DATA – Differential Address Trace Analysis: Finding Address-based Side-Channels in Binaries

Samuel Weiser*, Andreas Zankl*, Raphael Spreitzer*,
Katja Miller*, Stefan Mangard*, and Georg Sigl*

August 16th 2018 – 27th USENIX Security Symposium

Motivation
Side-channel Leakage

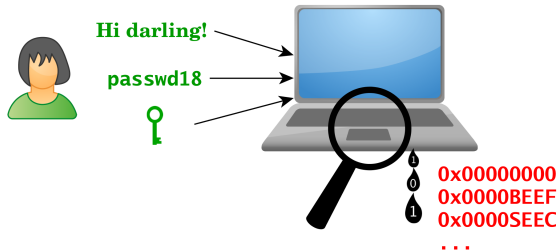
What is Address Leakage?

Motivation
Address Leakage

Secret information "somehow" leaked through **Memory access pattern**

Data leakage

```
1 int T[] =
2   {3,0,1,2};
3
4 a = T[sec];
```



Control-flow leakage

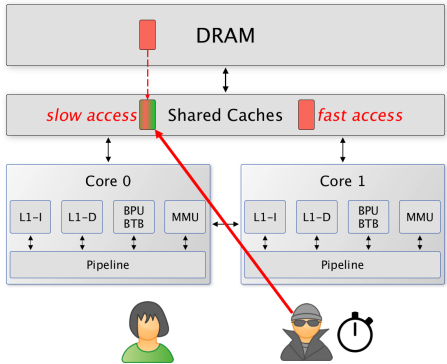
```
1 if (sec)
2   left();
3 else
4   right();
```

Secret-dependency

Motivation
Address Leakage

Data leakage

```
1 int T[] =
2   {3,0,1,2};
3
4 a = T[sec];
```



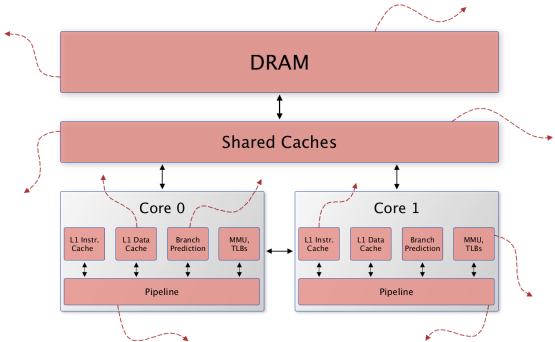
Control-flow leakage

```
1 if (sec)
2   left();
3 else
4   right();
```

Motivation
Address Leakage

Data leakage

```
1 int T[] =
2   {3,0,1,2};
3
4 a = T[sec];
```



Control-flow leakage

```
1 if (sec)
2   left();
3 else
4   right();
```

Capture all such attacks by **Address leakage**

Our Objective:

Analyze program
Find many address leaks
Be efficient

Methodology

Static analysis

- Symbolic execution
- Upper leakage bound (zero false negatives)
- Problems:
  - Imprecision (false positives)
  - Interpreted code
  - Performance

Dynamic analysis

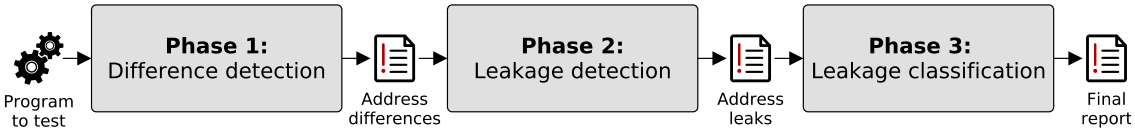
- Concrete execution
- Real leaks (zero false positives)
- Problem
  - Coverage (false negatives)

DATA – Differential Address-Trace Analysis

DATA
Our Contribution

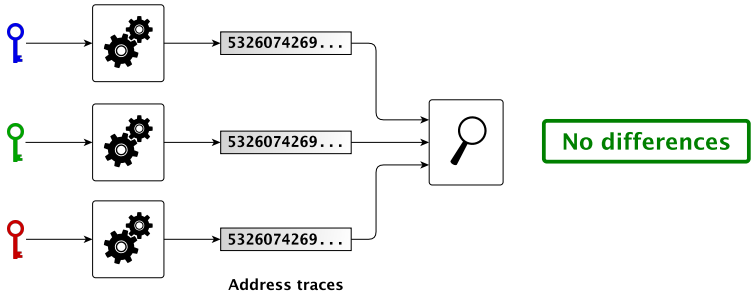
1. **Approach**
   - User specifies what is secret
   - Tool finds secret-dependent address leaks
   - Tool analyzes severity of leaks
2. **Accuracy**
   - Data and control-flow leaks
   - Low false positives & negatives
   - Non-determinism
3. **Practicality**
   - Fully automated
   - Fast and openly available
   - Found and fixed critical vulnerabilities in OpenSSL
   - Analyzed interpreted code (PyCrypto)

DATA
Overview



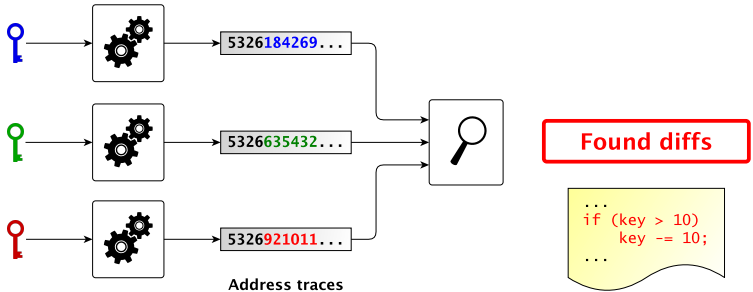
DATA

Phase 1: Difference Detection



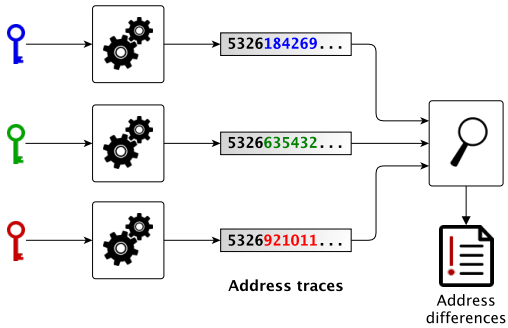
DATA

Phase 1: Difference Detection



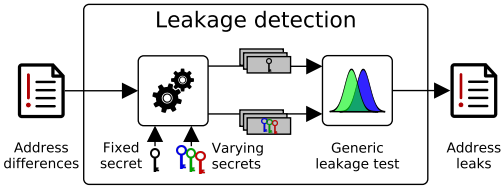
DATA

Phase 1: Difference Detection



- Reduce false negatives
- Binary instrumentation
- Capture all address leakage
- Sequential trace comparison
- Trace re-alignment on CF-leaks

DATA
Phase 2: Leakage Detection



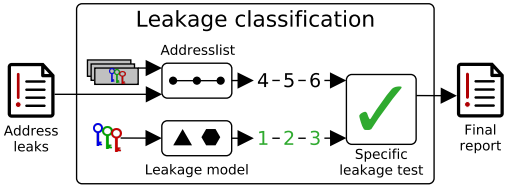
Trace recording

- Only instrument code with address differences
- Execute with fixed and varying input set
- Record short traces for each input set

Generic leakage test

- Build address distributions
- If not similar ⇒ **leak!**
- Accumulate in leak report

DATA
Phase 3: Leakage Classification



Preparation

- Collect list of addresses per leak
- Leakage model: property or part of secret inputs
- Build pairs: Addresslist ↔ LeakageModel(inputs)

Specific leakage test

- Test pairs for (non-)linear relations
- If related: **model ⇒ info loss**
- Accumulate in final report

Practical Results

Practical Results
Overview

Confirmed Known Leaks

- Symmetric ciphers – lookup tables
- AES bit-sliced – key schedule
- ECDSA – wNAF point multiplication

Found New Leaks

- DSA – bypass constant-time mod. inv.
- RSA – bypass constant-time mod. exp.
- AES-NI & PEM keys – hex parsing

Performance: < 4 CPU hours, < 4.5GB RAM, < 1GB storage

Conclusion

Conclusion
Takeaways

- **DATA - Differential Address Trace Analysis**
  - Any address-based leaks ..... caches, DRAM, etc.
  - Low false positives / negatives ..... guarantees / strategies
  - Severity ..... leakage models
- **Benefits for developers**
  - Automated ..... easy to use/no annotations
  - Efficient ..... interpreters, commodity hardware
  - Practical ..... new vulnerabilities in OpenSSL
- **Work in progress:** GUI, improved performance, your ideas...

`https://github.com/Fraunhofer-AISEC/DATA`