Formal Verification of SciTokens

Bach Hoang
SciAuth Student Fellow

Phuong Cao
NCSA co-mentor

Jim Basney
NCSA mentor
Authentication bugs in SciToken will have catastrophic consequences to NSF projects.

**NSF Critical Infrastructure**
- LIGO
- ACCESS
- LSST
- OSG

**Total Investments**
> $300M

**Trust & Safety Issues**
- Data breach
- Misuse of critical infrastructure
- Losing U.S. leading edges

Identity validation
Scope validation
Other bugs
Our initial approach is to understand specs, code, and manual auditing.

SciTokens specs are well-defined

SciTokens code is sophisticated (~5K LOC)

Where are the bugs?

Understand Scitoken Model

Manual auditing

- Input analysis (path, scope, audience)
- Concurrency analysis

Critical, confirmed bug reporting

- Time-of-check-time-of-use on scope permission.
- Path traversal attack on scope path (.././../)

These bugs allow unauthorized access of protected data. However, they are independent of SciToken’s model correctness.
Despite that SciToken has manual auditing, quality assurance (QA), and testing, they remain inadequate in unearthing critical bugs.

Formal verification of SciToken implementations is urgently needed.
Overview of Dafny: a verification-ready programming language

- A programming language with built-in specification constructs
- Supporting formal specification through
  - Preconditions,
  - Postconditions,
  - Loop Invariant
  - Termination specifications
- Formal reasoning through code using Hoare logic
  \[ \{P\}C\{Q\} \]

Putting Dafny in perspective

High-level programming language Dafny

Compiled to

Intermediate verification language Boogie

Verified by

Satisfiability modulo theories (SMT) Solver Z3

Why Dafny for scitokens?

- Static verification of programs at compile time, avoiding *data leak* and *system compromise* at runtime.
- Dafny compiler produces both the *proof* and *cross-platform, verified executable code*
  - Dafny code is compiled to .NET Common Intermediate Language (CIL)
  - CIL is translated into *six* languages including *compiled* (Java, Go, C++) and *interpreted* (Javascript, PHP, Python)
Example: verifying scitokens audience, a critical function, with Dafny.

Scitokens audience specs

Audience verification logic

Audience verification impl (scitokens.py)

Dafny implementation

Specs in natural language

Truth table

Python implementation

Errors & Counter examples
Example (2): verifying scitokens **scope** with Dafny.

```python
def _validate_scope(self, value):
    if not isinstance(value, str):
        raise InvalidAuthorizationResource
        ("Scope is invalid. Must be a space separated string")
    if self._test_access:
        if not self._test_path:
            norm_requested_path = '/'
        else:
            norm_requested_path =
            urllib.parse.urlparse(self._test_path)
    # Split on spaces
    for scope in value.split(' '):
        authz, norm_path = self._check_scope(scope)
        if (self._test_authz == authz)
            and norm_requested_path.startwith(norm_path):
            return True
    return False
```

```dafny
method Validate_Scope(
    value : string,
    test_access : bool,
    norm_requested_path : string,
    token_scope : seq<seq<string>>
)
returns 
  t : bool,
  result : seq<seq<string>>
requires |value| > 0;
```

```dafny
var scope := split1(value, ' ');
var iter := 0;
var authz := "";
var norm_path := "";
if (test_access == true) {
    while (iter < |scope|)
        invariant 0 <= iter <= |scope|;
        { authz, norm_path, j := Check_Scope(scope[iter]);
```
Summary & Future Work

SciTokens
Codebase & Specs

Formal Verification

Correct-by-construction
program synthesis

Found two bugs
- Time-of-check-time-of-use
- Path traversal attack

Verified three critical functions
- Validate audience
- Validate scope
- Check scope

Learned formal verification tools
- Hoare logic calculus
  \( \{P\}C\{Q\} \)
- SMT/SAT solvers (z3, dafny)

Generated correct implementation in six languages
- Compiled (Java, C++, Go)
- Interpreted (PHP, Javascript, Python)

Putting human-in-the-loop with formal verification.