R: Init}?

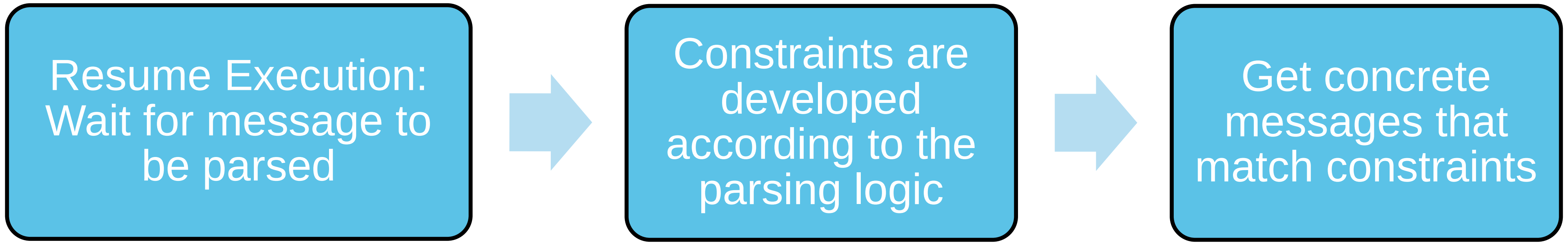


Probing for following message types

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



Probing for following message types


Resume Execution: Wait for message to be parsed
Constraints are developed according to the parsing logic
Get concrete messages that match constraints

```
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

→

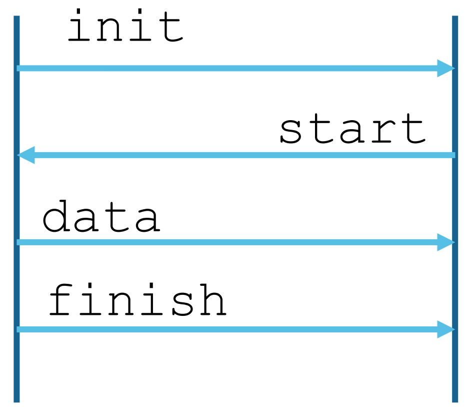
```
RCPT%20TO:*%0D%0A
```

Tying it all together

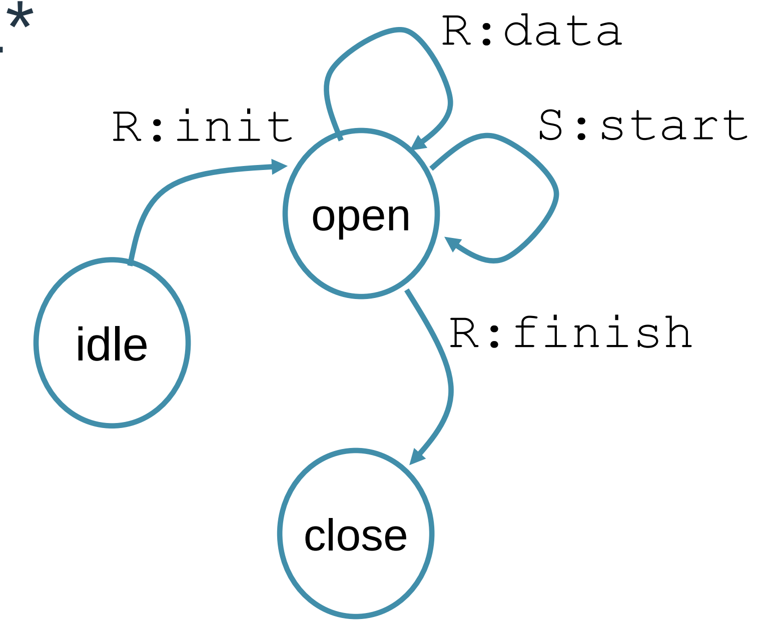
Use symbolic execution to learn if a given sequence of messages is valid and if so, what are the next messages the program expects to receive or is about to send.

`{R: init, S: start}` – A valid sequence. A next message is `data`.

`{R: data}` – Not a valid sequence.



Based on this information use a well-known algorithm (called L* algorithm) to reconstruct the protocol's state machine.

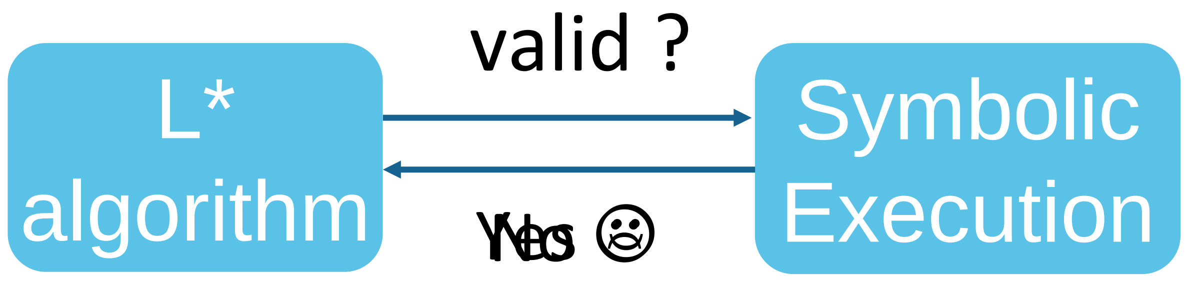


An illustrative example

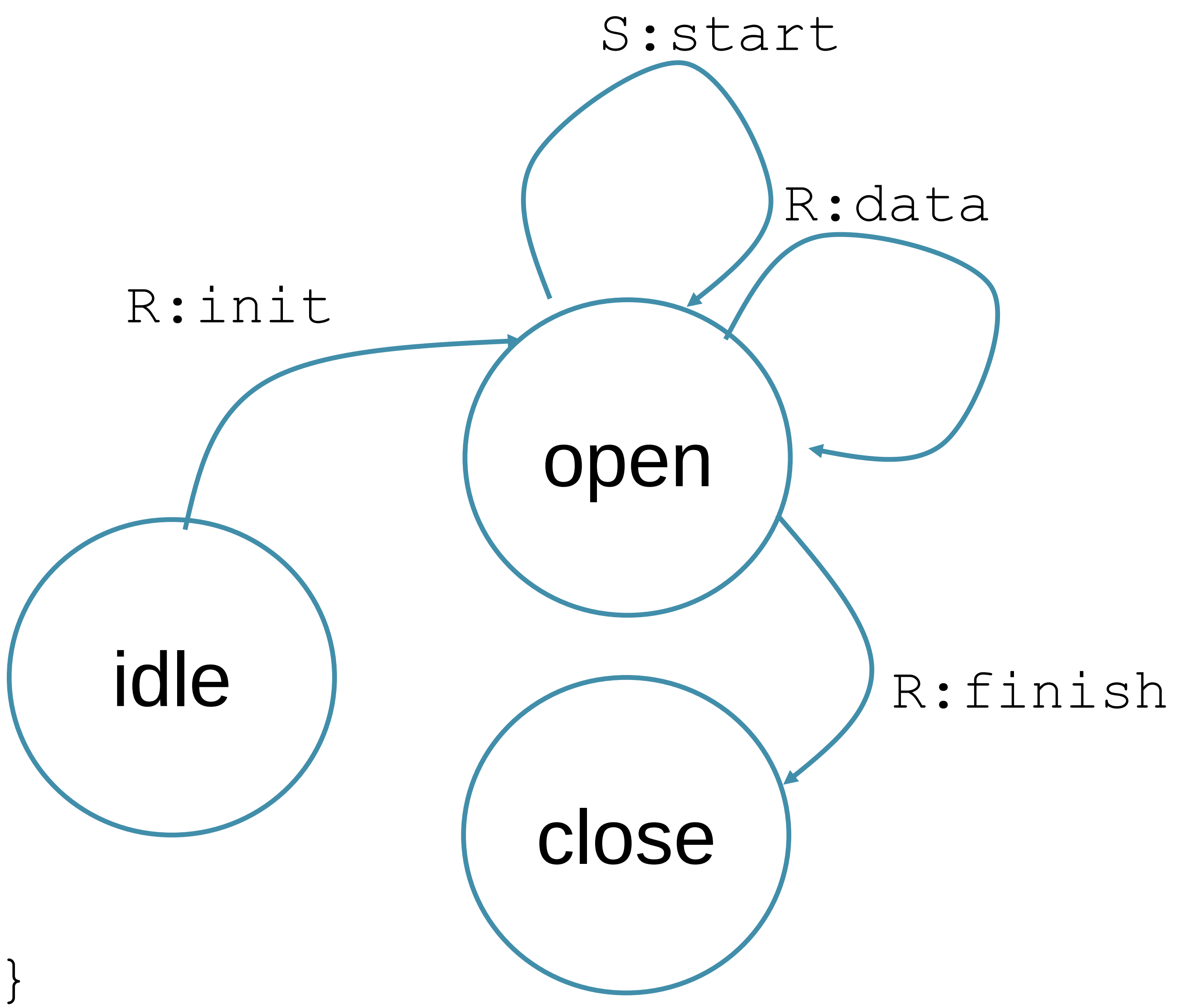
R:init
S:start
R: data
R: finish

Message Types

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



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



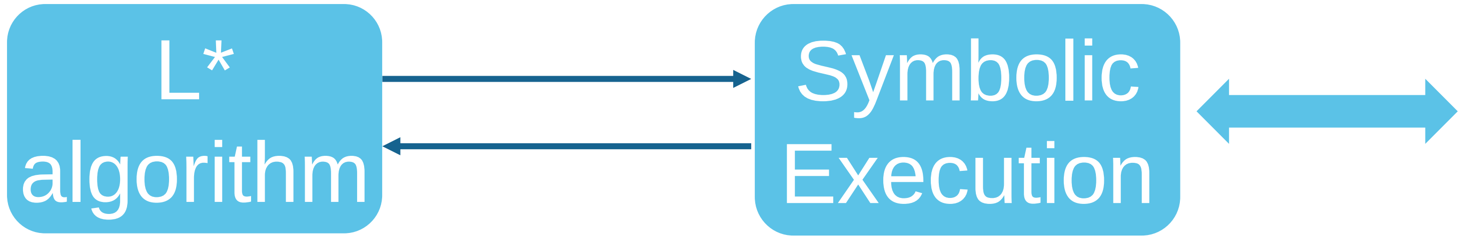
Caveats

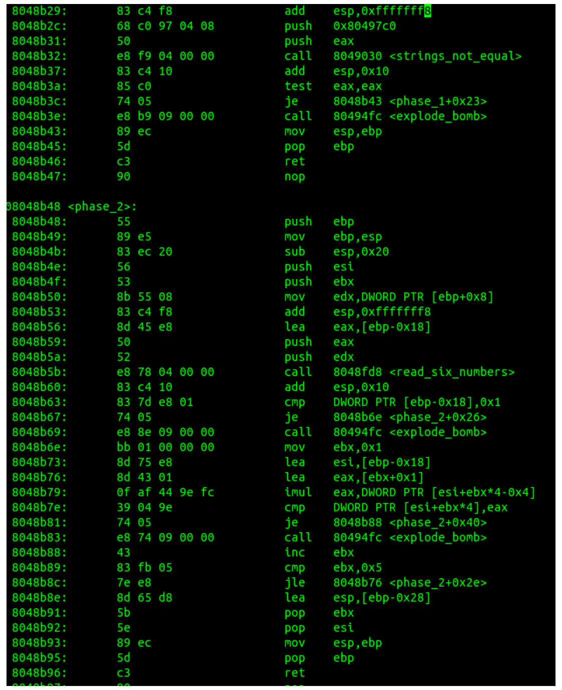
PISE’s interacts with the binary using symbolic execution.

This means that PISE is as good or as bad as the symbolic tool used.

Currently, PISE supports only Angr.

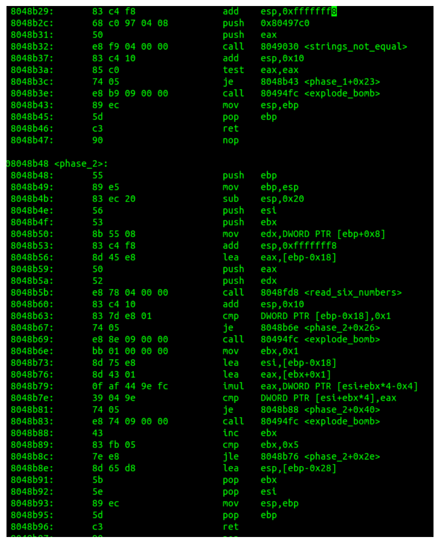
- Trouble supporting threads.
- Does not fully support windows API



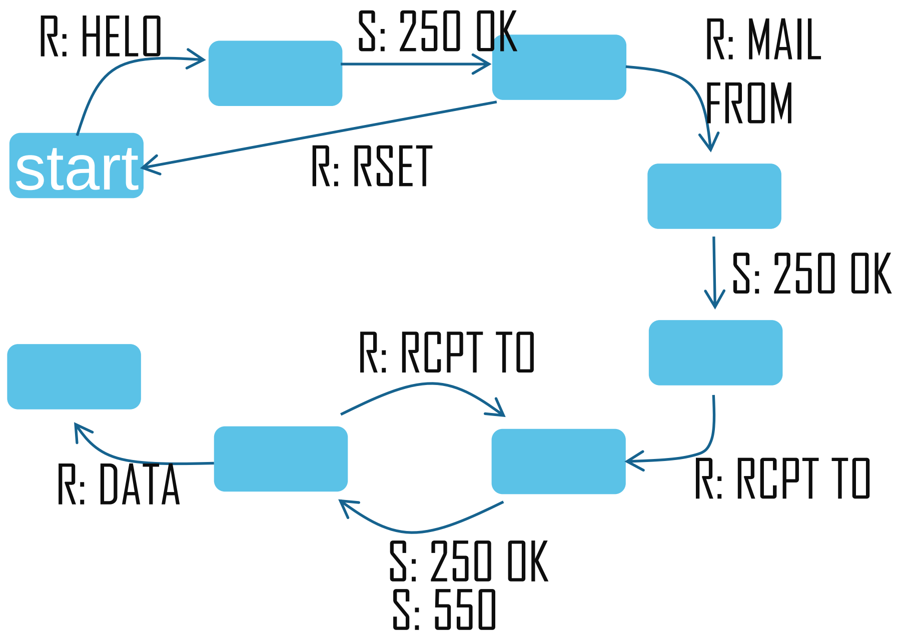


```
8048b29:       83 c4 f8                add    esp,0xfffffff8
8048b2c:       68 c0 97 04 08          push   0x80497c0
8048b31:       50                      push   eax
8048b32:       e8 f9 04 00 00          call   8049030 <strings_not_equal>
8048b37:       83 c4 10                add    esp,0x10
8048b3a:       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 f8                add    esp,0xfffffff8
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
```

Summary



```
 8048b29:       83 c4 f8                add    esp,0xfffffff8
 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 f8                add    esp,0xfffffff8
 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
```



Q

L* Algorithm

A

Symbolic Execution

https://github.com/ron4548/PISEServer

black hat USA 2022

Questions

https://github.com/ron4548/PISEServer