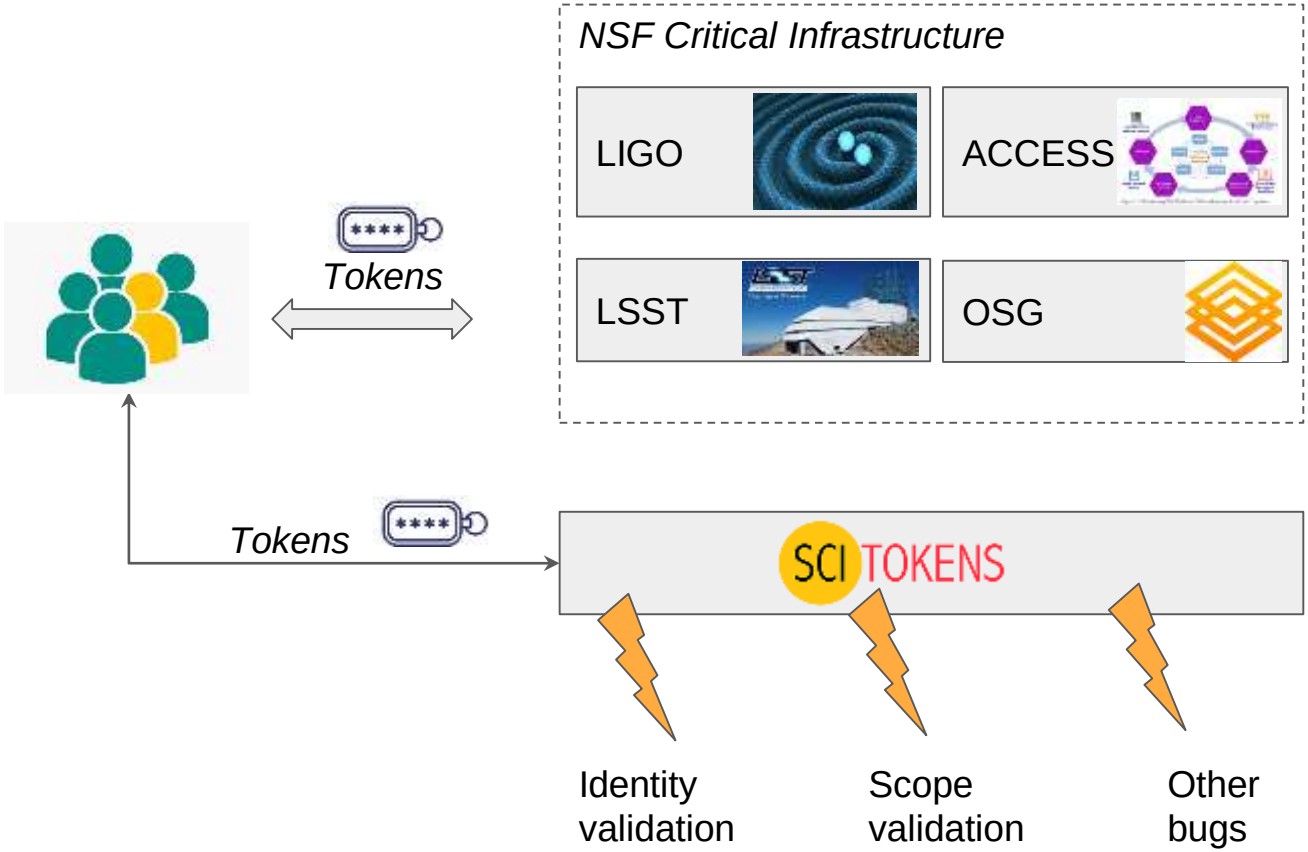
Formal Verification of SciTokens

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



Our initial approach is to understand specs, code, and manual auditing.


SciTokens Claims and Scopes

Language

Standard Claims

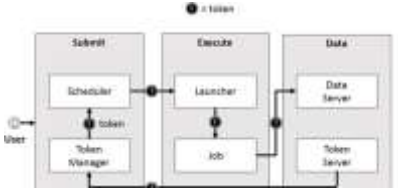
SciToken Claim Semantics

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

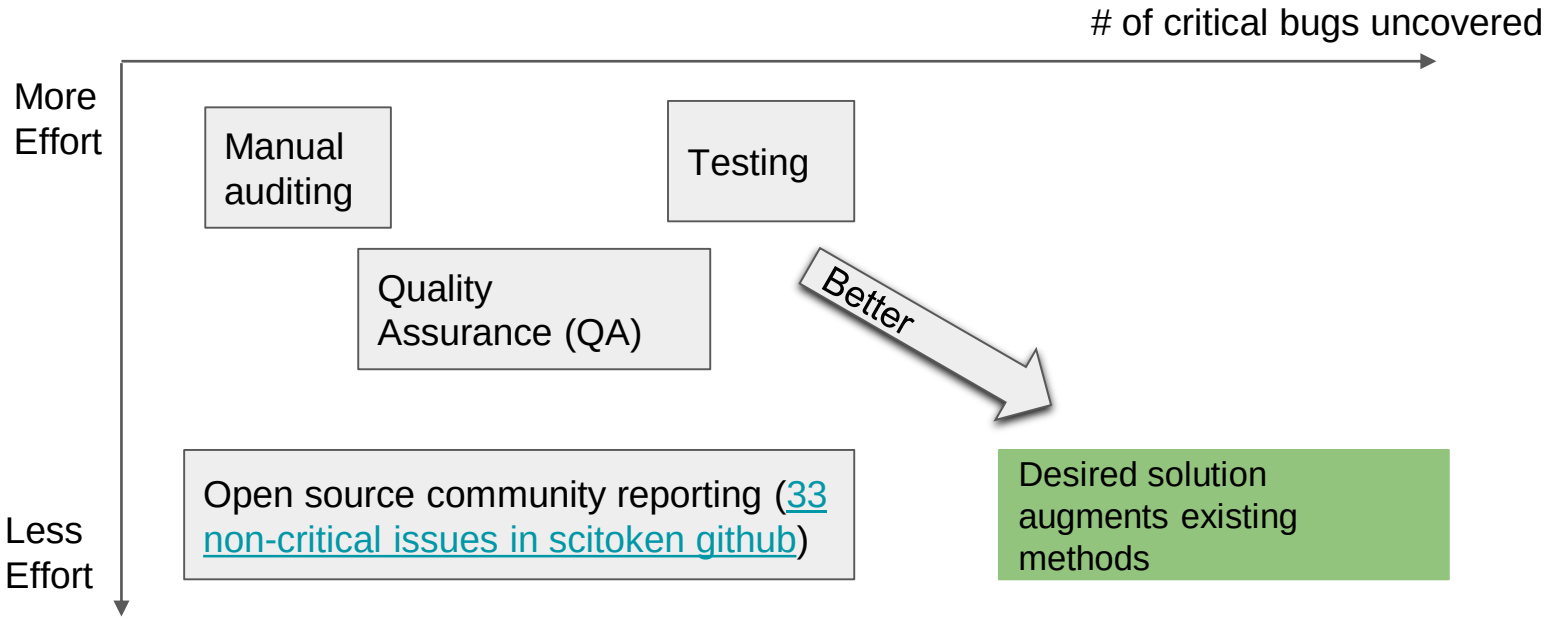
Critical, confirmed bug reporting

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

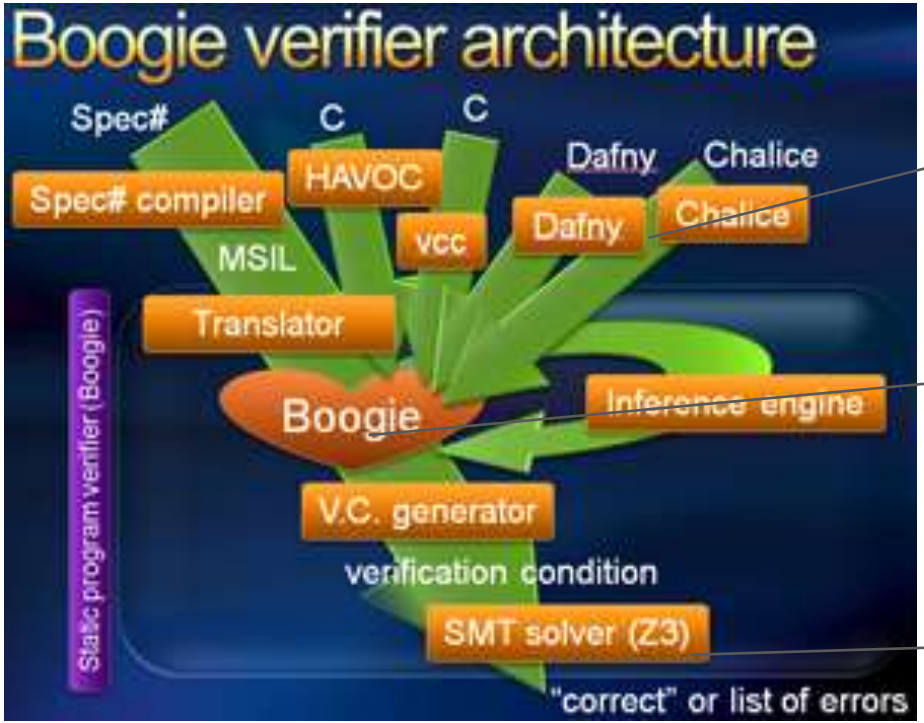
- A programming language with built-in specification constructs
- Supporting formal specification through
  - Preconditions,
  - Postconditions,
  - Loop Invariant
  - Termination specifications
- Formal reasoning through code using Hoare logic $\{P\}C\{Q\}$

```
method computeFactorial(n:int) returns (f:int)
  requires n >= 0
  ensures f == factorial(n)
{
var i := 0;
f := n;
while i < n-1
    invariant 0 <= i <= n-1;
    invariant f * fac
    {
    i:= i + 1;
    f := f * (n-i);
    }
}
```

Error: This loop invariant might not hold on entry

Hoare, C. A. R. (October 1969). "An axiomatic basis for computer programming". *Communications of the ACM*. **12** (10): 576–580. doi:10.1145/363235.363259.

Putting Dafny in perspective



High - level programming language
Dafny

Compiled to

Intermediate verification language
Boogie

Verified by

Satisfiability modulo theories (SMT) Solver
Z3

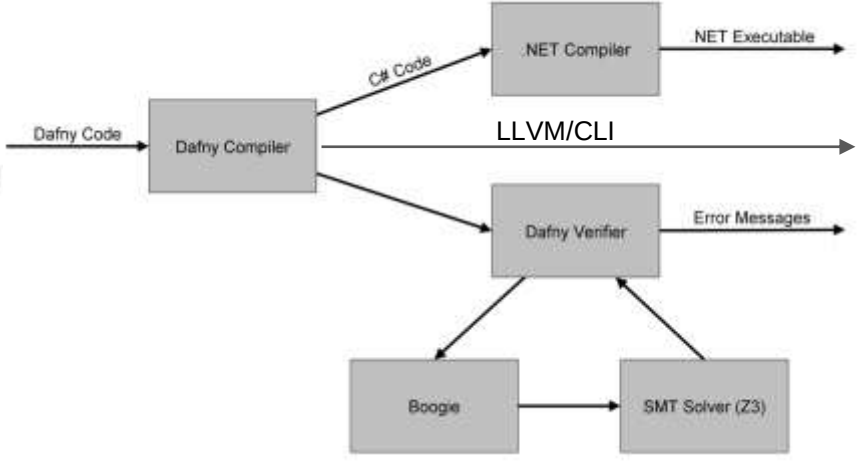
Microsoft Research

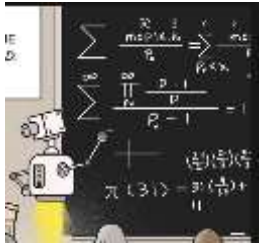
https://www.microsoft.com/en-us/research/project/boogie-an-intermediate-verification-language/

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**)

```
method computeFactorial(n:int) returns (f:int)
  requires n >= 0
  ensures f == factorial(n)
{
var i := 0;
f := n;
while i < n-1
    invariant 0 <= i <= n-1;
    invariant f * fac
    Error: This loop invariant might not hold on entry
    {
    i:= i + 1;
    f := f * (n-i);
    }
}
```





Computer Assisted Proof

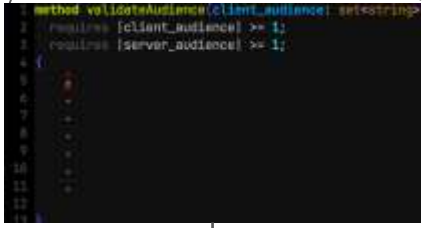
Example: verifying scitokens **audience**, a critical function, with Dafny.

implement *verify*

Scitokens audience specs → Audience verification logic → Audience verification impl (scitokens.py) ↔ Dafny implementation

- *aud* (Audience): A service the SciToken is authorized to access. For example, if the VO has write access to several storage services, this claim may be utilized to limit a token to a single endpoint URI. As in RFC7519, the `aud` claim is not necessarily a URI: for example, it might be the name of a target site. The service may accept several different possible audiences; the service endpoint at `https://storage.example.com` may accept an audience of either `Site_Example` or `https://storage.example.com` but ought to reject an audience of `https://www.google.com`. Additionally, the `aud` can be a special value of `ANY`, where it would allow any audience to match. The `aud` claim is OPTIONAL in version 1.0, mandatory in 2.0.

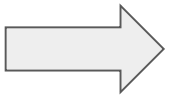
Client	Server	Result
ANY	ANY	Error
ANY	example.com	Success
example.com	ANY	Error
example.com	example.com	Success
notwork.com	example.com	Fail



Compile
Verify
Execute

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 =
            urltools.normalize_path(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.startswith(norm_path):
                return True
        return False
    else:
        # Split on spaces
        for scope in value.split(" "):
            authz, norm_path = self._check_scope(scope)
            self._token_scopes.add((authz, norm_path))
        return True
```

scitokens.py

```
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;
{
    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]);
```

Validate-scope.dfy (excerpt)

Summary & Future Work

SciTokens Codebase & Specs


SciToken Claims and Scopes Language

Standard Claims

Formal Verification

Found two bugs


Time-of-check-time-of-use


Path traversal attack

Verified three critical functions


Validate audience


Validate scope


Check scope

Correct-by-construction program synthesis

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.