Componolit
Secure Systems Engineering

Combining program verification with component-based architectures

Alexander Senier

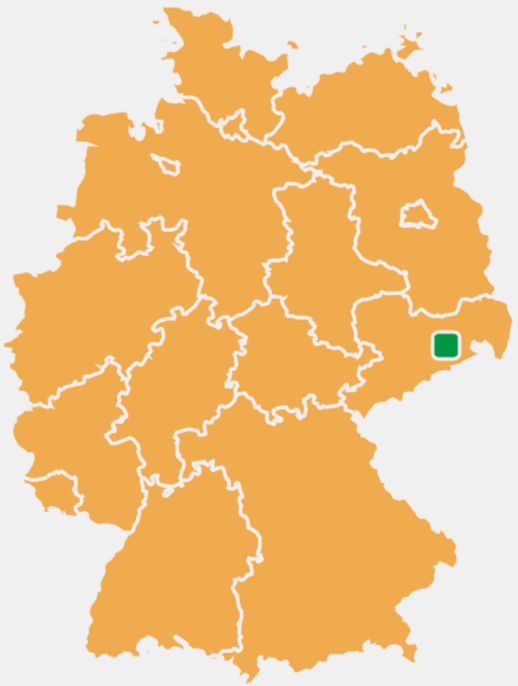
BOB 2018

Berlin, February 23rd, 2018

About Componolit

Componolit
Secure Systems Engineering

- **Topic:** Secure component-based systems
- **Focus:** Mobile devices, industrial security
- **Principles**
  - Free software
  - Open development
  - Security-by-design



Componolit
Secure Systems Engineering

What happens when we use what's best?

What's Best?
Mid-90ies: DOS+Pascal

Componolit
Secure Systems Engineering

```
program WriteName;
var
  i    : Integer;          {variable to be used for looping}
  Name : String;           {declares the variable Name as a string}
begin
  Write('Please tell me your name: ');
  ReadLn(Name);            {Return string entered by the user}
  for i := 1 to 100 do
  begin
    WriteLn('Hello ', Name)
  end;
 readln;
end.
```

What's Best?
End of 90ies: Linux+C

Componolit
Secure Systems Engineering

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
  goto fail;
... other checks ...
fail:
  ... buffer frees (cleanups) ...
  return err;
```

What's Best?
Mid 2000s: Linux/FreeBSD/NetBSD+Ada

Componolit
Secure Systems Engineering

```
type Day_type   is range    1 ..   31;
type Month_type is range    1 ..   12;
type Year_type  is range 1800 .. 2100;
type Hours is mod 24;
type Weekday is (Monday, Tuesday, Wednesday,
                 Thursday, Friday, Saturday, Sunday);
type Date is
   record
     Day   : Day_type;
     Month : Month_type;
     Year  : Year_type;
   end record;
```

What's Best?
Today and in the Future

Componolit
Secure Systems Engineering

That's what this talk is about.

Componolit
Secure Systems Engineering

“Use what’s best”

⇒

“Trustworthy systems”

Componolit
Secure Systems Engineering

¬ “Use what’s best”

⇒

???

What's Best?
Our answer (so far) – Outline

Componolit
Secure Systems Engineering

- **Problem**
  - Unsafe programming languages
  - Monolithic systems
- **Solution**
  - Component-based systems
  - Program verification
- **Future**
  - Verification of high-level models
  - Protocol verification

Problem
Unsafe Programming Languages

Componolit
Secure Systems Engineering

- Stragefright (July 2015)
  - Billions of devices affected
  - Remote code execution, privilege escalation
  - As easy as sending video/image
- Problem not solved since
  - > 350 bugs (critical/high)
    - Integer overflows
    - Integer underflows
    - Buffer overflows
    - Heap overflows

Stagefright malware is back! 'Worst Android bug in history' returns for a third time and could infect a BILLION phones

- **Stagefright bug lets attackers take control of older Android handsets**
- **Notorious bug is back for a third time, security research firm claims**
- **Israel-based NorthBit security researchers seem to show the hack in action**
- **It only affects handsets running software older than Android 4.0 and people are being told to upgrade, and install anti-malware apps**

By SARAH GRIFFITHS FOR MAILONLINE
PUBLISHED: 10:59 BST, 21 March 2016 | **UPDATED:** 23:17 BST, 21 March 2016

Share 307 shares 16 View comments

Just as stage fright can plague an actor repeatedly, a notorious Android bug that goes by the same name, is back for a third time.

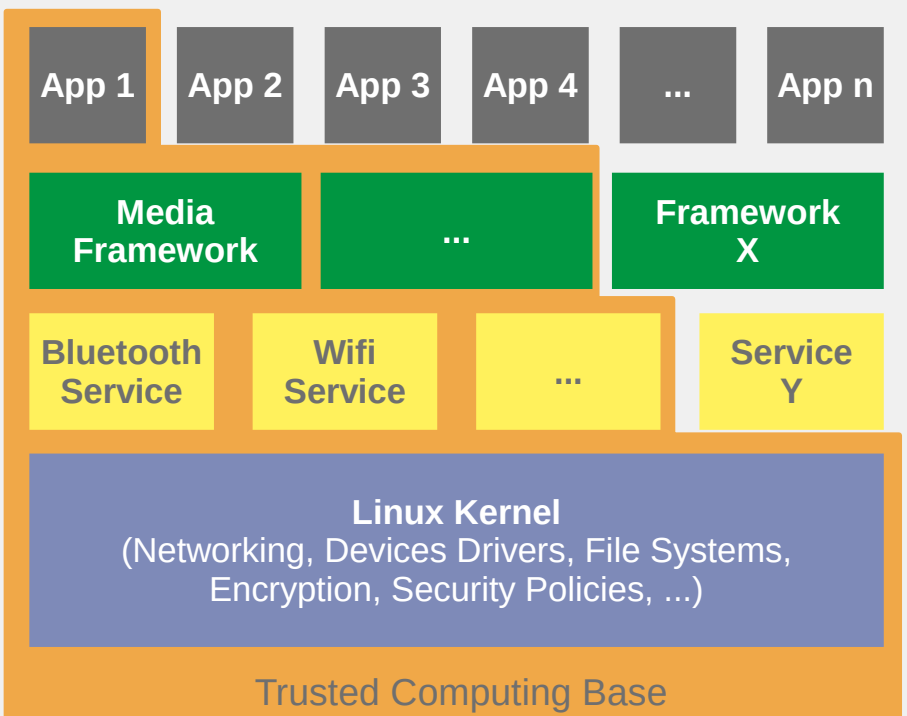
The 'worst Android vulnerability in the mobile OS history' with the potential to infect one billion handsets, allows cybercriminals to hack an Android smartphone in less than 10 seconds.

The newest version of Stagefright, also referred to as Metaphor, tricks a user into visiting a hacker's web page, containing a malicious multimedia file that when

Problem
Monolithic Systems

Componolit
Secure Systems Engineering

- Typical System Architecture



- **Most systems monolithic today**
  - Complex features
  - Large, shared services
  - Weak isolation
- **Consequences**
  - Large Trusted Computing Base
  - High error probability
  - Unrestricted error propagation

Solution
Our Constraints

Componolit
Secure Systems Engineering

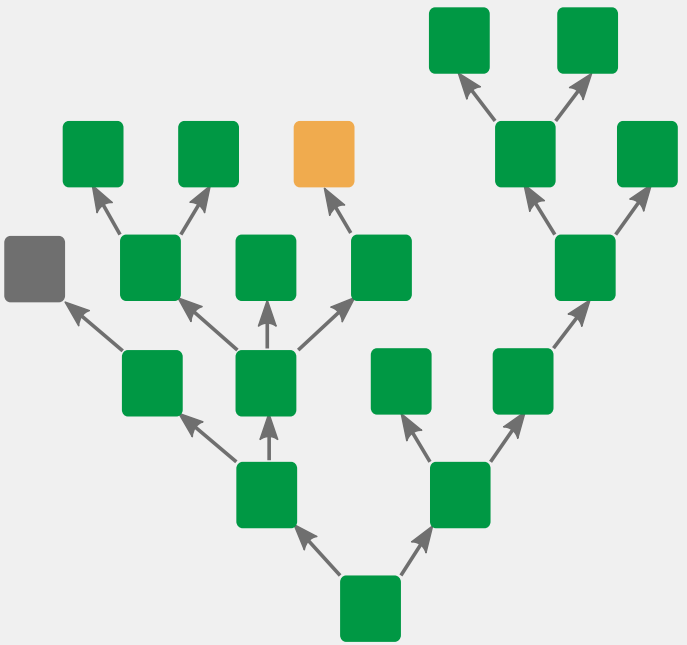
- Minimal Trusted Computing Base
- System/low-level programming
- Low overhead

Solution
The Genode OS Framework*

Componolit
Secure Systems Engineering

- **Recursive system structure**
  - Root: Microkernel
  - Parent: Responsibility + control
  - Isolation is default
  - Strict communication policy
- **Everything is a user-process**
  - Application
  - File systems
  - Drivers, Network stacks

- **Hierarchical System Architecture**



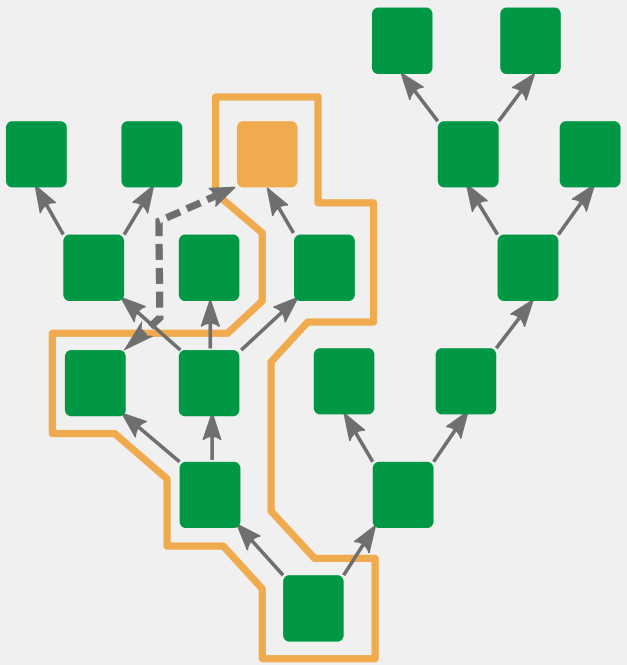
*) https://genode.org

Solution
Minimal Trusted Computing Base

Componolit
Secure Systems Engineering

- **Trusted Computing Base**
  - Software required for security
  - Parents in tree
  - Services used
- **TCB reduction**
  - Application-specific
  - Example: File system

- **Per-application TCB**



Componolit
Secure Systems Engineering

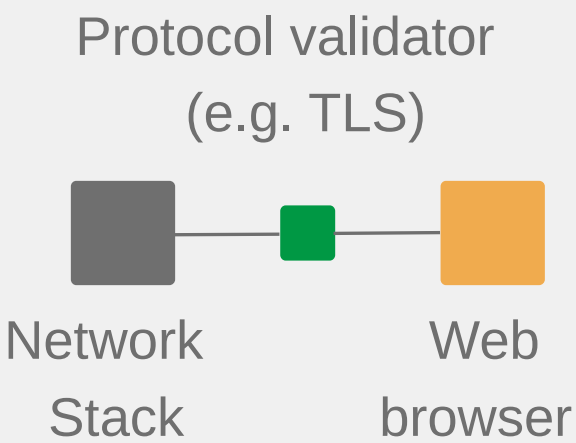
Does that mean we have to reimplement everything?

Strategy #1: Policy Objects

Componolit
Secure Systems Engineering

- Can’t reimplement everything
- **Solution: software reuse**
  - Untrusted software (gray)
  - Policy object (green)
  - Client software (orange)
- **Policy object**
  - Establishes assumptions of client
  - Sanitizes
  - Enforces additional policies

- **Policy objects**

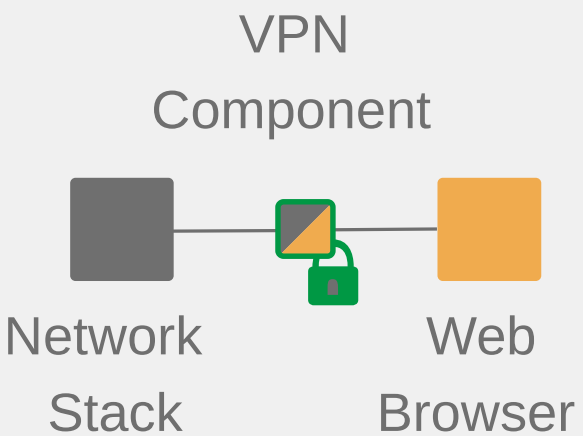


Architecture for Trustworthy Systems
Strategy #2: Trusted Wrappers

Componolit
Secure Systems Engineering

- Untrusted software (gray)
  - E.g. disk, file system, cloud
- Trusted wrapper
  - Mandatory encryption
- Client software (orange)
  - No direct interaction with untrusted components
  - Minimal attack surface

- **Trusted wrapper**

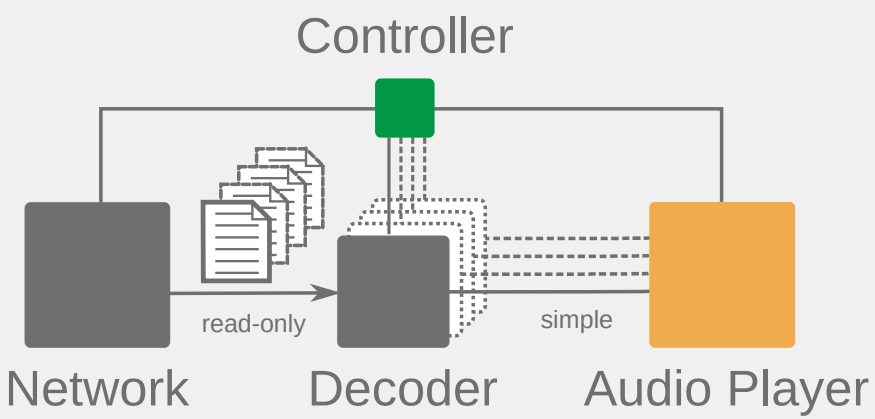


Strategy #3: Transient components

Componolit
Secure Systems Engineering

- **Untrusted software**
  - E.g. Media decoder
  - No chance to get this right!
- **Transient component**
  - Temporarily instantiate untrusted software for single file/stream
  - Expose only simple interfaces (e.g. PCM audio)
  - Cleanup on completion

- **Transient component**



Componolit
Secure Systems Engineering

But, what if trusted components fail?

High-assurance Implementation
A simple task: Calculating abs()

Componolit
Secure Systems Engineering

```
// Calculate absolute of X
1 int abs_value (int X)
2 {
3     if (X > 0) {
4         return X;
5     } else {
6         return -X;
7     };
8 }
```

```
// Let's try abs_value()
abs_value(-12345)       ⇒   12345
abs_value(56789)        ⇒   56789
abs_value(0)            ⇒   0
abs_value(-2147483648)  ⇒   -2147483648
```

High-assurance Implementation
At a glance: SPARK*

Componolit
Secure Systems Engineering

- **Language + verification toolset**
  - Imperative, object-oriented
  - Designed for error avoidance
  - Strong type system
  - Formal contracts

- **Depth of verification is flexible**
  - Data and control flow analysis
  - Dependency contracts
  - Absence of runtime errors
  - Functional correctness

*) http://spark-2014.org

High-assurance Implementation
SPARK benefits

Componolit
Secure Systems Engineering

- **Well-suited for system-level development**
  - Compiled using GCC (via GNAT Ada frontend)
  - Supports runtime-free mode (via profiles)
  - Integration of full Ada and bindings to C
- **Used in various critical and system-level projects**
  - Muen Separation Kernel (https://muen.sk)
  - Satellite software, air traffic control, secure workstation

High-assurance Implementation
Our previous example

Componolit
Secure Systems Engineering

```
 1 function Abs_Value (X : Integer) return Integer
 2 with
 3    -- Uncomment the following line to prove
 4    -- Pre  => X /= Integer'First,
 5    Post => Abs_Value'Result = abs (X)
 6 is
 7 begin
 8    if X > 0 then
 9       return X;
10    else
11       return -X;
12    end if;
13 end Abs_Value;
```

```
Proving...
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
abs_value.adb:11:14: medium: overflow check might fail (e.g. when Abs_Value'Result = 0 and X = -2147483648)
One error.
```

High-assurance Implementation
Bitwise swap using XOR

Componolit
Secure Systems Engineering

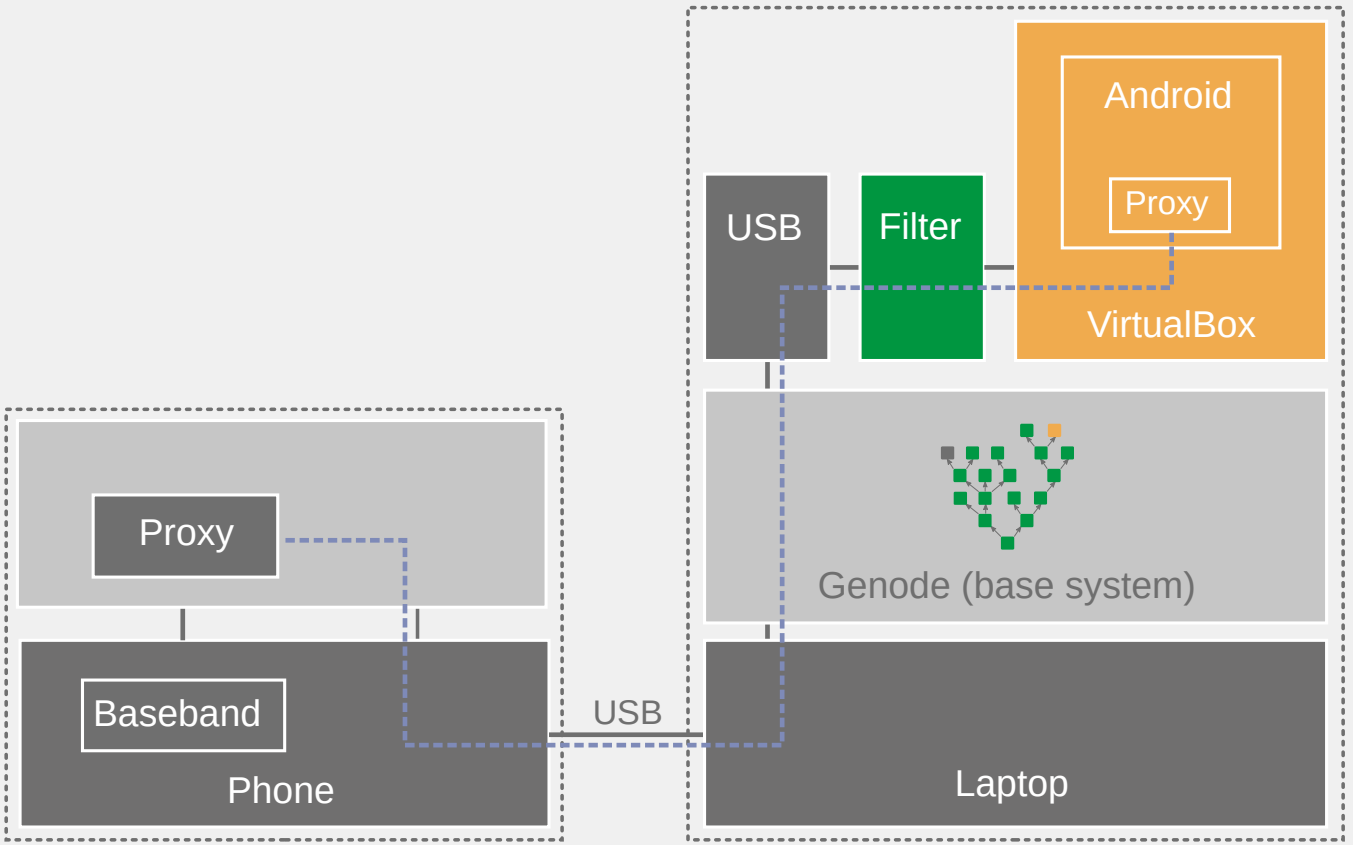
```
 1 with Interfaces; use Interfaces;
 2
 3 procedure Bitwise_Swap (X, Y : in out Unsigned_32) with
 4   Post => X = Y'Old and Y = X'Old
 5 is
 6 begin
 7   X := X xor Y;
 8   Y := X xor Y;
 9 -- Uncomment the following line to prove
10 -- X := X xor Y;
11 end Bitwise_Swap;
```

```
Proving...
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
bitwise_swap.adb:4:11: medium: postcondition might fail,
   cannot prove X = Y'old (e.g. when X = 0 and Y'Old = 4294967295)
One error.
```

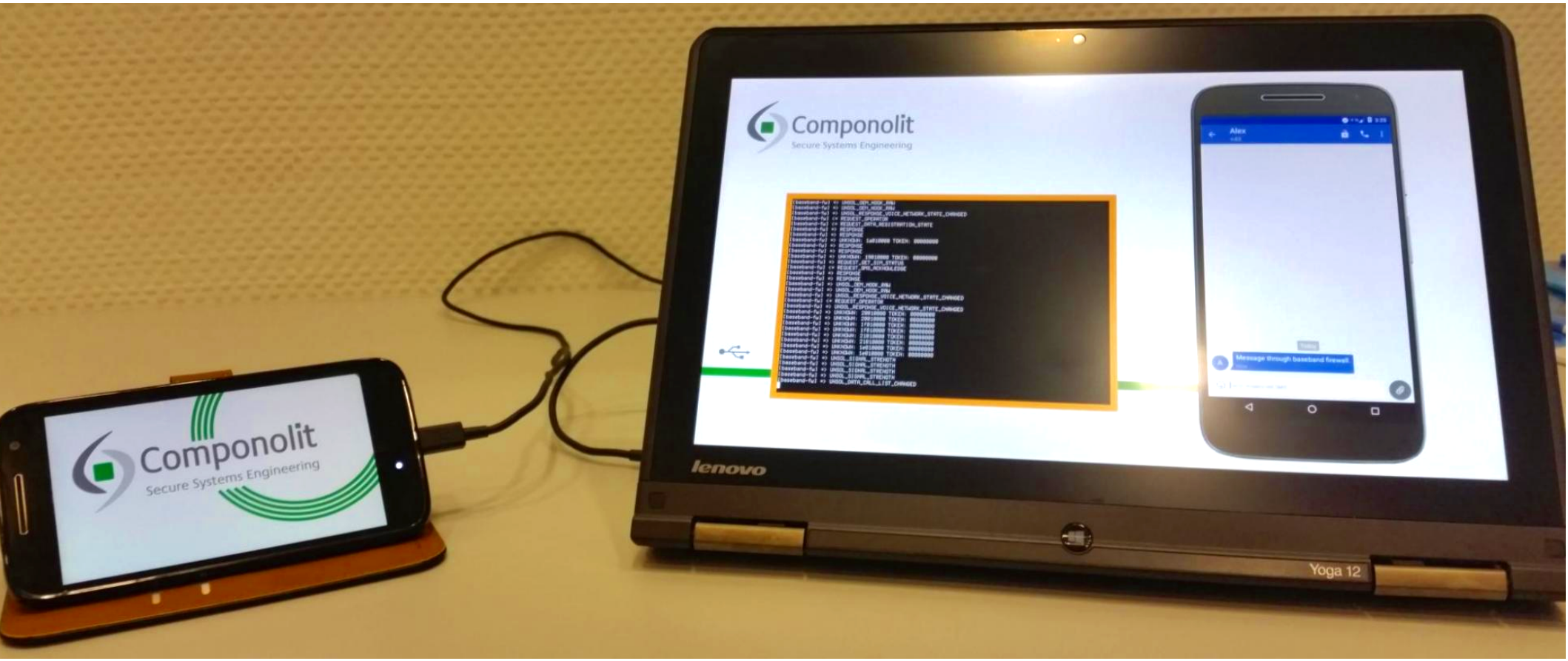
Componolit
Secure Systems Engineering

Let’s put it together.

Componolit Platform
Baseband firewall – Architecture



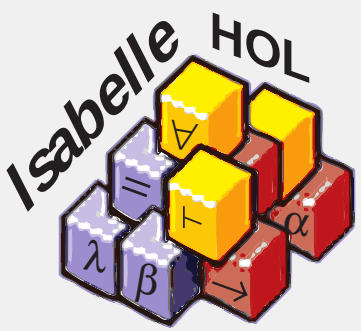
Componolit Platform
Baseband firewall – Implementation



What's Best?
Future: More Verification!

Componolit
Secure Systems Engineering

- **Interactive theorem proving**
  - Functional specification in Isabelle/HOL
  - Prove correspondence with SPARK program



- **Protocol verification**
  - See ourselves implementing communication protocols…
  - ...over and over again
- **Goal**
  - Closed specification of communication protocols
  - Verification of protocol properties using temporal logic
  - Generation of code

Interested in ideas!

Questions?

Componolit
Secure Systems Engineering

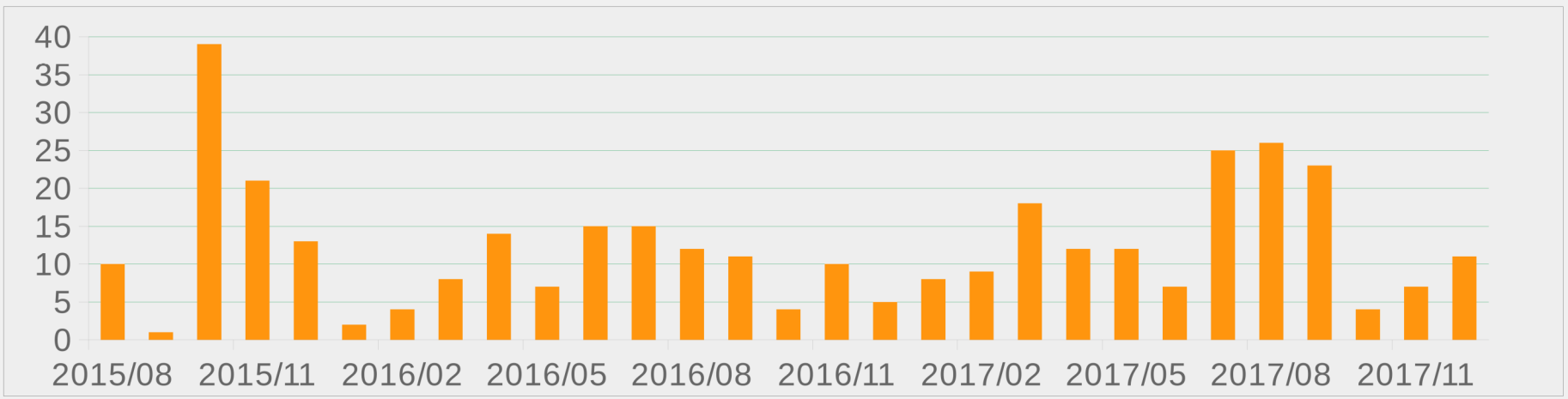
Alexander Senier
senier@componolit.com

@Componolit · componolit.com · github.com/Componolit


Componolit
Secure Systems Engineering

Stagefright
Bugs rated critical/high since 2015

Componolit
Secure Systems Engineering



Media-related Android vulnerabilities (Critical/high, based on https://source.android.com/security/bulletin/)