Componolit
Secure Systems Engineering

# Practical Verification for Software Engineers

Alexander Senier
CODE, 2020-11-11

# Software Security
# Security Vulnerabilities

Componolit
Secure Systems Engineering

- **CVE-1999-0015 – 5.0 MEDIUM – HP-UX, Windows, NetBSD, SunOS**
  - “Teardrop”
- **CVE-2014-0160 – 7.5 HIGH – OpenSSL**
  - “Heartbleed”: Improper Restriction of Operations within the Bounds of a Memory Buffer (CWE-119)
- **CVE-2017-0144 – 8.1 HIGH – Windows**
  - “EternalBlue”: Improper Input Validation (CWE-20)
- **CVE-2017-0785 – 6.5 MEDIUM – Android**
  - “BlueBorne”: Information Exposure (CWE-200)
- **CVE-2017-14315 – 7.5 HIGH – iOS**
  - “BlueBorne”: Improper Restriction of Operations within the Bounds of a Memory Buffer (CWE-119)
- **CVE-2018-10933 – 9.1 CRITICAL – libssh**
  - Improper Authentication (CWE-287)
- **CVE-2019-3560 – 7.5 HIGH – Fizz**
  - Loop with Unreachable Exit Condition (CWE-835)
- **CVE-2019-11477 – 7.5 HIGH – Linux**
  - Integer Overflow or Wraparound (CWE-190)

# Integer Overflow in Fizz

Componolit
Secure Systems Engineering

- **Fizz** [1]
  - TLS 1.3 implementation by Facebook in C++
- **Vulnerability** [2]
  - Infinite loop triggered by unauthenticated remote attacker (denial of service)



1) https://github.com/facebookincubator/fizz 2) https://securitylab.github.com/research/facebook-fizz-CVE-2019-3560

# How to prevent such bugs?

Componolit
Secure Systems Engineering

- **Software Quality Assurance** ⇨ Applied by Facebook
  - Code Reviews
  - Testing
  - Fuzzing
- **Static Code Analysis**
  - Variant Analysis ⇨ Applied by Semmle (acquired by GitHub) using CodeQL
  - Formal Verification

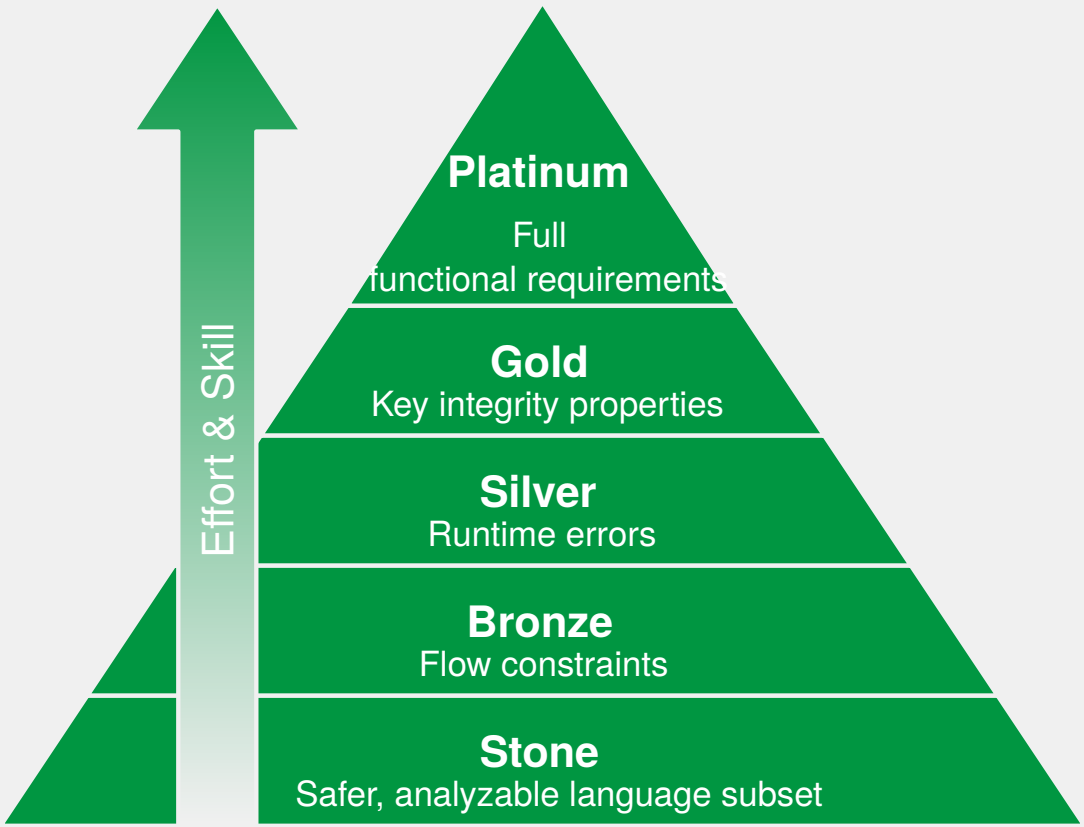
# Formal Verification
## SPARK

Componolit
Secure Systems Engineering


- **Programming Language and Verification Toolset**
  - Based on Ada
  - Compilable with GCC and LLVM
  - Interoperable with Ada, C, C++, Java
  - Customizable runtime
  - Contracts (preconditions, postconditions, invariants)
  - Open Source with commercial support

# SPARK
## Assurance Levels

Componolit
Secure Systems Engineering



# SPARK
## Guarantees and Limitations

Componolit
Secure Systems Engineering

- **Guarantees**
  - Formal verification gives guarantees that traditional software quality assurance cannot provide
  - Systems are secure and safe in all known potentially problematic situations

- **Limitations**
  - Every proof (and in fact every software) has assumptions
  - Proving higher-level properties is harder
  - Limited support for dynamic systems

SPARK

# Example: Integer Overflow in Fizz


```
type UInt16 is range 0 .. 2**16 - 1;
```

```
...
12          declare
13             Length : UInt16 := Read_UInt16 (Cursor);
14          begin
15             Length := Length + 5;
16             Trim_Start (Buf, Length);
...
```

SPARK

# Example: Integer Overflow in Fizz

Componolit
Secure Systems Engineering

```
type UInt16 is range 0 .. 2**16 - 1;
```

```
...
12          declare
13             Length : UInt16 := Read_UInt16 (Cursor);
14          begin
15             Length := Length + 5;
16             Trim_Start (Buf, Length);
...
```

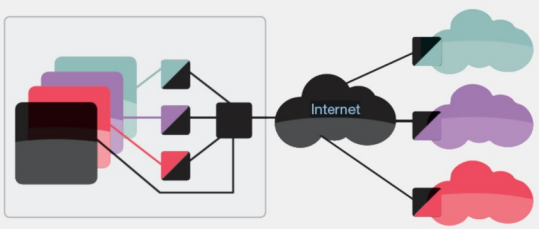
```
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
plaintext_record_layer.adb:15:30: medium: range check might fail (e.g. when
Length = 65531)
```

# SPARK
## Applications in Security

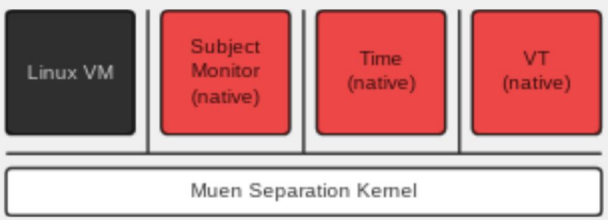
Componolit
Secure Systems Engineering

- Rockwell Collins
  **Turnstile/SecureOne**

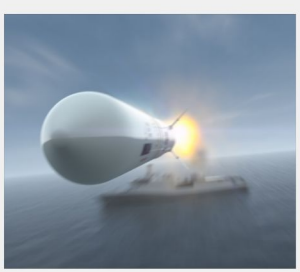

- secunet
  **SINA MLW**



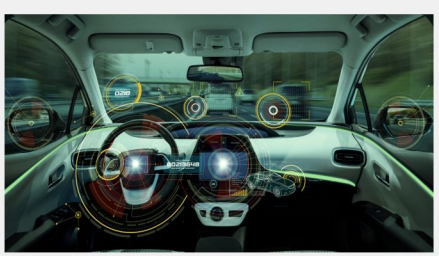

- codelabs
  **Muen**




- MBDA
  **EISR**



- NVIDIA
  **SP/FW**



- ANSSI
  **WooKey**

# Securing Existing Software

Componolit
Secure Systems Engineering

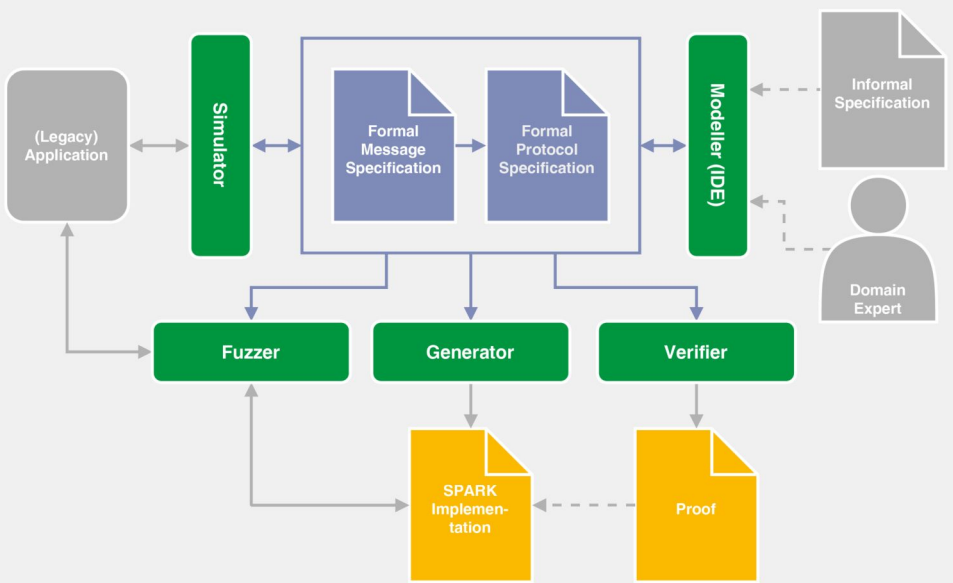
- **Current Situation**
  - Software usually written in unsafe languages (C, C++, …)
- **Migration to Language Supporting Formal Verification**
  - Very expensive when done manually
- **Options**
  - Only replace critical parts of software
  - Use code generation

# Protocol Verification
## RecordFlux

Componolit
Secure Systems Engineering

- Formal Specification of Messages (and Protocol Sessions)
- Model Verification
- Generation of Verifiable Binary Parsers
- Generation of Verifiable Message Serializers

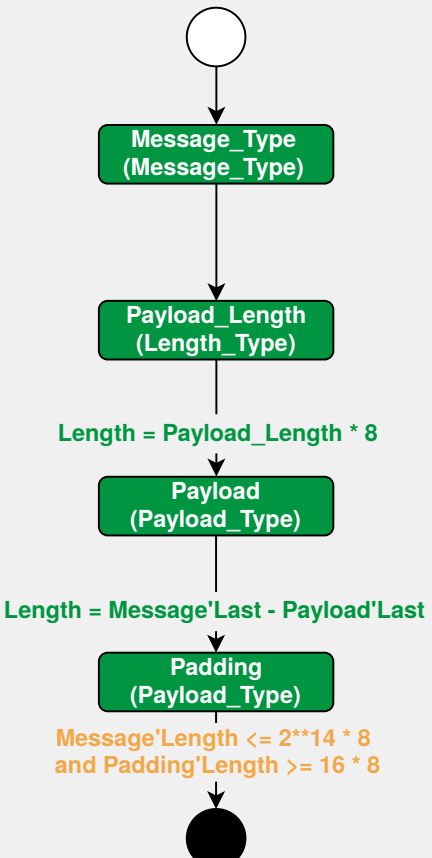
RecordFlux.



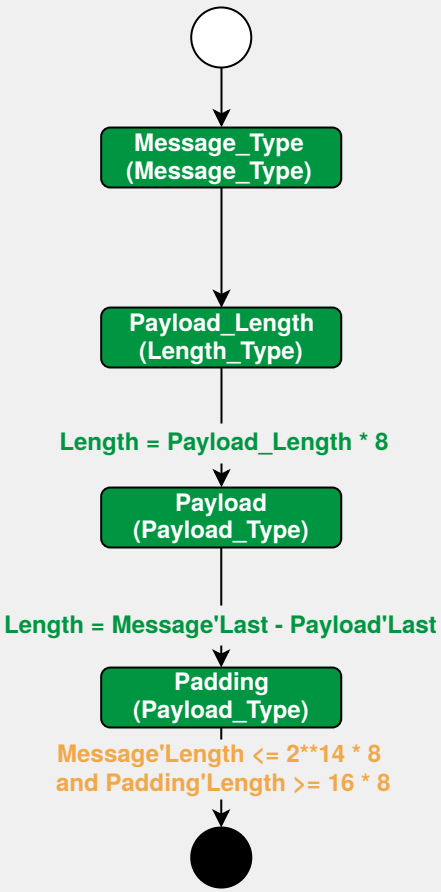
# RecordFlux
## Model

Componolit
Secure Systems Engineering

- Specification language enables precise definition of binary formats (and protocol sessions)
- Definition of complex data formats with value ranges, dependencies and restrictions
- Prevention of critical errors by automated correctness proofs at model level



# RecordFlux
# Specification Language




```
package TLS_Heartbeat is

   type Message_Type is (HEARTBEAT_REQUEST => 1, HEARTBEAT_RESPONSE => 2)
      with Size => 8;

   type Length_Type is range 0 .. 2**14 - 20 with Size => 16;

   type Heartbeat_Message is
      message
         Message_Type : Message_Type;
         Payload_Length : Length_Type
            then Payload
               with Length = Payload_Length * 8;
         Payload : Payload_Type
            then Padding
               with Length = Message'Last - Payload'Last;
         Padding : Payload_Type
            then null
               if Message'Length <= 2**14 * 8 and Padding'Length >= 16 * 8;
      end message;

end TLS_Heartbeat;
```

# RecordFlux
## Guarantees and Limitations

Componolit
Secure Systems Engineering

- **Guarantees**
  - Determinism
  - Liveness
  - Reachability
  - Coherency
  - Completeness

- **Limitations**
  - Some message schemes and complex invariants not supported yet
  - Support for protocol sessions in development

# RecordFlux
## Code Generation

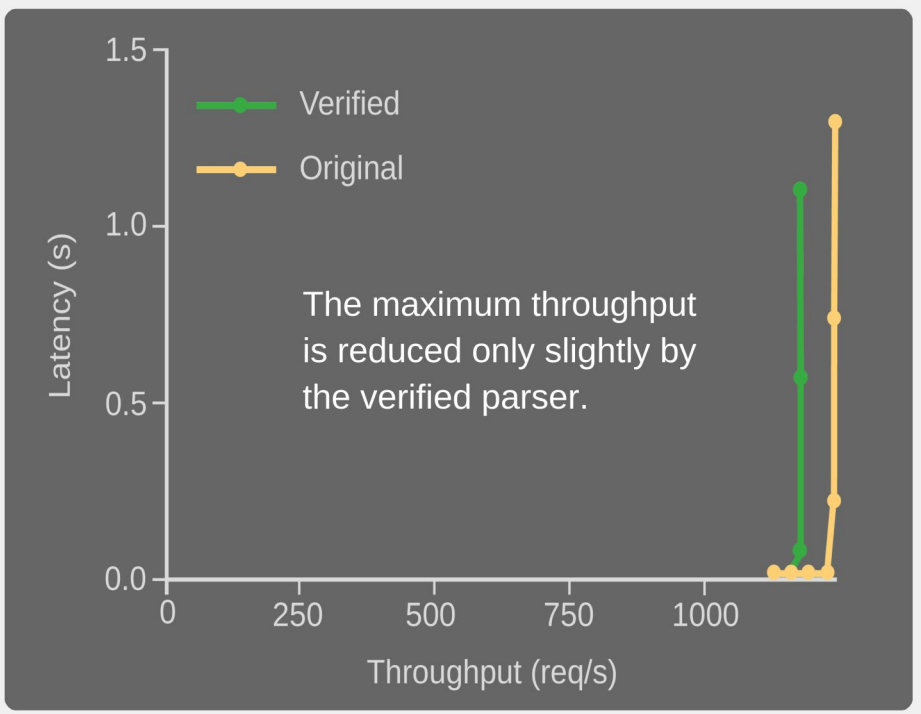
Componolit
Secure Systems Engineering

- Provable message parsers and serializers created in SPARK language
- Absence of runtime errors
- Functional correctness
  - Parsers guarantee received messages comply with specification
  - Serializers ensure creation of correct messages

# RecordFlux
## Case Study: Verified TLS Parser

Componolit
Secure Systems Engineering

- Minimizing attack surface by securing message parsers
- Formalization of TLS 1.3 with RecordFlux
- Replacing C++ parser of Fizz TLS library
- Critical vulnerabilities like CVE-2019-35602 now prevented by proven SPARK code



# RecordFlux
## Project GreenTLS

Componolit
Secure Systems Engineering

- Component-based high-assurance implementation of TLS 1.3
- Critical components in SPARK using RecordFlux
- Current State
  - Complete message specification
  - Initial design and protocol specification
  - Implementation of code generator in progress

https://github.com/Componolit/GreenTLS

Europäische Union
Europa fördert Sachsen.
ESF
Europäischer Sozialfonds
SACHSEN
Diese Maßnahme wird mitfinanziert durch Steuermittel auf Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

# Conclusion

Componolit
Secure Systems Engineering

- **Software Verification using SPARK**
  - Formal verification for software engineers
  - Already used in industries where safety/security matters
  - Flexible cost/benefit trade-off
- **Protocol Verification using RecordFlux**
  - Ensuring correctness of critical part of software: communication protocols
  - Reducing effort and implementation errors by high-level abstraction and automation

# Questions?

Componolit
Secure Systems Engineering

**Alexander Senier**
**senier@componolit.com**

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