SloT – Securing the Internet of Things through Distributed System Analysis

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

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

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

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

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

Leonardo B. Oliveira  
UFG, 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
Q: What happens when we run this Java program?

```java
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)
~$
```
Q: What about this C program?

```c
#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);
}
```
Q: Why is that?

Watch out! It’s overflowing!

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
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()
```

```
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
```

```
[ssssss ][&sfp][&ret ][a]
buffer
```

```
top of stack   bottom of stack
```

```
[Image 261x247 to 478x372]
```
BOF (Cont.)

```
be_or_not()
```

```
/bin/sh
```

```
bottom of memory
```

```
top of memory
```

```
[sssssssssss][&sfp][&ret ][a]
```

```
buffer
```

```
top of stack
```

```
bottom of stack
```
BOF (Cont.)

```
be_or_not()
```

```
/bin/sh
```

```
bottom of memory
```

```
top of memory
```

```
[xxxxxxxxxx&sfp][&ret ][a]
buffer
```

```
top of stack
```

```
bottom of stack
```
BOF (Cont.)

```
be_or_not()
```

```
/bin/sh
```

```
[ssssssssssssssssssssssssss&ret ][a]
buffer
```

```
bottom of memory
```

```
top of memory
```

```
top of stack
```

```
bottom of stack
```
BOF (Cont.)

```
be_or_not()

/bin/sh
```

```
bottom of memory
```
```python
[ssssssssssssssssssssssssssssinsert']][a]
buffer
```
```
top of memory
```
```
top of stack
```
```
bottom of stack
```
BOF (Cont.)

```
be_or_not()
```

```
/bin/sh
```

```
[ssssssssssssssssssssssssss&ret'][a]
buffer
```

top of memory

bottom of memory

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 \((\text{load/store})\) 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

- **Source**: An untrusted source e.g.: `scanf("%d", &i)`
- **Path**: A path connecting the source and the sink
- **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**

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

**ABC-protected code**

```c
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
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 individually
• 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

Program A

- `scanf`
- `argv`

Program B

- `scanf`
- `argv`

send → recv
recv ← send
Distributed System AB: sources

Single System AB

Program A

{ } \text{scanf} \hspace{1cm} \{ \} \text{argv}

Program B

\{ \} \text{send} \hspace{1cm} \{ \} \text{recv}

\{ \} \text{recv} \hspace{1cm} \{ \} \text{send}
Distributed System AB: sources

Single System AB

Program A

{ } scanf { } argv

send  recv

recv  send

Program B

{ } scanf { } argv
Distributed System AB: sources

Program A
{ } scanf
{ } argv

Rejected

Program B
{ } scanf
{ } argv

send -> recv
recv <- send
Distributed System AB: sources

Single System AB

Program A

Program B

buffer[i]
Program A

```
1  send(1);
2  ack = recv();
3  if (ack == 1) {
4      s = getc();
5        while (s != '0') {
6          send(s);
7          ack = recv();
8          if (ack != 1) {
9              break;
10          } else {
11              s = getc();
12          }
13      }
14      send(s);
15  }
```

Program B

```
1  msg = recv();
2  if (msg == 1) {
3      send(1);
4        do {
5          msg = recv();
6          putc(msg);
7          if (msg != '0')
8              send(1);
9          else
10          break;
11        } while (1);
12      }
13  else {
14      send(0);
15  }
```

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
(a)  
1. `send(1);`
2. `ack = recv();`
3. if `(ack == 1)` {
4.   `s = getc();`
5.   while `(s != '\0')` {
6.     `send(s)`
7.     `ack = recv();`
8.     if `(ack != 1)` {
9.       break;
10.   } else {
11.      `s = getc();`
12.   }
13. } `send(s);`

(b)  
1. `msg = recv();`
2. if `(msg == 1)` {
3.   `send(1);`
4.   do {
5.     `msg = recv();`
6.     `putc(msg);`
7.     if `(msg != '\0')`
8.     `send(1);`
9.   } else {
10.    break;
11.  }
12. } else {
13. `send(0);`
(a) 1. **send**(l); 2. **ack** = **recv**(); 3. if (**ack** == 1) { 4. \[s = getc();\] 5. while (s != '0') { 6. **send**(s) 7. \[**ack** = **recv**();\] 8. if (**ack** != 1) { 9. \[break;\] 10. } else { 11. \[s = getc();\] 12. } 13. } 14. **send**(s); 15. }

(b) 1. msg = **recv**(); 2. if (msg == 1) { 3. **send**(l); 4. do { 5. msg = **recv**(); 6. putc(msg); 7. if (msg != '0') 8. **send**(l); 9. else 10. break; 11. } while (l); 12. } else { 13. **send**(0); 14. }
(a) 1. \texttt{send(1)};
2. \texttt{ack = recv()}
3. if (\texttt{ack == 1}) {
4. \hspace{1em} \texttt{s = getc();}
5. \hspace{2em} while (\texttt{s != \textbackslash\textquoteleft\textbackslash\textquoteleft 0\textbackslash\textquoteleft}) {
6. \hspace{3em} \texttt{send(s)}
7. \hspace{3em} \texttt{ack = recv();}
8. \hspace{4em} if (\texttt{ack != 1}) {
9. \hspace{5em} break;
10. \hspace{4em} } else {
11. \hspace{6em} \texttt{s = getc();}
12. \hspace{5em} }
13. \hspace{2em} }
14. \hspace{1em} \texttt{send(s)};
15. }

(b) 1. \texttt{msg = recv();}
2. if (\texttt{msg == 1}) {
3. \hspace{1em} \texttt{send(1)};
4. \hspace{2em} do {
5. \hspace{3em} \texttt{msg = recv();}
6. \hspace{3em} \texttt{putc(msg);}
7. \hspace{4em} if (\texttt{msg != \textbackslash\textbackslash 0\textbackslash\textquoteleft})
8. \hspace{5em} \texttt{send(1)};
9. \hspace{4em} } else {
10. \hspace{6em} break;
11. \hspace{5em} }
12. \hspace{2em} }
13. \hspace{1em} else {
14. \hspace{2em} \texttt{send(0)};
15. }
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.

How to find out which sends link to a recv?
SIoT key contribution #3

- Elevator, an algorithm to selectively link \texttt{ sends/recvs}
  1. Elevator assigns levels to sends and recvs
  2. Program A’s \texttt{ sends} and Program B’s \texttt{ recvs} in the same level are thus linked together
Elevator: Illustration

- Consider the Echo Client and Server programs

(a) Echo Client

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

(b) Echo Server

1. `msg = recv();`
2. `if (msg == 1) {
   send(1);
   do {
      if (msg != '\0')
         send(1);
      else
         break;
   } while (1);
} else {
   send(0);
}
Echo Client CFG

A: \textbf{send}(1) \quad B: \textbf{ack} = \textbf{recv}()

\begin{align*}
s &= \text{getc}() \\
\text{(ack} &= \text{==} 1) ?
\end{align*}

\begin{align*}
\text{(s} &= \text{!=} \text{'0')}? \\
\textbf{C: send}(s)
\end{align*}

\begin{align*}
\textbf{D: ack} &= \textbf{recv}() \\
\text{(ack} &= \text{!=} 1) ?
\end{align*}

\begin{align*}
\textbf{E: send}(s)
\end{align*}

(a) Echo Client

1. \textbf{send}(1);
2. \textbf{ack} = \textbf{recv}();
3. \textbf{if} (\text{ack} == 1) {
4. \quad \textbf{s} = \text{getc}();
5. \quad \textbf{while} (\text{s} != '\text{'0'}') {
6. \quad \quad \textbf{send}(s)
7. \quad \quad \textbf{ack} = \textbf{recv}();
8. \quad \quad \textbf{if} (\text{ack} != 1) {
9. \quad \quad \quad \textbf{break};
10. \quad \quad } \text{else} {
11. \quad \quad \quad \textbf{s} = \text{getc}();
12. \quad \quad }\}
13. \quad }\}
14. \textbf{send}(s);
Extract Echo Client’s Send-Graph

CFG - Echo Client

A: send(1)
B: ack = recv()

s = getc()

(s != '0')?

C: send(s)

D: ack = recv()

(ack != 1)?

E: send(s)

(a) Echo Client

1  send(1);
2  ack = recv();
3  if (ack == 1) {
4      s = getc();
5      while (s != '\0') {
6          send(s);
7          ack = recv();
8          if (ack != 1) {
9              break;
10         } else {
11             s = getc();
12           }
13       }
14  }
15 }
Extract Echo Client’s Send-Graph

CFG - Echo Client

A: \texttt{send(1)}

\textbf{s} = \texttt{getc()}

(s != ‘\0’)?

C: \texttt{send(s)}

E: \texttt{send(s)}

D: \texttt{ack = recv()}

(ack != 1)?

B: \texttt{ack = recv()}

(ack == 1) :

\textbf{s} != ‘\0’?

SG – Send-Graph

A: \texttt{send(1)}

C: \underline{\texttt{send(s)}}

E: \underline{\texttt{send(s)}}
Extract Echo Client’s Receive-Graph

CFG - Echo Client

A: \texttt{send(1)}

B: \texttt{ack = recv()}

\(s = \text{getc()}\)

\((\text{ack} == 1)?\)

\((s \neq '\\0')?\)

C: \texttt{send(s)}

D: \texttt{ack = recv()}

\((\text{ack} != 1)?\)

E: \texttt{send(s)}

SG - Send-Graph

A: \texttt{send(1)}

C: \texttt{send(s)}

E: \texttt{send(s)}
Extract Echo Client’s Receive-Graph

CFG - Echo Client

A: send(1) → B: ack = recv()

s = getc() → (ack == 1)?

(s != ‘\0’)?

C: send(s) → E: send(s)

D: ack = recv()

(ack != 1)?

s = getc()

SG - Send-Graph

A: send(1)

C: send(s) → E: send(s)
Extract Echo Client’s Receive-Graph
Ditto for Echo Server

CFG - Echo Server

- F: msg = `recv()`
- H: msg = `recv()`
- G: `send(1)`
- J: `send(0)`
- I: `send(1)`
- (msg != '\0')?
- putc(msg)
- (msg == 1)?

SG - Send-Graph

- G: `send(1)`
- I: `send(1)`
- J: `send(0)`

RG - Receive-Graph

- F: msg = `recv()`
- H: msg = `recv()`
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
```
Level Assignment

**SG – Echo Client**

- A: `send(1)`
- C: `send(s)`
- E: `send(s)`

**Level of Senders:**

<table>
<thead>
<tr>
<th>level</th>
<th>set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{ start }</td>
</tr>
</tbody>
</table>

**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
```
Elevator (Graph G) /* toy version*/

\begin{verbatim}
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{verbatim}

Echo Client: Send-Graph

Level of Senders:

<table>
<thead>
<tr>
<th>level</th>
<th>set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{ start }</td>
</tr>
<tr>
<td>1</td>
<td>{ A }</td>
</tr>
</tbody>
</table>
Level Assignment

**Level of Senders:**

<table>
<thead>
<tr>
<th>Level</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{ start }</td>
</tr>
<tr>
<td>1</td>
<td>{ A }</td>
</tr>
<tr>
<td>2</td>
<td>{ C, E }</td>
</tr>
</tbody>
</table>

**Elevator (Graph G) /* toy version*/:

```plaintext
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
```

**Echo Client:** Send-Graph
Level Assignment

Echo Client: Send-Graph

**Level of Senders:**

<table>
<thead>
<tr>
<th>level</th>
<th>set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{ start }</td>
</tr>
<tr>
<td>1</td>
<td>{ A }</td>
</tr>
<tr>
<td>2</td>
<td>{ C, E }</td>
</tr>
<tr>
<td>3</td>
<td>{ C, E }</td>
</tr>
</tbody>
</table>

Elevator (Graph G) /* toy version*/

```plaintext
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
```

the algorithm halts whenever sets stop changing

Level of Senders:

<table>
<thead>
<tr>
<th>level</th>
<th>set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{ start }</td>
</tr>
<tr>
<td>1</td>
<td>{ A }</td>
</tr>
<tr>
<td>2</td>
<td>{ C, E }</td>
</tr>
<tr>
<td>3</td>
<td>{ C, E }</td>
</tr>
</tbody>
</table>
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
```

Level of Senders:

<table>
<thead>
<tr>
<th>level</th>
<th>set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{ start }</td>
</tr>
<tr>
<td>1</td>
<td>{ A }</td>
</tr>
<tr>
<td>2</td>
<td>{ C, E }</td>
</tr>
<tr>
<td>3</td>
<td>{ C, E }</td>
</tr>
</tbody>
</table>
Level Assignment: Ditto for Recv-Graph

Echo Client: Send-Graph

A: \textit{send}(1)

C: \textit{send}(s) \rightarrow E: \textit{send}(s)

Level of Senders:

\begin{align*}
\text{level} & \quad \text{set} \\
0 & \quad \{ \text{start} \} \\
1 & \quad \{ A \} \\
2 & \quad \{ C, E \} \\
3 & \quad \{ C, E \}
\end{align*}

Echo Server: Recv-Graph

F: \textit{msg = recv()}

H: \textit{msg = recv()}

Level of Receivers:

\begin{align*}
\text{level} & \quad \text{set} \\
0 & \quad \{ \text{root} \} \\
1 & \quad \{ F \} \\
2 & \quad \{ H \} \\
3 & \quad \{ H \}
DCFG construction final step

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

• Introduction
• Goal
• Solution
  – Conception
  – Refinement
  – Development
• Results
• Conclusion
SloT Coding

Core

- Program_1..2.bc
- Setup: Send/Recvs Functions Names
- Send-Graphics
- Receive-Graphics

Merging

- Merge
- DistSysDem.bc

Linking

- Elevator
- DCFG
- DistDepGraph
- DDG

BOF case study

- DistVulArrays
  - DistVulArrays Graph
  - DistVulArrays Statistics
SloT is publicly available

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
SiIoT 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

<table>
<thead>
<tr>
<th>Application</th>
<th>Instructions</th>
<th>Time (s)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>netdb client/server</td>
<td>57,877</td>
<td>66.24</td>
<td>210.03</td>
</tr>
<tr>
<td>ping / new-ipv6</td>
<td>47,422</td>
<td>63.58</td>
<td>167.36</td>
</tr>
<tr>
<td>ipv6-rpl-collect udp-sender/sink</td>
<td>48,800</td>
<td>80.08</td>
<td>173.37</td>
</tr>
<tr>
<td>ipv6-rpl-udp client/server</td>
<td>48,226</td>
<td>66.31</td>
<td>169.90</td>
</tr>
<tr>
<td>udp-ipv6 client/server</td>
<td>48,800</td>
<td>80.08</td>
<td>167.39</td>
</tr>
<tr>
<td>coap-client / rest-server</td>
<td>51,258</td>
<td>54.36</td>
<td>179.68</td>
</tr>
</tbody>
</table>
ABCs Insertion

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

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

- DAQ
- Shunt resistor
- Iris sensor node
SloT Dynamic Analysis

Energy savings compared to Baseline

- netdb-client: 1.74%
- udp-client: 2.52%
- rpl-upd-client: 1.67%
- coap-client: 3.55%
- udp-sender: 31.18%
- ping6: 24.83%
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 $\{C_1, C_2\}$, Send-Graphs $\{S_1, S_2\}$ and Receive-Graphs $\{R_1, R_2\}$.

**Output:** a DCFG $D$

▷ Set the SEND and RECV levels

foreach $G_i \in \{S_1, S_2\} \cup \{R_1, R_2\}$ do

\[ n \leftarrow 0 \]
\[ L_{G_i, n} \leftarrow \{\text{root}\} \]

▷ 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

\[ S_{\text{success}} \leftarrow \text{successors of } v \]
\[ L_{G_i, n+1} \leftarrow L_{G_i, n+1} \cup S_{\text{success}} \]
\[ n \leftarrow n + 1 \]

▷ Link SENDs and RECVs of the same level

$D \leftarrow C_1 \cup C_2$

for $k \leftarrow 1$ to $n$ do

foreach $v_s \in L_{S_1, k}$ and $v_r \in L_{R_2, k}$ do

add an edge from $v_s$ to $v_r$ in $D$

foreach $v_r \in L_{R_1, k}$ and $v_s \in L_{S_2, k}$ do

add an edge from $v_s$ to $v_r$ in $D$
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

\[
y = 0.0073x^3 - 0.0429x^2 + 0.1997x - 0.0366 \\
R^2 = 0.99547
\]
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
buffer[i] = a;

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mrmovl -12(%ebp), %eax</td>
<td>Load 'i'</td>
</tr>
<tr>
<td>mrmovl -4(%ebp), %edx</td>
<td></td>
</tr>
<tr>
<td>rrmovl %eax, %edi</td>
<td></td>
</tr>
<tr>
<td>sal $2, %edi</td>
<td></td>
</tr>
<tr>
<td>addl %ebp, %edi</td>
<td></td>
</tr>
<tr>
<td>rmmovl %edx, -2060(%edi)</td>
<td></td>
</tr>
</tbody>
</table>
buffer[i] = a;
ABCs’ Computational Cost

buffer[i] = a;

\[
\begin{align*}
mrmovl \ -12(\%ebp), \ %eax & \quad \text{Load ‘i’} \\
mrmovl \ -4(\%ebp), \ %edx & \quad \text{Load ‘a’} \\
rrmovl \ %eax, \ %edi & \\
sall \ $2, \ %edi & \quad \text{Adjust base register} \\
addl \ %ebp, \ %edi & \\
rmmovl \ %edx, \ -2060(\%edi) & \\
\end{align*}
\]
buffer[i] = a;

\[
\begin{align*}
\text{mrmovl} \ (-12(\%ebp), \%eax) & \quad \text{Load ‘i’} \\
\text{mrmovl} \ (-4(\%ebp), \%edx) & \quad \text{Load ‘a’} \\
\text{rrmovl} \ %eax, \%edi & \quad \text{Adjust base register} \\
\text{sall} \ $2, \%edi & \\
\text{addl} \ %ebp, \%edi & \\
\text{rmmovl} \ %edx, \ -2060(\%edi) & \quad \text{Move}
\end{align*}
\]

ABCs’ Computational Cost
ABCs’ Computational Cost

if ( (i>= 0) && (i < BUFFERSIZE) )

mrmovl -12(%ebp), %edi
irmovl $0, %ebx
subl %ebx, %edi
js L3

mrmovl -12(%ebp),%edi
irmovl $511, %ebx
subl %ebx, %edi
jg L3

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

buffer[i] = a;

Load ‘i’
Load ‘i’
Load ‘i’
Load ‘a’
Adjust base register
Move
ABCs’ Computational Cost

\[
\text{if( (i >= 0) \&\& (i < BUFFERSIZE) )}
\]

- \text{mrmovl -12(%ebp), %edi}  \quad \text{Load ‘i’}
- \text{irmovl $0, %ebx}  \quad \text{Load lower index}
- \text{subl %ebx, %edi}  \quad \text{Load ‘i’}
- \text{js L3}
- \text{mrmovl -12(%ebp),%edi}  \quad \text{Load ‘i’}
- \text{irmovl $511, %ebx}  \quad \text{Load upper index}
- \text{subl %ebx, %edi}  \quad \text{Load ‘i’}
- \text{jg L3}
- \text{buffer[i] = a;}
- \text{mrmovl -12(%ebp), %eax}  \quad \text{Load ‘i’}
- \text{mrmovl -4(%ebp), %edx}  \quad \text{Load ‘a’}
- \text{rrmovl %eax, %edi}  \quad \text{Adjust base register}
- \text{sall $2, %edi}
- \text{addl %ebp, %edi}
- \text{rmmovl %edx, -2060(%edi)}  \quad \text{Move}
ABCs’ Computational Cost

if( (i >= 0) 
  && (i < BUFFERSIZE) )
  buffer[i] = a;

Load ‘i’
Load lower index
Compare
Load ‘i’
Load upper index
Compare
Load ‘i’
Load ‘a’
Adjust base register
Move
Full BOF example
Vulnerable Code

```c
#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
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
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