Formal Verification of Cloud Services

Bryan Parno
Online and Mobile Security

• 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
## Large-Scale Data Breaches Affect Millions of Users

Number of compromised records in recent large-scale data breaches

<table>
<thead>
<tr>
<th>Company</th>
<th>Year</th>
<th>Number of Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adobe</td>
<td>2013</td>
<td>152m</td>
</tr>
<tr>
<td>eBay</td>
<td>2014</td>
<td>145m</td>
</tr>
<tr>
<td>Heartland</td>
<td>2009</td>
<td>130m</td>
</tr>
<tr>
<td>T.J. Maxx</td>
<td>2007</td>
<td>94m</td>
</tr>
<tr>
<td>AOL</td>
<td>2005</td>
<td>92m</td>
</tr>
<tr>
<td>Sony PSN</td>
<td>2011</td>
<td>77m</td>
</tr>
<tr>
<td>US Military</td>
<td>2009</td>
<td>76m</td>
</tr>
<tr>
<td>Target</td>
<td>2014</td>
<td>70m</td>
</tr>
<tr>
<td>Evernote</td>
<td>2013</td>
<td>50m</td>
</tr>
<tr>
<td>Ashley Madison</td>
<td>2015</td>
<td>32m</td>
</tr>
</tbody>
</table>

Source: Media Reports
The Ironclad Project

Chris Hawblitzel
Jon Howell
Manos Kapritsos
Rustan Leino
Jay Lorch
Arjun Narayan

Bryan Parno
Michael Lowell Roberts
Srinath Setty
Danfeng Zhang
Brian Zill
The Ironclad Project

Ironclad Apps
[OSDI 2014]

- App
- Std. Lib
- Common
- UDP/IP
- Datatypes
- RSA
- Ethernet
- SHA-256
- BigNum
- Net Driver
- PM Driver
- Math
- Late launch
- Segs
- IOMMU
- Device IO
- GC

Hardware specs

IronFleet
[SOSP 2015]

Everest HTTPS
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
• Secure hardware
• Software verification
Ironclad combines:

- Late launch
- Secure hardware
- Software verification

Secure Remote Equivalence

Entire software stack

Reasonable effort

push ebp
mov ebp, esp
sub esp, 4
mov eax, 8
Verification goals

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

• Rapid development

• Non-goal: Verify existing code

• Long-term: Performance matches unsafe code
Prior work

• Small components
  – E.g., RSA-OAEP
  – Localized guarantee

• Single layers
  – No end-to-end guarantee
Our formal, end-to-end guarantee

• End-to-end secure communication with provably secure assembly code
• Implies:
  – No buffer overflows
  – No code injection
  – No type-safety flaws
  – No information disclosures
  – No crypto impl flaws
  ...

Math
TPM Driver
Net Driver
UDP/IP
Datatypes
RSA
Ethernet
SHA-256
BigNum
Std. Lib
Common
App
Late launch
Segs
IOMMU
Device IO
GC
IO
Verifiability, high-level implementation

procedure CheckPrimality(p:int) returns (b:bool)

requires p >= 0;
ensures b == IsPrime(p);

{ 
  var divisor := 2;
  while divisor < p
    invariant 2 <= divisor <= p;
    
    if (divisor >= eax) goto loopEnd;
    
    mov edx, 2
    ... 
    cmp edx, eax 
    jae loopEnd

  p % x != 0;
}

Low-level spec

Verifier

Verifiability, assembly implementation

Verifica3on methodology

High-level spec

Ironclad spec translator

Ironclad compiler

Verifiability, high-level implementation

Ironclad assembler + linker

predicate IsPrime(p:int)

{ 
  2 <= p &&
  forall x :: 2 <= x < p ==>
  p % x != 0
}

procedure CheckPrimality(p:int) returns (b:bool)

requires p >= 0;
ensures b == IsPrime(p);

{ 
  var divisor := 2;
  while divisor < p
    invariant 2 <= divisor <= p;
    
    if (divisor >= eax) goto loopEnd;
    
    mov edx, 2
    ... 
    cmp edx, eax 
    jae loopEnd

  p % x != 0;
}
Verification methodology: Benefits

- **High-level spec**: Verifiable, high-level implementation in Dafny
- **Ironclad compiler**
- **Low-level spec**: Verifiable assembly language (BoogieX86)
- **Verifier**: Simple and declarative
- **Assembler + Linker**: Performance optimization
- **Low-level verification**: Rapid development
- **Arbitrarily complex**

**Benefits**:
- Rapid development
- Arbitrarily complex
- Performance optimization
- Simple and declarative
- Low-level verification
Ironclad Apps

Password Protector
password  letmein
123456  dragon
1234567 1111
abc123  baseball
monkey  iloveyou
qwerty  trustno1

Notary

Trusted Incrementer

Differentially Private DB

Key pair
Database
Privacy budget

Insert datum
Query

0373 0027
1288 9823
Key Challenge: Specifying security

![Diagram showing the relationship between hardware specs, software components, and app specifications.](image-url)
Specifying the DiffPriv DB

datatype DiffPrivState(
    keys: RSA,
    budget: real,
    db: seq<Row>
)

predicate Transition(old_state: DiffPrivState,
    new_state: DiffPrivState,
    request: Request,
    reply: Reply)
{
    match request
    ...
    case Query =>
        request.spend < old_state.budget
        && new_state.budget == old_state.budget - request.spend
        && reply.answer == RunQuery(request.query, old_state.db)
        + ComputeNoise(request.spend)
        && ...
}
Key Challenge: Specifying security

```
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);
```
procedure instr_inb(..., x: reg)

\[ F(\text{all possible inputs}) \rightarrow \text{all permitted output words} \]

procedure instr_outb(..., x: reg)

requires ????

push ebp
mov ebp, esp
sub esp, 4
mov eax, 8

Connecting the app to the hardware
Connecting the app to the hardware

**Key idea**
State-machine-based relational verification

```plaintext
procedure instr_inb(..., x:reg)
  public(x);

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

public_request reply
  push ebp
  mov ebp, esp
  sub esp, 4
  mov eax, 8

public_reply
```

**Guarantee**
System’s output is indistinguishable from that of the abstract app
Evaluation questions

• How does verification affect development?

• How does verification affect performance?
Eval: Developer experience

• Rapid feedback via aggressive caching:
  – Method-level via Dafny’s IDE plugins
  – File-level via new Dafny *include* feature
  – Cloud-aided verification

---

**Full Build**
Sequential: 30+ hours
Cloud: 8 minutes
Eval: Developer experience

Verified software works!

1st Version: Secure, but non-functional
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: 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: Performance

SHA-256

- v1
- v2
- Current
- OpenSSL

Cut by 84%
Within 30% of OpenSSL

RSA private (1024)

- v1
- v2
- v3
- Current
- OpenSSL

Improved by 32,000x
Only 5.6x to go...
The Ironclad Project

Ironclad Apps
[OSDI 2014]

IronFleet
[SOSP 2015]

Everest HTTPS

Crypto Algorithms
- X.509
- ASN.1
- TLS
- RSA
- SHA
- ECDH
- 4Q
- Network buffers

Network buffers

Hardware specs

App

STD Lib
Common
UDP/IP
Datatypes
RSA
Ethernet
SHA-256
BigNum
Net Driver
PM Driver
Math
Late Launch
Segs
IOMMU
Device IO
GC

IronFleet
[SOSP 2015]
IronFleet: Proving Practical Distributed Systems Correct [SOSP 2015]
"not one of the properties claimed invariant in [PODC] is actually true of it."

Under the same assumptions made in the Chord papers, the [SIGCOMM] version of the protocol is not correct, and not one of the properties claimed invariant in [PODC] is actually invariantly true of it. The [PODC] version satisfies one invariant, but is still not correct. The results are presented by means of counterexamples to the invariants in Section 4. In preparation for the results, Section 2 gives a brief summary of the protocol and failure assumptions, and Section 3 introduces the invariants.
We show how to build complex, efficient distributed systems whose implementations are provably safe and live.

System will not take incorrect actions

Implementations are correct, not just abstract protocols
Our verified distributed systems

IronRSL
Replicated state library

IronKV
Sharded key-value store

Complex with many features:
- state transfer
- log truncation
- dynamic view-change timeouts
- batching
- reply cache
Running example:
IronRSL replicated state library

Safety property:
Equivalence to single machine

Liveness property:
Clients eventually get replies
Verification rules out all bugs by construction

- Invariant violations
- Race conditions
- Integer overflow
- Buffer overflow
- Parsing errors
- Marshalling errors
- Deadlock
- Livelock
Background: Refinement

Specification is a simple, logically centralized state machine

```plaintext
type SpecState = int

function SpecInit(s:SpecState) : bool
{
    s == 0
}

function SpecNext(s:SpecState, s':SpecState) : bool
{
    s' == s + 1
}

function SpecRelation(realPackets:set<Packet>, s:SpecState) : bool
{
    forall p, i :: p in realPackets &&
        p.msg == MarshallCounterVal(i) ==> i <= s
}
```
Implementacon

```
method Main()
{
    var s: ImplState;
    s := ImplInit();
    while (true) {
        s := EventHandler(s);
    }
}
```

Host implementation is a single-threaded event-handler loop
Proving correctness is hard

- Subtleties of distributed protocols
  - Maintaining global invariants
  - Ensuring progress
  - Dealing with hosts acting concurrently
- Complexities of implementation
  - Using efficient data structures
  - Memory management
  - Avoiding integer overflow
Two-level refinement
Distributed System Layer

\[
\text{seq<int>}
\]
\[
\text{array<uint64>}
\]

function ProtocolNext(s: HostState, s’: HostState) : bool

method EventHandler(s: HostState) returns (s’: HostState)

type Message = MessageRequest() | MessageReply() | ...

type Packet = array<byte>

Abstract Host

Implementation

Refines
Distributed System Layer

Spec Refines Distributed System Model

Is Part Of

Abstract Host Refines Implementation
Concurrency containment strategy

Ideal World

Host A, Step 1 | Host B, Step 1 | Host A Step 2

Real World

R C R S C S R C C S S
Concurrency containment strategy

Ideal World

Host A, Step 1
Host B, Step 1
Host A, Step 2

Reduction obligation: Step $\triangle R^*C^*S^*$

Real World
Evaluation questions

• How does verification affect development?

• How does verification affect performance?
Developer experience: Proof burden

<table>
<thead>
<tr>
<th>Library</th>
<th>Functional:</th>
<th>DS Safety:</th>
<th>DS Live:</th>
</tr>
</thead>
<tbody>
<tr>
<td>IronFleet</td>
<td>3.5:1</td>
<td>5.4:1</td>
<td>7.8:1</td>
</tr>
<tr>
<td>Ironclad</td>
<td>4.8:1</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>seL4</td>
<td>25.7:1</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Verdi</td>
<td></td>
<td>94.0:1</td>
<td></td>
</tr>
</tbody>
</table>

Lines of code

Safety
Liveness
IronRSL performance

- Challenges reasoning about the heap
- Each optimization requires a proof

We trade some performance for strong correctness, but no fundamental reason verified code should be slower

Adding batching (~2 person-months) improved performance by 5x
IronKV performance

Max throughput: 28,800 req/sec
Throughput up to 75% of Redis
The Ironclad Project

Ironclad Apps
[OSDI 2014]

IronFleet
[SOSP 2015]

Everest HTTPS

Crypto Algorithms
X.509
ASN.1
TLS
***
RSA
SHA
ECDH
4Q
Crypto Algorithms
Network buffers

Transactions

Std. Lib
Common
UDP/IP
Datatypes
RSA
Ethernet
SHA-256
BigNum
Net Driver
PM Driver
Math
Late launch
Segs
IOMMU
Device IO
GC

Hardware specs

App

IronFleet Apps

App

Ironclad Apps

App
The HTTPS Ecosystem is critical

• Most widely deployed security protocol?
  – 40% all Internet traffic (+40%/year)
• Web, cloud, email, VoIP, 802.1x, VPNs, ...
The HTTPS Ecosystem is complex
The HTTPS Ecosystem is buggy

• 20 years of attacks & fixes
  Buffer overflows
  Memory management
  Incorrect state machines
  Lax certificate parsing
  Weakly or badly implemented crypto
  Side channels
  Error-inducing APIs
  Flawed standards
  ...

• Many implementations
  OpenSSL, Schannel, NSS, ...
  Still patched every month!

Services & Applications

Clients

Edge  cURL  WebKit  Skype  IIS  Apache  Nginx

Servers

Certification Authority

X.509  ASN.1

HTTPS

TLS

RSA  SHA

Untrusted network (TCP, UDP, ...)
# Unsolved Attacks against HTTPS

<table>
<thead>
<tr>
<th>SSL Stripping (Marlinspike)</th>
<th>Cookie-based Attacks (various variants)</th>
<th>CRIME / BREACH (Rizzo, Duong et al.)</th>
<th>Virtual Host Confusion (Delignat-Lavaud)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS is optional in HTTP and can be disabled by an active attacker</td>
<td>Shared cookie database for HTTP and HTTPS can be used to mount various session fixation and login CSRF attacks.</td>
<td>Attackers can easily mount adaptive chosen-plaintext attacks. Encryption after compression can leak secrets through length.</td>
<td>HTTPS servers do not correlate transport-layer and HTTP identities, leading to origin confusion</td>
</tr>
<tr>
<td>Mitigated by correct use of HTTP Strict Transport Security (HSTS)</td>
<td>Mitigated by new binding proposals (ChannelID, Token Binding). Mitigation is not widely implemented.</td>
<td>Mitigated by refreshing secrets (e.g. CSRF tokens). Some protocol-specific mitigations (QUICK, HTTP2)</td>
<td>Mitigated by configuration of HTTPS servers with strict host rules</td>
</tr>
<tr>
<td>Mitigation is not widely used and vulnerability is still widespread in practice.</td>
<td>Extremely difficult to mitigate in all browsers with current technologies. Can be used to attack almost every website.</td>
<td>Ad-hoc mitigation; attack is still widespread in practice as HTTP compression remains popular.</td>
<td>Ad-hoc mitigation; attack is still widespread in practice.</td>
</tr>
</tbody>
</table>

---

### Related Dates
- 2006
- 2007
- 2008
- 2009
- 2010
- 2011
- 2012
- 2013
High-Profile TLS Attacks

- **Crypto failures**
  - MD5

- **Protocol weaknesses**
  - Renegotiation Attack
  - BEAST (Rogaway 02)
  - ECDHE Cross-protocol Attack

- **Implementation bugs**
  - OpenSSL entropy
  - CRIME
  - Lucky13
  - Heartbleed
  - POODLE
  - EarlyCCS
  - POODLE E
  - FREAK
  - SLOTH
  - DROWN

- **Dates**
  - 2007
  - 2008
  - 2009
  - 2010
  - 2011
  - 2012
  - 2013
  - 2014

- **Encryption and Hashing Issues**
  - RC4
  - RSA 512 bit
  - SHA1
  - MD5
  - OpenSSH entropy
  - CRIM
  - E
  - RSA 512 bit
  - SHA1
  - High-Profile TLS AOacks
A Timeline of Recent PKI Failures

Crypto failures
- Debian OpenSSL entropy bug
- Bleichenbacher’s e=3 attack on PKCS#1 signatures
- HashClash rogue CA (MD5 collision) Stevens et al.
- Flame malware NSA/GCHQ attack against Windows CA
- 512 bit Korean School CAs

Basic constraints not enforced (recurring catastrophic bug)
- OpenSSL null prefix
- Usage-unrestricted VeriSign certificates
- GnuTLS X509v1

Name constraints failures
- VeriSign NetDiscovery
- VeriSign hack
- Comodo hack
- DigiNotar hack

Formatting & semantics
- VeriSign hack
- Trustwave
- TÜRKTRUS

CA failures
- ANSSI
- DROWN KeyUsage
- BERSerk
- OpenSSL CVE-2015-1793

The SHAppening

2006 2007 2008 2009 2010 2011 2012 2013
## Side Channel Challenge (Attacks)

<table>
<thead>
<tr>
<th>Protocol-level side channels</th>
<th>Traffic analysis</th>
<th>Timing attacks against cryptographic primitives</th>
<th>Memory &amp; Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS messages may reveal information about the internal protocol state or the application data</td>
<td>Combined analysis of the time and length distributions of packets leaks information about the application</td>
<td>A remote attacker may learn information about crypto secrets by timing execution time for various inputs</td>
<td>Memory access patterns may expose secrets, in particular because caching may expose sensitive data (e.g. by timing)</td>
</tr>
<tr>
<td>• Hello message contents (e.g. time in nonces, SNI)</td>
<td>• CRIME/BREACH (adaptive chosen plaintext attack)</td>
<td>• Bleichenbacher attacks against PKCS#1 decryption and signatures</td>
<td>• OpenSSL key recovery in virtual machines</td>
</tr>
<tr>
<td>• Alerts (e.g. decryption vs. padding alerts)</td>
<td>• User tracking</td>
<td>• Timing attacks against RC4 (Lucky 13)</td>
<td>• Cache timing attacks against AES</td>
</tr>
<tr>
<td>• Record headers</td>
<td>• Auto-complete input theft</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Remote timing attacks are practical

- **Bleichenbacher**
- **Vaudenay**
- **AES cache timing**

### Side-channel leaks in Web applications

- **ECDSA timing**
- **Tag size**

### Timing attacks against AES

- **CRIME**
- **Lucky13**
- **BREACH**
- **DROWN**

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>...</td>
</tr>
<tr>
<td>2006</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
</tr>
</tbody>
</table>
Everest:
Deploying Verified-Secure Implementations in the HTTPS Ecosystem
Everest Goals

- Fully verified replacement
- Widespread deployment
- Trustworthy, usable tools

```
$ apt-get install verified_https
$ /etc/init.d/apache2 restart
```
Research Questions

• How do we decide whether new protocols are secure?
  – Especially when interoperating with insecure protocols

• Can we make verified systems as fast as unverified?

• How do we handle advanced threats?
  – Ex: Side channels

• Why should we trust automated verification tools?

• How can verification be more accessible?
  – Especially to non-experts in verification
Summary

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

• IronFleet extends these techniques to prove the safety and liveness of distributed systems

• Everest will showcase the power of verification and its applicability to real-world security problems

• Verification of systems code is possible, and we’re scaling it to even larger more complex systems

https://github.com/Microsoft/Ironclad

Thank you!
parno@cmu.edu