

Hardening Zeek in Research & Education

Phuong Cao

National Center for Supercomputing Applications

Zeek Workshop, Berkeley, CA, September 2026

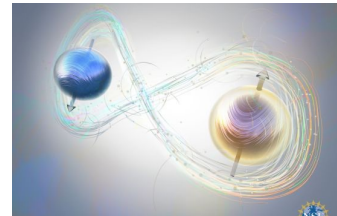
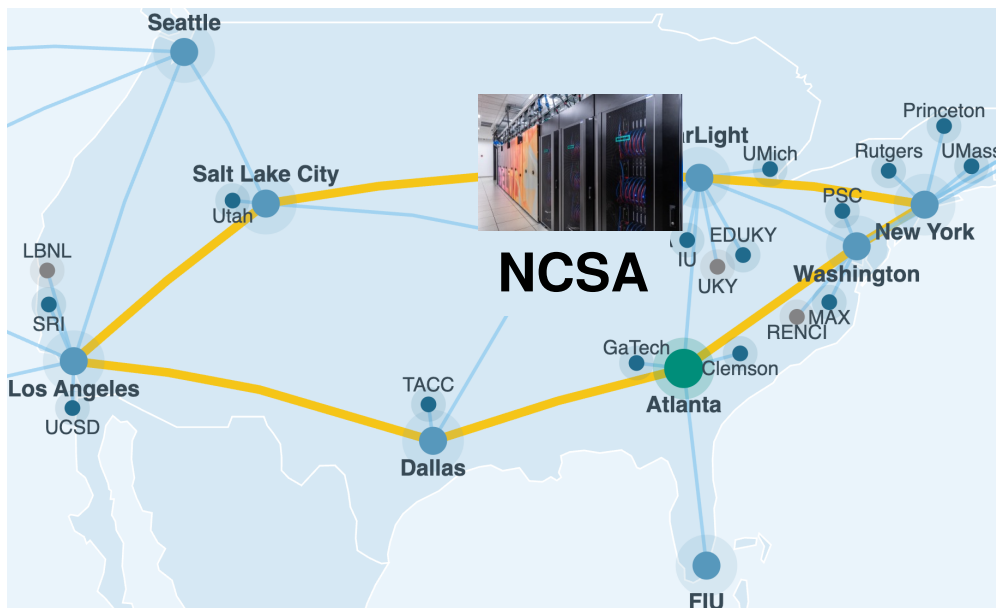
Protecting Scientific Research



Windows on the Universe
(Vera Rubin observatory)



AI-Enabled Dataset



Quantum Leap
(HPC-Quantum Integration)



Surgical Robots

Zeek - Foundation of Network Security Measurements Research

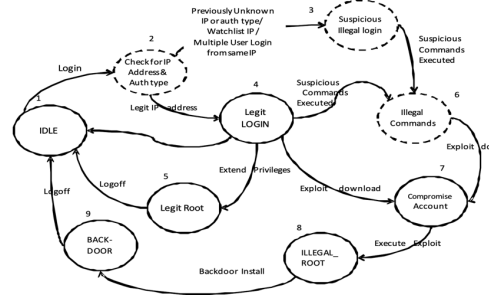


NCSA-ICSI dedicated Zeek development -> 2.0

NCSA acted as a production testbed, proving that Zeek could handle 100+ Gbps research networks.

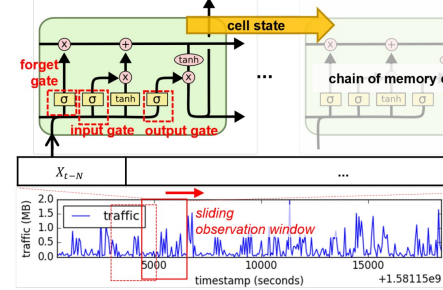
Funded by NSF Software Development for Cyberinfrastructure

Data-driven Security



A. Sharma et al., IEEE DSN

AI-Driven Ransomware



K. Chung et al., IEEE CSR

Hardening Zeek

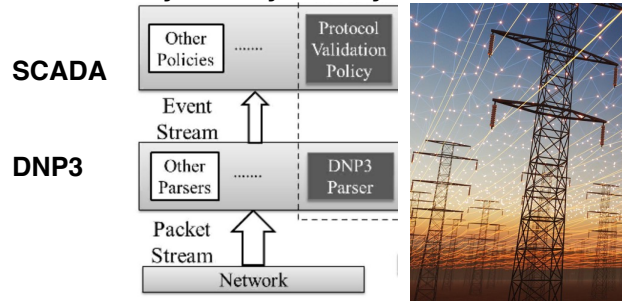
2010

2015

2020

2025

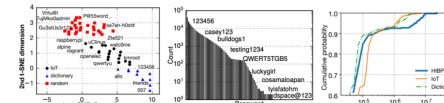
Expanding Zeek into Cyber-Physical Systems



H. Lin et al., IEEE Transactions on Smart Grid

SSH Backdoor & Honeypot

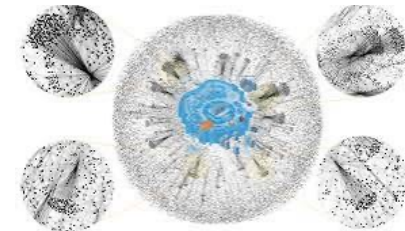
Client	Version	Count	Release Year
sshlib	0.1	76.7M	2010
sshlib	0.5.2	1.8M	2011
libssh2	1.7.0	26.8M	2016
paramiko	2.4.0	25.1K	2017
Go	N/A	19.4M	—
PIITTV	N/A	20.4M	—



corelight

P. Cao et al., USENIX NSDI

Security Data Lake



L. Yang et al., USENIX Security

Challenges of Parsing *Legacy* and *Emerging* Protocols

Zeek parses untrusted, attacker-controlled traffic at line rate

Two generations of protocol parsers:

- *binpac* (legacy, C++-generated, manual bounds reasoning)
- *Spicy* (newer, safer, but migration is in progress)

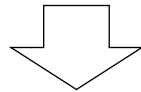
Legacy protocols, e.g. *DNS*, *LDAP*, are implemented in both binpac and Scipy, respectively.

Challenges of Parsing *Legacy* and *Emerging* Protocols (cont.)

Why focusing on protocols parsing again?

- *Increasing Post Quantum Cryptography (PQC) traffic.*
- *Initial measurements on PQC adoption at NCSA and peer sites (IEEE QCE'24)*
- *Ensuring the visibility of Zeek on PQC traffic's metadata.*

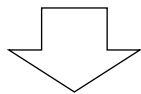
Emerging PQC protocols (e.g., ML-DSA) have new fields and Zeek relies on external libraries such as OpenSSL 3.5 to provide metadata



Motivated to understand and harden Zeek's protocols parsing logic

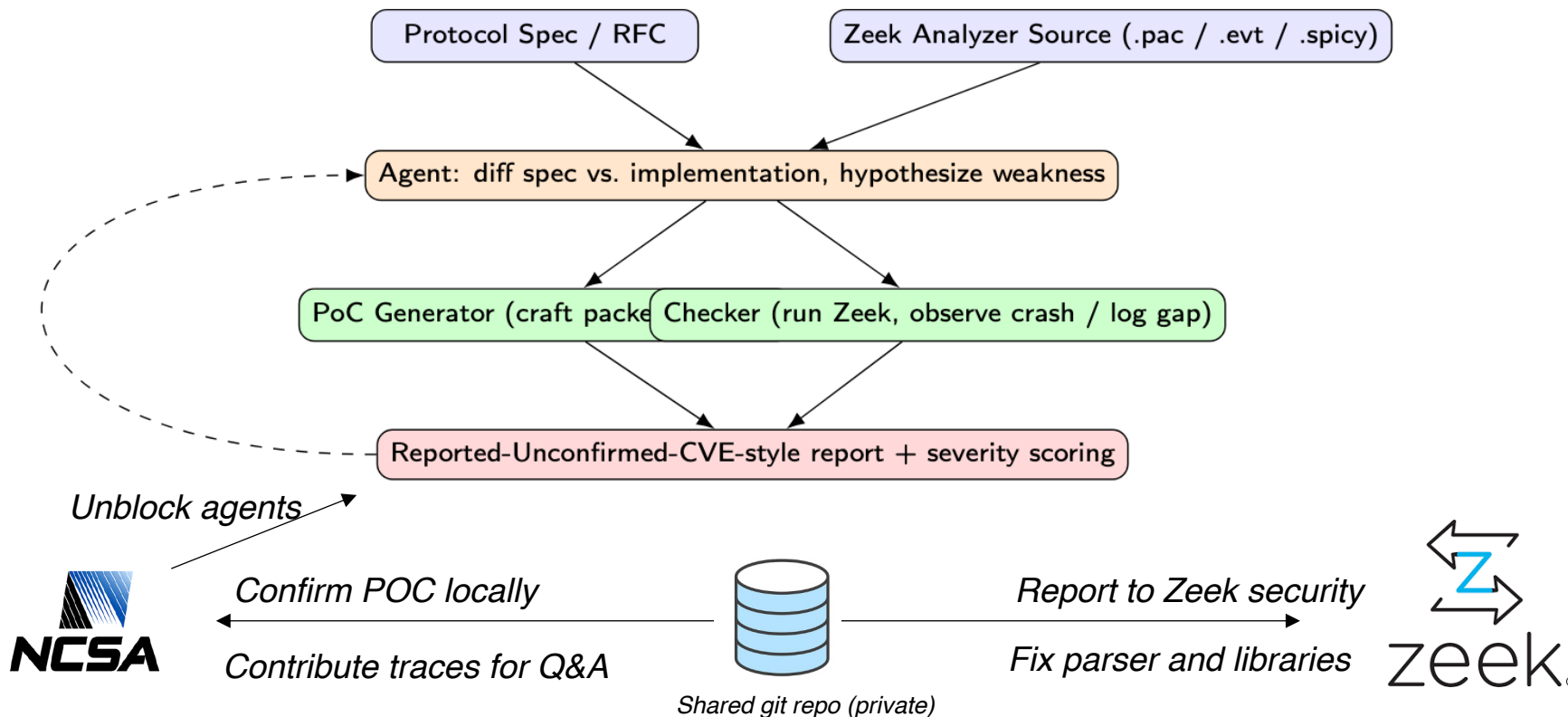
Manual vs. Agentic Harness

- Manual protocol-fuzzing / code audit doesn't scale across dozens of analyzers and thousands of lines of .pac grammar.
- **An agentic harness = an LLM agent + tools (repo search, PCAP crafting, static analysis, sandboxed execution) driven by a structured prompt loop.**
- Goal: turn “*read the spec, read the code, look for mismatches*” into a repeatable, checkable pipeline instead of one-off human review



- Simply prompting LLMs without domain insights and specific focus did not work: convincing CVE-like report but POC did not trigger crash nor evasion

Agentic Harness + Human-in-the-loop (NCSA + Zeek)



Triumvirate Agents in the Pipeline

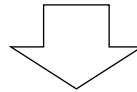
Static-analysis agent

- Reads .pac/.cc sources
- Looks for unchecked arithmetic, unsigned underflow, unbounded lengths, missing state validation
- Cross-references CMake build flags to find which code path is actually compiled (binpac vs. Spicy)

PoC / exploitation agent

- Crafts malformed packets (pcap) programmatically using Python
- Drives live sockets or offline pcap replay against a Zeek instance
- Confirms whether the hypothesized weakness actually reaches the vulnerable code path

Reporting agent: normalizes findings into CVSS-scored write-ups with root cause, affected lines, and remediation.



Human review and fix

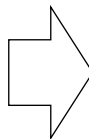
Collaborative Bug Fix Successes

ML-DSA / PQC Key Type and Key Length Not Logged in X.509 #5321

`X509::ParseCertificate()` in `X509.cc` determines the key type and key length of an X.509 certificate's public key by checking `EVP_PKEY_base_id()` against three known types: `EVP_PKEY_RSA`, `EVP_PKEY_DSA`, `EVP_PKEY_EC`. For any other key type including the ML-DSA, Falcon, and SLH-DSA algorithms natively supported by OpenSSL 3.5, the code falls through without setting either field.

OpenSSL 3.5.0 (released 2025) added native support for ML-DSA-44/65/87. Zeek containers built against OpenSSL 3.5+ (e.g., `zeek/zeek:8.0` on RHEL 10) will successfully parse ML-DSA X.509 certificates, yet omit the key metadata.

Reproducer: [pqc_key_blindspot.pcap.zip](#)



Oxxon on Apr 14

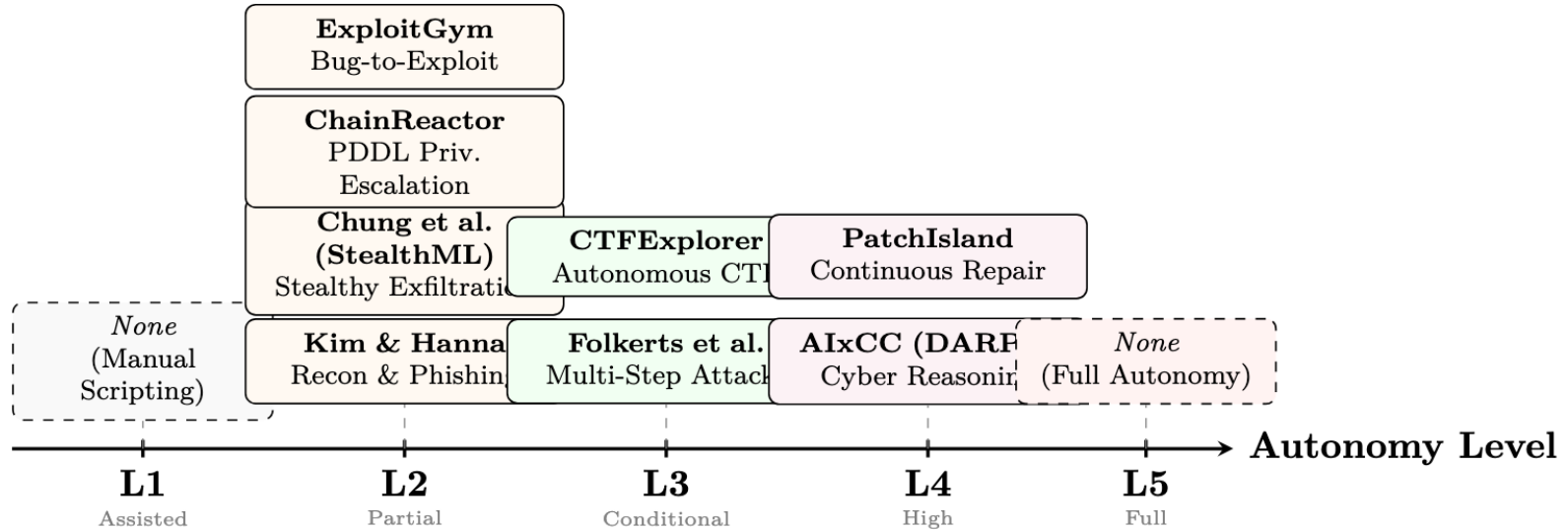
Thank you very much for pointing this out. This should be fixed by [#5344](#)

Oxxon added 2 commits that reference this issue on Apr 14

Add X509 test covering ML-DSA-44. ...

Add X509 test with a ML-DSA-44 certificate. ...

Where are we in the Agentic Autonomy Level?



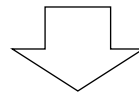
Planning Prompts

Spec-vs-Implementation Diff

Cryptographic-Agility Visibility

Resource-Exhaustion / DoS Hunting

“You are auditing Zeek’s protocol analyzer for <PROTOCOL>.
1. Summarize the RFC-defined field semantics and bounds
(e.g. uint24 length, sequence numbers, KEX algorithm IDs).
2. Read the corresponding .pac/.spicy source.
3. For each field, state whether the implementation enforces
the same bound the spec implies. Flag any:
- unsigned subtraction that could underflow
- shift amounts not bounded by the operand width
- lengths used to allocate/reassemble without a cap
4. Output candidate weaknesses as (file, line, description).”



LDAP Request Not Logged

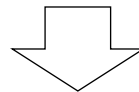
Planning Prompts

Spec-vs-Implementation Diff

Cryptographic-Agility Visibility

Zeek's <SSH/TLS> analyzer classifies key-exchange/cert algorithms into a fixed enum (e.g. KEX_ECC, KEX_DH). New PQC algorithms (ML-KEM, ML-DSA) are being added to real-world traffic. For the analyzer:

- 1. Find where new algorithm name strings are matched against the existing enum/state machine.*
- 2. Determine if a new algorithm family is silently folded into a semantically wrong bucket, or dropped entirely.*
- 3. Assess whether this causes incorrect parsing, or just a loss of visibility*



Resource-Exhaustion / DoS Hunting

**ML-DSA / PQC Key Type and Key Length
Not Logged in X.509**

Planning Prompts

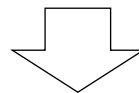
Spec-vs-Implementation Diff

Cryptographic-Agility Visibility

Resource-Exhaustion / DoS Hunting

Given a length-prefixed record format parsed by Zeek (e.g. uint24 msg_length in TLS Handshake), determine:

- 1. The maximum value representable by the length field.*
- 2. What Zeek does while the announced length has not yet arrived (buffering, allocation, state retained).*
- 3. Whether an attacker who sends the header but withholds the body can force disproportionate resource commitment per connection, and whether this is cheap to repeat.*



DNS Compression (Unbounded)

Deep Dive: Unbounded Recursion in DNS Compression

Spec-vs-Implementation Diff

Cryptographic-Agility Visibility

Resource-Exhaustion / DoS Hunting

File

src/analyzer/protocol/dns/DNS.cc

Call graph

```
ParseAnswerHeader (line 283)
├─ ExtractName (line 288)           ← entry point per RR
│   └─ while ExtractLabel (line 473)
│       └─ if compression pointer (line 515):
│           ExtractName (line 540) ← UNBOUNDED RECURSION
│               └─ while ExtractLabel ...
```

Deep Dive: Unbounded Recursion in DNS Compression

Spec-vs-Implementation Diff

Cryptographic-Agility Visibility

Resource-Exhaustion / DoS Hunting

```
def _dns_response(txid: int, depth: int = N_MAX) -> bytes:
    flags = 0x8180 # QR=1 AA=0 RD=1 RA=1
    header = struct.pack('!HHHHH', txid, flags, 1, depth, 0, 0)
    question = b'\x01q\x00' + struct.pack('!HH', 1, 1)
    payload = header + question

    first_offset = len(payload)
    payload += b'\x00' + struct.pack('!HHIH', 1, 1, 0, 4) + b'\x7f\x00\x00\x01'
    last_offset = first_offset

    for _ in range(1, depth):
        cur_offset = len(payload)
        payload += struct.pack('!H', 0xC000 | last_offset)
        payload += struct.pack('!HHIH', 1, 1, 0, 4) + b'\x7f\x00\x00\x01'
        last_offset = cur_offset

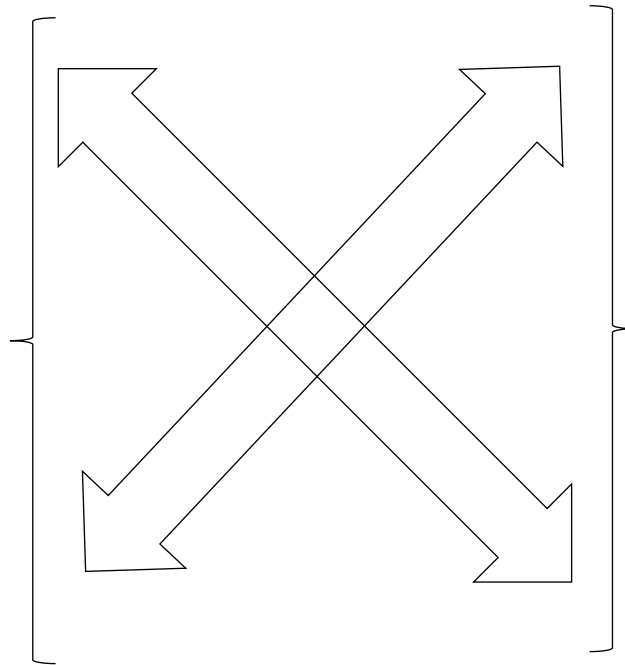
    return payload
```

Same Prompt, Multiple LLMs

Spec-vs-Implementation Diff

Cryptographic-Agility Visibility

Resource-Exhaustion / DoS Hunting



GLM 4.8

Kimi K2

Qwen-3

GPT-OSS

Gemma

Minimax M2

Summary & Future Work

Zeek's Important Role

- Continued Measurements-based Research Enablement
- Protocol Evolution Challenges: DNS, PQC
- Agentic Harness Architecture
- Zeek Community Collaboration

Future Work

- More automations:
 - Specs & RFC -> Code
 - Formally Verified Parser
 - Visibility at other layers across stack
- Addressing agents harness failures
- Collaboration between agents

Collaboration, Discussions, and Questions: [Phuong Cao • pcao3@illinois.edu](mailto:pcao3@illinois.edu)