Ironclad Apps: End-to-End Security via Automated Full-System Verification

Chris Hawblitzel  Jon Howell  Jay Lorch  Arjun Narayan

Bryan Parno  Danfeng Zhang  Brian Zill
Online and Mobile Security

• Chase Online, the Chase Mobile app and the Chase Mobile website use Secure Socket Layer (SSL) technology

• We periodically review our operations and business practices to make sure they comply with the corporate policies and procedures we follow to protect confidential information
An *Ironclad app* guarantees to remote parties that *every instruction* it executes adheres to a high-level *security spec.*

My password will never leak
Ironclad combines:

• Late launch
• Trusted Computing
• Software verification

```assembly
push ebp
mov ebp, esp
sub esp, 4
mov eax, 8
```
Ironclad combines:

- Late launch
- Trusted Computing
- Software verification

Entire software stack

Reasonable effort

Secure Remote Equivalence

```
push ebp
mov ebp, esp
sub esp, 4
mov eax, 8
```
Verification implies:

- No buffer overflows
- No code injection
- No type-safety flaws
- No information disclosures
- No crypto implementation flaws
- Absence of side channels
- Liveness
- Physical security

We always know what the app will do with private data!
Verification goals

• End-to-end security
  – Complete
  – Low-level

• Rapid development

• Non-goal: Verify existing code

• Long-term: Performance matches unsafe code
Verification methodology

High-level spec

Verifiable, high-level implementation

Ironclad spec translator

Low-level spec

Verifiable assembly language

Verifier

Ironclad compiler

-predicate

IsPrime(p):
\[ 2 \leq p \land \forall x :: 2 \leq x < p \implies p \not\equiv 0 \mod x \]

procedure CheckPrimality(p:int) returns (b:bool)
\{ 
  requires p \geq 0;
  ensures b \equiv \text{IsPrime}(p);
  \{
    var divisor := 2;
    while divisor < p
      invariant 2 \leq divisor \leq p;
      \{
        ... 
      }
    \}
  \}
\}

\text{call} \text{edx} := \text{Mov}(2);

\text{loop}:

\begin{itemize}
  \item \text{invariant} 2 \leq \text{edx} < \text{eax};
  \item \text{invariant} \text{MemInv}(...);
  \item if (edx \geq eax)
    \{ \text{goto loopEnd}; \}
\end{itemize}

\text{mov} \text{edx}, 2

\text{loop}:

\begin{itemize}
  \item \text{invariant} 2 \leq \text{edx} < \text{eax};
  \item \text{invariant} \text{MemInv}(...);
  \item if (edx \geq eax)
    \{ \text{goto loopEnd}; \}
\end{itemize}

\text{cmp} edx, eax
\text{jae} loopEnd

= Trusted

= Untrusted
Verification methodology: Benefits

High-level spec

Rapid development

Ironclad compiler

Verifiable, high-level implementation

Verifiable assembly language

Low-level spec

Simple and declarative

Verifier

Low-level verification

Assembler + Linker

Arbitrarily complex
Writing trustworthy specifications

== 3439 pages == 1,364 lines of spec ( < 60 instructions )

== 795 pages == 296 lines of spec (secure randomness + attestation)

procedure instr_Add(...) x:reg, y:reg)
ensures x := (x + y) % 0x100000000;
...

type core = core(regs:[int]int, eip:int, ..., segments, paging, ...);
type machine = machine(cores:[int]core, mem:[int]int, io:IOState);

Hardware specs
Writing trustworthy specifications

```plaintext
predicate ValidTransition( old_state: NotaryState, 
                         new_state: NotaryState, 
                         request: Request, 
                         response: Response, ...
)
{
    match request
    {
        case AdvanceCounter => 
            response.AdvanceCntrResponse
            && new_state.ctr == old_state.ctr + 1
            && response.sig == RSA_Sign( old_state.ctr, request )
            && ...
    }
}

function SHA256( messageBits: seq<int> ) : seq<int>
    requires |messageBits| < power2(64); 
    requires IsBitSeq( messageBits );
{
    ...
}
```

Idiomatic
Architecture

App spec

Lib specs

OS specs

Hardware specs

App Common

Std. Lib

UDP/IP

Datatypes

RSA

Ethernet

SHA-256

BigNum

Net Driver

TPM Driver

Core Math

Late launch

Segments

IOMMU

Device IO

GC

Verve++
procedure CheckPrimality(p:int) returns (b:bool) {
  requires p >= 0;
  ensures b == IsPrime(p);
  {
    var divisor := 2;
    while divisor < p {
      invariant 2 <= divisor <= p;
      {
        ...
      }
    }
  }
}

F( all possible inputs ) \rightarrow all permitted output words

procedure instr_outb(..., x:reg) requires ????
Solution: Relational verification

procedure instr_inb(..., x:reg) 
    ensures public(x);

procedure instr_outb(..., x:reg) 
    requires public(x);

Declassifier

Declassify X by proving the abstract app would have output X
Rapid verification

Automated tools
- Dafny
- SymDiff
- BOOGIE
- Z3

Modular verification

IronBuild

Shared verification
Ironclad Apps

Password Protector
password  letmein
123456  dragon
12345678  1111
abc123  baseball
monkey  iloveyou
qwerty  trustno1

Notary

Trusted Incrementer

Differentially Private DB

Key pair
Database
Privacy budget

Insert datum
Query
Lessons learned

Automated ≠ Automatic

- ☑️ Non-recursive functions
- ☑️ Addition & subtraction
- ☑️ Mul/div/mod by small constants
- ☒️ Forall/exists
- ☐️ Recursive functions
- ☐️ General mul/div/mod

Verification works!

opaque attributes

Custom math library

1\textsuperscript{st} Version: Secure, but non-functional

How to write concise specs

How to write libraries

Benefits of refinement types

…
Eval: Proof burden

Proof hints : Implementation LoC

- Apps
- Crypto (SHA, HMAC, RSA)
- BigInt Lib
- Math Lib
- Std. Lib (bytes, words, arrays)
- UDP/IP/Ethernet
- Network Driver
- TPM Driver
- OS

Ratio

Average 4.8 : 1
Previously > 25 : 1

~3 person-years
Previously 22+ pys
Eval: System size

Software
- Trusted Spec: 1796 LoC
- Implementation: 6971 LoC

Hardware
- Trusted Spec: 1750 LoC

- SW Impl : Spec = 3.9 : 1
- 41,566 instructions
- 23.1 : 1

Categories:
- Apps
- Crypto (SHA, HMAC, RSA)
- BigInt Lib
- Math Lib
- Std. Lib (bytes, words, arrays)
- UDP/IP/Ethernet
- Network Driver
- TPM Driver
- OS

Lines
Eval: Performance

SHA-256

Cut by 84%
Within 30% of OpenSSL

RSA private (1024)

Improved by 8300x
Still 22x too slow 😞
Related work

• Early security kernels
  – Examples: KVM/370, VAX VMM, SCOMP, GEMSOS
  – Formally specified, but no connection to implementation

• Recent verified systems
  – Examples: seL4, VCC, PROSPER, CompCert, Jitk
  – Focus on one layer
  – Many verify C code => Good performance
  – Typically less automation => More human proof burden
Conclusions

• Ironclad guarantees end-to-end security to remote parties: Every instruction meets the app’s security spec

• Achieved via:
  – New and modified tools
  – A methodology for rapid verification of systems software

• Verification of systems code is quite feasible!

http://research.microsoft.com/ironclad

Thank you!
ironclad@microsoft.com