



Security Analysis of LLVM Bitcode Files for Mobile Platforms

Dan S. Wallach (Rice University)

TIZEN™
**DEVELOPER
CONFERENCE**
2013
SAN FRANCISCO

Preface

- **This is ongoing research at Rice, supported by Samsung**
- **I don't speak for Samsung**

- **Core team at Rice**
 - Vivek Sarkar (PI)
 - Dan Wallach (Co-PI)
 - Michael Burke (Senior Research Scientist)
 - Jisheng Zhao (Research Scientist)
 - Deepak Majeti (PhD student)
 - Dragos Sbirlea (PhD student)
 - Bhargava Shastry (PhD student)
- **Additional contributors at Rice**
 - Keith Cooper
 - Swarat Chaudhuri

Security goals

- **Analyze apps submitted by developers, prior to deployment**
- **Static analysis (process applications without running them)**
 - Automation to assist a human analyst
- **Versus other models**
 - Apple iOS: unspecified process, apparently labor intensive and slow
 - Google Play Store: 100% automated, fast but problems get through

Web vs. native Tizen apps

Source: "Tizen Overview and Architecture", Seokjae Jeong, Samsung Electronics, Oct 2012

Web application

Web Framework

Runtime
Core

Tizen Web
API
Plug-In

Installer
Core

App
Security
Core

Java Script Core

WebKit2

Native application

Core

App FW

MM

Conn

Location

PIM

Telephony

System

Graphics
& Input

...

Web vs. native Tizen apps

Source: "Tizen Overview and Architecture", Seokjae Jeong, Samsung Electronics, Oct 2012

Web application

Native application

Our work:
"native" (compiled
LLVM bitcode)
applications

Runtime
Core

App
Security
Core

Java Script Core

WebKit2

Core

App FW

MM

Conn

Location

PIM

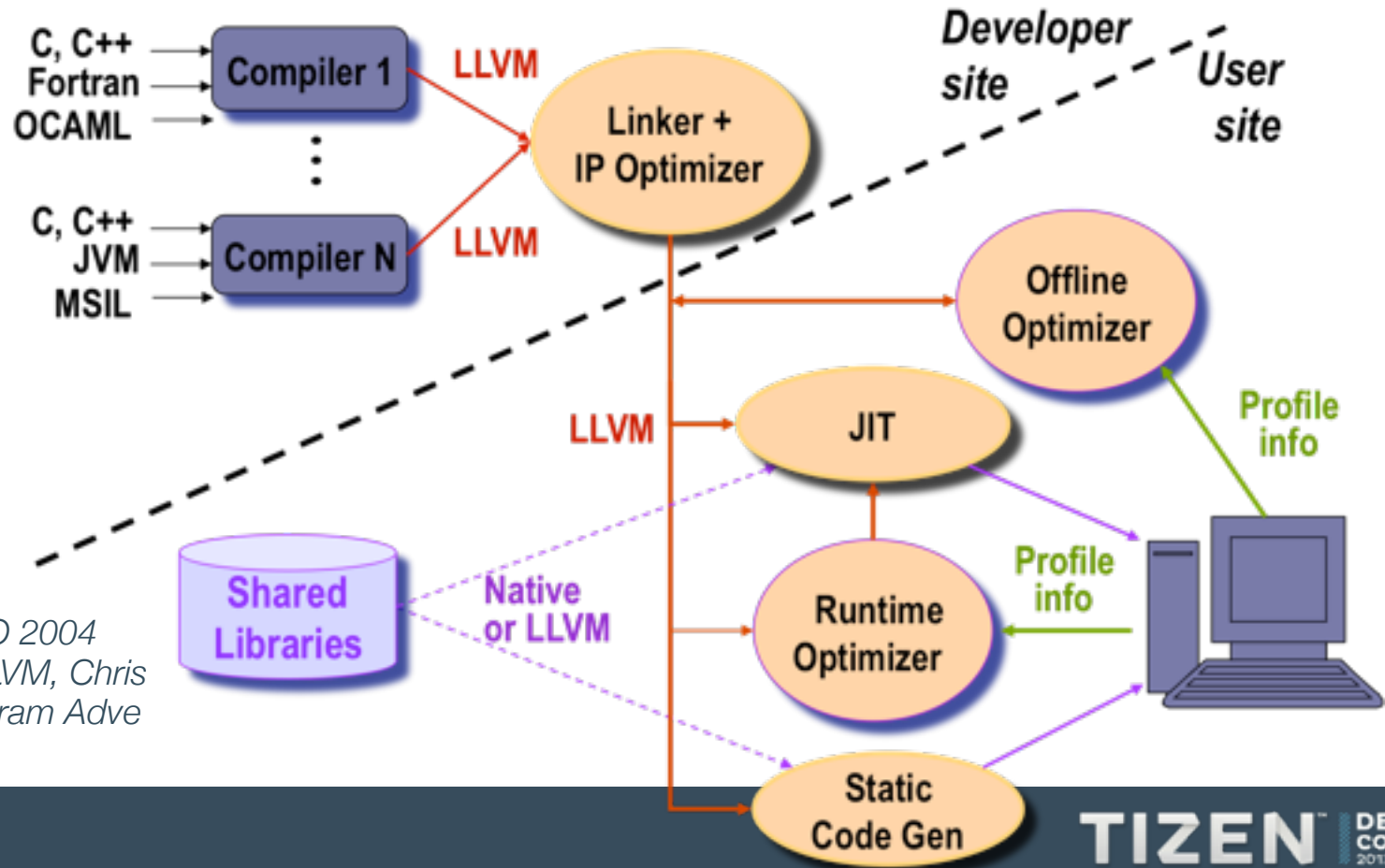
Telephony

System

Graphics
& Input

...

LLVM: A Low-Level Virtual Machine



Source: CGO 2004
tutorial on LLVM, Chris
Lattner & Vikram Adve

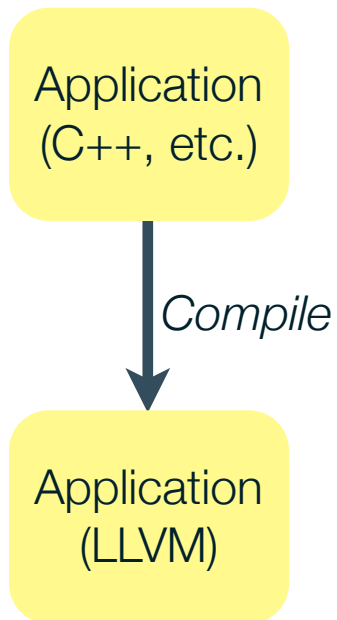
LLVM is a great way to ship code

- **Designed to be analyzed / optimized**
 - Maintains high-level semantics of source code
 - Independent of any specific CPU architecture (unlike machine code)
 - Deals better with multiple source languages (unlike Java bytecode)
 - High runtime performance
- **Open source / widely used in industry**
 - Notably adopted in Apple's Xcode toolchain
- **Straightforward to analyze for security purposes**

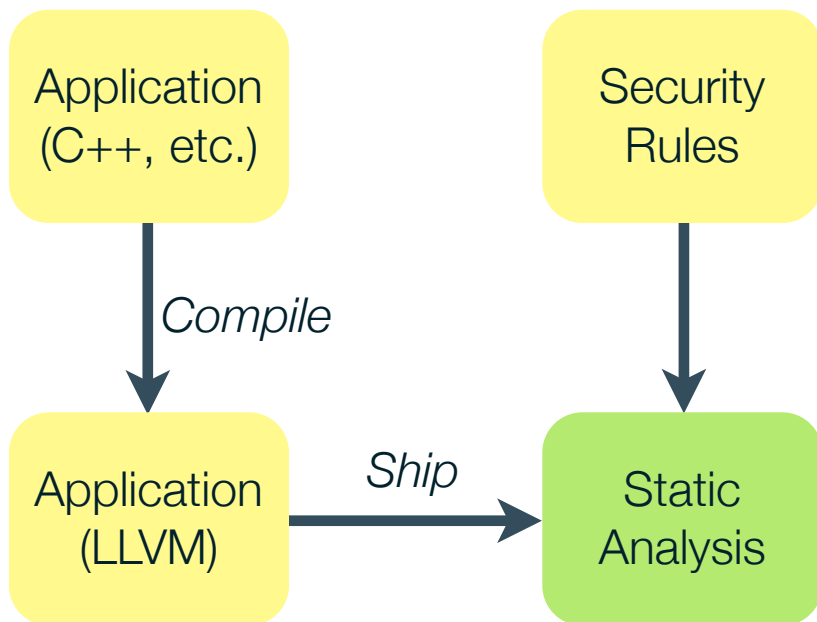
Overall Pipeline

Application
(C++, etc.)

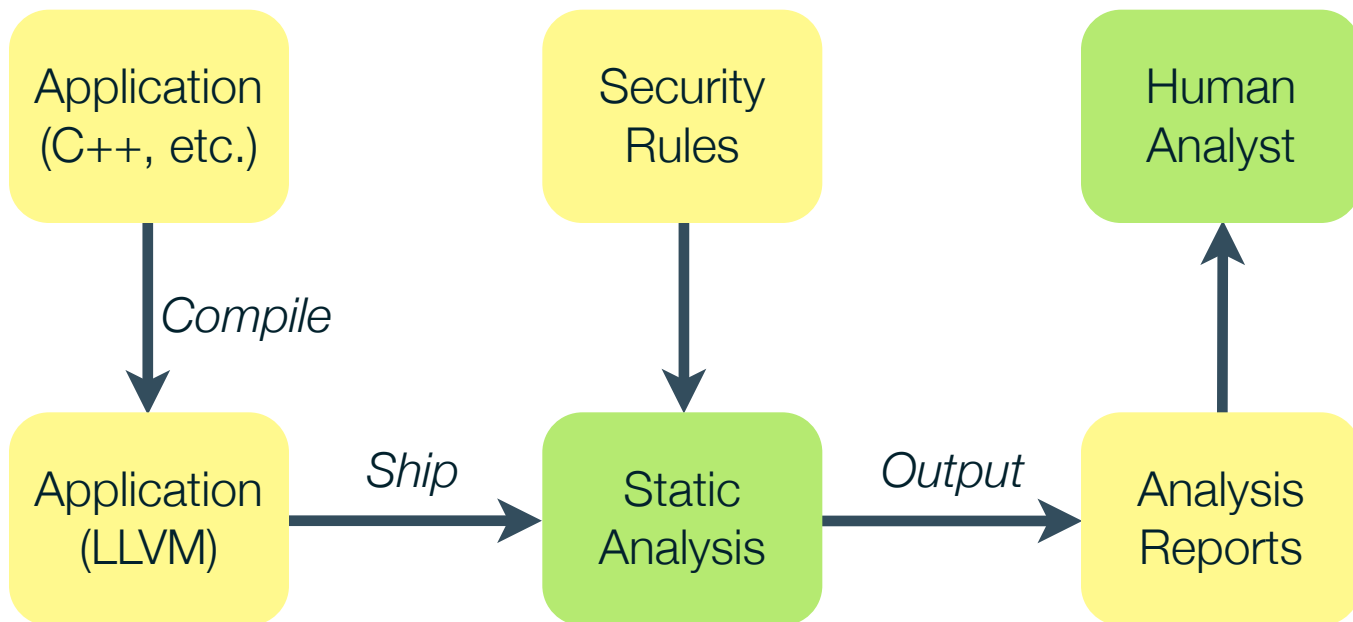
Overall Pipeline



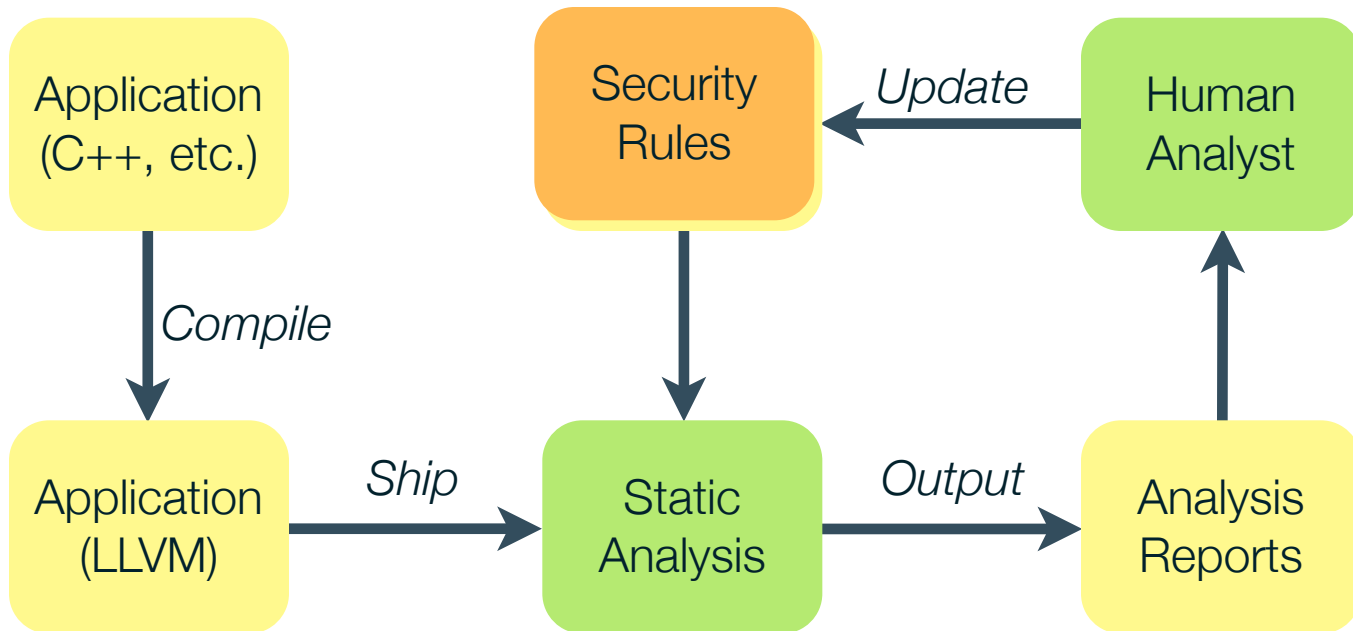
Overall Pipeline



Overall Pipeline



Overall Pipeline



Possible analyses

- **Blacklist for APIs used**
 - Easy/fast to implement
- **Information flow**
 - Captures our intuition of policies we want to enforce
 - Example: “GPS to Internet”

Example policy: “GPS to Internet”

```
void UpdateLocation() {
    time t = System::GetTime(); // secs since epoch
    if((t-lastLocationTime) > 60) {
        lastLocation = System::GetLocation();
        lastLocationTime = t;
    }
}

time lastLocationTime = 0;
Location lastLocation = Location(0,0);
Socket homeServer;

void EventLoop() {
    UpdateLocation();

    String s = "User now at: " + lastLocation.toString();

    homeServer.send(s);
}
```

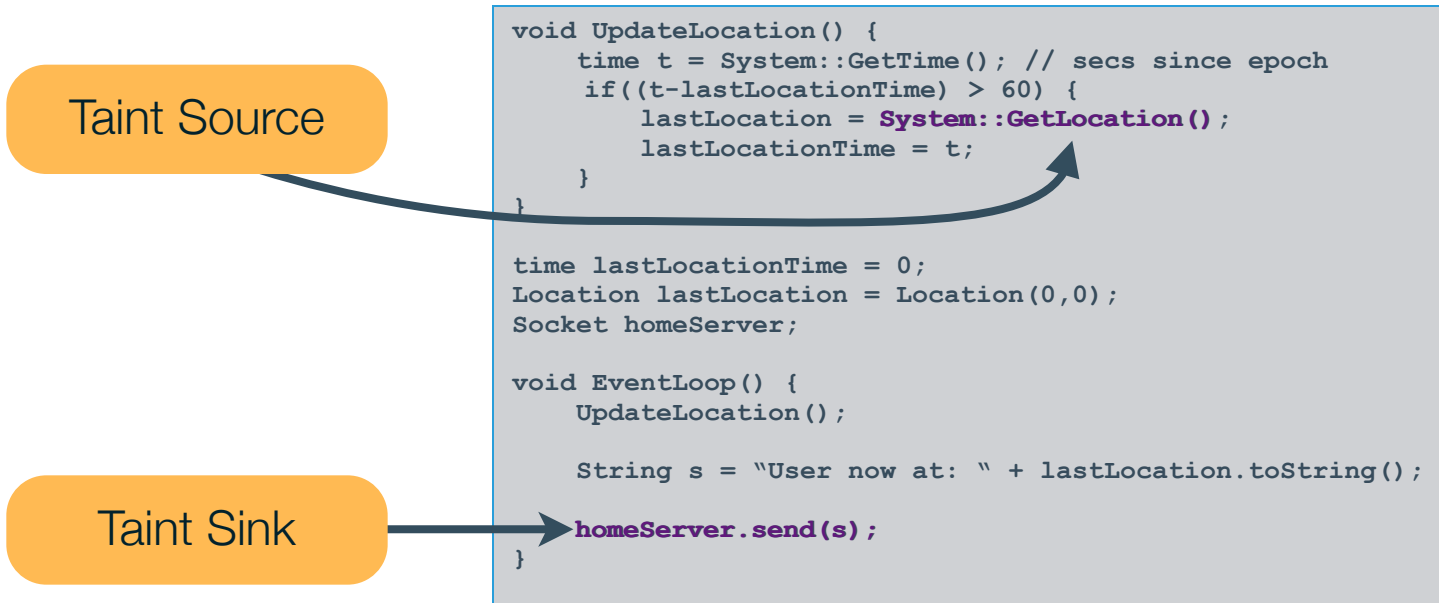

Example policy: “GPS to Internet”

Taint Source

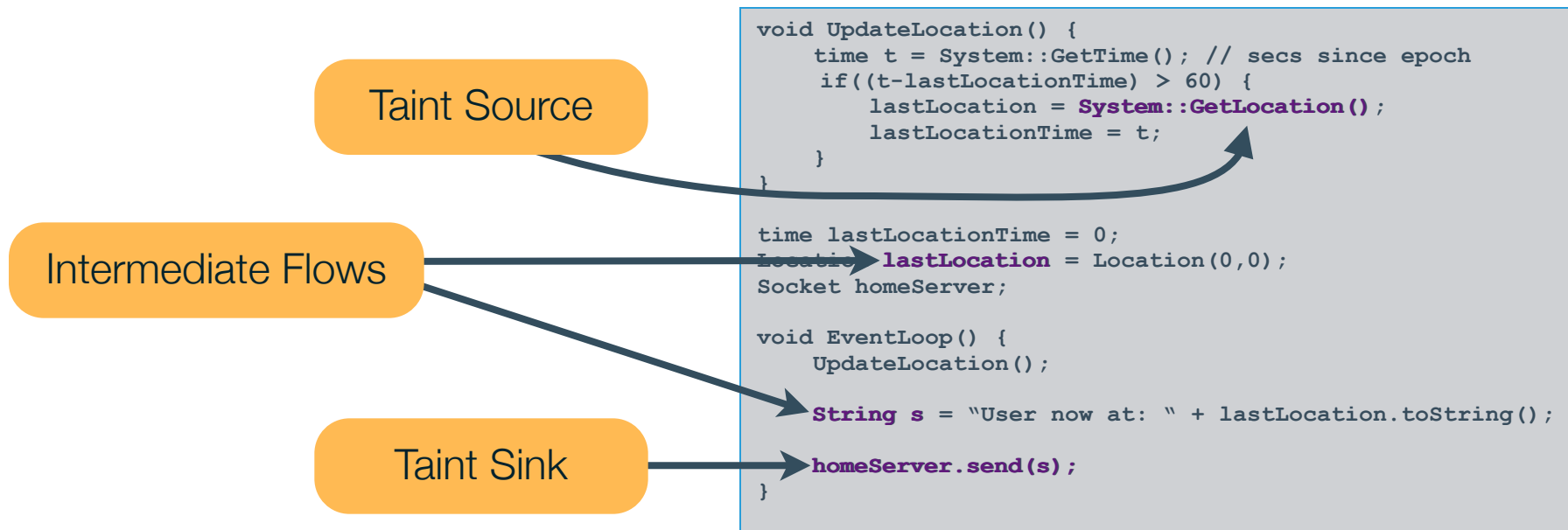


```
void UpdateLocation() {  
    time t = System::GetTime(); // secs since epoch  
    if((t-lastLocationTime) > 60) {  
        lastLocation = System::GetLocation() ;  
        lastLocationTime = t;  
    }  
}  
  
time lastLocationTime = 0;  
Location lastLocation = Location(0,0);  
Socket homeServer;  
  
void EventLoop() {  
    UpdateLocation();  
  
    String s = "User now at: " + lastLocation.toString();  
  
    homeServer.send(s);  
}
```

Example policy: “GPS to Internet”



Example policy: “GPS to Internet”



Example policy: “GPS to Internet”

Conditionals

Taint Source

Intermediate Flows

Taint Sink

```
void UpdateLocation() {  
    time t = System::GetTime(); // secs since epoch  
    if((t-lastLocationTime) > 60) {  
        lastLocation = System::GetLocation();  
        lastLocationTime = t;  
    }  
}  
  
time lastLocationTime = 0;  
Location lastLocation = Location(0,0);  
Socket homeServer;  
  
void EventLoop() {  
    UpdateLocation();  
  
    String s = "User now at: " + lastLocation.toString();  
  
    homeServer.send(s);  
}
```

The diagram illustrates the flow of taint through the provided code. Arrows indicate the following paths:

- Conditionals:** An arrow points from the 'Conditionals' label to the `if` statement in the `UpdateLocation()` function.
- Taint Source:** An arrow points from the 'Taint Source' label to the `System::GetLocation()` call within the `if` block.
- Intermediate Flows:** Two arrows originate from this label. One points to the `lastLocation` variable assignment, and the other points to the `lastLocation.toString()` call in the `EventLoop()` function.
- Taint Sink:** An arrow points from the 'Taint Sink' label to the `homeServer.send(s)` call.

Information flow complexities

- **Conditional flows**
 - Assume all branches taken
- **Pointer aliasing**
 - Don't always know where a pointer points
- **OO method dispatch**
 - Don't always know where a callsite will go
- **Challenge: Conservative analysis vs. false positives**

```
void UpdateLocation() {
    time t = System::GetTime(); // secs since epoch
    if((t-lastLocationTime) > 60) {
        lastLocation = System::GetLocation();
        lastLocationTime = t;
    }
}

time lastLocationTime = 0;
Location lastLocation = Location(0,0);
Socket homeServer;

void EventLoop() {
    UpdateLocation();

    String s = "User now at: " + lastLocation.toString();

    homeServer.send(s);
}
```

Information flow is a well-studied problem

- **Control-flow / data-flow analyses already used in compilers**
- **“Declassifying” operations to support special cases**
 - Example: system module trusted to give coarse location
- **False positives are an important challenge**
 - Conservatively flagging every possible flow would be overwhelming
 - Instead, pragmatic focus on “most likely” vulnerabilities

Prioritizing bug reports

- **Heuristics can help sort bug severity**
 - Shorter distance between source and sink $\Rightarrow$ higher severity
 - More sensitive sources (GPS, contacts) $\Rightarrow$ higher severity
 - More conditionals (harder to exploit) $\Rightarrow$ lower severity
- **Heuristics, sources, and sinks are refined by analysts**
 - Learn from previous experience
 - Learn from (the inevitable) security exploits

SSA-based analysis example

- **SSA = Static Single Assignment**
- **Each definition is given a unique name**
- **SSA-based sparse analysis**

```
X = ...  
if (cond) {  
    x = TaintSource();  
    y = x + 1;  
    TaintSink(y);  
} else  
    TaintSink(x);
```


SSA-based analysis example

- **SSA = Static Single Assignment**
- **Each definition is given a unique name**
- **SSA-based sparse analysis**

```
x0 = ...  
if (cond) {  
    x1 = TaintSource();  
    y1 = x1 + 1;  
    TaintSink(y1);  
} else  
    TaintSink(x0);  
x2 =  $\phi$ (x0, x1);
```

SSA-based analysis example

- **SSA = Static Single Assignment**
- **Each definition is given a unique name**
- **SSA-based sparse analysis**

```
x0 = ...  
if (cond) {  
    x1 = TaintSource();  
    y1 = x1 + 1;  
    TaintSink(y1);  
} else  
    TaintSink(x0);  
x2 =  $\phi$ (x0, x1);
```

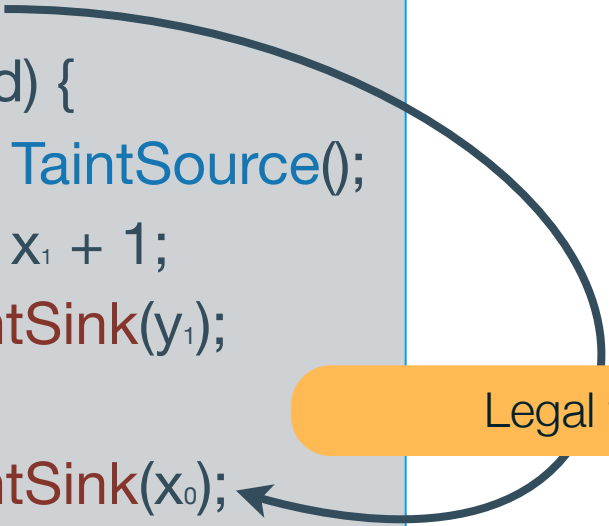
Simple forbidden flow

SSA-based analysis example

- **SSA = Static Single Assignment**
- **Each definition is given a unique name**
- **SSA-based sparse analysis**

```
x0 = ...  
if (cond) {  
    x1 = TaintSource();  
    y1 = x1 + 1;  
    TaintSink(y1);  
} else  
    TaintSink(x0);  
x2 =  $\phi$ (x0, x1);
```

Legal flow

A curved arrow originates from the variable x_0 in the first line of the code block and points to the argument x_0 in the `TaintSink(x0);` statement within the `else` branch. An orange rounded rectangle labeled "Legal flow" is positioned over the arrow.

SSA-based analysis example

- **SSA = Static Single Assignment**
- **Each definition is given a unique name**
- **SSA-based sparse analysis**

```
x0 = ...  
if (cond) {  
    x1 = TaintSource();  
    y1 = x1 + 1;  
    TaintSink(y1);  
} else  
    TaintSink(x0);  
x2 =  $\phi$ (x0, x1);
```

x₂ inherits taint from x₁

Pseudo-uses

- **Tainted conditional expressions**
 - Taint is applied to all subsequent values that depend on the condition
- **Similar issues**
 - Array storage
 - Method dispatch

```
x0 = TaintSource();  
if (x0 == ...)  
    y0 = "yes";  
else  
    y1 = "no";  
y2 =  $\phi$ (y0, y1);  
TaintSink(y2);
```

Pseudo-uses

- **Tainted conditional expressions**
 - Taint is applied to all subsequent values that depend on the condition
- **Similar issues**
 - Array storage
 - Method dispatch

```
x0 = TaintSource();  
if (x0 == ...)  
    y0 = "yes";  
else  
    y1 = "no";  
y2 =  $\phi$ (y0, y1);  
TaintSink(y2);
```

y₀ inherits taint from x₀

Pseudo-uses

- **Tainted conditional expressions**
 - Taint is applied to all subsequent values that depend on the condition
- **Similar issues**
 - Array storage
 - Method dispatch

```
x0 = TaintSource();  
if (x0 == ...)  
    y0 = "yes";  
else  
    y1 = "no";  
y2 =  $\phi$ (y0, y1);  
TaintSink(y2);
```

y₀ inherits taint from x₀

y₁ inherits taint from x₀

Pseudo-uses

- **Tainted conditional expressions**
 - Taint is applied to all subsequent values that depend on the condition
- **Similar issues**
 - Array storage
 - Method dispatch

```
x0 = TaintSource();
```

```
if (x0 == ...)
```

```
    y0 = "yes";
```

y₀ inherits taint from x₀

```
else
```

```
    y1 = "no";
```

y₁ inherits taint from x₀

```
y2 =  $\phi$ (y0, y1);
```

```
TaintSink(y2);
```

Forbidden flow

Under the hood: Taint analysis

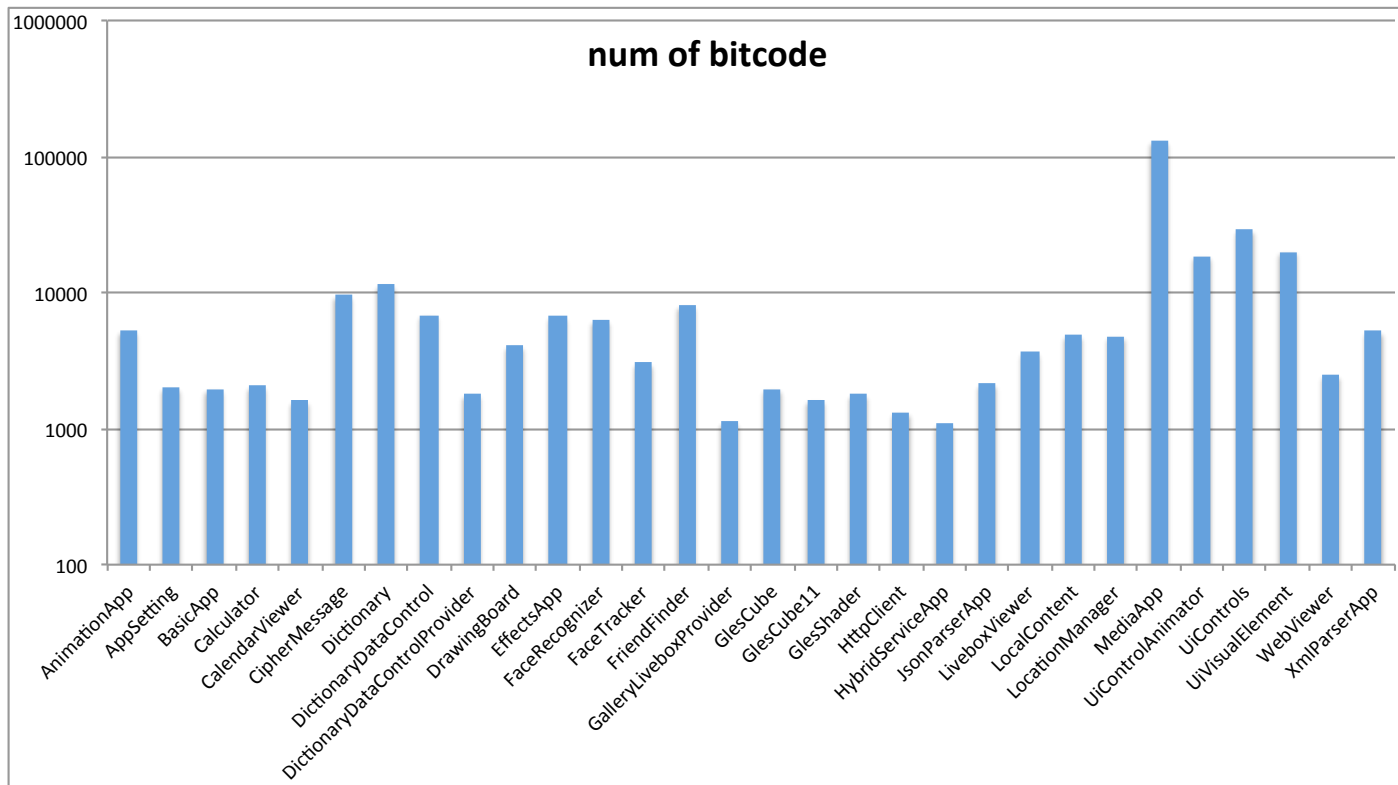
- **Simple lattice of taint values for every variable**
 - Untainted = top
 - Tainted = bottom
 - $\text{join}(\text{Tainted}, \text{Untainted}) = \text{Tainted}$
- **SSA-based sparse conditional constant propagation (SCCP) algorithm**
 - Already implemented in LLVM, extended by Rice for taint analysis
- **Important extensions**
 - Use of control dependences to insert “pseudo uses”
 - Use of Array SSA form for efficient alias analysis

Preliminary results

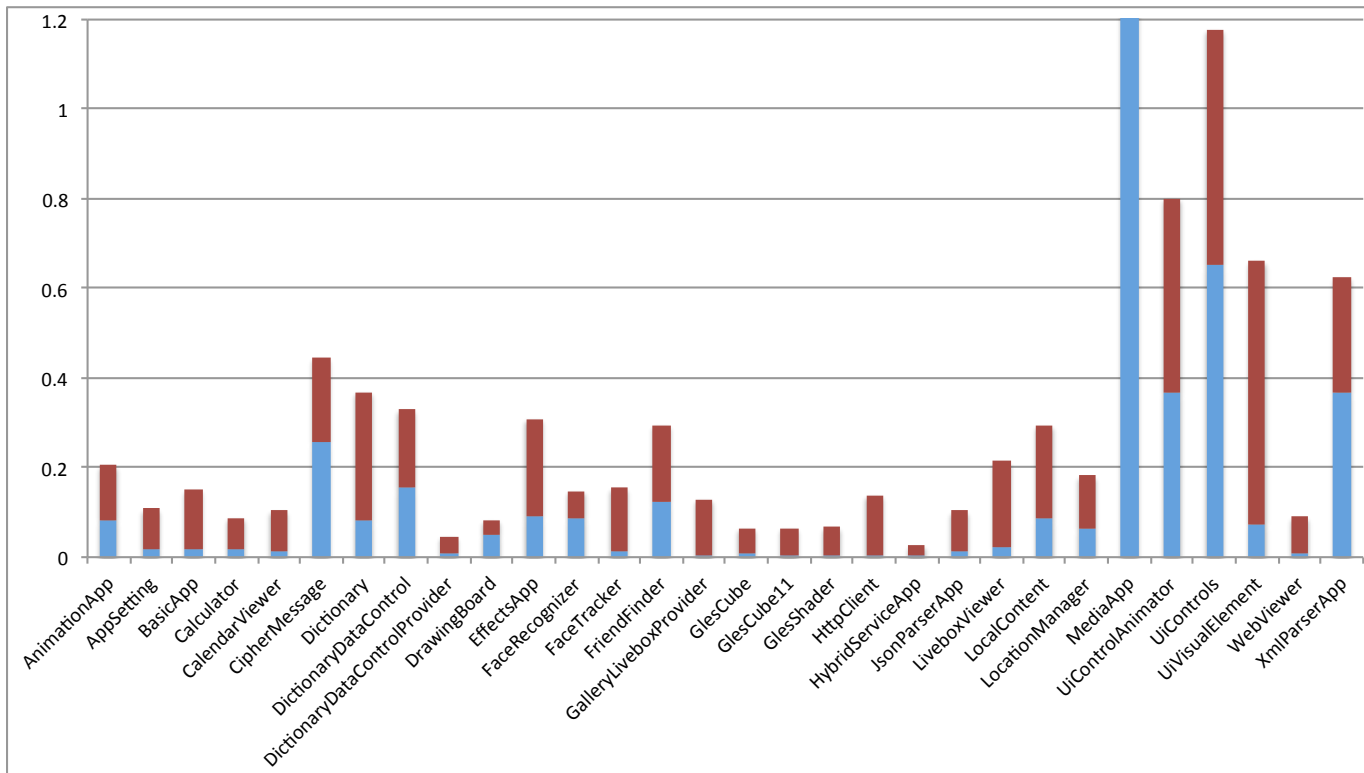
- Analyzed 30 Tizen apps (from Samsung)
- Flagged one privacy leak (*FriendFinder*)
 - Can transmit contact information over Bluetooth
- No errors (false positives or false negatives)

```
...  
W_char* str = GetImagePathPtr(); // taint source  
...  
String s = str;  
BluetoothOppClient::PushFile(s); // taint sink  
...
```

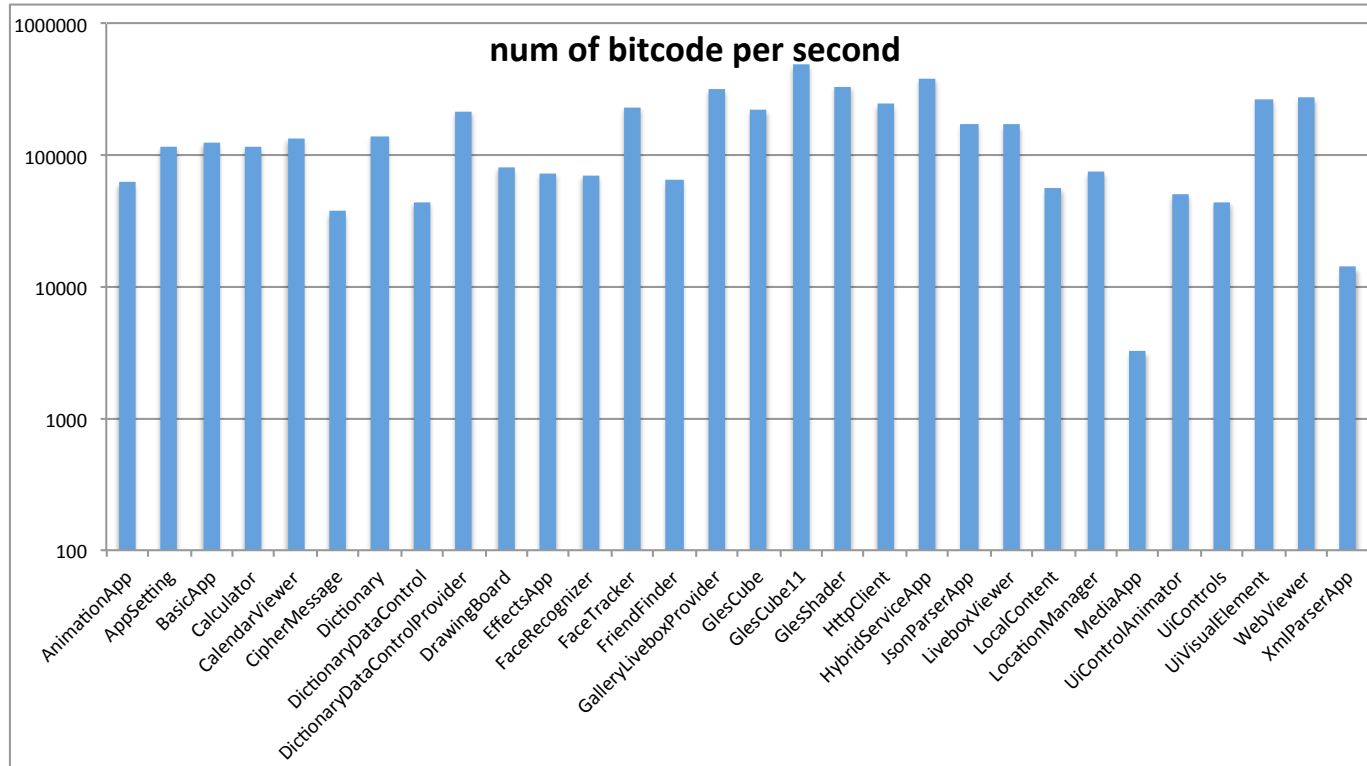
Preliminary performance



Preliminary performance



Preliminary performance



Future work

- **Full-system analysis (system libraries, kernel, etc.)**
 - Currently just analyzing the local code of the app
 - Special handling for callbacks
- **Dynamic analysis (virtual machine, runtime instrumentation)**
- **Tizen web apps**

An abstract graphic on the left side of the image. It features a large, white, three-dimensional ribbon-like shape that is twisted and folded. This shape is decorated with blue and white diagonal stripes. Surrounding this central element are numerous small, blue, triangular and polygonal shapes of various sizes, some of which are also striped. These shapes appear to be floating or falling from the top left towards the bottom right, creating a sense of motion and depth. The background is a light, neutral color with a subtle gradient.

TIZENTM

DEVELOPER CONFERENCE

2013

SAN FRANCISCO