UFMG

DEPARTMENT OF COMPUTER SCIENCE
FEDERAL UNIVERSITY OF MINAS GERAIS, BRAZIL

SIoT – Securing the Internet of Things through Distributed System Analysis

Fernando A. Teixeira
UFMG, Brazil
teixeira@dcc.ufmg.br

Gustavo V. Machado
UFMG, Brazil
gvieira@dcc.ufmg.br

Fernando M. Q. Pereira
UFMG, Brazil
fernando@dcc.ufmg.br

Hao Chi Wong
Intel Corporation, CA
hao-chi.wong@intel.com

José M. S. Nogueira
UFMG, Brazil
jmarcos@dcc.ufmg.br

Leonardo B. Oliveira
UFMG, Brazil
leob@dcc.ufmg.br

Agenda

- Introduction
- Goal
- Solution
- Results
- Conclusion

Agenda

- Introduction
- Goal
- Solution
- Results
- Conclusion

Agenda

- Introduction
  - IoT
  - C language
  - Buffer Overflow
- Goal
- Solution
- Results
- Conclusion

Agenda

- Introduction
  - IoT
  - C language
  - Buffer Overflow
- Goal
- Solution
- Results
- Conclusion

IoT

- Capabilities
  - It’s made up with constrained devices
- Computing Paradigm
  - A distributed system and usually exchange a large number of messages
- Programming language
  - Apps are often written in C, which is inherently unsafe

IoT

- Capabilities
  - It’s made up with constrained devices
- Computing Paradigm
  - A distributed system and usually exchange a large number of messages
- Programming language
  - Apps are often written in C, which is inherently unsafe

C is unsafe because it does not check array-bounds

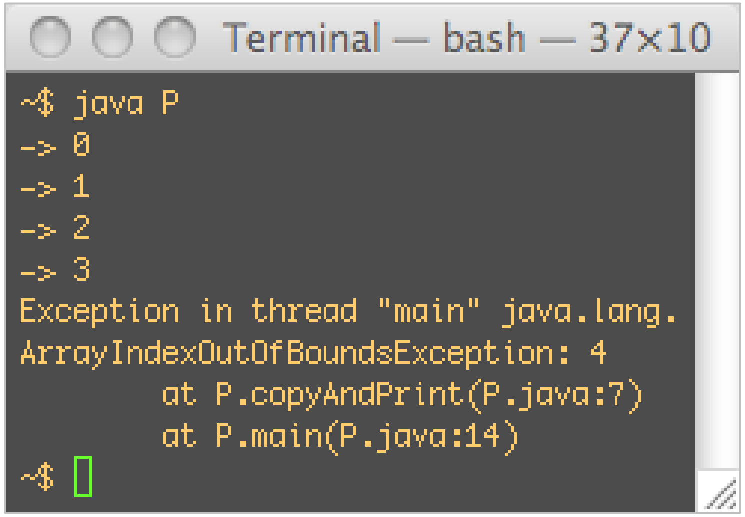
Agenda

- Introduction
  - IoT
  - C language
  - Buffer Overflow
- Goal
- Solution
- Results
- Conclusion

Java

Q: What happens when we run this Java program?

```
public class P {
  final static int SIZE = 4;

  static void copyAndPrint(byte[] v) {
    byte[] buf = new byte[SIZE];
    for (int i = 0; i < v.length; i++) {
      buf[i] = v[i];
      System.out.println("-> " + buf[i]);
    }
  }

  public static void main(String args[]) {
    byte[] v = {0, 1, 2, 3, 4};
    copyAndPrint(v);
  }
}
```

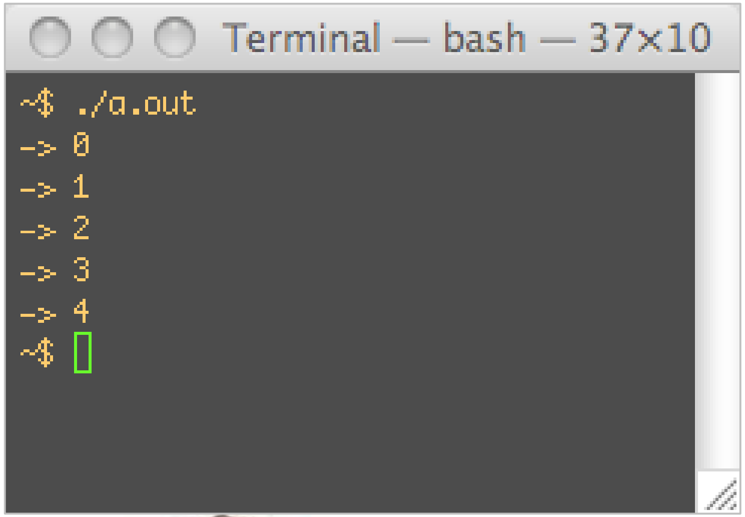


```
~$ java P
-> 0
-> 1
-> 2
-> 3
Exception in thread "main" java.lang.
ArrayIndexOutOfBoundsException: 4
        at P.copyAndPrint(P.java:7)
        at P.main(P.java:14)
~$
```

C language

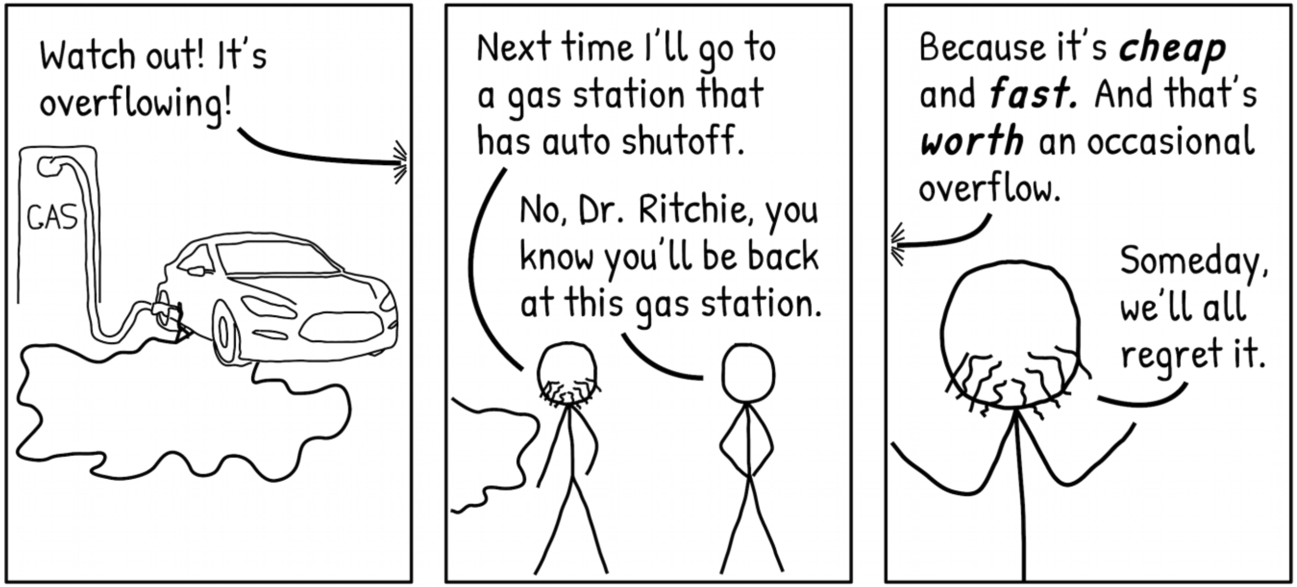
Q: What about this C program?

```
#include <stdio.h>

#define SIZE 4

void copyAndPrint(char v[], int n) {
  char buf[SIZE];
  int i;
  for (i = 0; i < n; i++) {
    buf[i] = v[i];
    printf("-> %d\n", buf[i]);
  }
}

int main() {
  char v[] = {0, 1, 2, 3, 4};
  copyAndPrint(v, SIZE + 1);
}
```



```
~$ ./a.out
-> 0
-> 1
-> 2
-> 3
-> 4
~$ 
```

Q: Why is that?


Watch out! It's overflowing!
GAS
Next time I'll go to a gas station that has auto shutoff.
No, Dr. Ritchie, you know you'll be back at this gas station.
Because it's ***cheap*** and ***fast.*** And that's ***worth*** an occasional overflow.
Someday, we'll all regret it.

Buffer Overflow.

icanbarelydraw.com CC BY-NC-ND 3.0

C unsafety: Outcomes

- Morris worm
  - Buffer over-write that compromised around 10% of computers connected to the Internet back in 1988
- Heartbleed
  - Buffer over-read that compromised half a million web servers in 2014
- IoT vulnerability
  - Due the unsafe nature of C, IoT apps are vulnerable to buffer overflow attacks, too

C unsafety: Outcomes

- Morris worm
  - Buffer over-write that compromised around 10% of computers connected to the Internet back in 1988
- Heartbleed
  - Buffer over-read that compromised half a million web servers in 2014
- IoT vulnerability
  - Due the unsafe nature of C, IoT apps are vulnerable to buffer overflow attacks, too

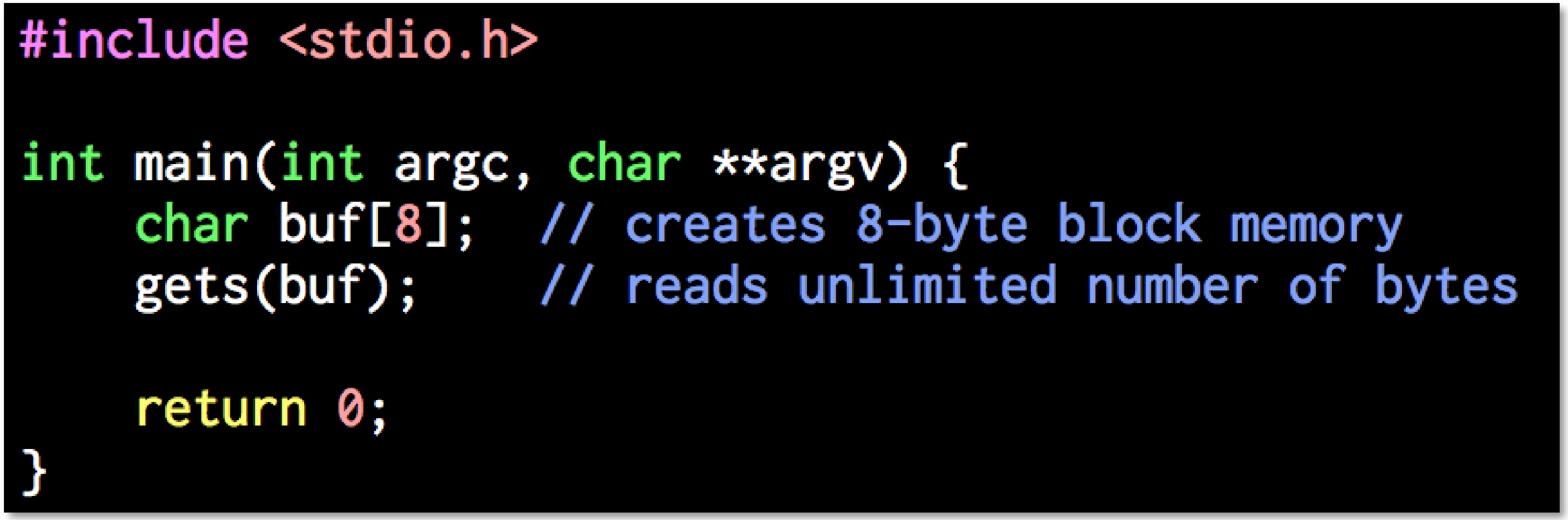
Buffer Overflow?

Agenda

- Introduction
  - IoT
  - C language
  - Buffer Overflow
- Goal
- Solution
- Results
- Conclusion

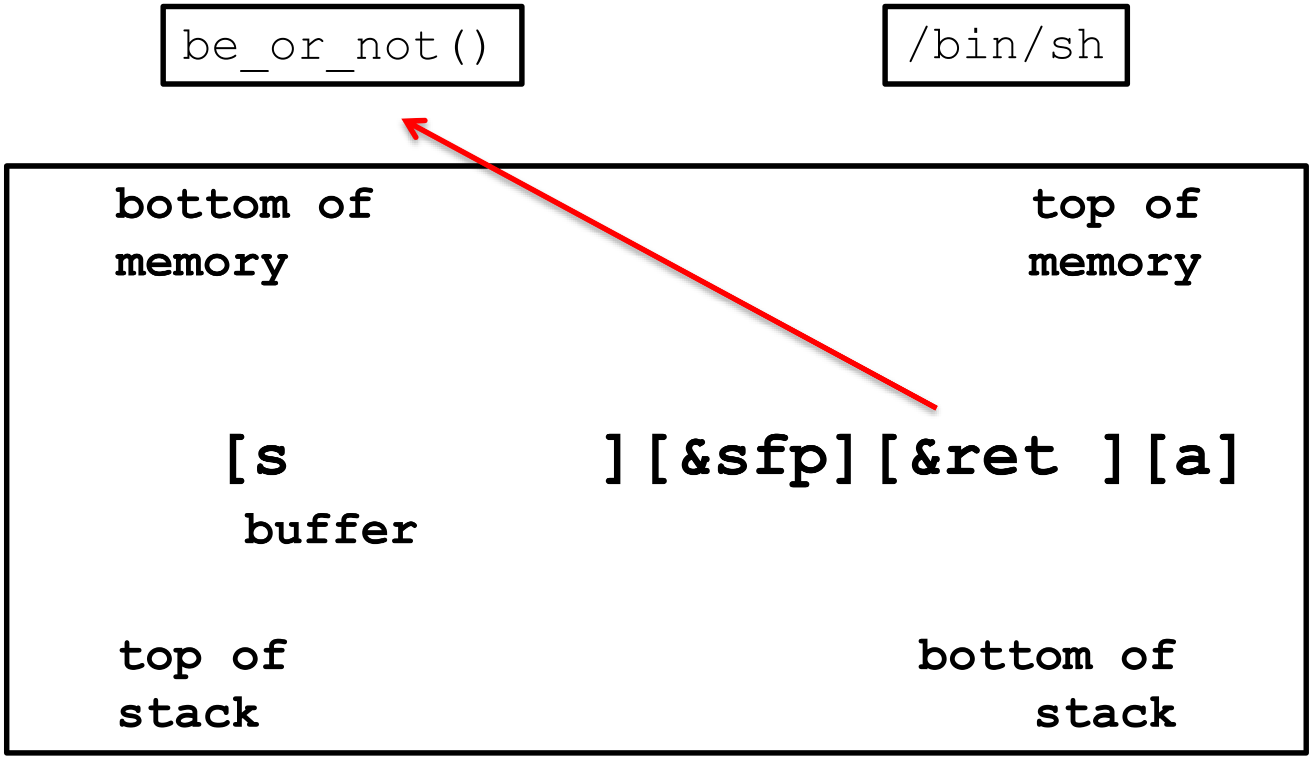
Buffer Overflow

- A buffer overflow happens when the memory space that guides the execution flow is overwritten
- The idea is to manipulate arrays w/o bound checks



```c
#include <stdio.h>

int main(int argc, char **argv) {
    char buf[8];  // creates 8-byte block memory
    gets(buf);    // reads unlimited number of bytes

    return 0;
}
```

BOF (Cont.)



be_or_not()

/bin/sh

bottom of memory

top of memory

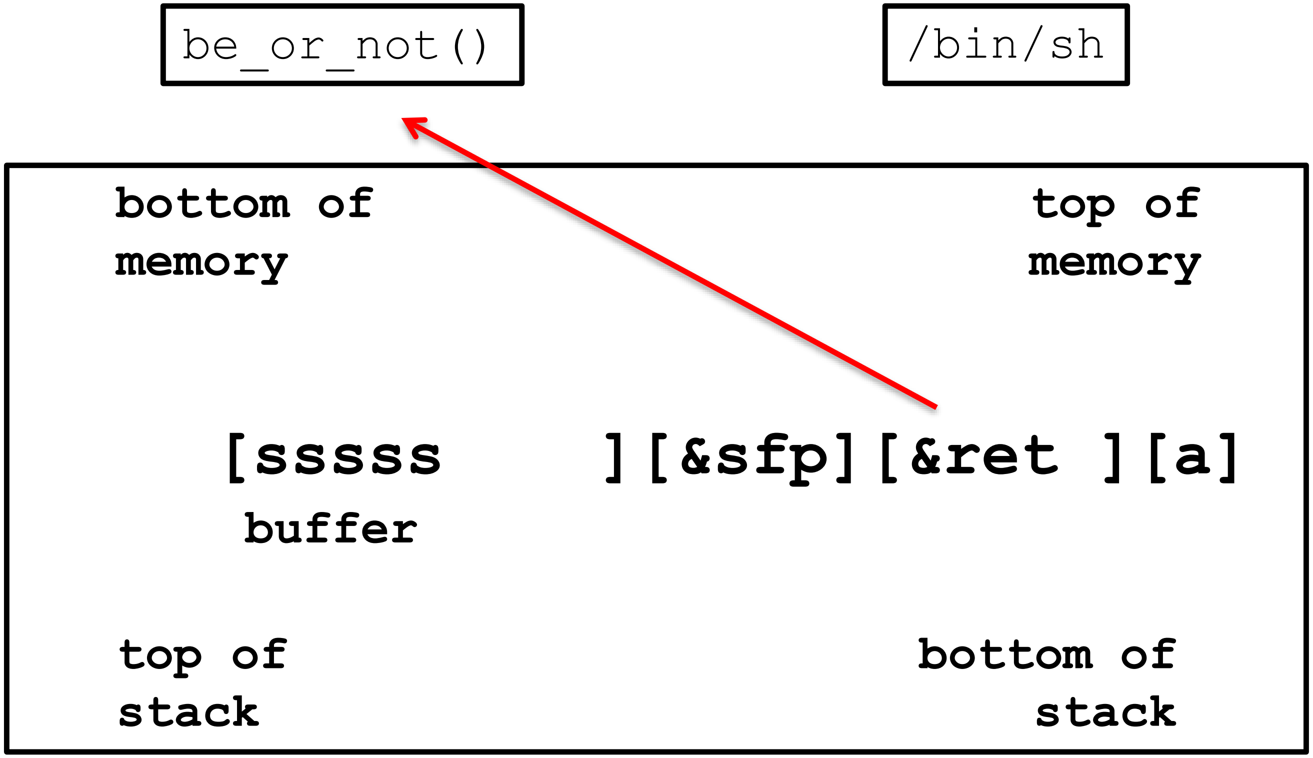
[s][&sfp][&ret][a]

buffer

top of stack

bottom of stack

BOF (Cont.)



be_or_not()

/bin/sh

bottom of memory

top of memory

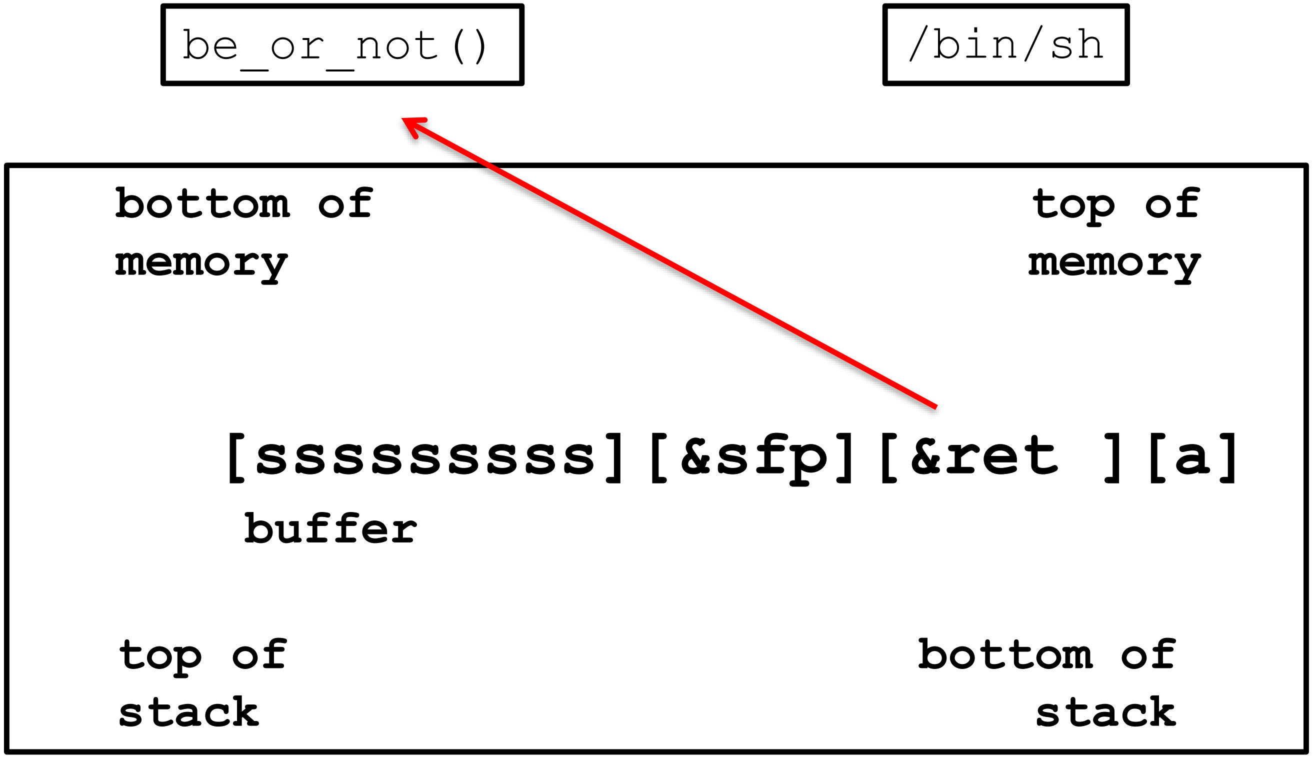
[sssss][&sfp][&ret][a]

buffer

top of stack

bottom of stack

BOF (Cont.)



be_or_not()

/bin/sh

bottom of memory

top of memory

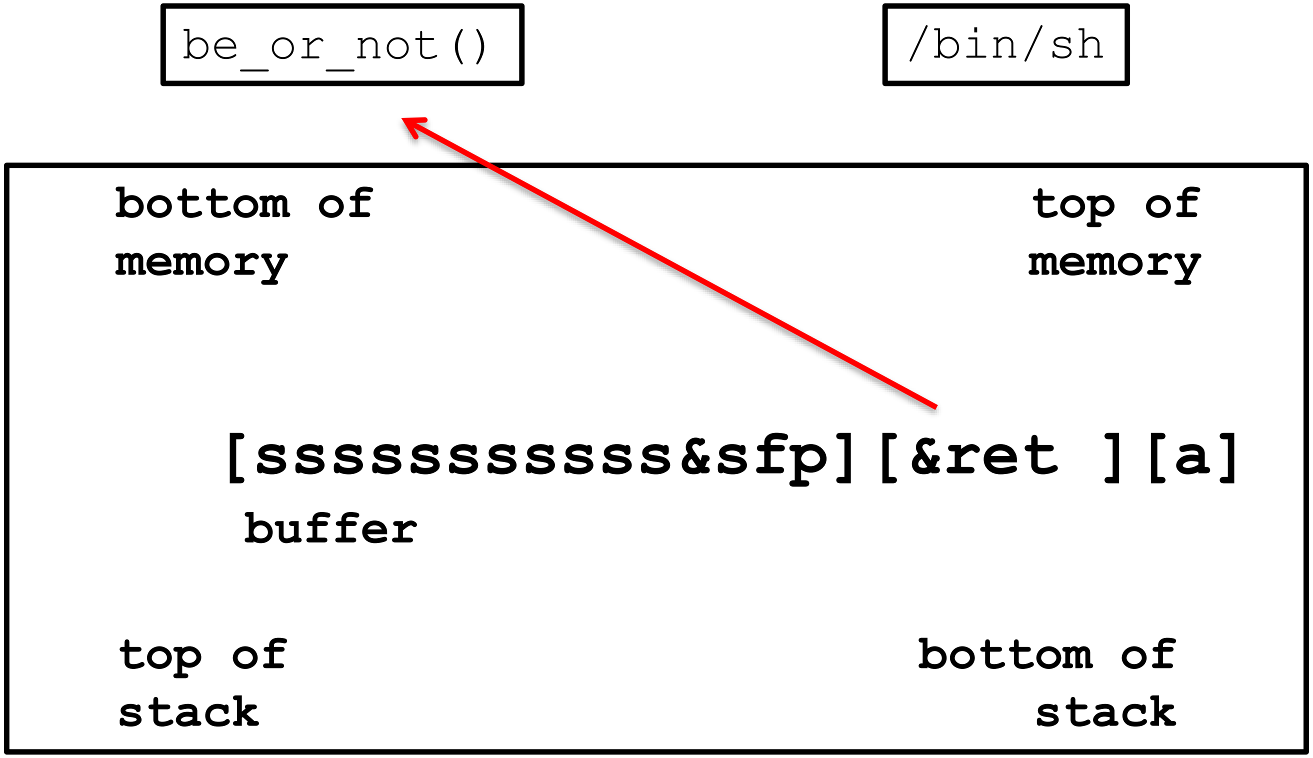
[sssssssss][&sfp][&ret][a]

buffer

top of stack

bottom of stack

BOF (Cont.)



be_or_not()

/bin/sh

bottom of memory

top of memory

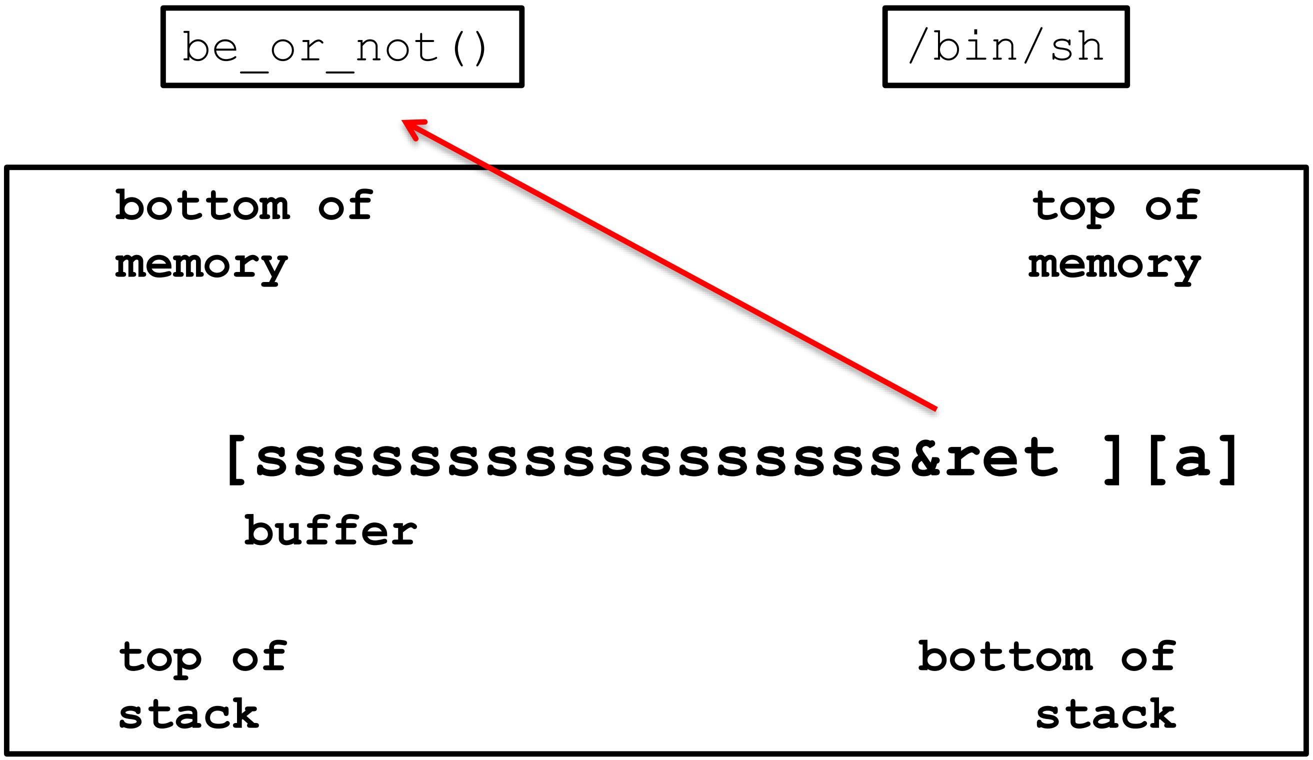
[ssssssssssss&sfp][&ret][a]

buffer

top of stack

bottom of stack

BOF (Cont.)



be_or_not()

/bin/sh

bottom of memory

top of memory

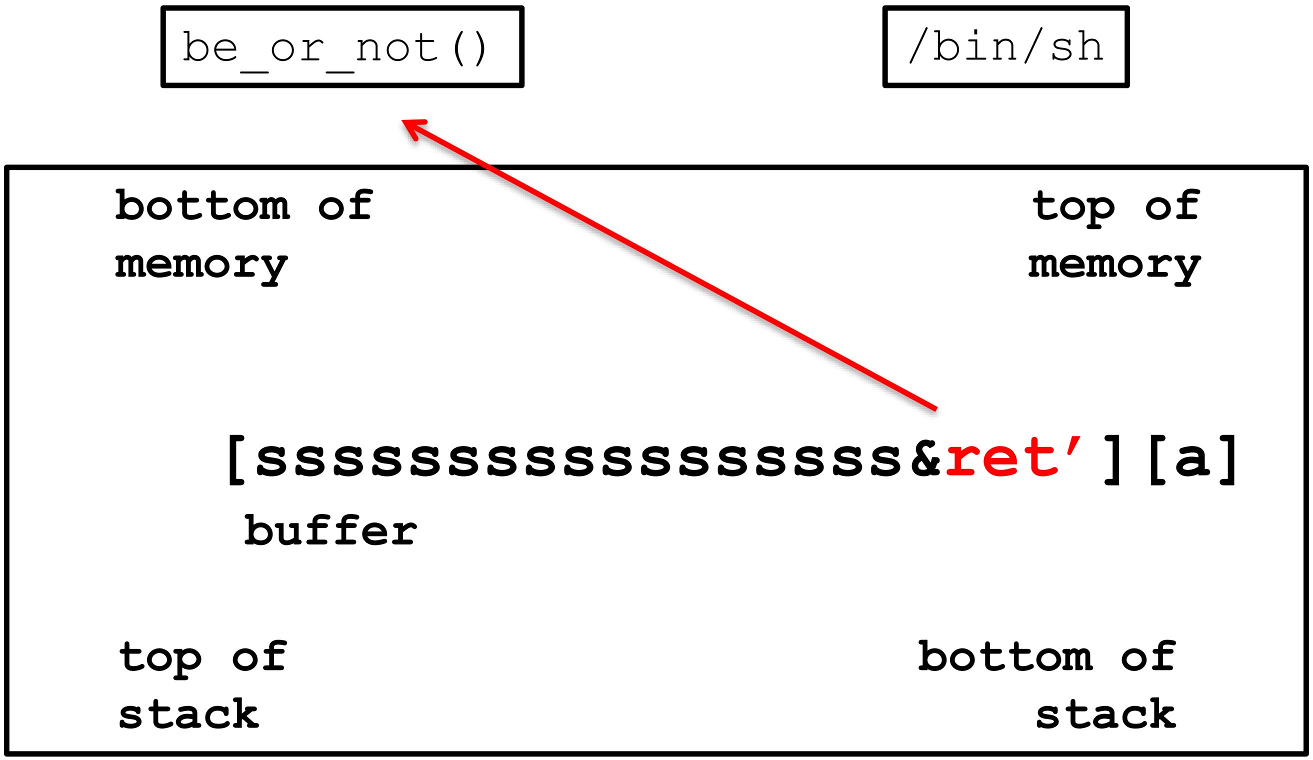
[sssssssssssssssss&ret][a]

buffer

top of stack

bottom of stack

BOF (Cont.)



be_or_not()

/bin/sh

bottom of
memory

top of
memory

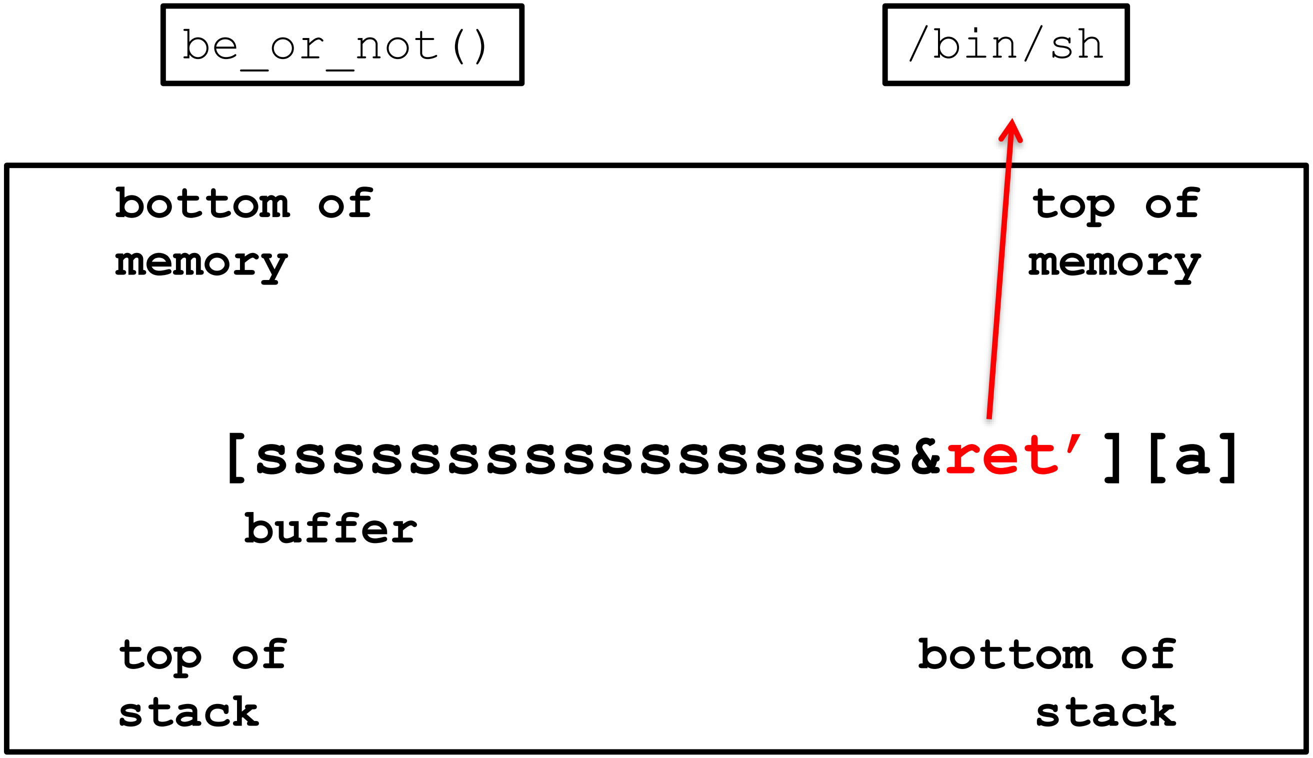
[sssssssssssssssss&ret′][a]

buffer

top of
stack

bottom of
stack

BOF (Cont.)



be_or_not()

/bin/sh

bottom of
memory

top of
memory

[sssssssssssssssss&ret']][a]

buffer

top of
stack

bottom of
stack

Ok, I got it! IoT apps are written and C and then they’re particularly vulnerable to BOFs.

There are a bunch of BOF prevention mechanisms out there, though.

Can’t we just pick out one and apply in IoT?

No! Because they are inadequate *as-is* to IoT.

BOF Prevention: existing proposals

- There are many proposals for BOF prevention in the context of Internet
  - E.g. SAFECode, SoftBounds, AddressSanitizer, etc.
- They are effective in that they protect memory accesses (`load/store)` via Array-Bound Checks

BOF Prevention: existing proposals

- There are many proposals for BOF prevention in the context of Internet
  - E.g. SAFECode, SoftBounds, AddressSanitizer
- They are effective in that they protect memory accesses (`load/store`) via Array-Bound Checks

Problem

- They tend to slow down the programs too much
- E.g. AddressSanitizer (Serebryany et al. 2012) incurs on average 73% of overhead in a conventional machine

BOF Prevention: existing proposals

- There are many proposals for BOF prevention in the context of Internet
  - E.g. SAFECode, SoftBounds, AddressSanitizer
- They are effective in that they protect memory accesses (`load/store`) via Array-Bound Checks

Problem

- They tend to slow down the programs too much
- E.g. AddressSanitizer (Serebryany et al. 2012) incurs on average 73% of overhead in a conventional machine

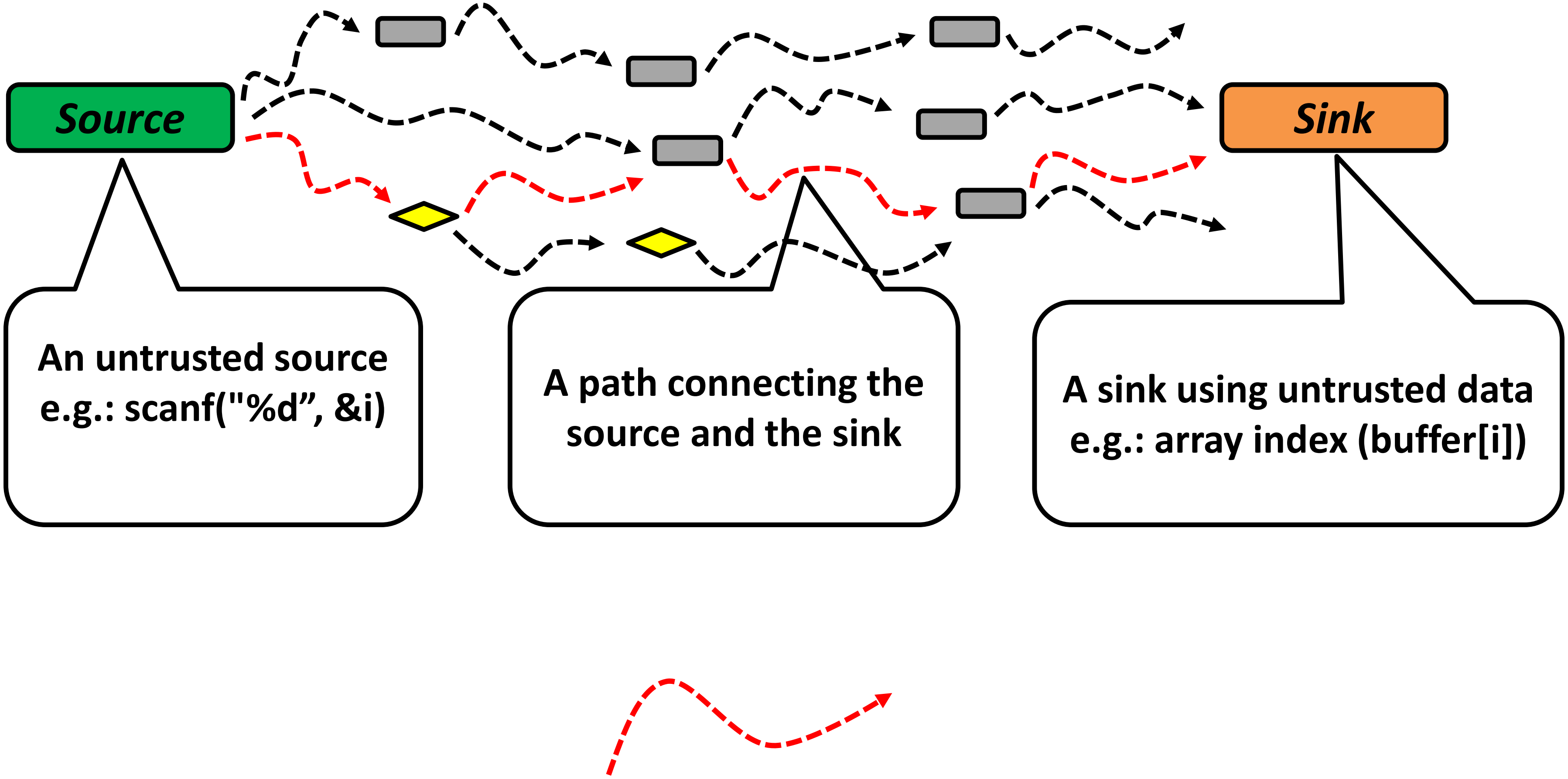
Constrained devices like *things* cannot afford this overhead

Q: How do existing proposals for preventing BOF work?

How to protect code against BOFs

- Many of the existing proposals to secure code against BOFs have three phases
  1. They first find buffers reachable from untrusted sources, at compiling time

Stage 1: buffers reachable from untrusted sources



An untrusted source e.g.: scanf("%d", &i)

A path connecting the source and the sink

A sink using untrusted data e.g.: array index (buffer[i])

Tainted flow

How to protect code against BOFs

- Many of the existing proposals to secure code against BOFs have three phases
  1. They first find buffers reachable from untrusted sources, at compiling time
  2. They thus guard buffers by inserting Array Bounds Checks (ABCs) prior to buffers use

Stage 2: ABC insertion

Vulnerable code

```
int main(int argc, char
**argv){
 int buffer[BUFSIZE];
 int a,i,j;
 ...
 for(i;i<j;i++){
   ...

   buffer[i] = a;
   ...
   }
   ...
}
```

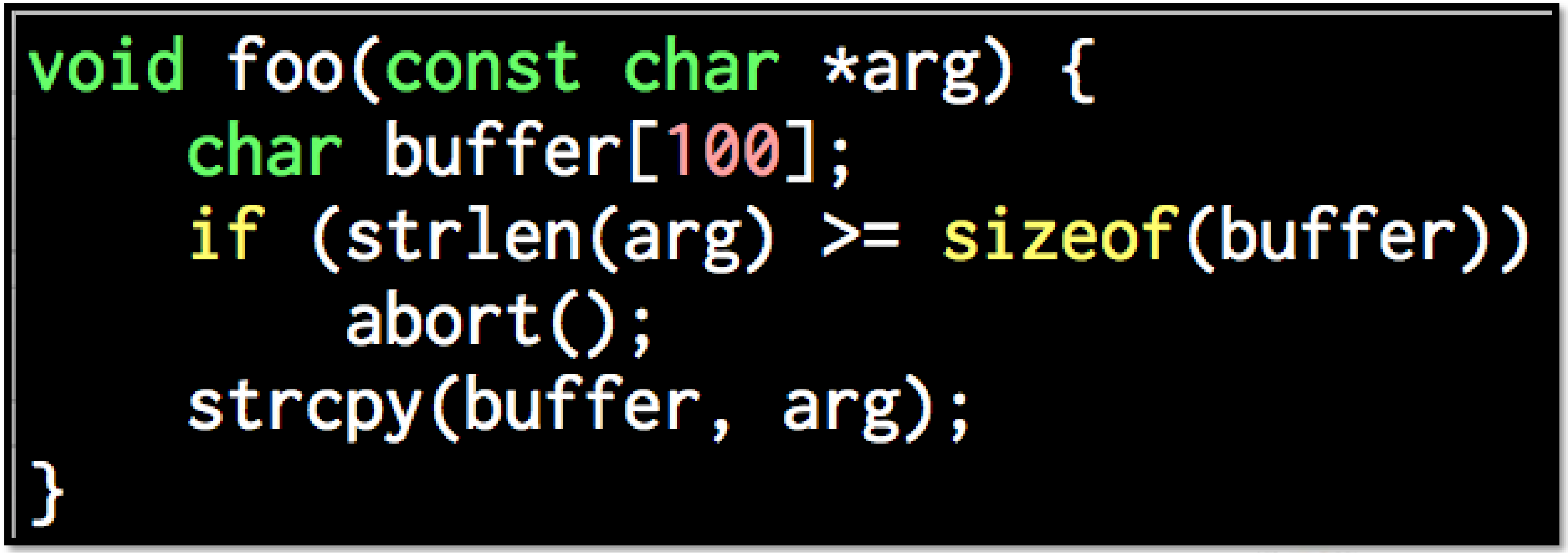
ABC-protected code

```
int main(int argc, char
**argv){
 int buffer[BUFSIZE];
 int a,ij,;
 ...
 for(i;i<j;i++){
   ...
   if((i >= 0)&&(i < BUFSIZE))
     buffer[i] = a;
   ...
   }
   ...
}
```

How to protect code against BOFs

- Many of the existing proposals to secure code against BOFs have three phases
  1. They first find buffers reachable from untrusted sources, at compiling time
  2. They thus guard buffers by inserting Array Bounds Checks (ABCs) prior to buffers use
  3. If an ABC is not satisfied at execution time, they then abort programs

Stage 3: Illegal memory access is aborted



```c
void foo(const char *arg) {
    char buffer[100];
    if (strlen(arg) >= sizeof(buffer))
        abort();
    strcpy(buffer, arg);
}
```

Agenda

- Introduction
- Goal
- Solution
- Results
- Conclusion

Goal

- Our goal is to come up with a BOF prevention mechanism tailor-made for IoT
- Solutions must therefore be
  1. Secure against BOFs
  2. Light enough to be run in battery-powered devices

Agenda

- Introduction
- Goal
- Solution
- Results
- Conclusion

Agenda

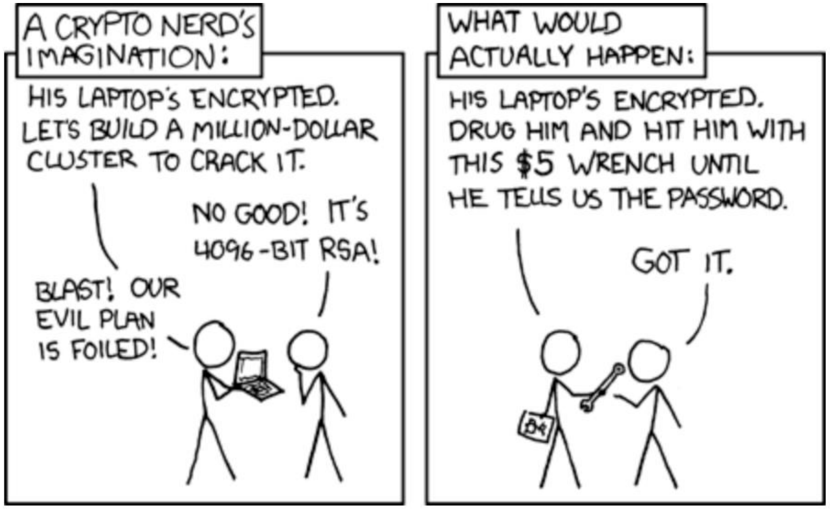
- Introduction
- Goal
- Solution
  - Conception
  - Refinement
  - Development
- Results
- Conclusion

Agenda

- Introduction
- Goal
- Solution
  - Conception
  - Refinement
  - Development
- Results
- Conclusion

Assumptions/Attack Model

- Nodes run authentic programs
  - E.g. they could employ Trusted Platform Module (TPM)
- The communication channel is secure
  - Crypto solutions like DTLS (Kothmayr et al.), TinyPBC (Oliveira et al.), or SPINS (Perrig et al.) could be adopted


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Assumptions/Attack Model (Cont.)

- Attackers have control over the input data that the nodes receive from its environment
  - This includes data captured by the sensors or input from the user interfaces
  - But excludes data coming from network interfaces as we assume a secure communication channel.

How to lose weight?

SIoT challenge #1

Idea

- Recall existing proposals find buffers reachable from (untrusted) sources
- They analyze programs of a distributed system as disjoint/independent programs
  - E.g., they end up analyzing a client and its respective server programs <u>individually</u>
- Therefore the list of untrusted sources include not only conventional (e.g. `get`) and network (e.g. `recv`) sources

Idea (Cont.)

- The higher the number of untrusted sources, the higher the number of reachable arrays
- And the higher the number of reachable arrays, the higher the number of ABCs and thus the overhead

Idea (Cont.)

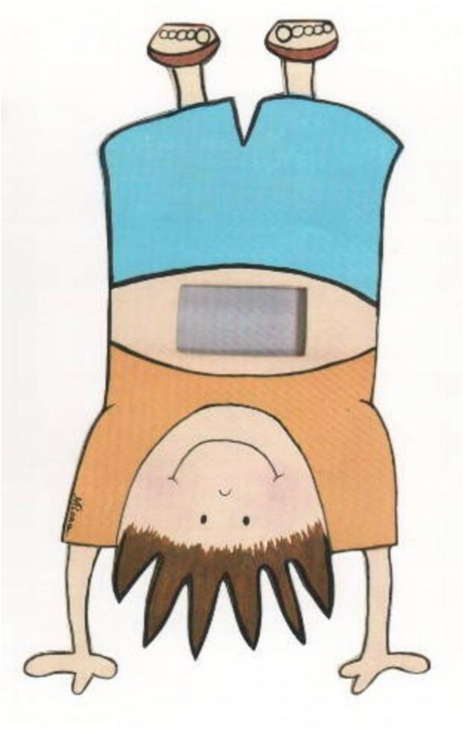
- The higher the number of untrusted sources, the higher the number of reachable arrays
- And the higher the number of reachable arrays, the higher the number of ABCs and thus the overhead

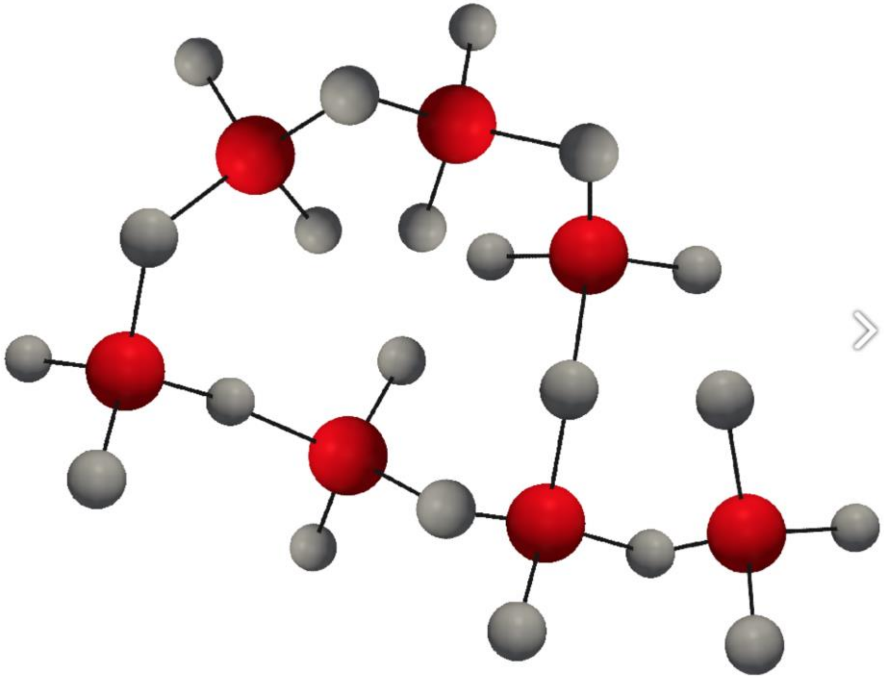
So, what if we decrease the number of untrusted sources?

How can we turn untrusted sources into something else, though?

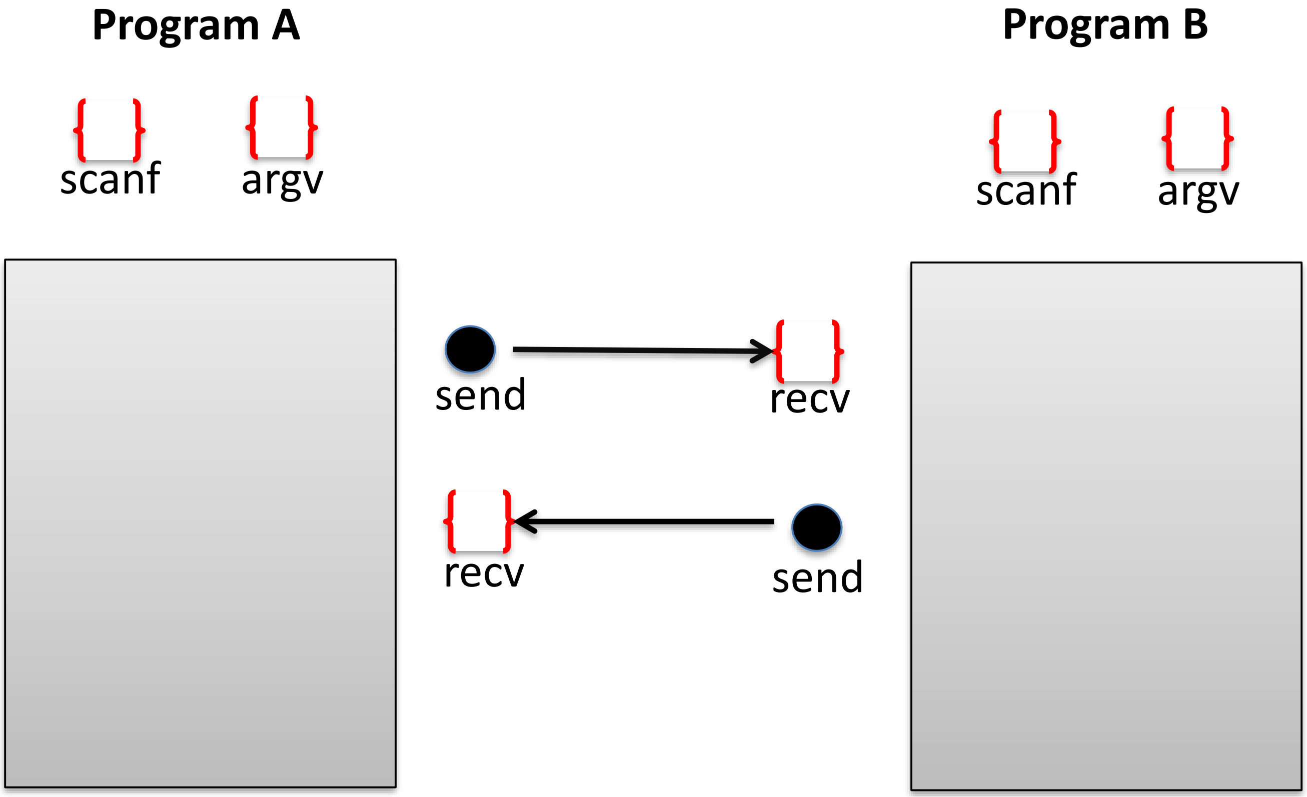
SIoT key contribution #1

- Look at a distributed system programs from another perspective

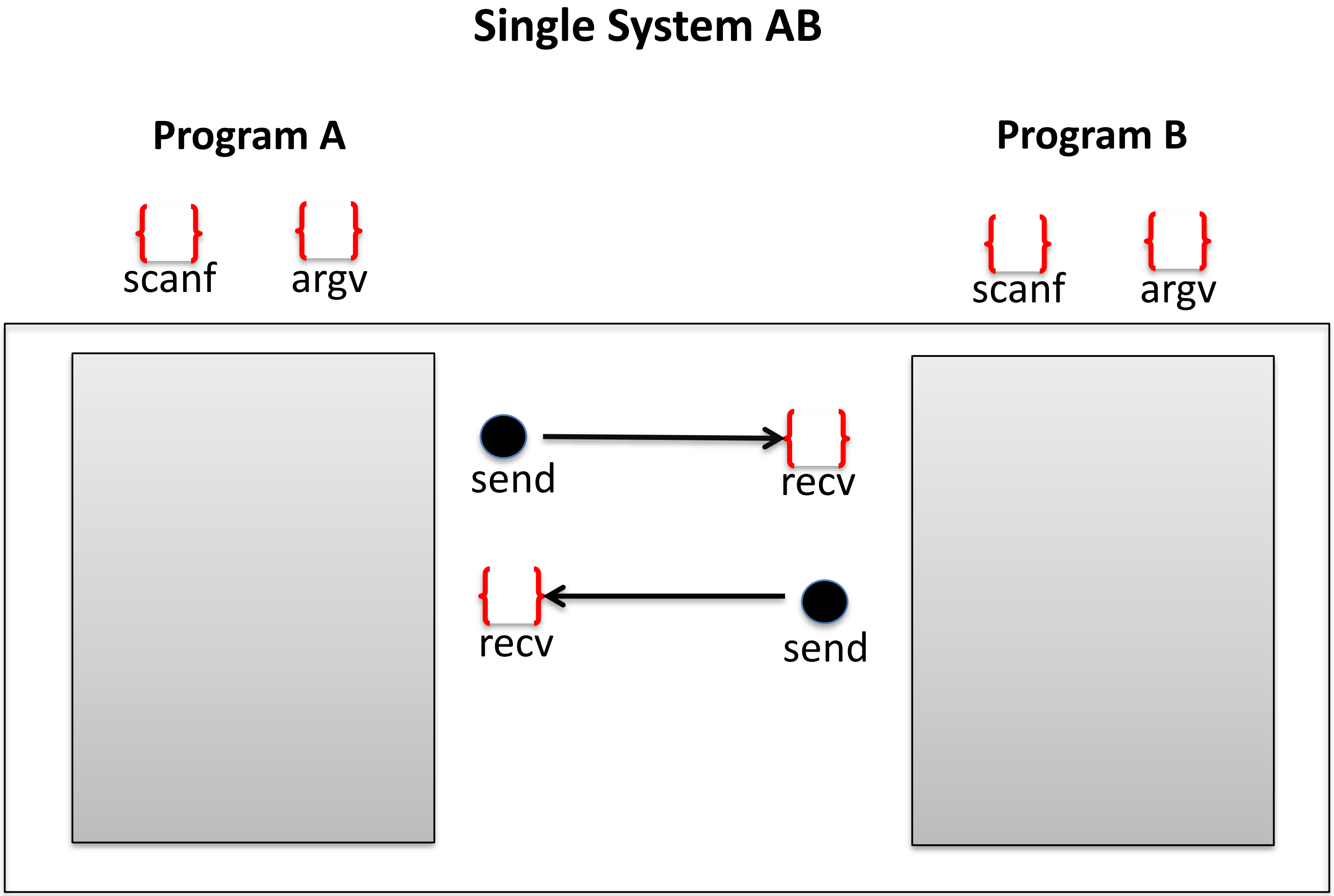




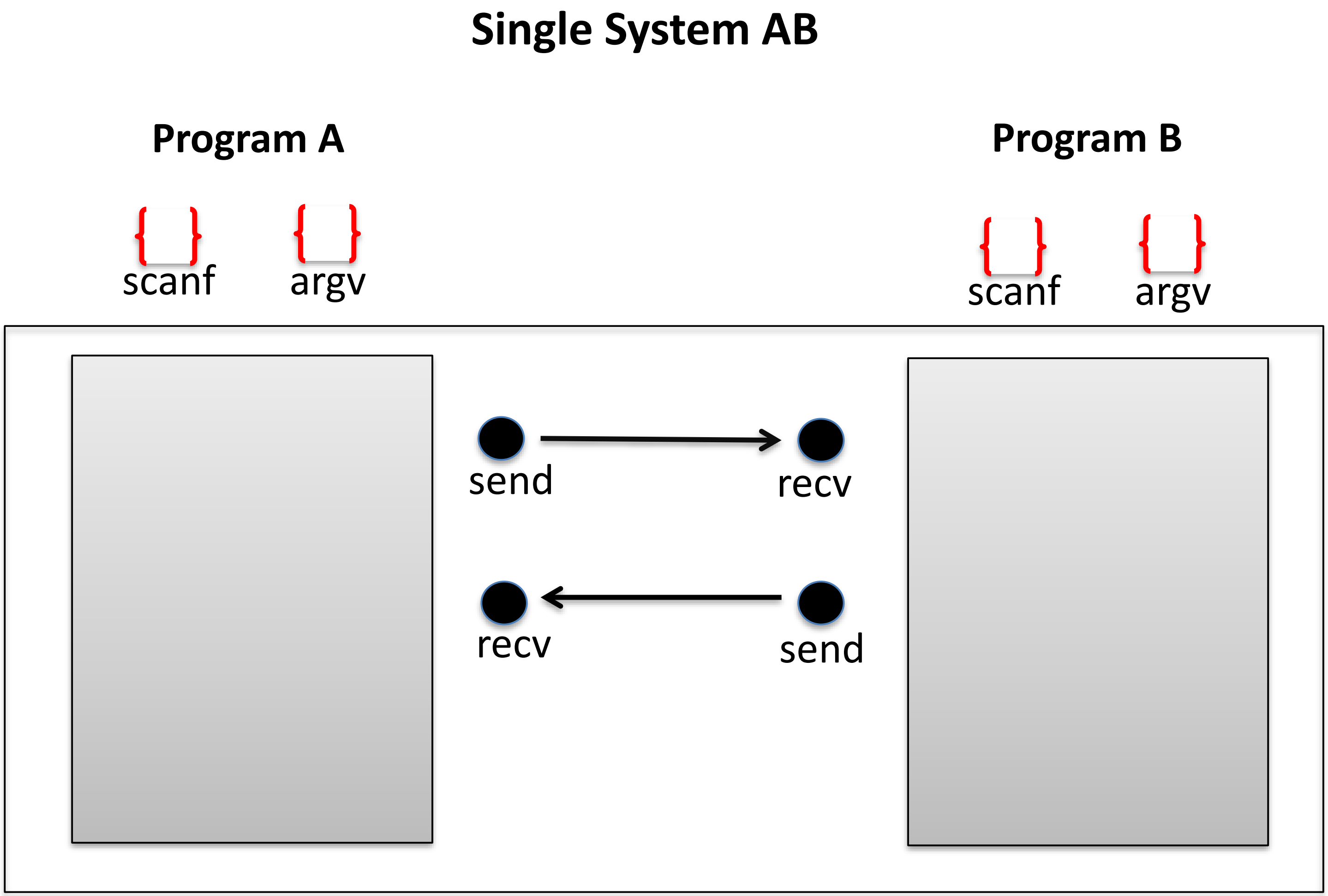
Programs A and B: sources



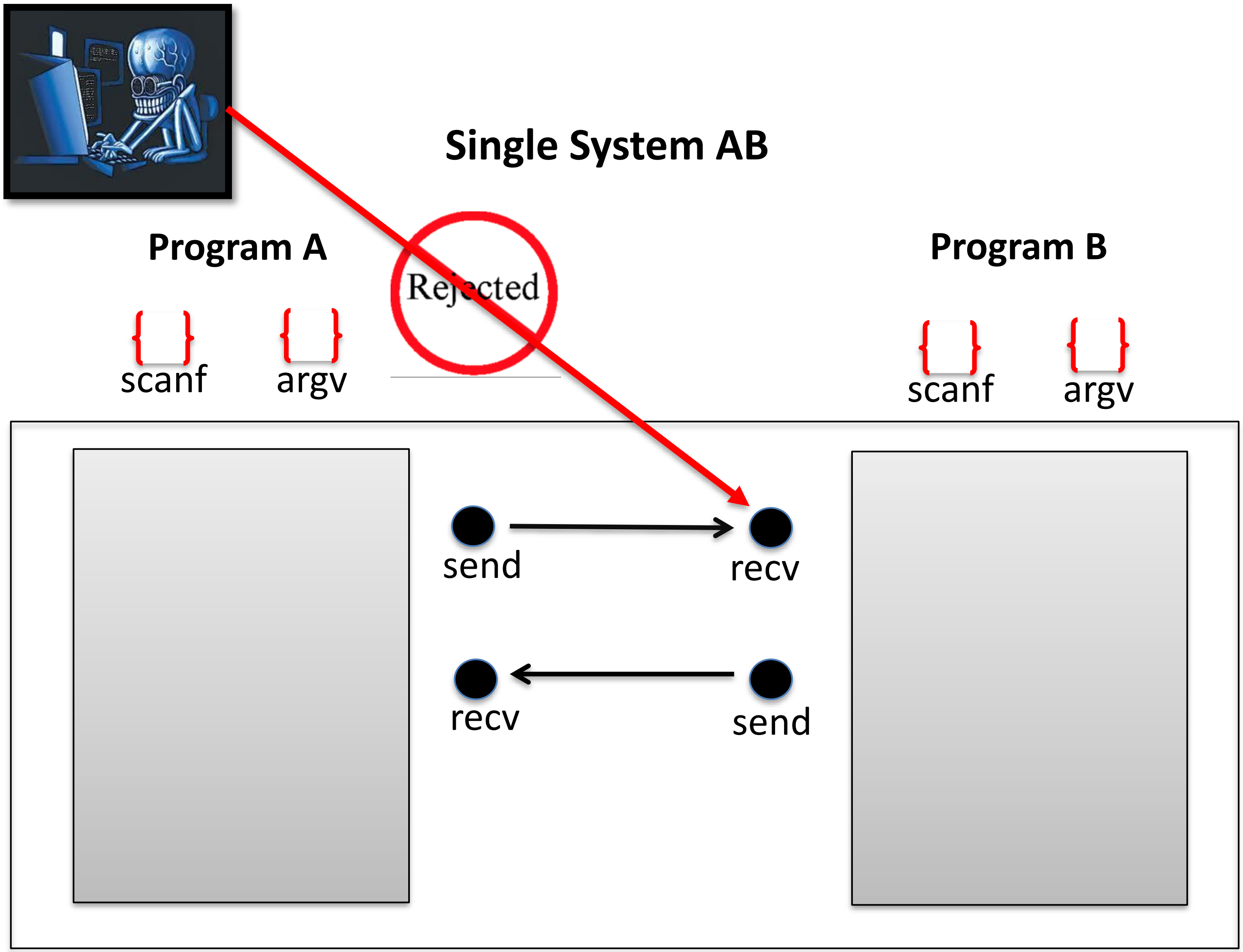
Distributed System AB: sources



Distributed System AB: sources



Distributed System AB: sources



Single System AB

Program A

scanf

argv

Rejected

Program B

scanf

argv

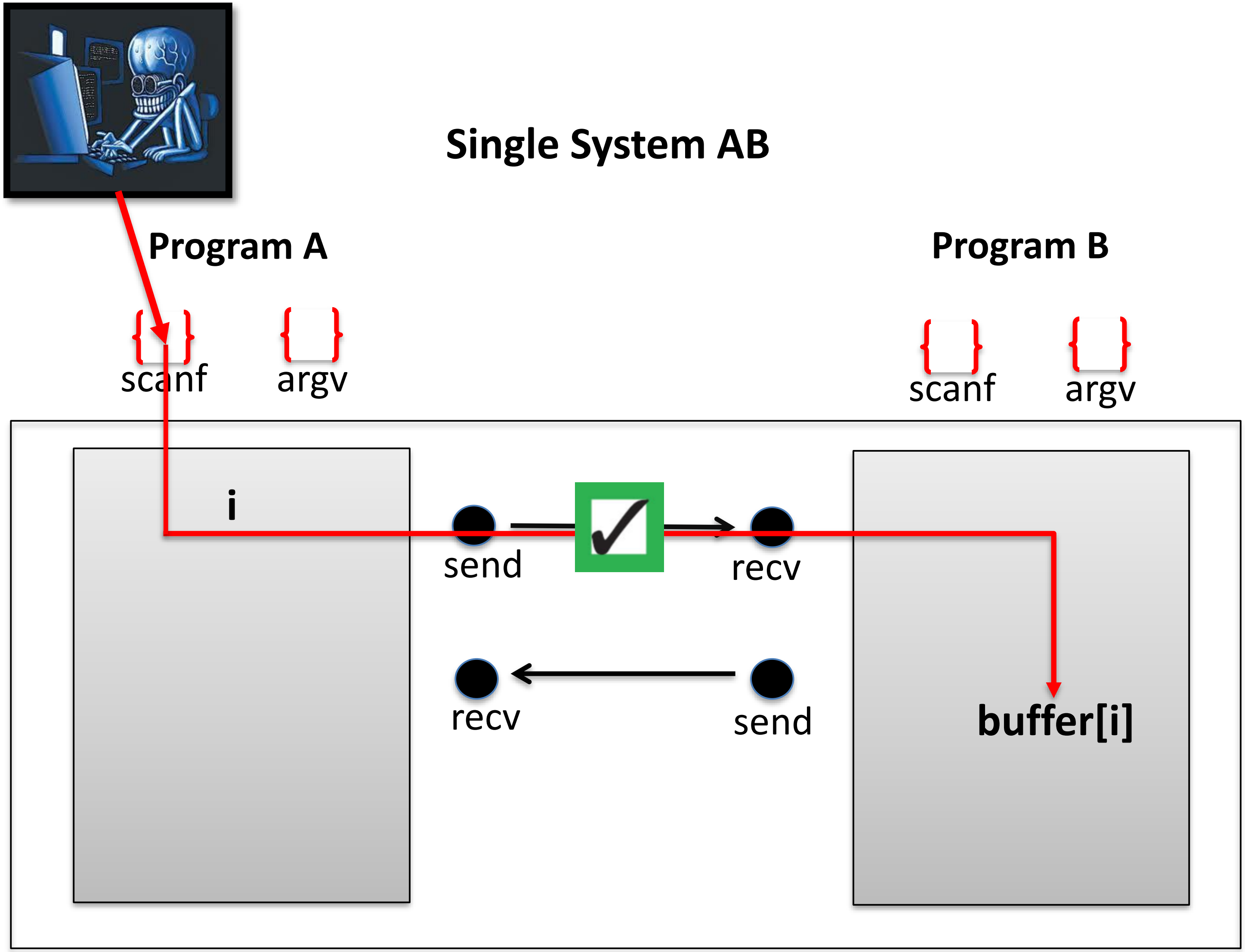
send

recv

recv

send

Distributed System AB: sources



Single System AB

Program A

scanf

argv

Program B

scanf

argv

i

send

recv

recv

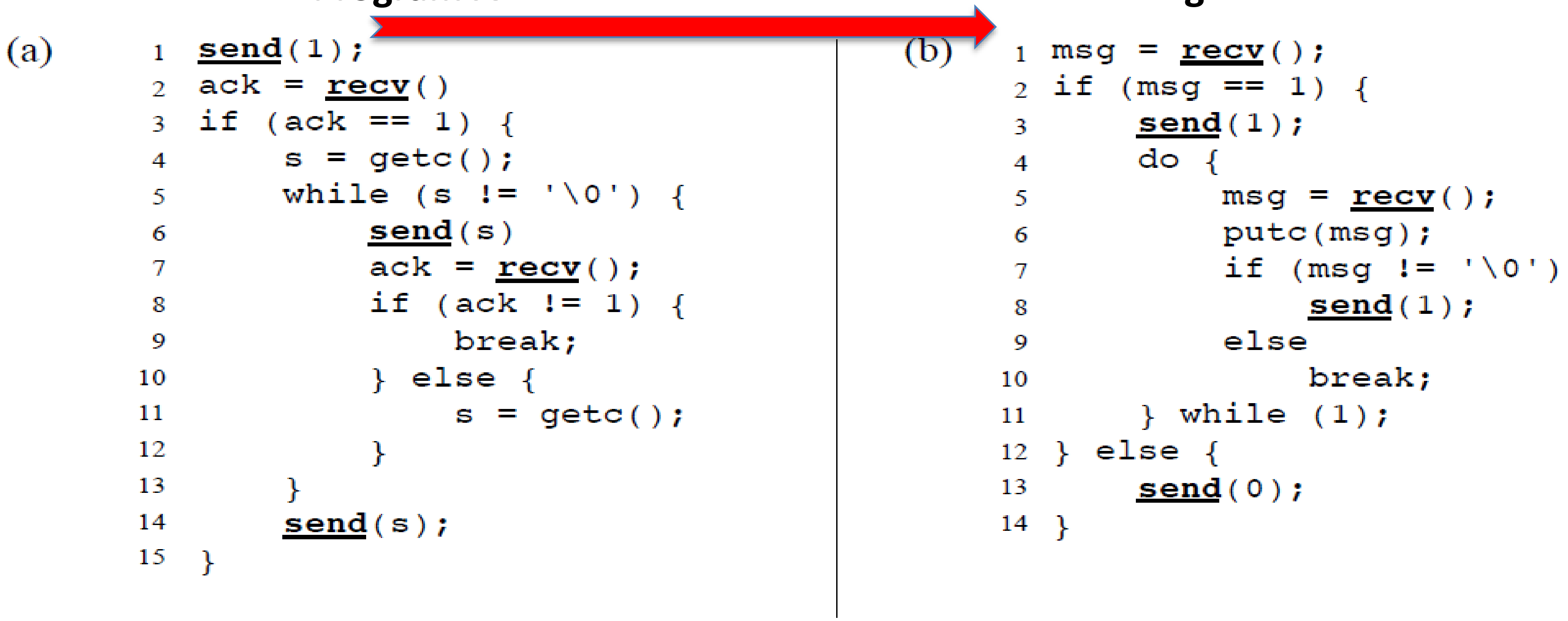
send

buffer[i]

Distributed System AB

Program A

Program B



(a)

```
send(1);
ack = recv()
if (ack == 1) {
     s = getc();
     while (s != '\0') {
          send(s)
          ack = recv();
          if (ack != 1) {
               break;
          } else {
               s = getc();
          }
     }
     send(s);
}
```

(b)

```
msg = recv();
if (msg == 1) {
     send(1);
     do {
          msg = recv();
          putc(msg);
          if (msg != '\0')
               send(1);
          else
               break;
     } while (1);
} else {
     send(0);
}
```

No longer considered a vulnerability

Agenda

- Introduction
- Goal
- Solution
  - Conception
  - Refinement
  - Development
- Results
- Conclusion

SIoT challenge #2

- Face the lack of data-structures to analyze distributed systems
- Control Flow Graphs (CFGs) are the core data-structure in program analysis
- CFGs are not expressive enough to represent programs that communicate over a network, though
  - E.g., they do not handle message exchange between nodes

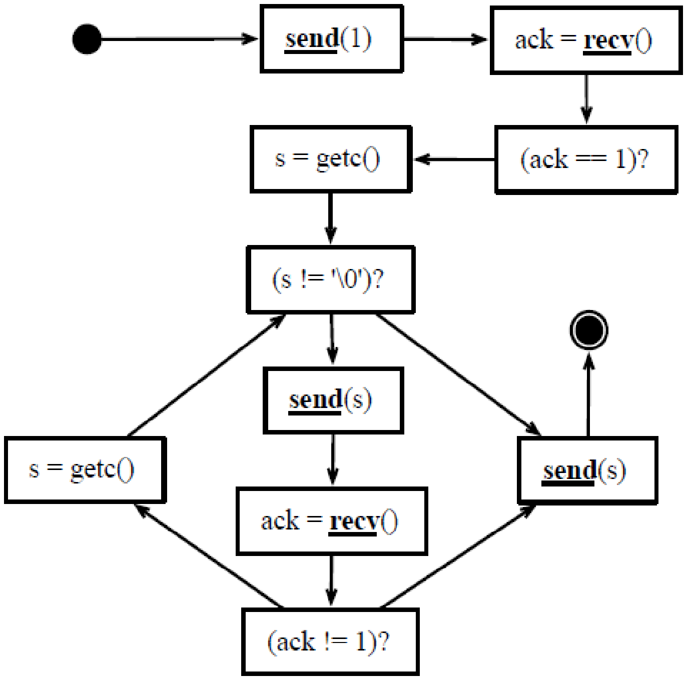
SIoT key contribution #2

- Distributed Control Flow Graph (DCFG)
- DCFGs are data structures able to bind together the CFGs of all individual programs that constitute a system

CFG

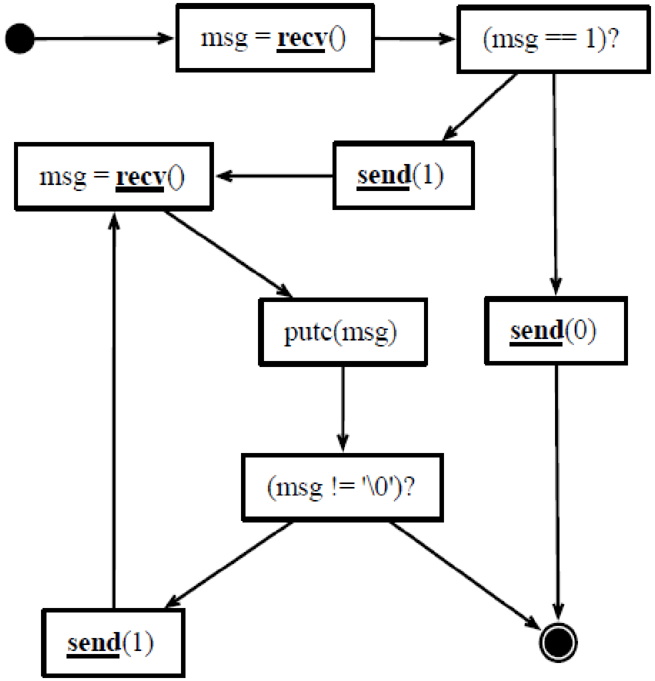
(a)

```
send(1);
ack = recv()
if (ack == 1) {
    s = getc();
    while (s != '\0') {
        send(s)
        ack = recv();
        if (ack != 1) {
            break;
        } else {
            s = getc();
        }
    }
    send(s);
}
```

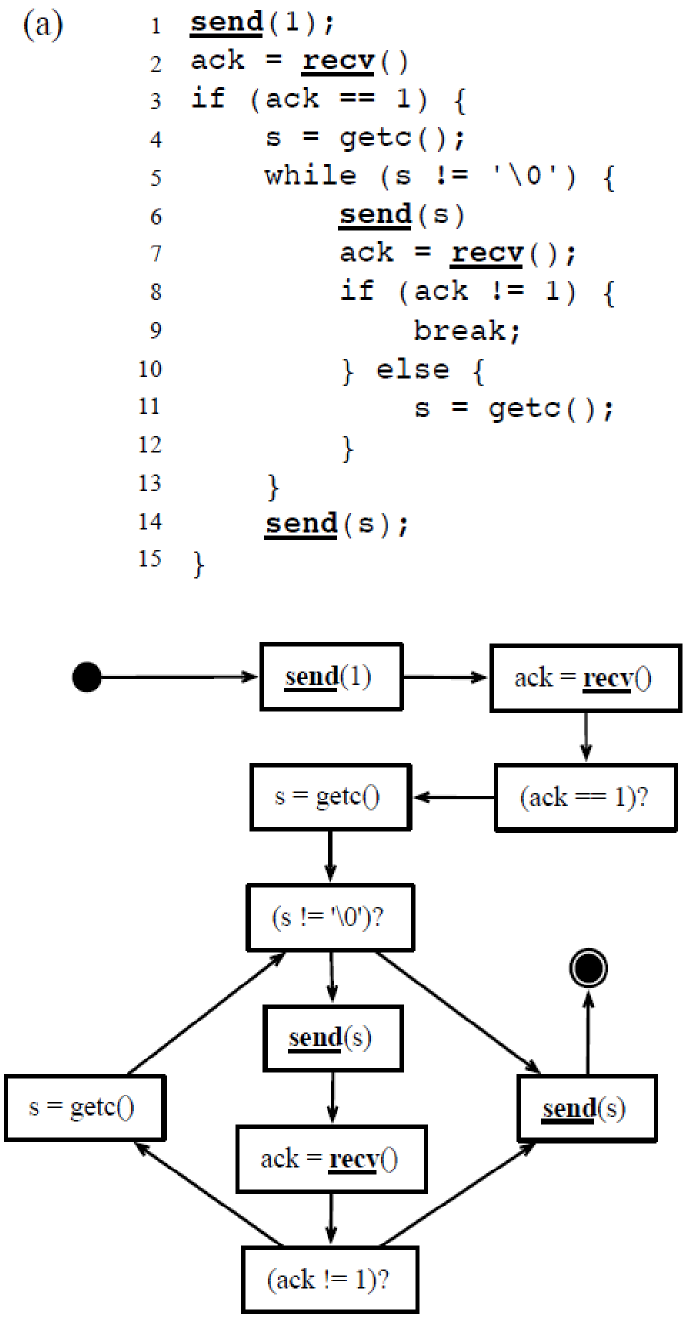


(b)

```
msg = recv();
if (msg == 1) {
    send(1);
    do {
        msg = recv();
        putc(msg);
        if (msg != '\0')
            send(1);
        else
            break;
    } while (1);
} else {
    send(0);
}
```

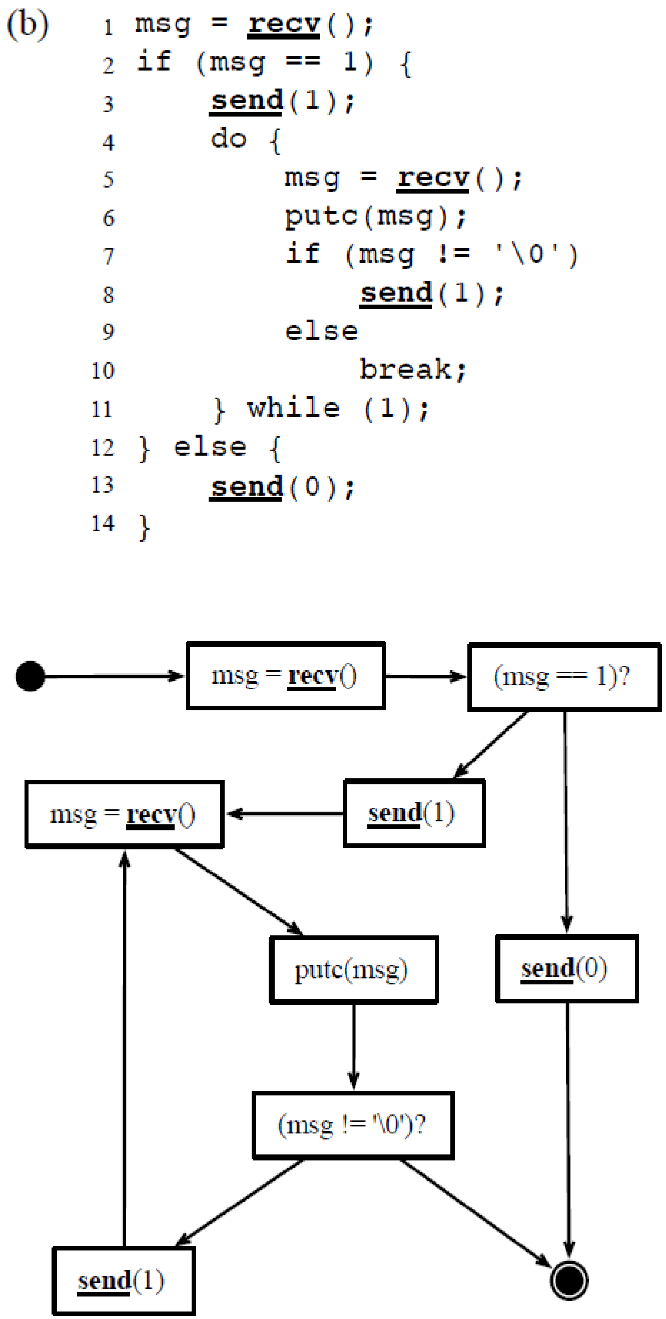


DCFG



(a)

```
send(1);
ack = recv()
if (ack == 1) {
    s = getc();
    while (s != '\0') {
        send(s)
        ack = recv();
        if (ack != 1) {
            break;
        } else {
            s = getc();
        }
    }
    send(s);
}
```



(b)

```
msg = recv();
if (msg == 1) {
    send(1);
    do {
        msg = recv();
        putc(msg);
        if (msg != '\0')
            send(1);
        else
            break;
    } while (1);
} else {
    send(0);
}
```

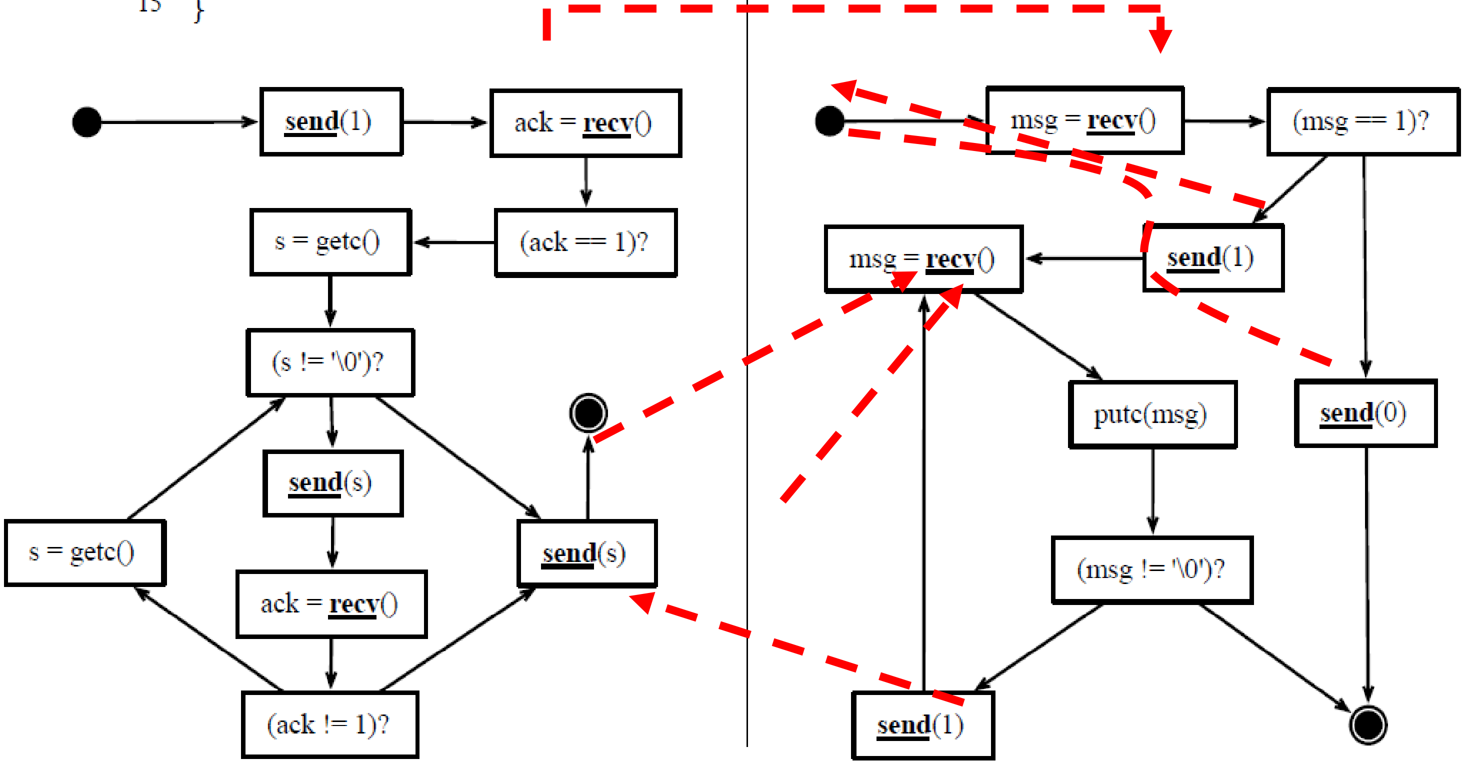
DCFG

(a)

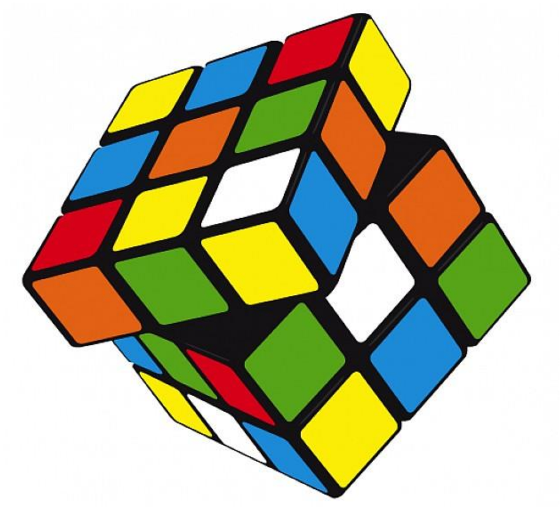
```
send(1);
ack = recv()
if (ack == 1) {
    s = getc();
    while (s != '\0') {
        send(s)
        ack = recv();
        if (ack != 1) {
            break;
        } else {
            s = getc();
        }
    }
    send(s);
}
```

(b)

```
msg = recv();
if (msg == 1) {
    send(1);
    do {
        msg = recv();
        putc(msg);
        if (msg != '\0')
            send(1);
        else
            break;
    } while (1);
} else {
    send(0);
}
```

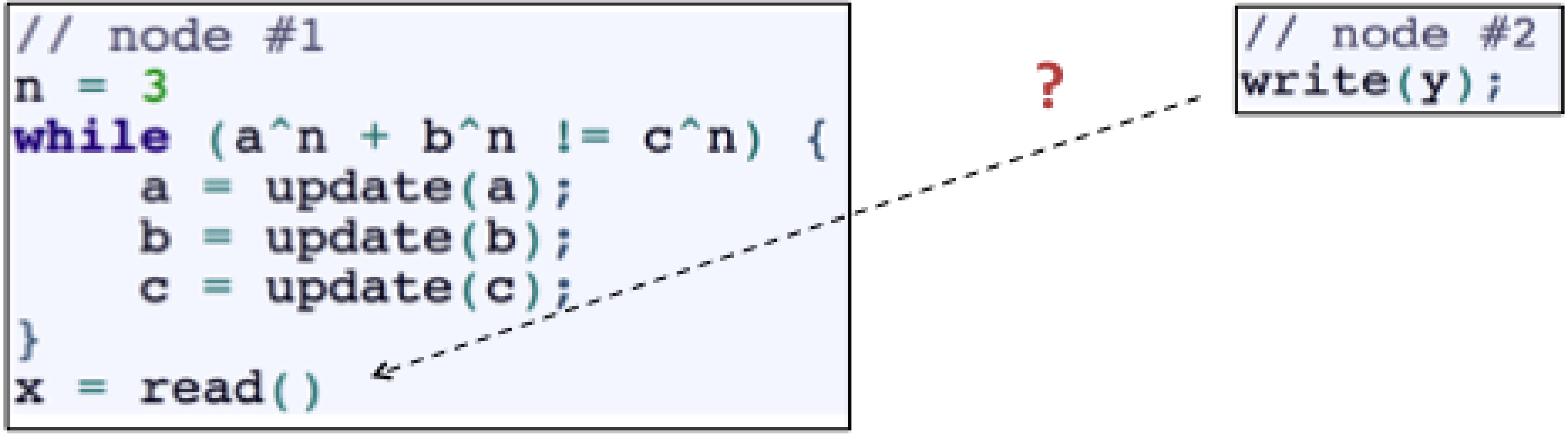


It’s not that easy, though



SIoT challenge #3

- To build a DCFG, we have to link the `sends` of a program A with the `recvs` of a program B and vice-versa



```
// node #1
n = 3
while (a^n + b^n != c^n) {
      a = update(a);
      b = update(b);
      c = update(c);
}
x = read()
```

```
// node #2
write(y);
```

How to find out which `sends` link to a `recv`?

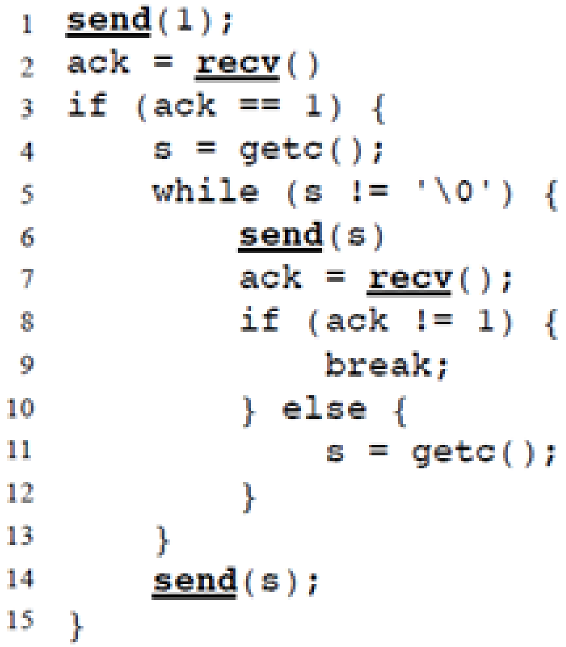
SIoT key contribution #3

- Elevator, an algorithm to selectively link `sends/recvs`
  1. Elevator assigns levels to sends and recvs
  2. Program A's `sends` and Program B's `recvs` in the same level are thus linked together

Elevator: Illustration

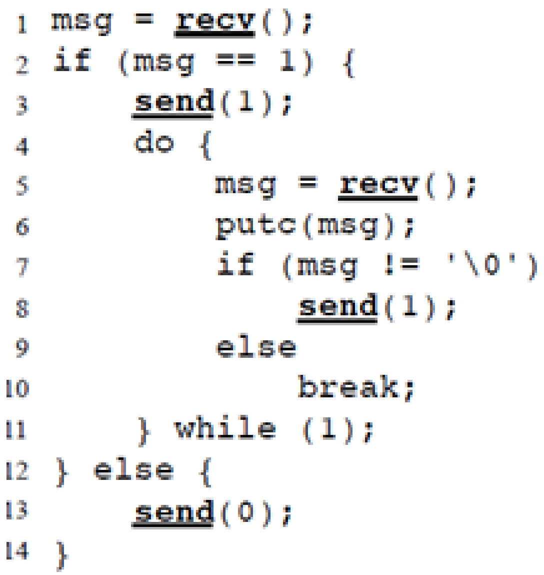
- Consider the Echo Client and Server programs

(a) Echo Client



```
send(1);
ack = recv()
if (ack == 1) {
    s = getc();
    while (s != '\0') {
        send(s)
        ack = recv();
        if (ack != 1) {
            break;
        } else {
            s = getc();
        }
    }
    send(s);
}
```

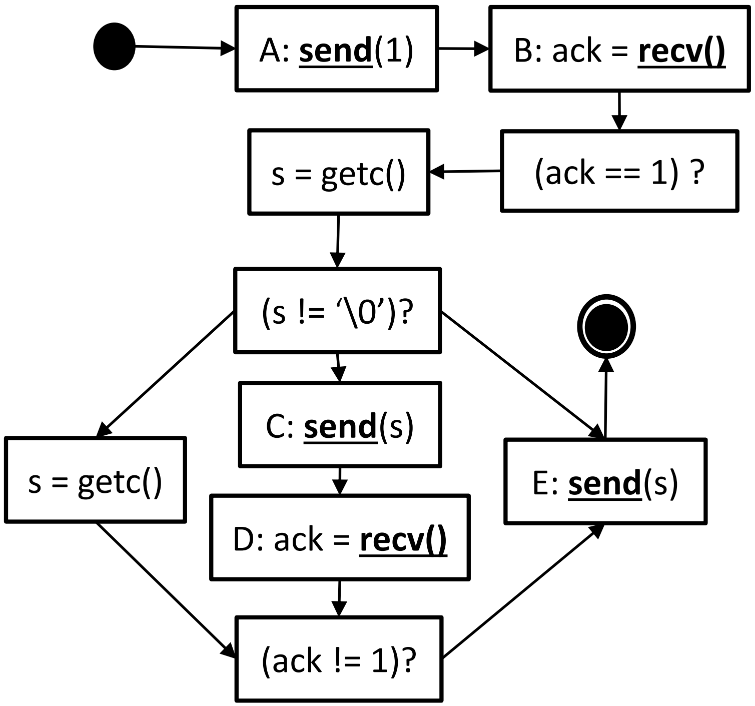
(b) Echo Server



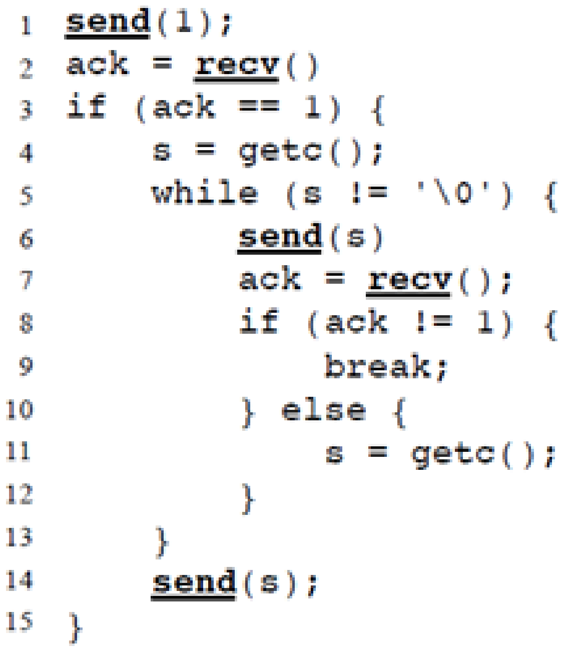
```
msg = recv();
if (msg == 1) {
    send(1);
    do {
        msg = recv();
        putc(msg);
        if (msg != '\0')
            send(1);
        else
            break;
    } while (1);
} else {
    send(0);
}
```

Echo Client CFG

CFG - Echo Client



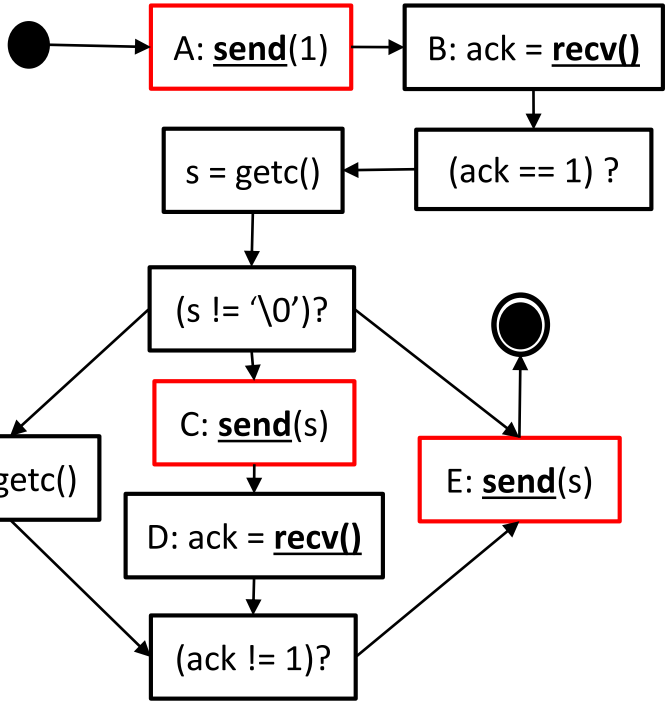
(a) Echo Client



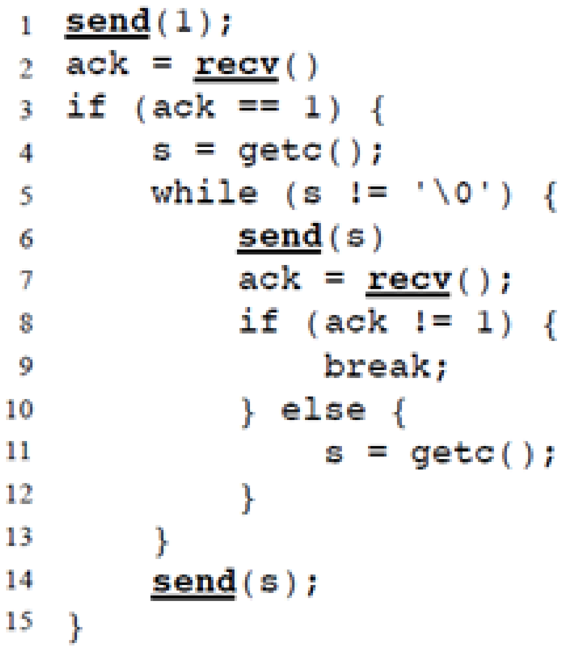
```
send(1);
ack = recv()
if (ack == 1) {
        s = getc();
        while (s != '\0') {
                send(s)
                ack = recv();
                if (ack != 1) {
                        break;
                } else {
                        s = getc();
                }
        }
        send(s);
}
```

Extract Echo Client's Send-Graph

CFG - Echo Client



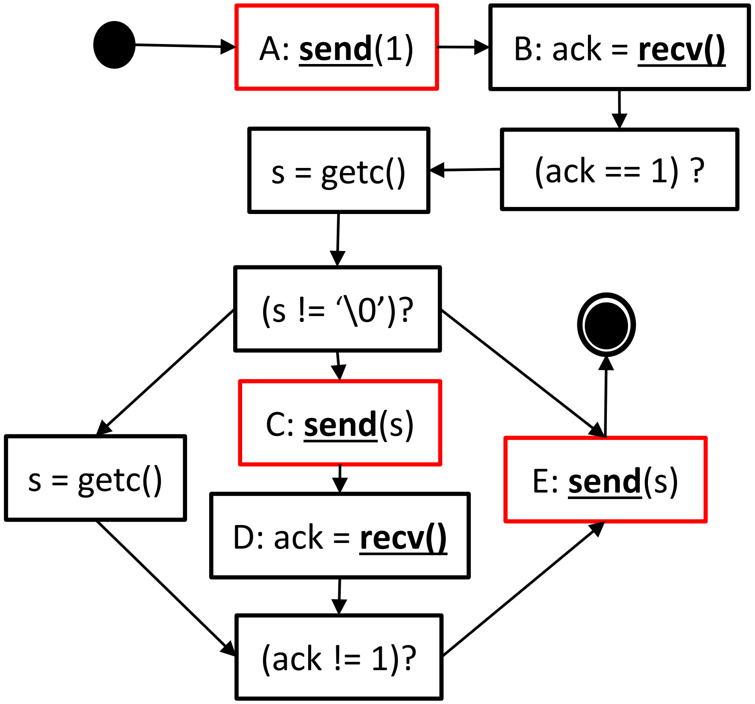
(a) Echo Client



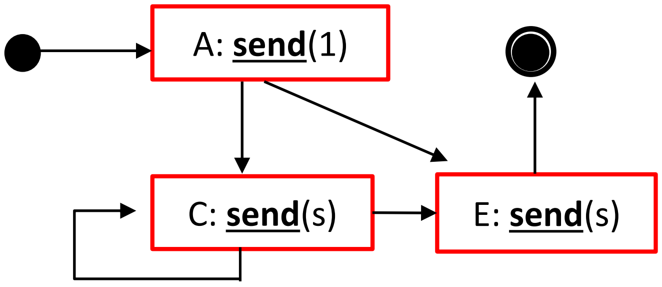
```
send(1);
ack = recv()
if (ack == 1) {
        s = getc();
        while (s != '\0') {
                send(s)
                ack = recv();
                if (ack != 1) {
                        break;
                } else {
                        s = getc();
                }
        }
        send(s);
}
```

Extract Echo Client’s Send-Graph

CFG - Echo Client

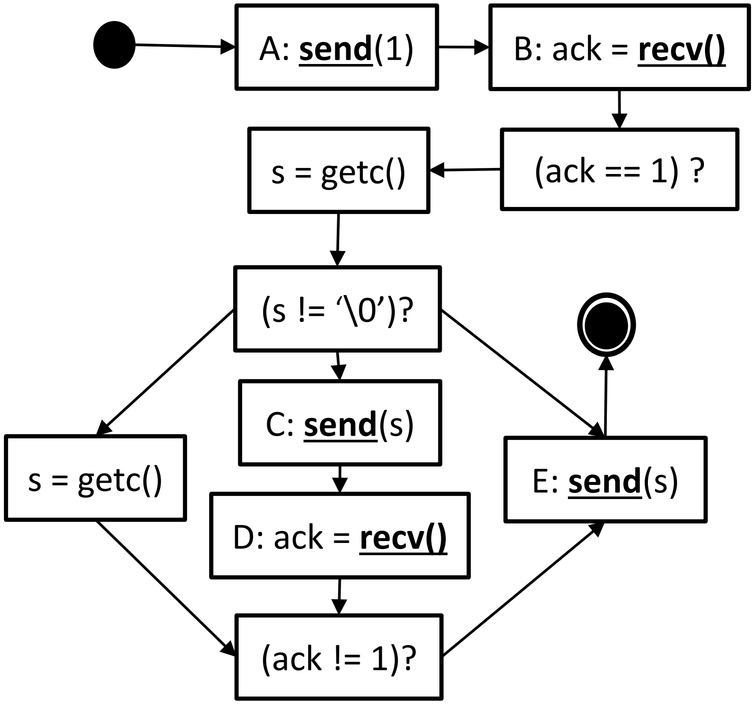


SG – Send-Graph

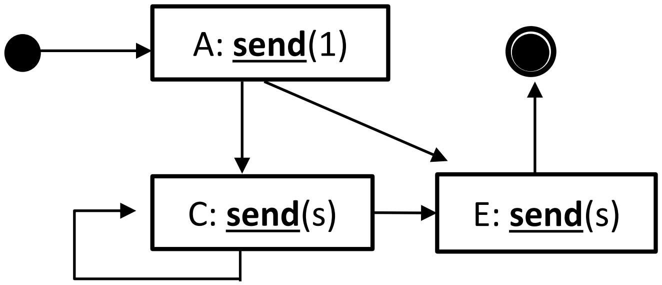


Extract Echo Client’s Receive-Graph

CFG - Echo Client

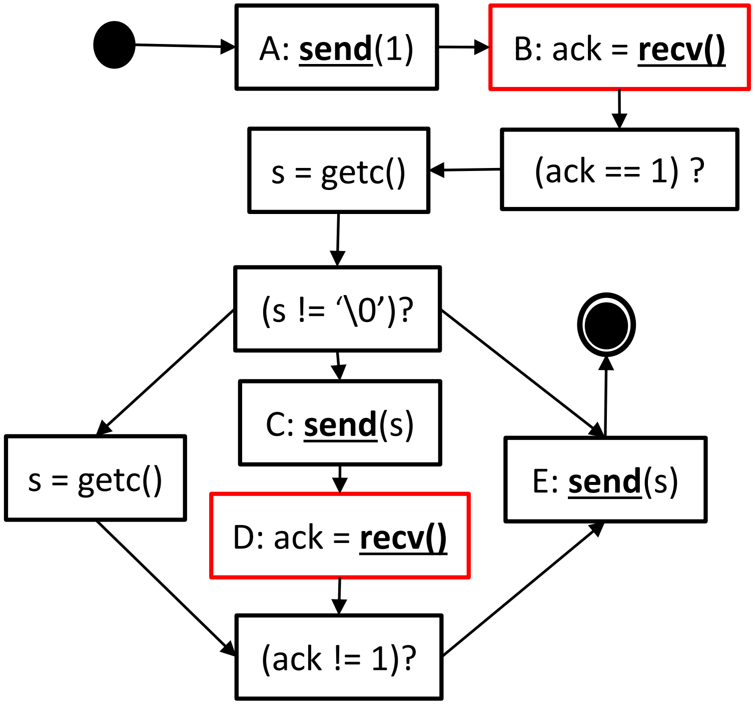


SG - Send-Graph

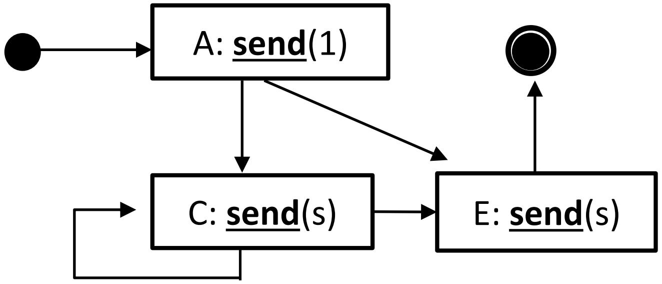


Extract Echo Client's Receive-Graph

CFG - Echo Client

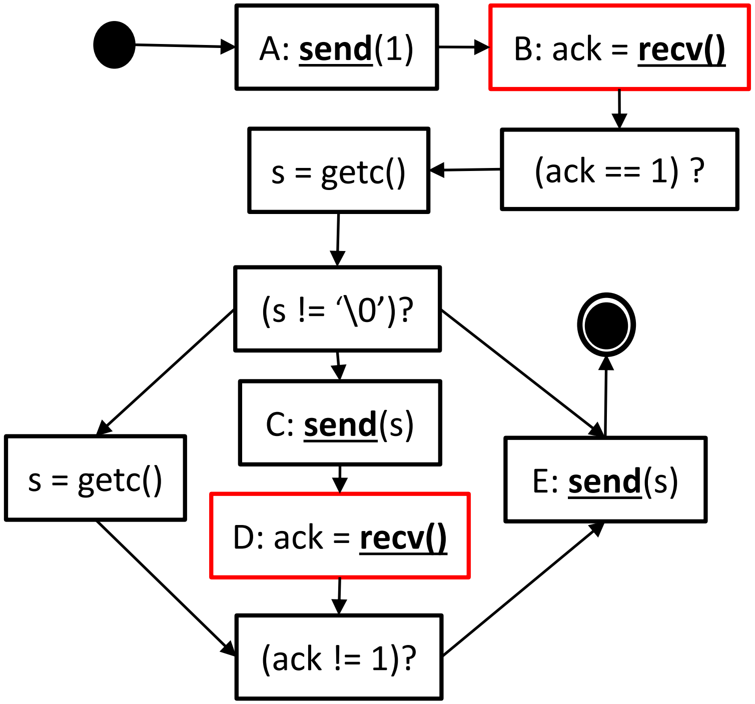


SG - Send-Graph



Extract Echo Client’s Receive-Graph

CFG - Echo Client



A: **send**(1)

B: ack = **recv()**

(ack == 1) ?

s = getc()

(s != ‘\0’)?

C: **send**(s)

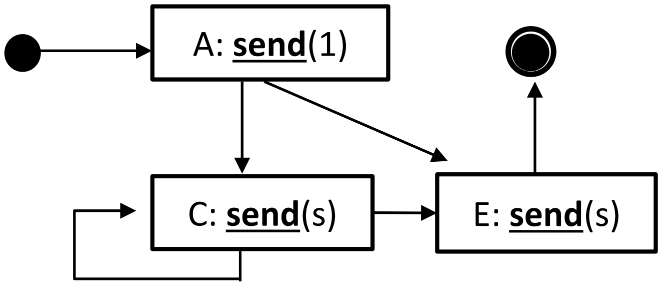
s = getc()

E: **send**(s)

D: ack = **recv()**

(ack != 1)?

SG - Send-Graph

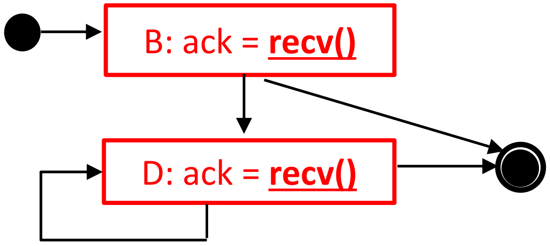


A: **send**(1)

C: **send**(s)

E: **send**(s)

RG - Receive-Graph

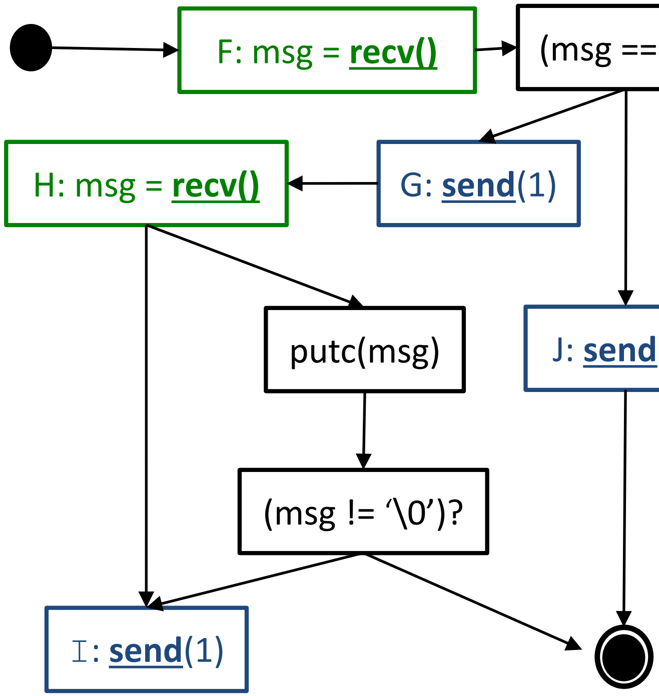


B: ack = **recv()**

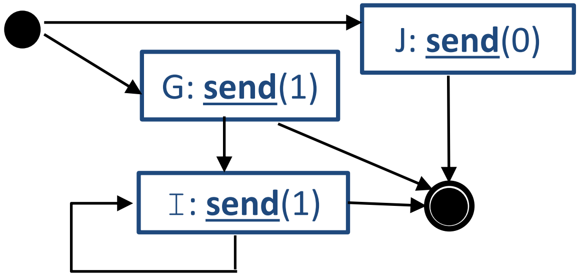
D: ack = **recv()**

Ditto for Echo Server

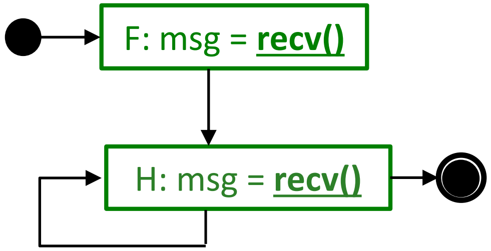
CFG - Echo Server



SG - Send-Graph

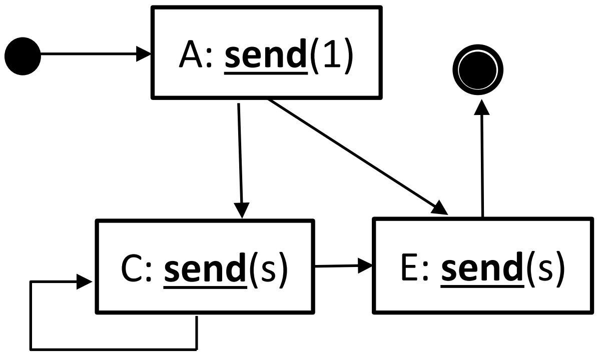


RG - Receive-Graph



Level Assignment

Echo Client: Send-Graph

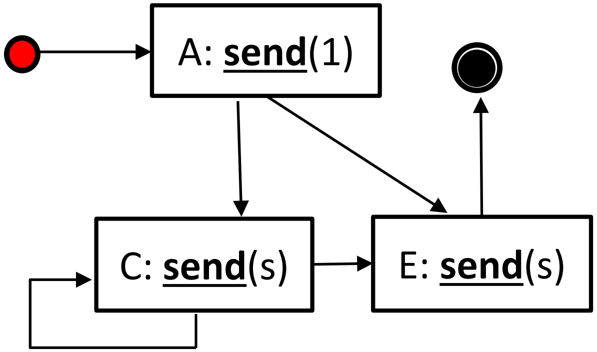


```
Elevator (Graph G)  /* toy version*/
    start.level := 0
    While { sets are different }
        For each vertex v in G
            If v is reachable from u
                then v.level := (u.level+1)
            end
        end
    end
end
```

$$\begin{aligned} level(mg, 0) &= \{start\} \\ level(mg, n) &= \{v \mid \overrightarrow{uv} \in mg \wedge u \in level(mg, n-1)\} \end{aligned}$$

Level Assignment

SG – Echo Client



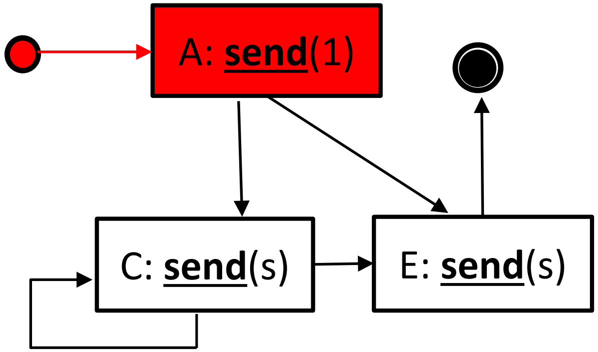
```
Elevator (Graph G)  /* toy version*/
    start.level := 0
    While { sets are different }
        For each vertex v in G
            If v is reachable from u
                then v.level := (u.level+1)
            end
        end
    end
end
```

Level of Senders:

level	*set*
0	**{ start }**

Level Assignment

Echo Client: Send-Graph



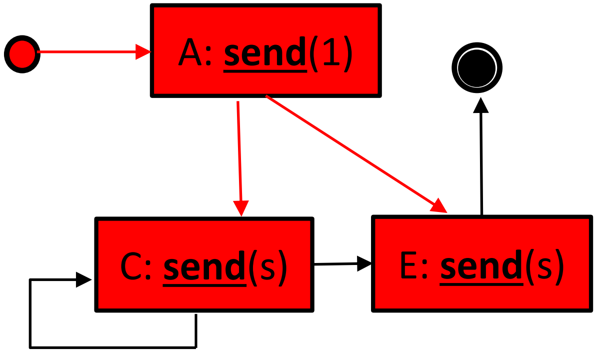
```
Elevator (Graph G)  /* toy version*/
    start.level := 0
    While { sets are different }
        For each vertex v in G
            If v is reachable from u
                then v.level := (u.level+1)
            end
        end
    end
end
```

Level of Senders:

level	*set*
0	**{ start }**
1	**{ A }**

Level Assignment

Echo Client: Send-Graph



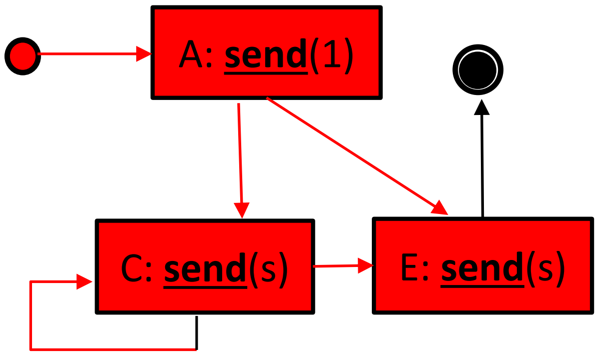
```
Elevator (Graph G)  /* toy version*/
    start.level := 0
    While { sets are different }
        For each vertex v in G
            If v is reachable from u
                then v.level := (u.level+1)
            end
        end
    end
end
```

Level of Senders:

level	*set*
0	**{ start }**
1	**{ A }**
2	**{ C, E }**

Level Assignment

Echo Client: Send-Graph



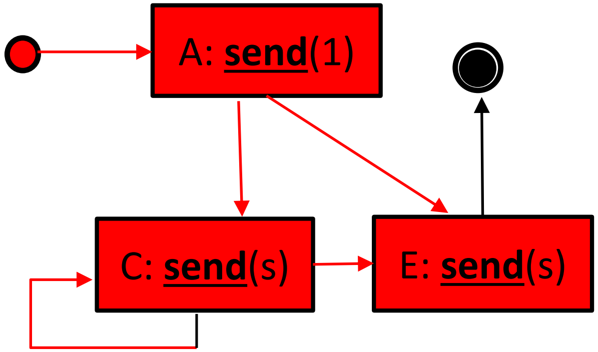
Level of Senders:

level	*set*
0	**{ start }**
1	**{ A }**
2	**{ C, E }**
3	**{ C, E }**

```
Elevator (Graph G)  /* toy version*/
        *start*.level := 0
        While { sets are different }
                For each vertex v in G
                        If *v* is reachable from *u*
                                then *v*.level := (*u*.level+1)
                        end
                end
        end
end
```

Level Assignment

Echo Client: Send-Graph



```
Elevator (Graph G)  /* toy version*/
        start.level := 0
        While { sets are different }
                For each vertex v in G
                        If v is reachable from u
                                then v.level := (u.level+1)
                        end
                end
        end
end
```

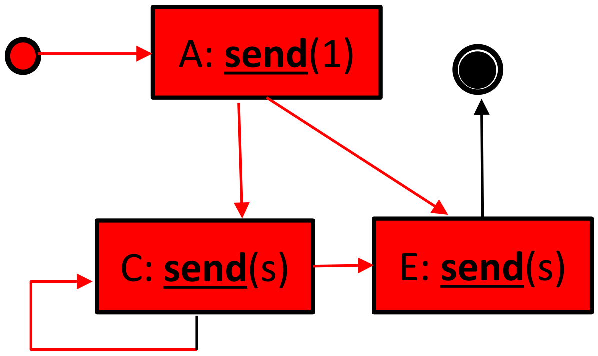
<u>Level of Senders:</u>

level	*set*
0	**{ start }**
1	**{ A }**
2	**{ C, E }**
3	**{ C, E }**

the algorithm halts whenever sets stop changing

Level Assignment

Echo Client: Send-Graph



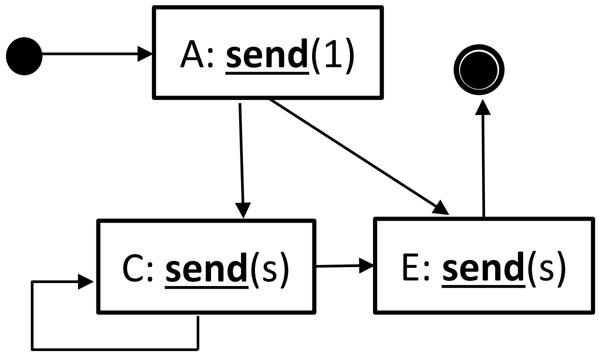
Level of Senders:

level	*set*
0	**{ start }**
1	**{ A }**
2	**{ C, E }**
3	**{ C, E }**

```
Elevator (Graph G)  /* toy version*/
        start.level := 0
        While { sets are different }
                For each vertex v in G
                        If v is reachable from u
                                then v.level := (u.level+1)
                        end
                end
        end
end
```

Level Assignment: Ditto for Recv-Graph

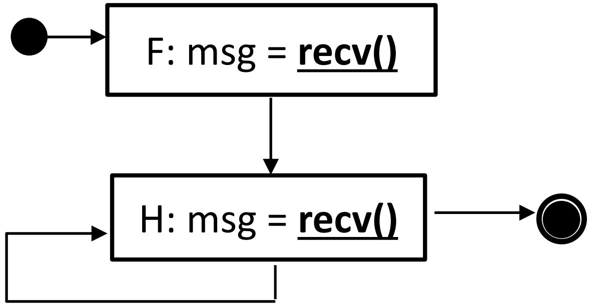
Echo Client: Send-Graph



Level of Senders:

level	*set*
0	**{ start }**
1	**{ A }**
2	**{ C, E }**
3	{ C, E }

Echo Server: Recv-Graph

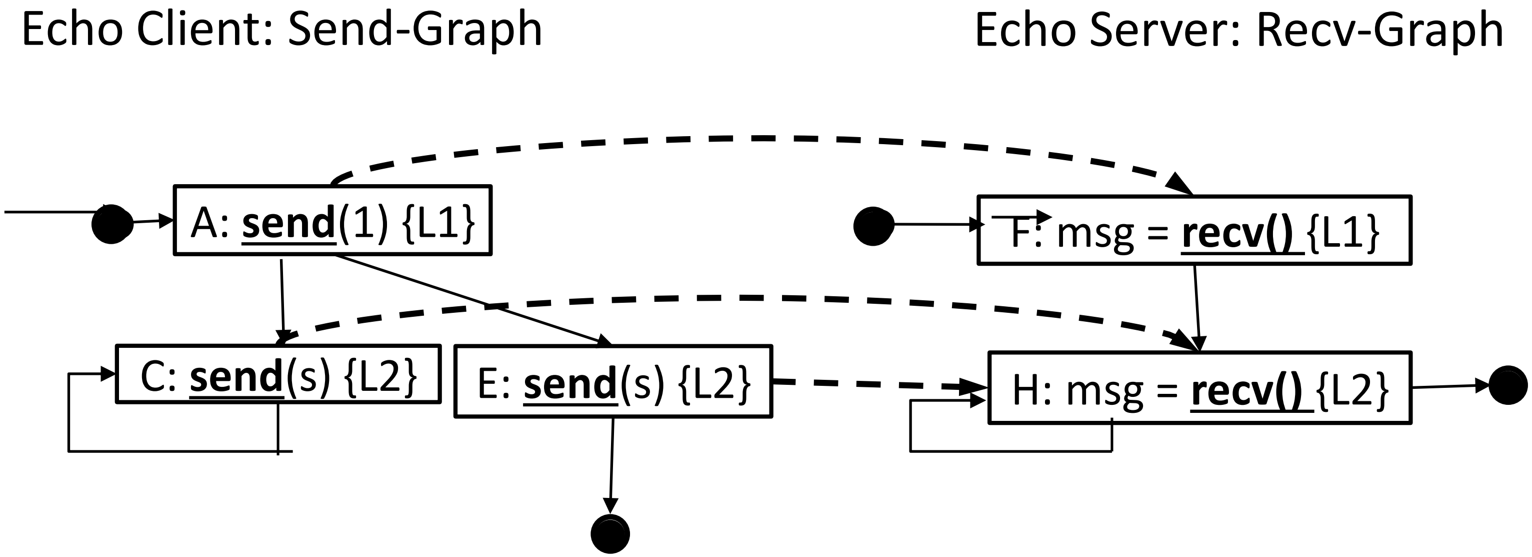


Level of Receivers:

level	*set*
0	**{ root }**
1	**{ F }**
2	**{ H }**
3	{ H }

DCFG construction final step

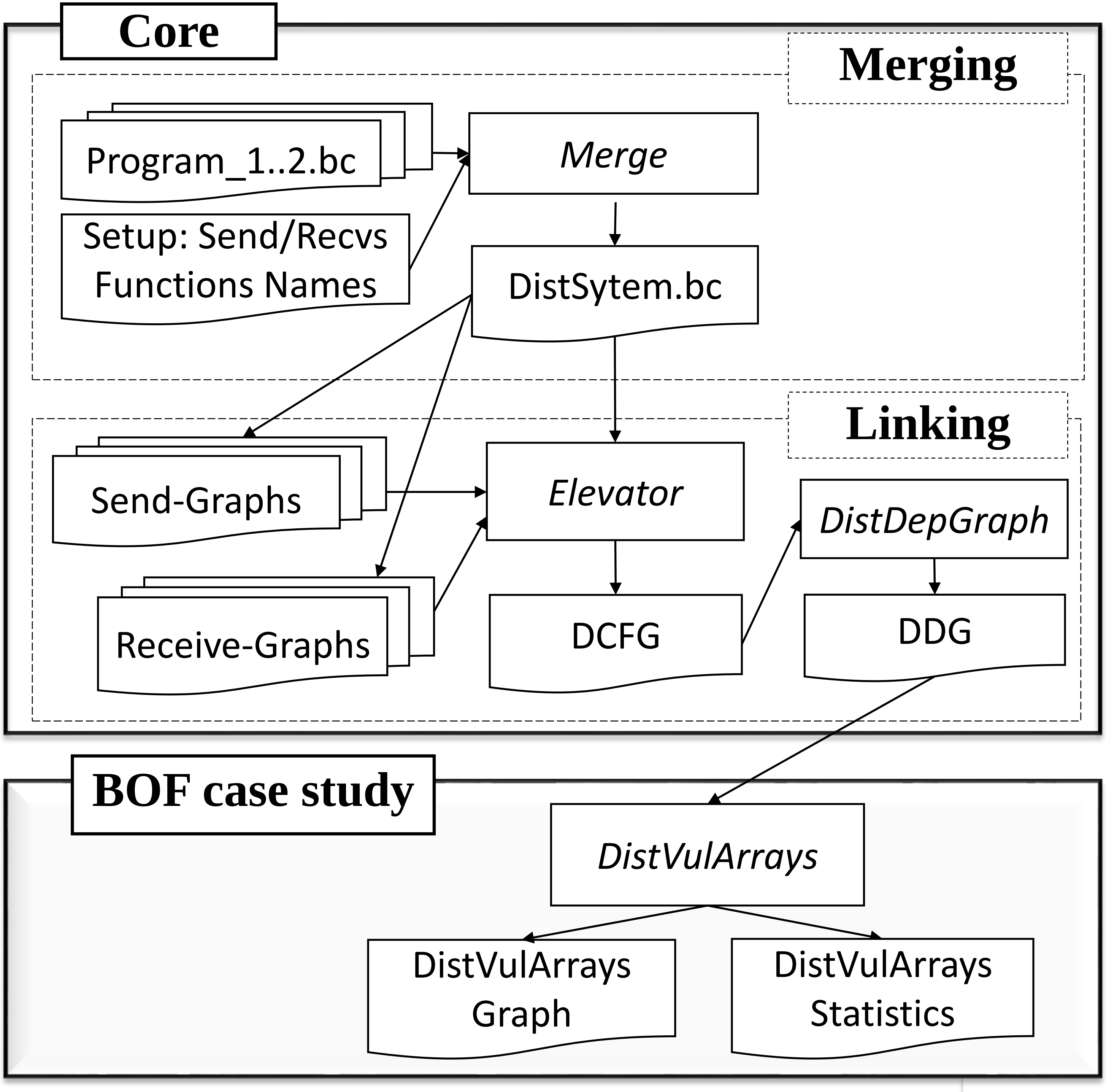
- Link Client’s `sends` with Server’s `recvs` if they are belong to the same level, and vice-versa



Agenda

- Introduction
- Goal
- Solution
  - Conception
  - Refinement
  - Development
- Results
- Conclusion

SIoT
Coding



Core

Merging

Program_1..2.bc

Setup: Send/Recvs
Functions Names

Merge

DistSytem.bc

Linking

Send-Graphs

Receive-Graphs

Elevator

DCFG

DistDepGraph

DDG

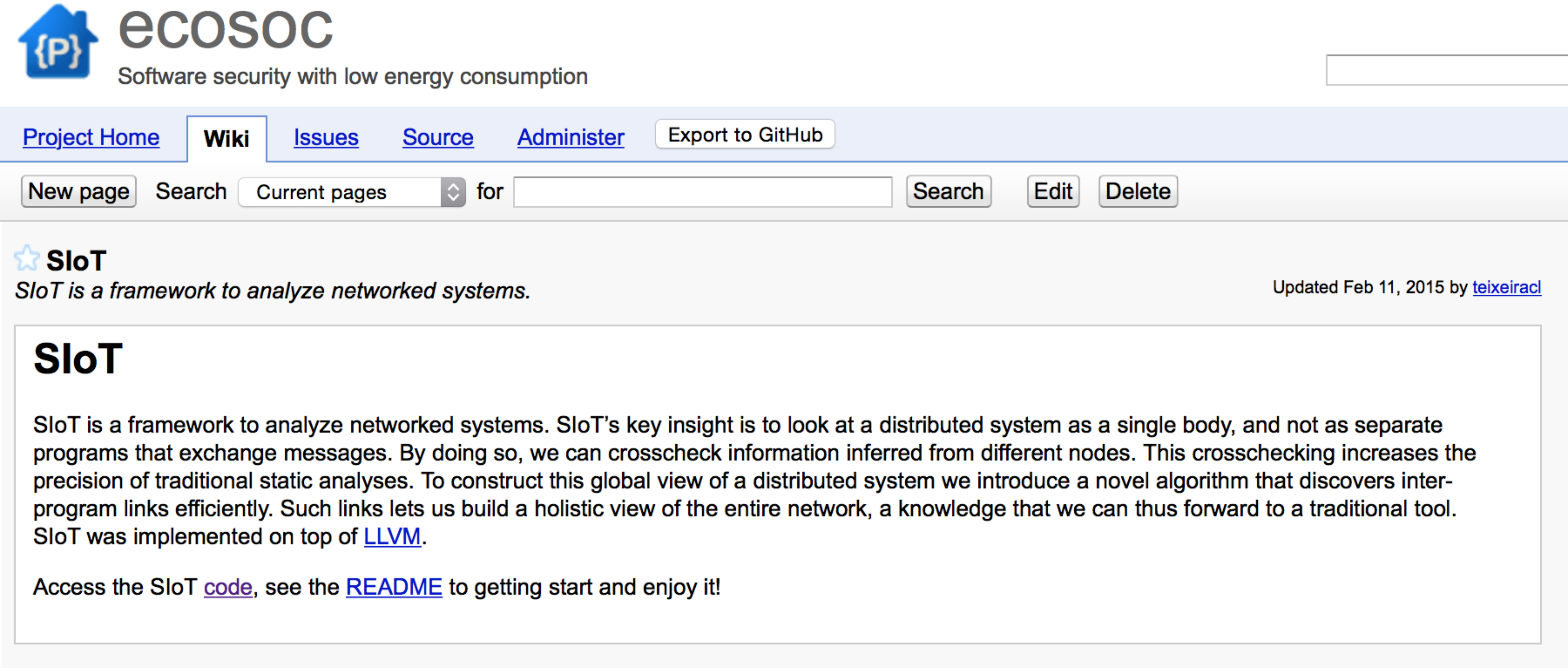
BOF case study

DistVulArrays

DistVulArrays
Graph

DistVulArrays
Statistics

SIoT is publicly available



SIoT

SIoT is a framework to analyze networked systems.

Updated Feb 11, 2015 by teixeiracl

SIoT

SIoT is a framework to analyze networked systems. SIoT's key insight is to look at a distributed system as a single body, and not as separate programs that exchange messages. By doing so, we can crosscheck information inferred from different nodes. This crosschecking increases the precision of traditional static analyses. To construct this global view of a distributed system we introduce a novel algorithm that discovers inter-program links efficiently. Such links lets us build a holistic view of the entire network, a knowledge that we can thus forward to a traditional tool. SIoT was implemented on top of LLVM.

Access the SIoT code, see the README to getting start and enjoy it!

https://code.google.com/p/ecosoc/wiki/SIoT

Agenda

- Introduction
- Goal
- Solution
- Results
- Conclusion

Evaluation

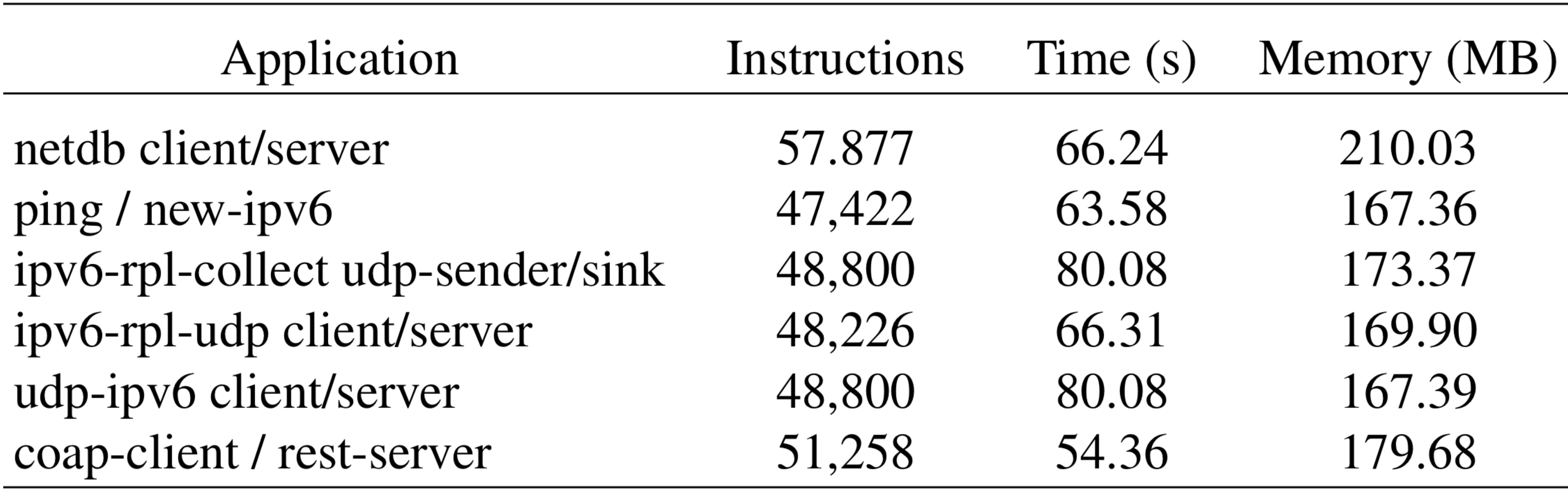
- We have compared SIoT against the state-of-the-art approach
  - Tainted flow analysis followed by ABCs insertion
  - We called this approach *Baseline*
  - Our hypothesis was that SIoT would insert less ABCs than Baseline and thus end up being more efficient
- We have used real IoT code in our evaluation
  - I.e., ContikiOS applications

The tado° smart thermostat and intelligent heating system

Contiki: The Open Source OS for the Internet of Things

SIoT Static Analysis

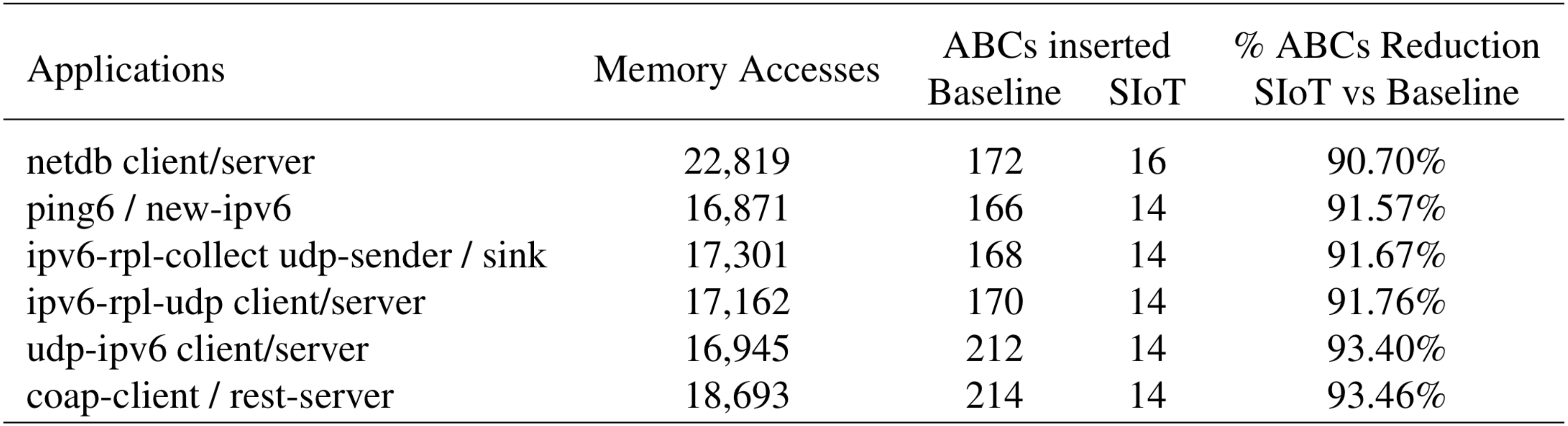
- It takes on average 66s and consumed 170 MB of RAM
  – In a Intel Core i7 2.2GHz laptop
  – Memory obtained via Valgrind
  – Time taken through Unix time
- It’s done offline and does not represent a burden to nodes



Application	Instructions	Time (s)	Memory (MB)
netdb client/server	57.877	66.24	210.03
ping / new-ipv6	47,422	63.58	167.36
ipv6-rpl-collect udp-sender/sink	48,800	80.08	173.37
ipv6-rpl-udp client/server	48,226	66.31	169.90
udp-ipv6 client/server	48,800	80.08	167.39
coap-client / rest-server	51,258	54.36	179.68

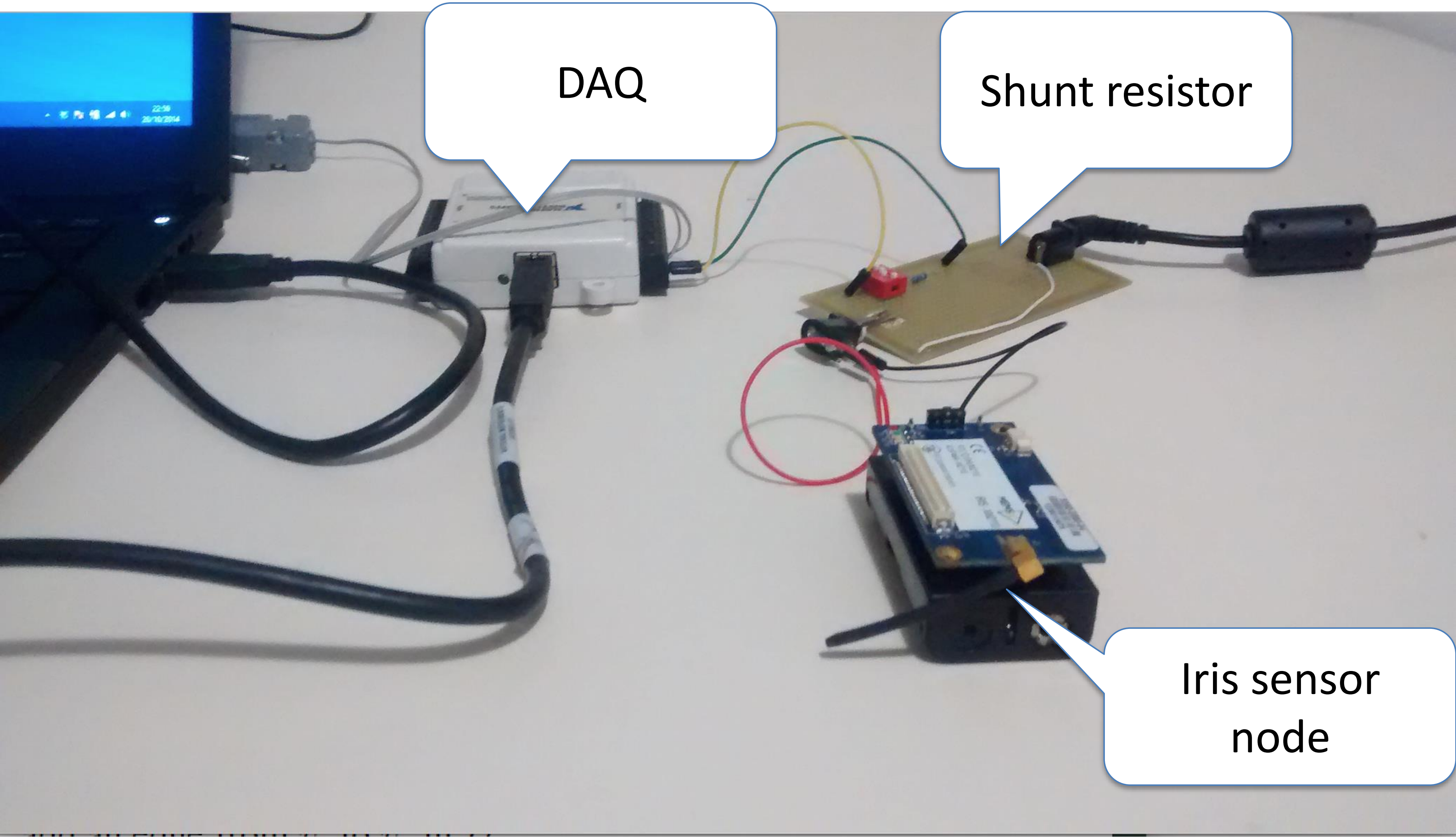
ABCs Insertion

- SIoT reduces the number of ABCs insertion by around 10x compared to Baseline in our benchmark



Applications	Memory Accesses	ABCs inserted		% ABCs Reduction
		Baseline	SIoT	SIoT vs Baseline
netdb client/server	22,819	172	16	90.70%
ping6 / new-ipv6	16,871	166	14	91.57%
ipv6-rpl-collect udp-sender / sink	17,301	168	14	91.67%
ipv6-rpl-udp client/server	17,162	170	14	91.76%
udp-ipv6 client/server	16,945	212	14	93.40%
coap-client / rest-server	18,693	214	14	93.46%

Experiment Setup



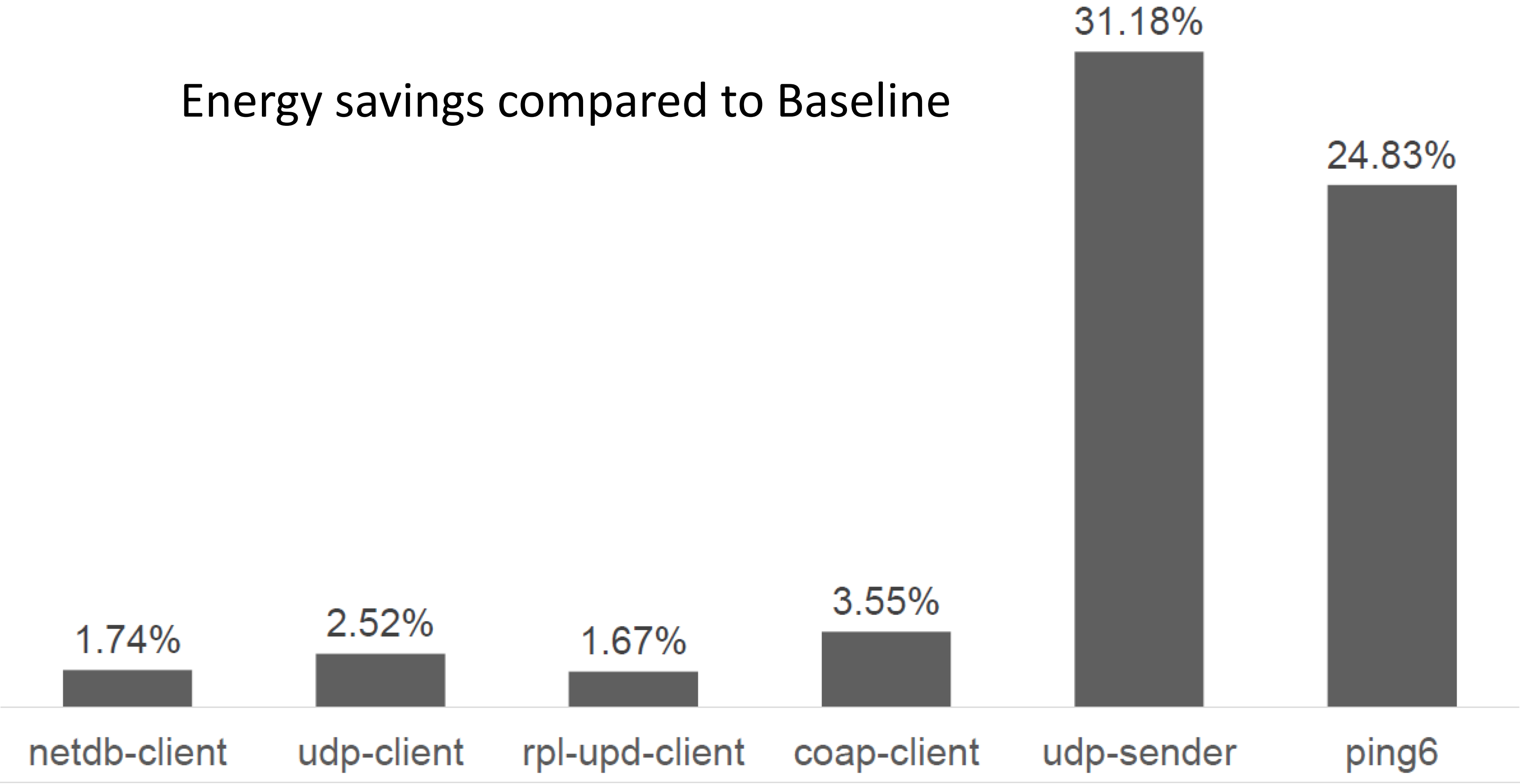
DAQ

Shunt resistor

Iris sensor node

SIoT Dynamic Analysis

Energy savings compared to Baseline



Agenda

- Introduction
- Goal
- Solution
- Results
- Conclusion

Conclusions

SIoT

1. Sees and analyze individuals programs of a distributed system as a single system
2. Protects IoT code around 20% more energy-efficiently than state-of-the-art approach
3. Is publicly available

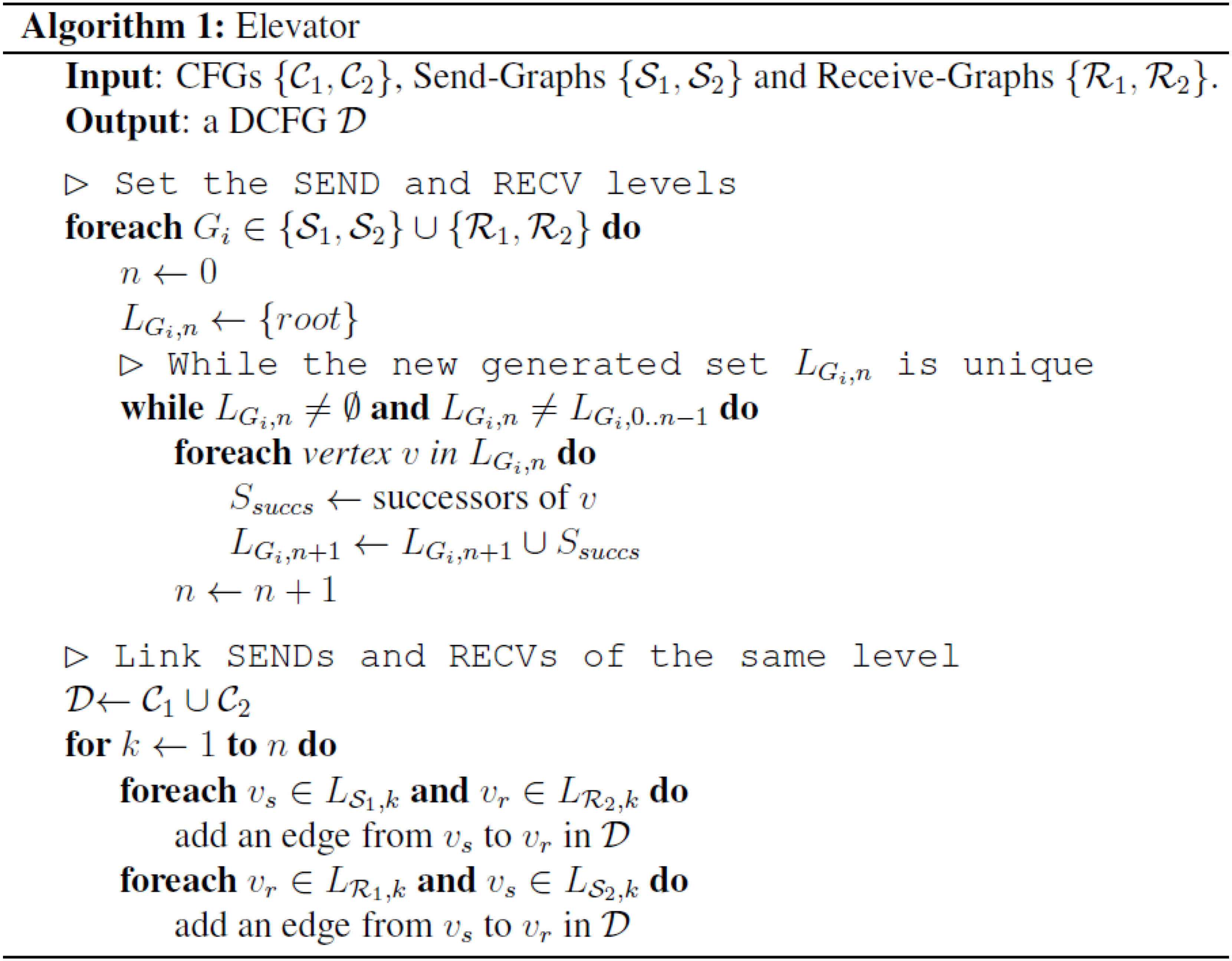
Besides, we came up with

1. Distributed Control Flow Graphs – DCFGs
2. Elevator, an alg. that is able to link `sends/recvs`

Thanks

www.ecosoc.dcc.ufmg.br

leonardo.barbosa@dcc.ufmg.br

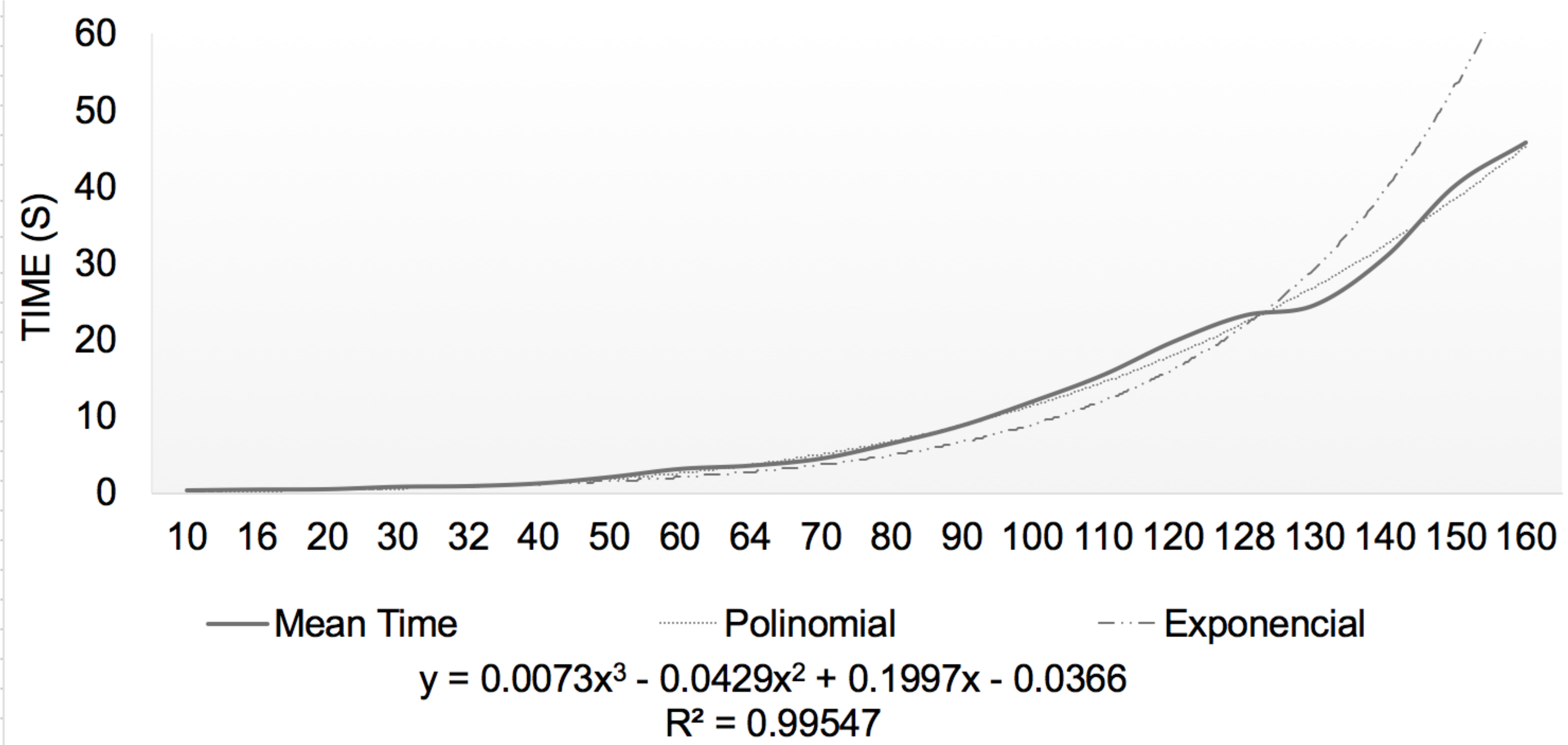


Algorithm 1: Elevator

```
Input: CFGs $\{\mathcal{C}_1, \mathcal{C}_2\}$, Send-Graphs $\{\mathcal{S}_1, \mathcal{S}_2\}$ and Receive-Graphs $\{\mathcal{R}_1, \mathcal{R}_2\}$.
Output: a DCFG $\mathcal{D}$

▷ Set the SEND and RECV levels
foreach $G_i \in \{\mathcal{S}_1, \mathcal{S}_2\} \cup \{\mathcal{R}_1, \mathcal{R}_2\}$ do
    $n \leftarrow 0$
    $L_{G_i,n} \leftarrow \{root\}$
    ▷ While the new generated set $L_{G_i,n}$ is unique
    while $L_{G_i,n} \neq \emptyset$ and $L_{G_i,n} \neq L_{G_i,0..n-1}$ do
        foreach vertex $v$ in $L_{G_i,n}$ do
            $S_{succs} \leftarrow$ successors of $v$
            $L_{G_i,n+1} \leftarrow L_{G_i,n+1} \cup S_{succs}$
        $n \leftarrow n + 1$

▷ Link SENDs and RECVs of the same level
$\mathcal{D} \leftarrow \mathcal{C}_1 \cup \mathcal{C}_2$
for $k \leftarrow 1$ to $n$ do
    foreach $v_s \in L_{\mathcal{S}_1,k}$ and $v_r \in L_{\mathcal{R}_2,k}$ do
        add an edge from $v_s$ to $v_r$ in $\mathcal{D}$
    foreach $v_r \in L_{\mathcal{R}_1,k}$ and $v_s \in L_{\mathcal{S}_2,k}$ do
        add an edge from $v_s$ to $v_r$ in $\mathcal{D}$
```

Elevator Asymptotic Complexity

- Runtime as a function of the number of CGFs vertices
- In practice, O(*n^3*) where *n* is the number of vertices



Related Work

- There are works that already focused on IoT software correctness and security
  - Cooprider et al. Safe TinyOS (Sensys'07)
  - Li and Regehr. Kleenet. (IPSN'10)
  - Sasnauskas et al. T-Check. (IPSN'10)
- These works don’t look at IoT as a single system and we believe ours is complementary to their strategies
  - I.e., SIoT can potentially improve their numbers

ABC Cost

ABCs' Computational Cost

buffer[i] = a;

```
mrmovl -12(%ebp), %eax     Load 'i'
mrmovl -4(%ebp), %edx
rrmovl %eax, %edi
sall $2, %edi
addl %ebp, %edi
rmmovl %edx, -2060(%edi)
```

ABCs’ Computational Cost

buffer[i] = a;

```
mrmovl -12(%ebp), %eax     Load ‘i’
mrmovl -4(%ebp), %edx      Load ‘a’
rrmovl %eax, %edi
sall $2, %edi
addl %ebp, %edi
rmmovl %edx, -2060(%edi)
```

ABCs' Computational Cost

buffer[i] = a;

```
mrmovl -12(%ebp), %eax      Load 'i'
mrmovl -4(%ebp), %edx       Load 'a'
rrmovl %eax, %edi
sall $2, %edi               Adjust base register
addl %ebp, %edi
rmmovl %edx, -2060(%edi)
```

ABCs' Computational Cost

```
                 mrmovl -12(%ebp), %eax       Load 'i'
                 mrmovl -4(%ebp), %edx        Load 'a'
                 rrmovl %eax, %edi
buffer[i] = a;   sall $2, %edi                Adjust base register
                 addl %ebp, %edi
                 rmmovl %edx, -2060(%edi)     Move
```

ABCs’ Computational Cost

if((i>= 0)	mrmovl -12(%ebp), %edi	Load ‘i’
	irmovl $0, %ebx	
	subl %ebx, %edi	
	js L3	
(i < BUFFERSIZE))	mrmovl -12(%ebp),%edi	Load ‘i’
	irmovl $511, %ebx	
	subl %ebx, %edi	
	jg L3	
buffer[i] = a;	mrmovl -12(%ebp), %eax	Load ‘i’
	mrmovl -4(%ebp), %edx	Load ‘a’
	rrmovl %eax, %edi	
	sall $2, %edi	Adjust base register
	addl %ebp, %edi	
	rmmovl %edx, -2060(%edi)	Move

ABCs’ Computational Cost

if((i>= 0)	mrmovl -12(%ebp), %edi	Load ‘i’
	irmovl $0, %ebx	Load lower index
	subl %ebx, %edi	
	js L3	
(i < BUFFERSIZE))	mrmovl -12(%ebp),%edi	Load ‘i’
	irmovl $511, %ebx	Load upper index
	subl %ebx, %edi	
	jg L3	
buffer[i] = a;	mrmovl -12(%ebp), %eax	Load ‘i’
	mrmovl -4(%ebp), %edx	Load ‘a’
	rrmovl %eax, %edi	
	sall $2, %edi	Adjust base register
	addl %ebp, %edi	
	rmmovl %edx, -2060(%edi)	Move

ABCs' Computational Cost

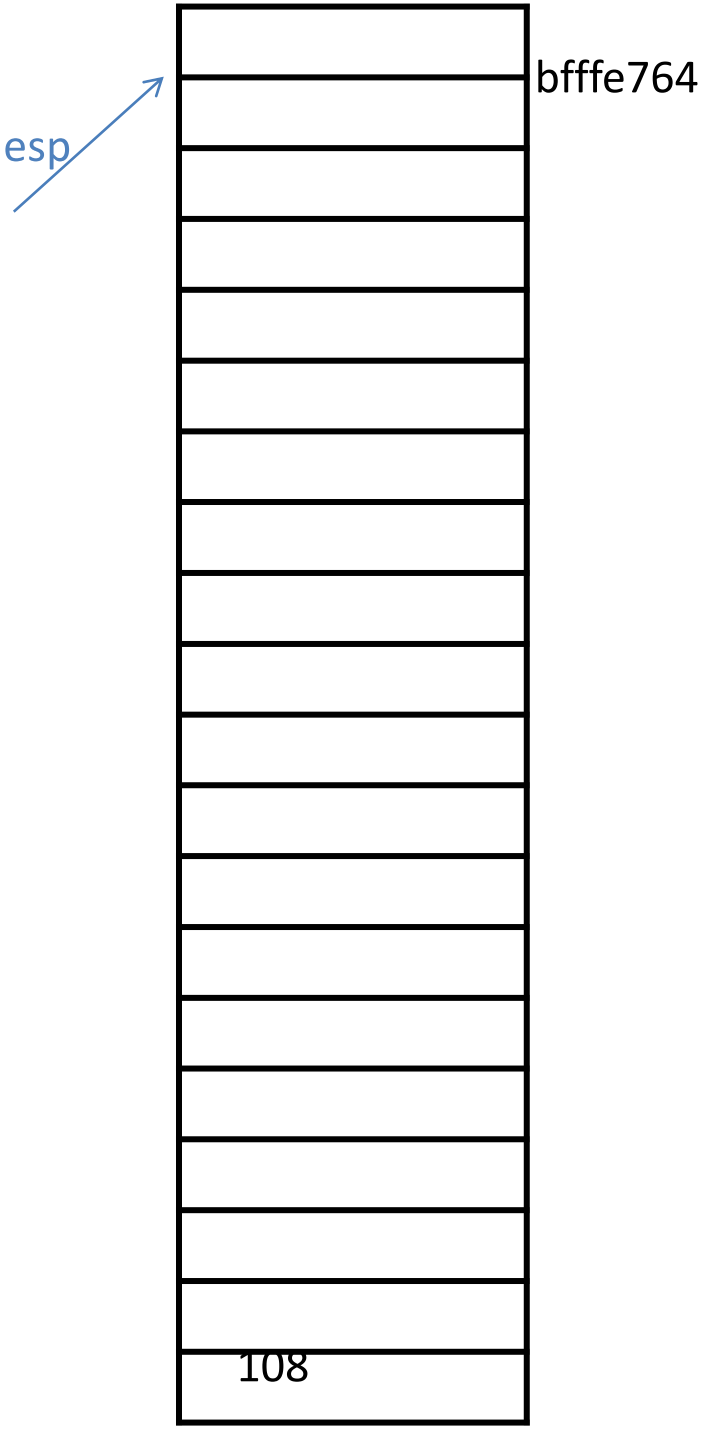
```
                       mrmovl -12(%ebp), %edi      Load 'i'
                       irmovl $0, %ebx             Load lower index
if( (i>= 0)            subl %ebx, %edi
                       js L3                       Compare

                       mrmovl -12(%ebp),%edi       Load 'i'
                       irmovl $511, %ebx           Load upper index
(i < BUFFERSIZE) )     subl %ebx, %edi
                       jg L3                       Compare

                       mrmovl -12(%ebp), %eax      Load 'i'
                       mrmovl -4(%ebp), %edx       Load 'a'
                       rrmovl %eax, %edi
buffer[i] = a;         sall $2, %edi               Adjust base register
                       addl %ebp, %edi
                       rmmovl %edx, -2060(%edi)    Move
```

Full BOF example

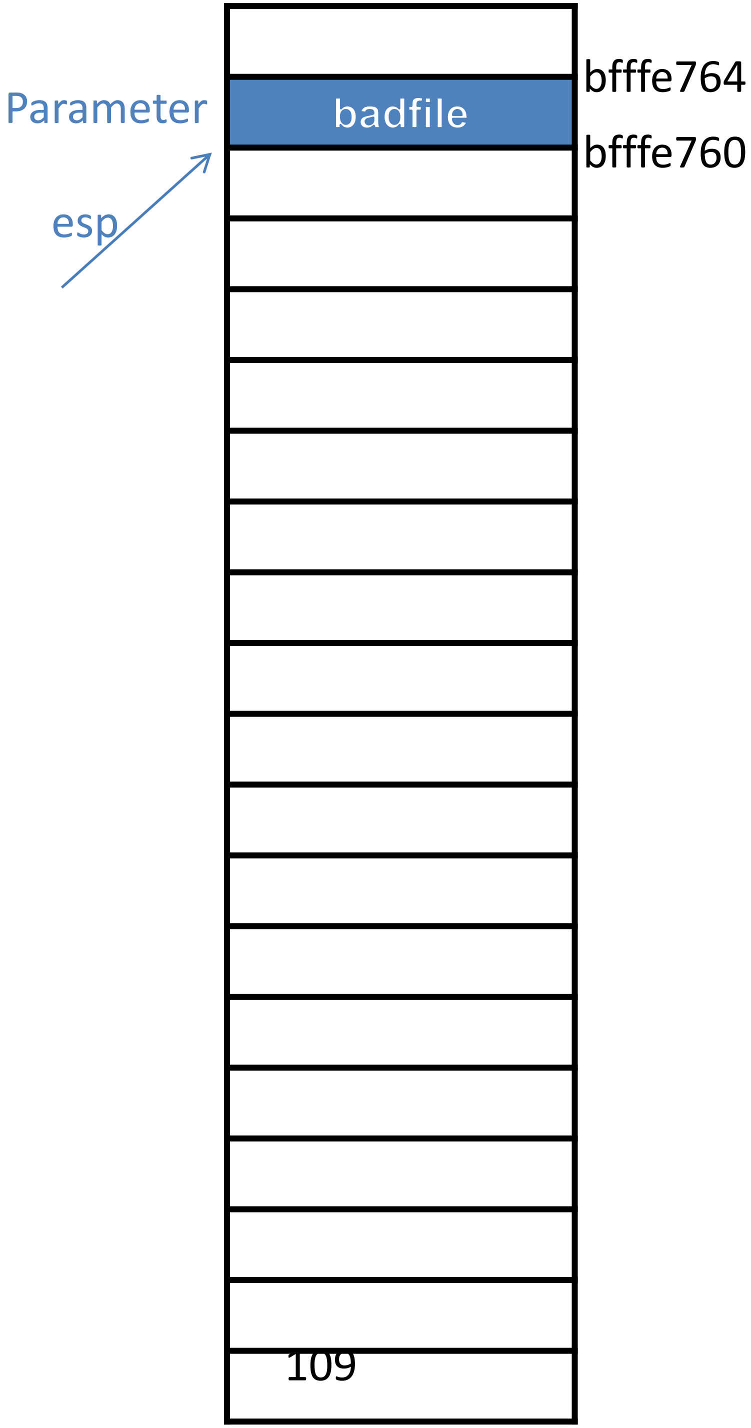
Vulnerable Code

```
#include <stdio.h>
void foo(FILE *badfile){
   char buffer[12];
   ...
   fread(buffer,sizeof(char),517,badfile);
   ...
   return 1;
}

int main() {
   FILE *badfile;
   badfile = fopen(“file”,”r”);
   foo(badfile);
   fclose(badfile);
   return 1;
}
```

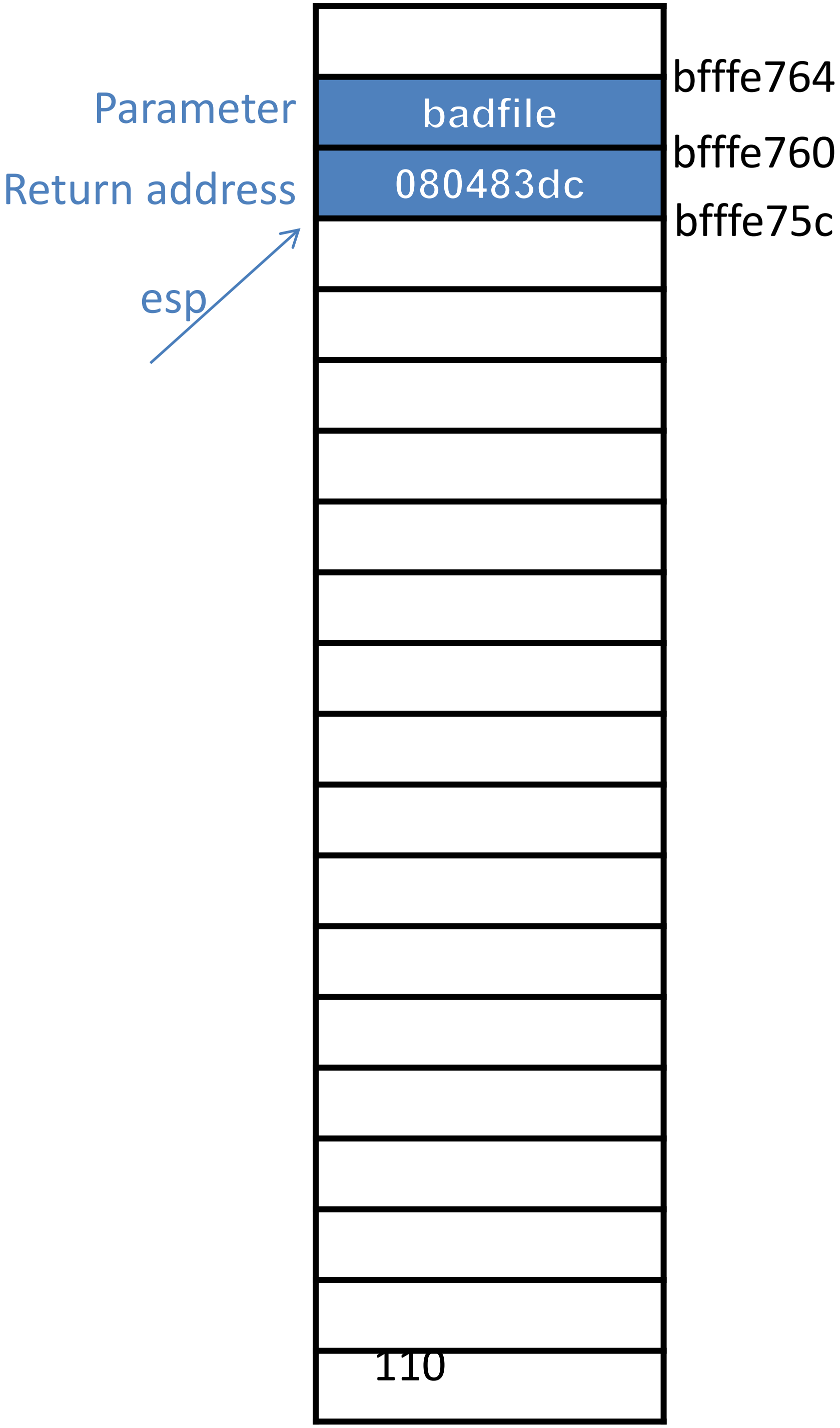
```
foo:
    pushl   %ebp
    movl %esp, %ebp
    subl $40, %esp
    ...
    movl 8(%ebp), %edx
    movl %edx, 12(%esp)
    movl $517, 8(%esp)
    movl $1, 4(%esp)
    movl %eax, (%esp)
    call fread
    movl $1, %eax
    leave
    ret
main:
    ...
    movl %eax, (%esp)
    call foo
    ...
    $1, %eax
    leave
    ret
```



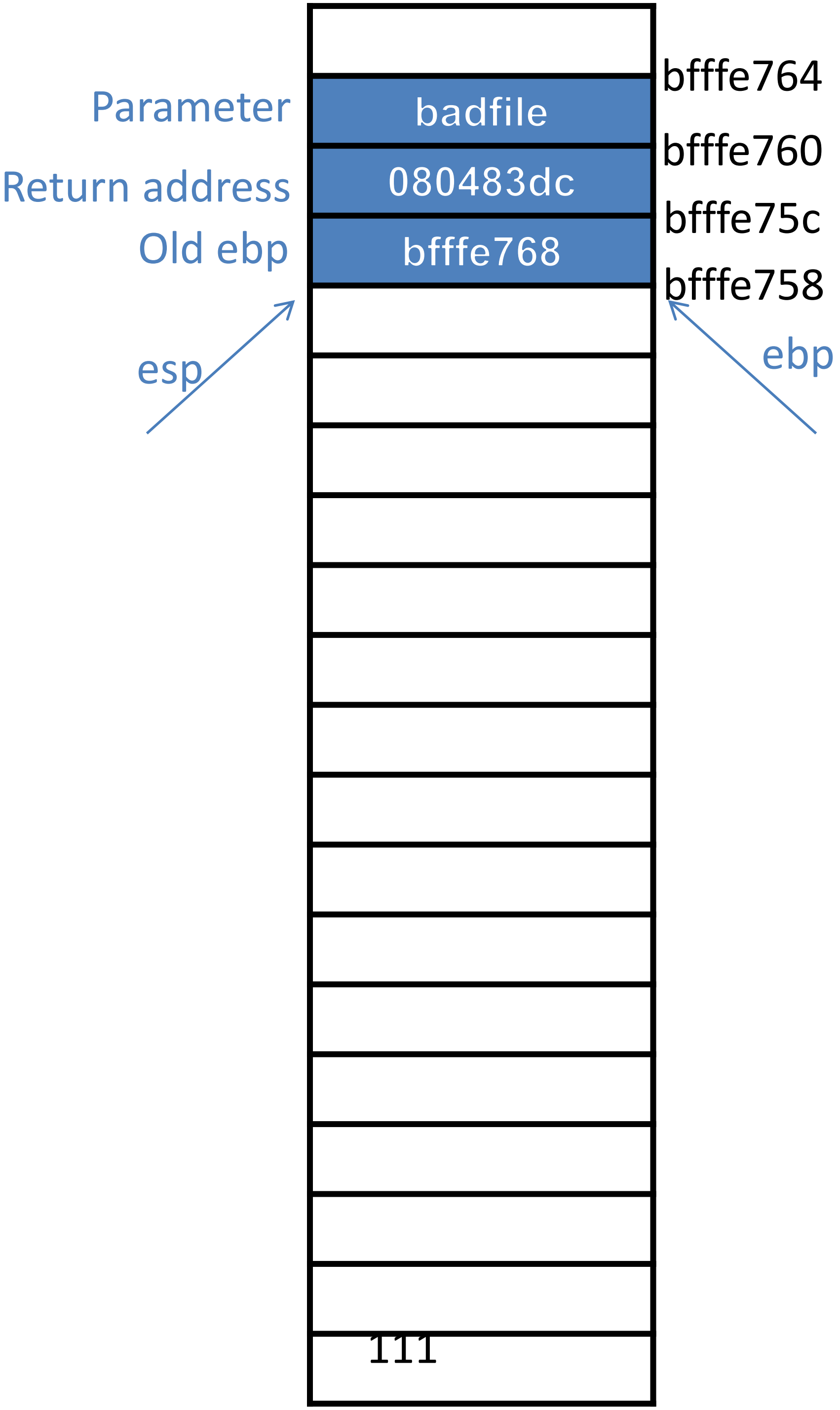
```
foo:
    pushl   %ebp
    movl %esp, %ebp
    subl $40, %esp
    ...
    movl 8(%ebp), %edx
    movl %edx, 12(%esp)
    movl $517, 8(%esp)
    movl $1, 4(%esp)
    movl %eax, (%esp)
    call fread
    movl $1, %eax
    leave
    ret
main:
    ...
    movl %eax, (%esp)
    call foo
    ...
    $1, %eax
    leave
    ret
```



```
foo:
    pushl   %ebp
    movl %esp, %ebp
    subl $40, %esp
    ...
    movl 8(%ebp), %edx
    movl %edx, 12(%esp)
    movl $517, 8(%esp)
    movl $1, 4(%esp)
    movl %eax, (%esp)
    call fread
    movl $1, %eax
    leave
    ret
main:
    ...
    movl %eax, (%esp)
    call foo
    ...
    $1, %eax
    leave
    ret
```



```
foo:
     pushl    %ebp
     movl %esp, %ebp
     subl $40, %esp
     ...
     movl 8(%ebp), %edx
     movl %edx, 12(%esp)
     movl $517, 8(%esp)
     movl $1, 4(%esp)
     movl %eax, (%esp)
     call fread
     movl $1, %eax
     leave
     ret
main:
     ...
     movl %eax, (%esp)
     call foo
     ...
     $1, %eax
     leave
     ret
```



bfffe764

Parameter badfile

bfffe760

Return address 080483dc

bfffe75c

Old ebp bfffe768

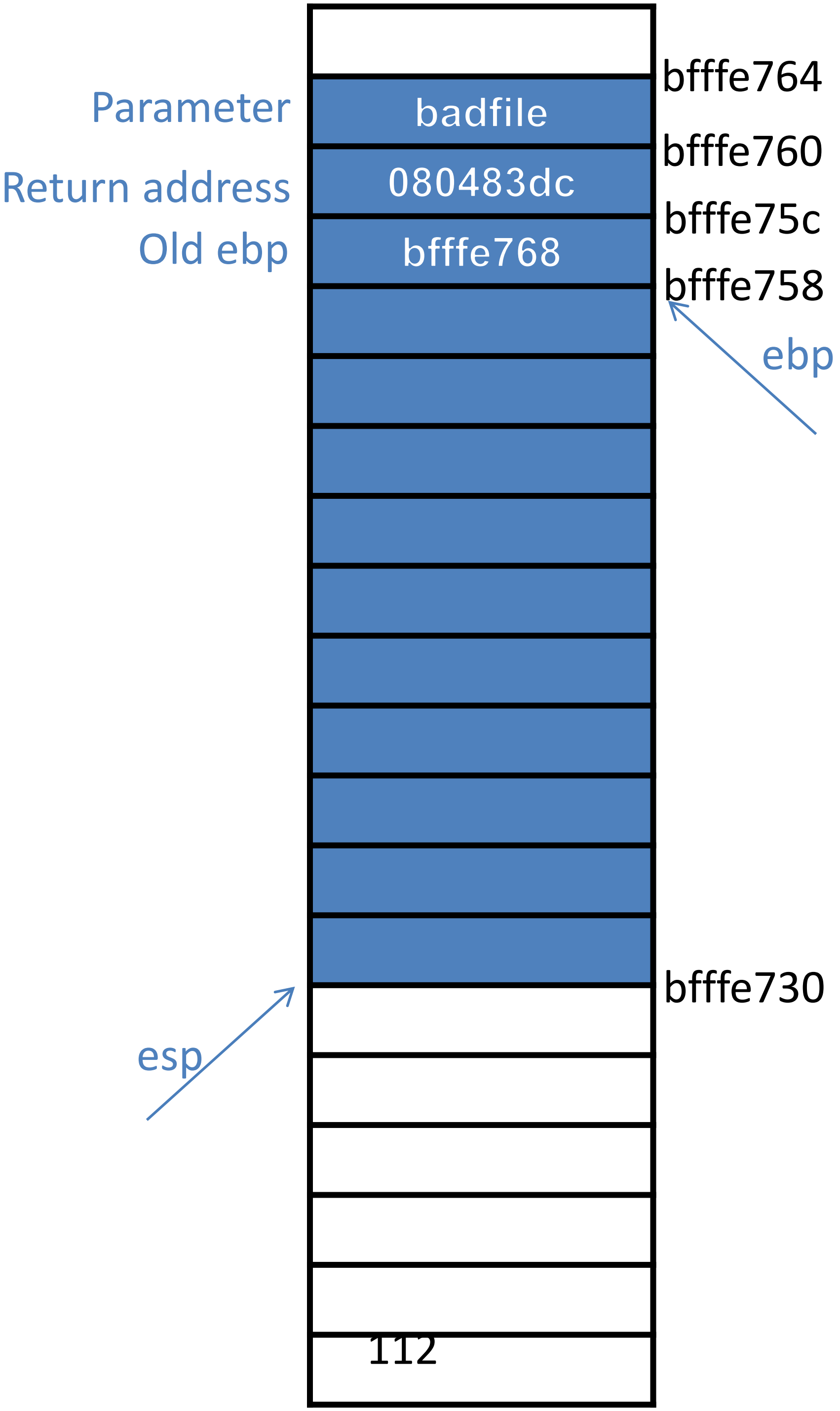
bfffe758

esp

ebp

111

```
foo:
    pushl   %ebp
    movl %esp, %ebp
    subl $40, %esp
    ...
    movl 8(%ebp), %edx
    movl %edx, 12(%esp)
    movl $517, 8(%esp)
    movl $1, 4(%esp)
    movl %eax, (%esp)
    call fread
    movl $1, %eax
    leave
    ret
main:
    ...
    movl %eax, (%esp)
    call foo
    ...
    $1, %eax
    leave
    ret
```



Parameter

Return address

Old ebp

badfile

080483dc

bfffe768

bfffe764

bfffe760

bfffe75c

bfffe758

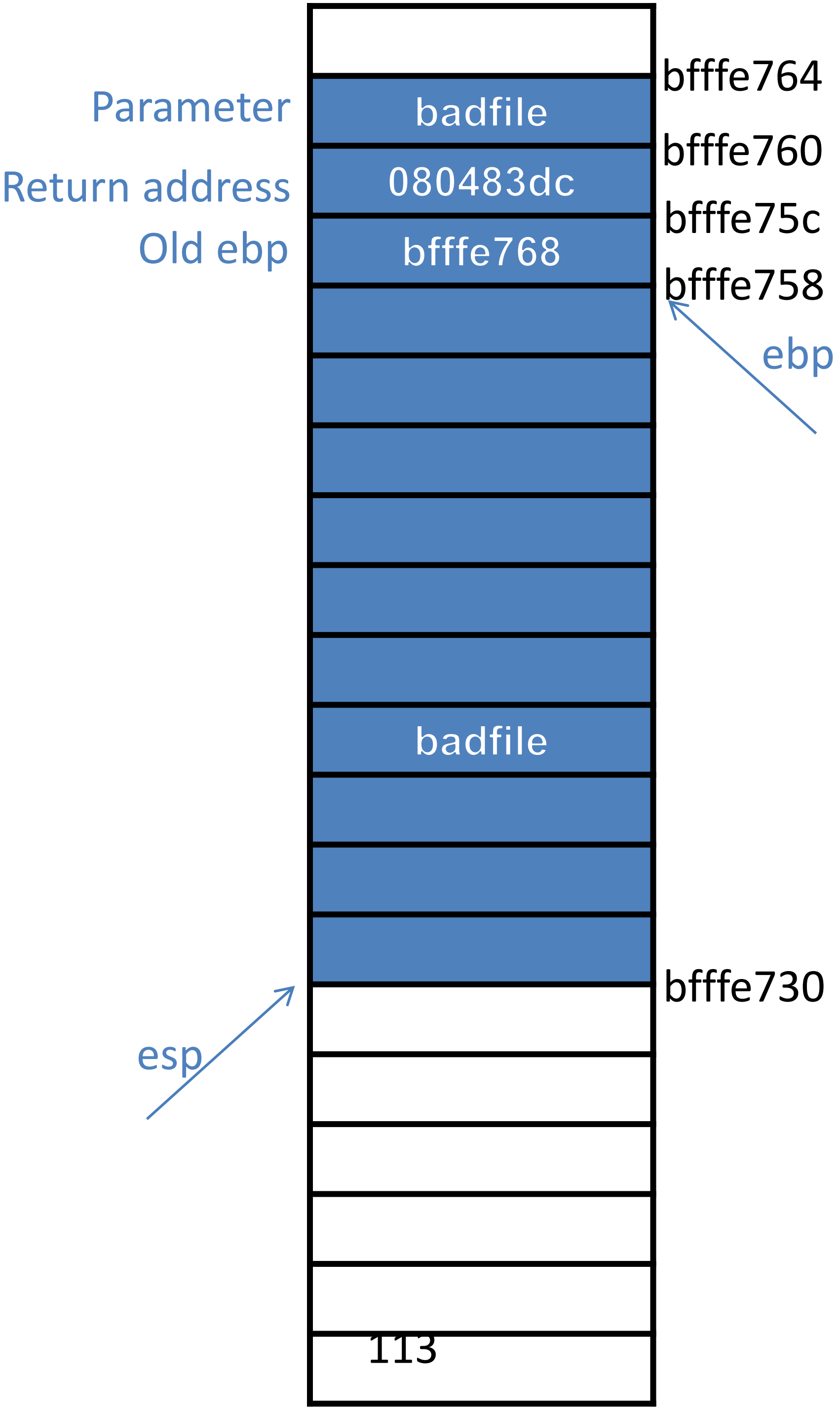
ebp

bfffe730

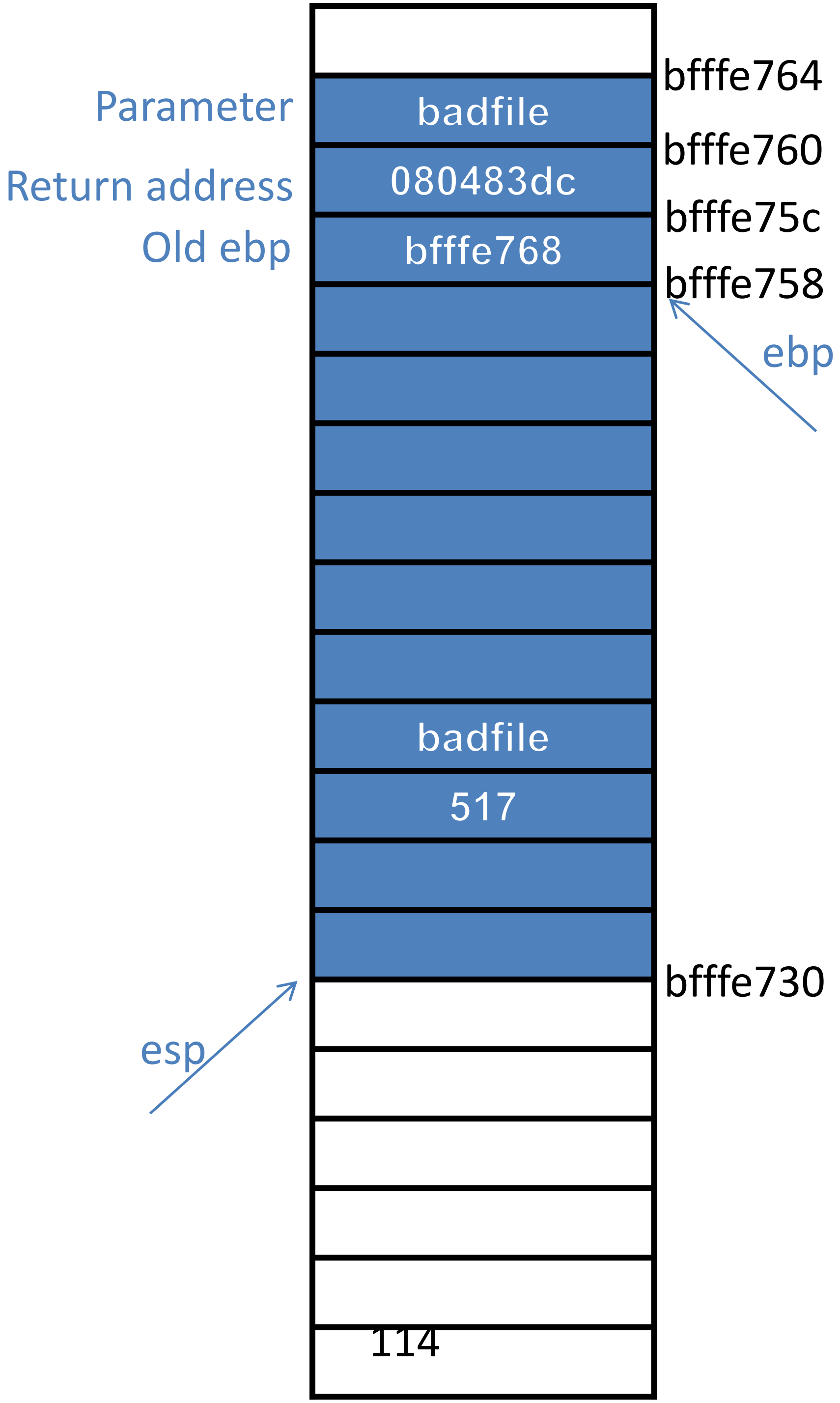
esp

112

```
foo:
    pushl    %ebp
    movl %esp, %ebp
    subl $40, %esp
    ...
    movl 8(%ebp), %edx
    movl %edx, 12(%esp)
    movl $517, 8(%esp)
    movl $1, 4(%esp)
    movl %eax, (%esp)
    call fread
    movl $1, %eax
    leave
    ret
main:
    ...
    movl %eax, (%esp)
    call foo
    ...
    $1, %eax
    leave
    ret
```



```
foo:
    pushl   %ebp
    movl %esp, %ebp
    subl $40, %esp
    ...
    movl 8(%ebp), %edx
    movl %edx, 12(%esp)
    movl $517, 8(%esp)
    movl $1, 4(%esp)
    movl %eax, (%esp)
    call fread
    movl $1, %eax
    leave
    ret
main:
    ...
    movl %eax, (%esp)
    call foo
    ...
    $1, %eax
    leave
    ret
```



bfffe764

Parameter badfile

bfffe760

Return address 080483dc

bfffe75c

Old ebp bfffe768

bfffe758

ebp

badfile

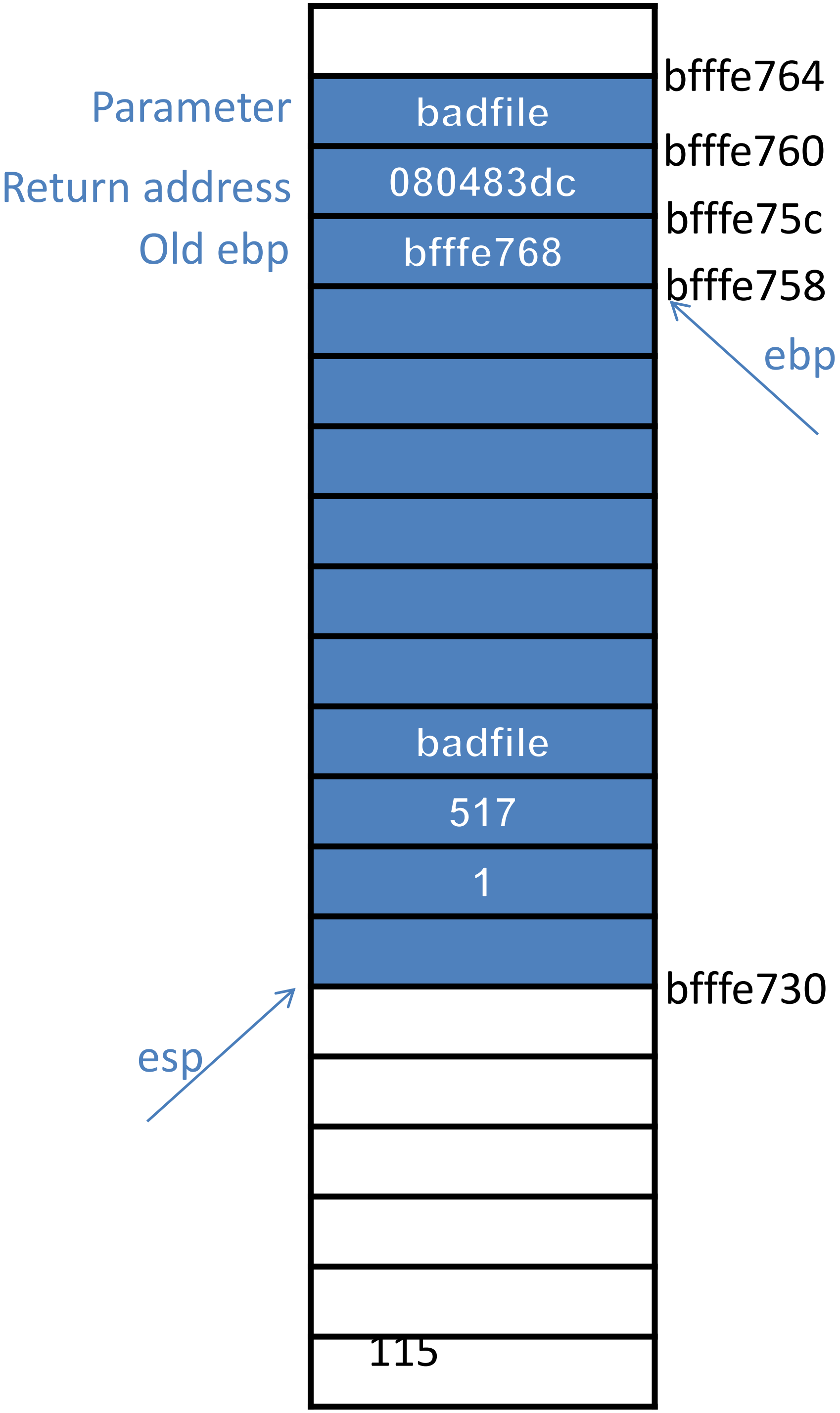
517

bfffe730

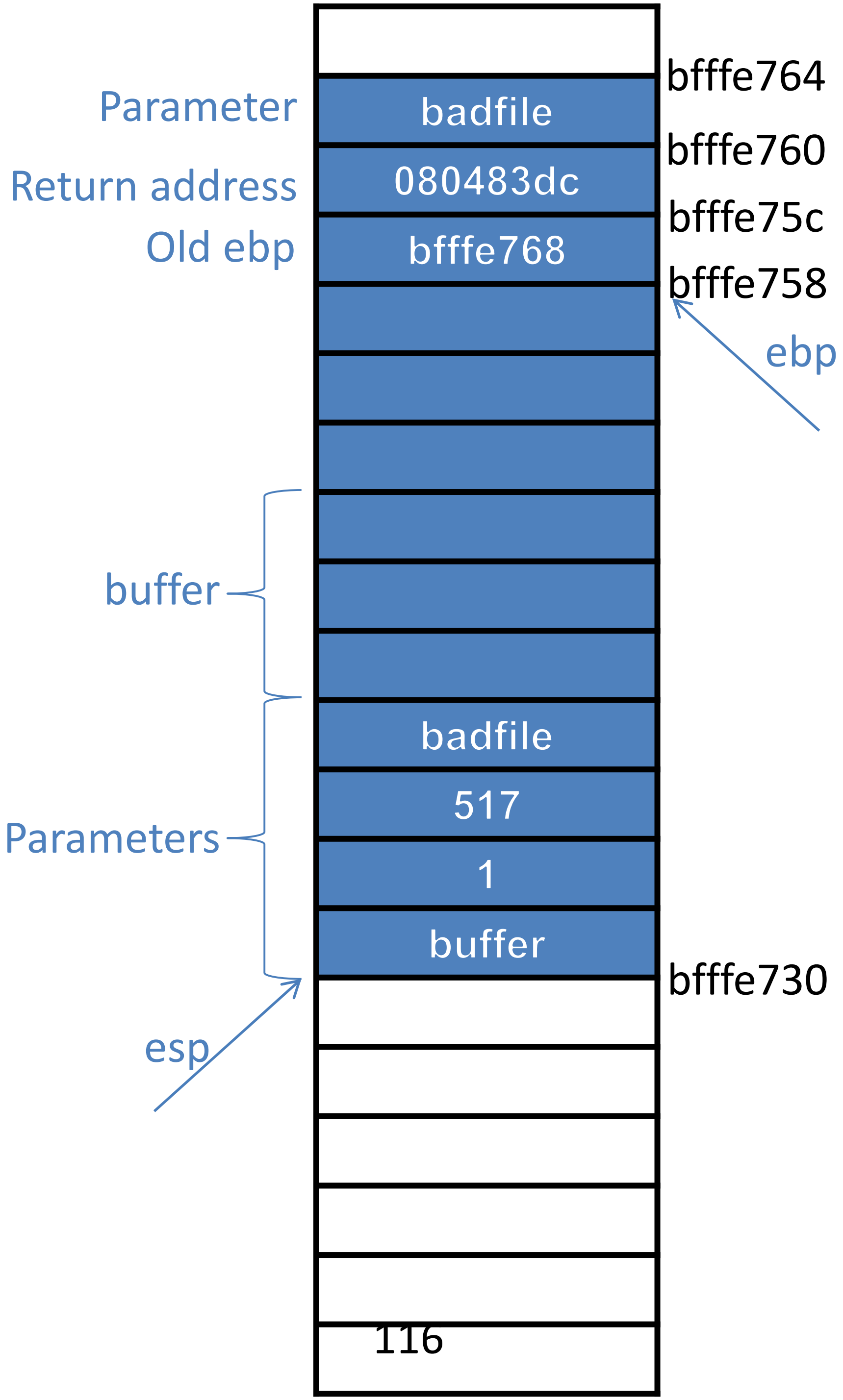
esp

114

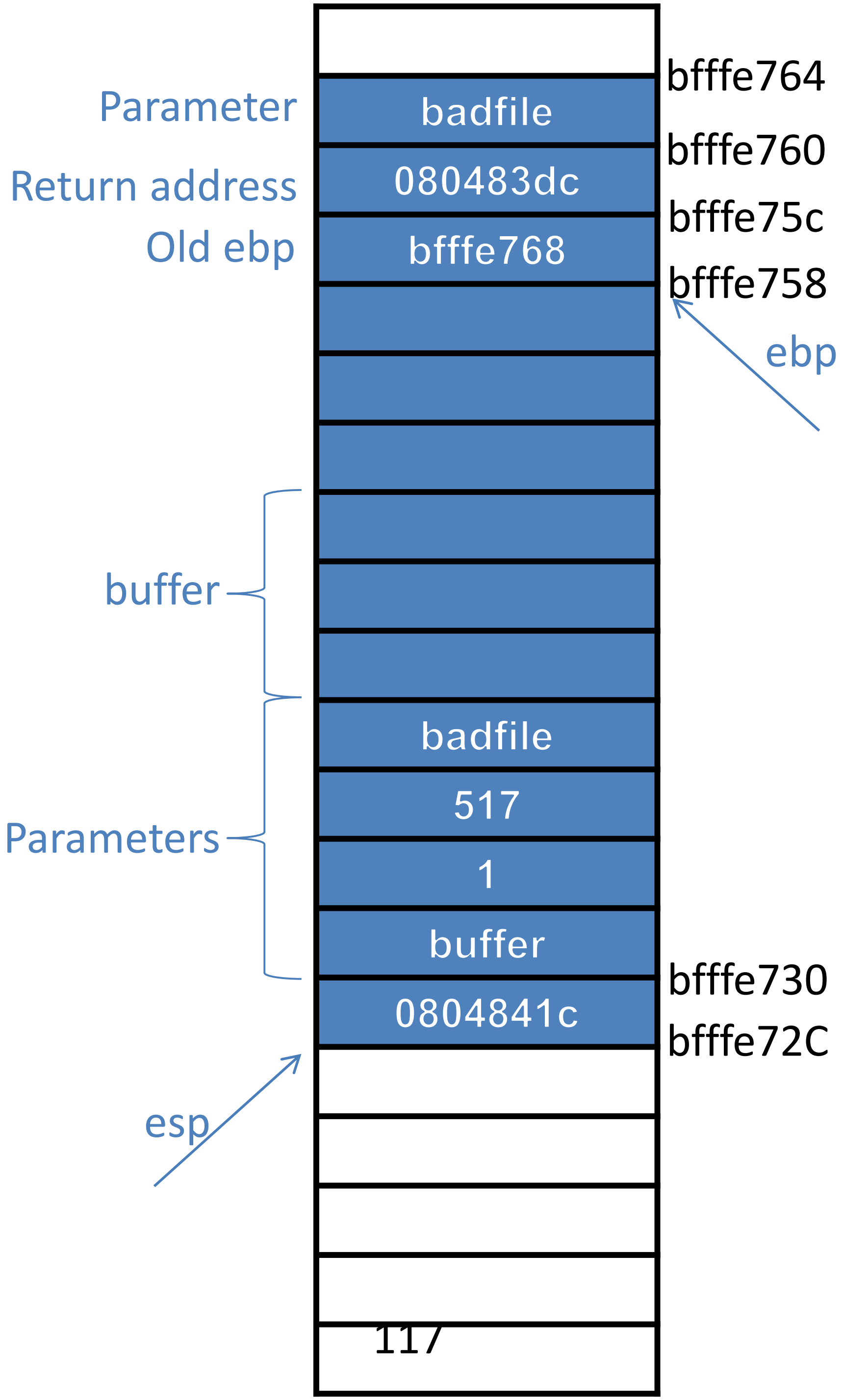
```
foo:
    pushl   %ebp
    movl %esp, %ebp
    subl $40, %esp
    ...
    movl 8(%ebp), %edx
    movl %edx, 12(%esp)
    movl $517, 8(%esp)
    movl $1, 4(%esp)
    movl %eax, (%esp)
    call fread
    movl $1, %eax
    leave
    ret
main:
    ...
    movl %eax, (%esp)
    call foo
    ...
    $1, %eax
    leave
    ret
```



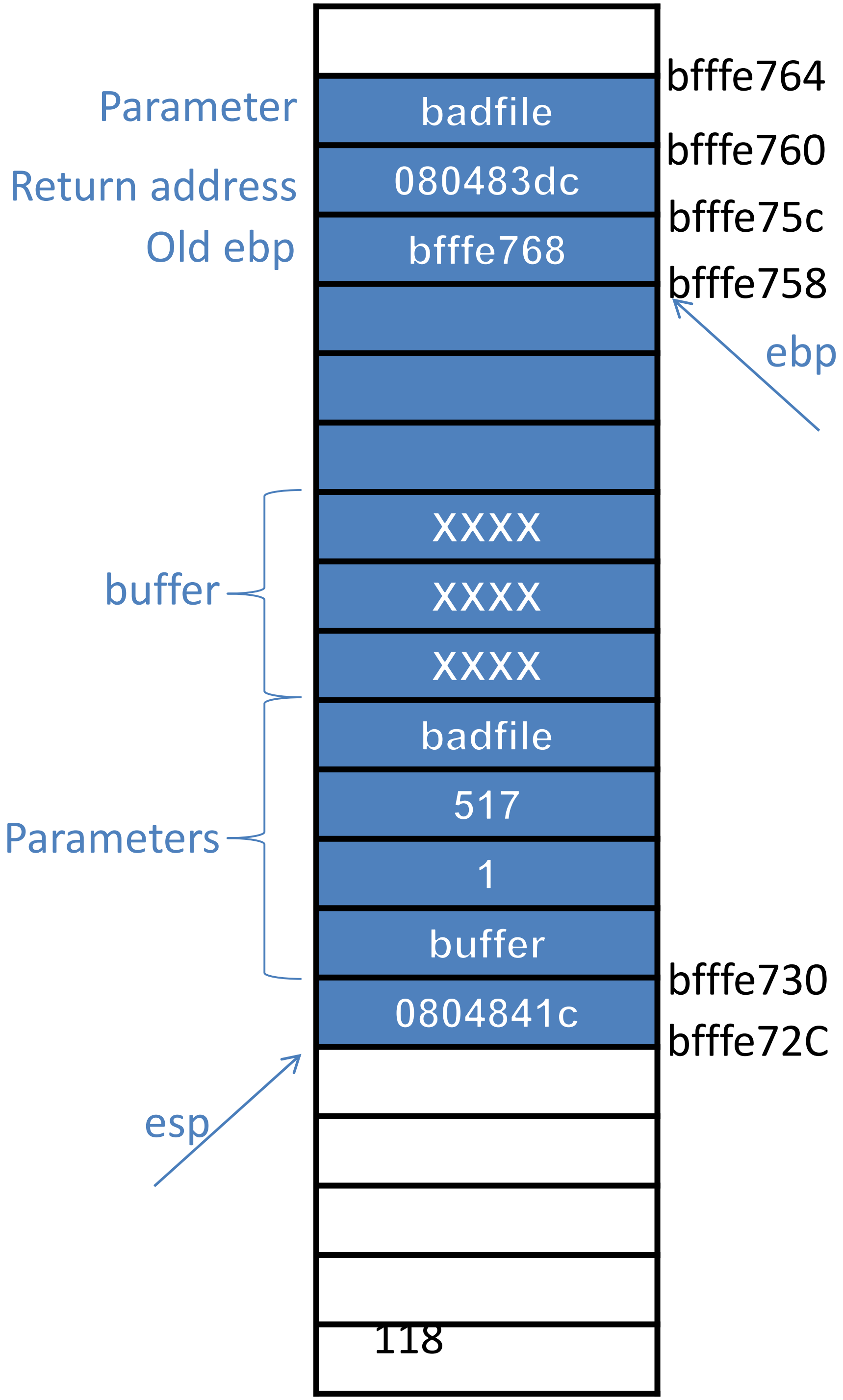
```
foo:
    pushl   %ebp
    movl %esp, %ebp
    subl $40, %esp
    ...
    movl 8(%ebp), %edx
    movl %edx, 12(%esp)
    movl $517, 8(%esp)
    movl $1, 4(%esp)
    movl %eax, (%esp)
    call fread
    movl $1, %eax
    leave
    ret
main:
    ...
    movl %eax, (%esp)
    call foo
    ...
    $1, %eax
    leave
    ret
```



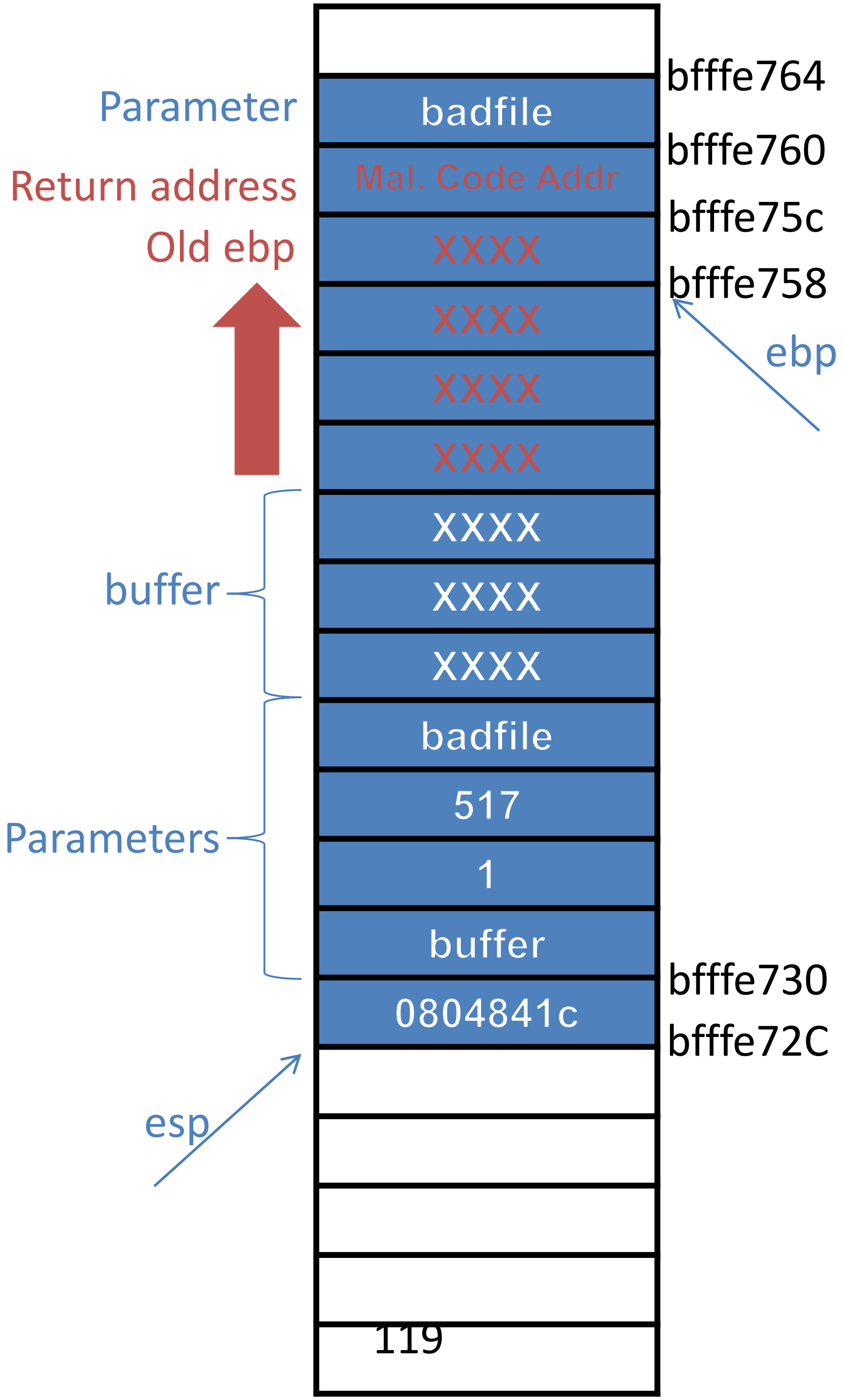
```
foo:
     pushl    %ebp
     movl %esp, %ebp
     subl $40, %esp
     ...
     movl 8(%ebp), %edx
     movl %edx, 12(%esp)
     movl $517, 8(%esp)
     movl $1, 4(%esp)
     movl %eax, (%esp)
     call fread
     movl $1, %eax
     leave
     ret
main:
     ...
     movl %eax, (%esp)
     call foo
     ...
     $1, %eax
     leave
     ret
```



```
foo:
     pushl    %ebp
     movl %esp, %ebp
     subl $40, %esp
     ...
     movl 8(%ebp), %edx
     movl %edx, 12(%esp)
     movl $517, 8(%esp)
     movl $1, 4(%esp)
     movl %eax, (%esp)
     call fread
     movl $1, %eax
     leave
     ret
main:
     ...
     movl %eax, (%esp)
     call foo
     ...
     $1, %eax
     leave
     ret
```



```
foo:
     pushl    %ebp
     movl %esp, %ebp
     subl $40, %esp
     ...
     movl 8(%ebp), %edx
     movl %edx, 12(%esp)
     movl $517, 8(%esp)
     movl $1, 4(%esp)
     movl %eax, (%esp)
     call fread
     movl $1, %eax
     leave
     ret
main:
     ...
     movl %eax, (%esp)
     call foo
     ...
     $1, %eax
     leave
     ret
```



Parameter
Return address
Old ebp
buffer
Parameters
esp
ebp
badfile
Mal. Code Addr
XXXX
XXXX
XXXX
XXXX
XXXX
XXXX
XXXX
badfile
517
1
buffer
0804841c
bfffe764
bfffe760
bfffe75c
bfffe758
bfffe730
bfffe72C