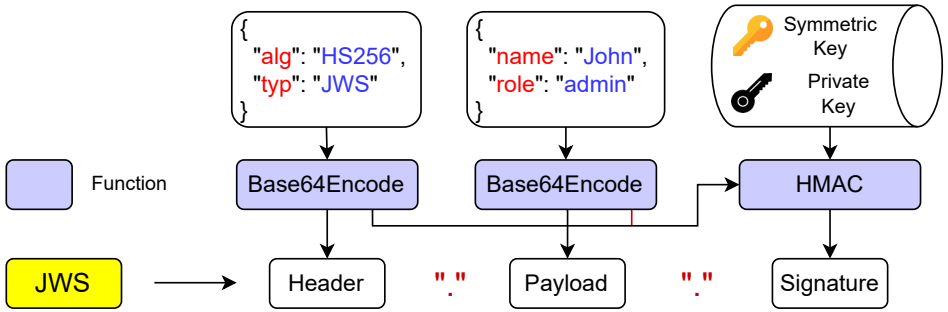
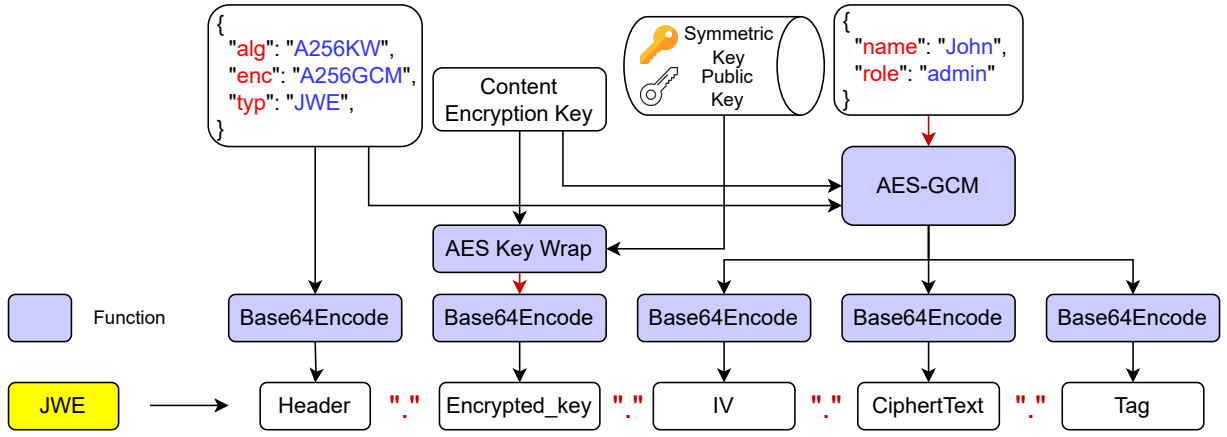
Token Time Bomb: Evaluating JWT Implementations for Vulnerability Discovery

Jingcheng Yang[1], Enze Wang[1], Jianjun Chen, Qi Wang, Yuheng Zhang, Haixin Duan, Wei Xie, Baosheng Wang

What is a JSON WEB TOKEN (JWT)?

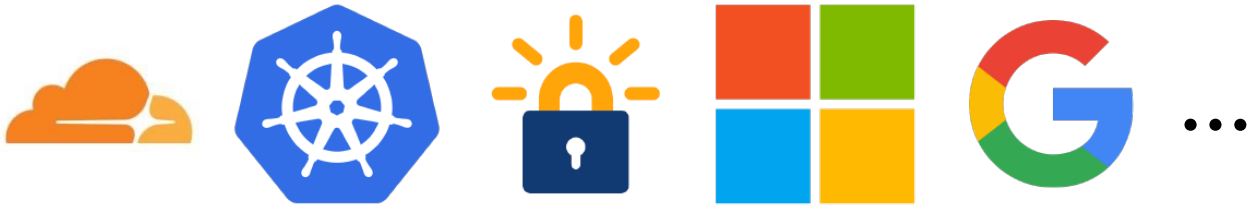
- JSON Web Token — compact, self-contained credential
  - JSON Web Signature:
    - Signed for integrity, payload visible
  - JSON Web Encryption:
    - Encrypted for confidentiality, payload hidden



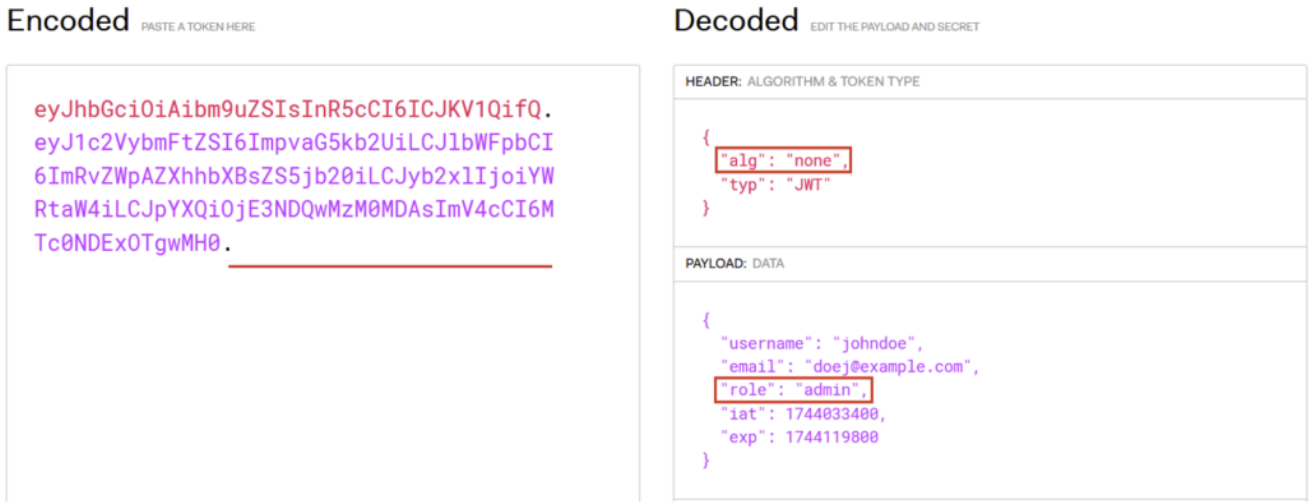


Vulnerabilities in JWT implementations

- **JWT is widely adopted by industry leaders**



- **JWT is highly flexible and feature-rich, which inadvertently creates opportunities for implementation vulnerabilities**



CVE-2023-29357 (Microsoft SharePoint)
forged JWT → admin privileges → remote code execution

Our Motivation & Goals

- **The majority of JWT vulnerabilities have been discovered manually in prior research**
- **This method may result in some vulnerabilities in JWT implementations being overlooked**

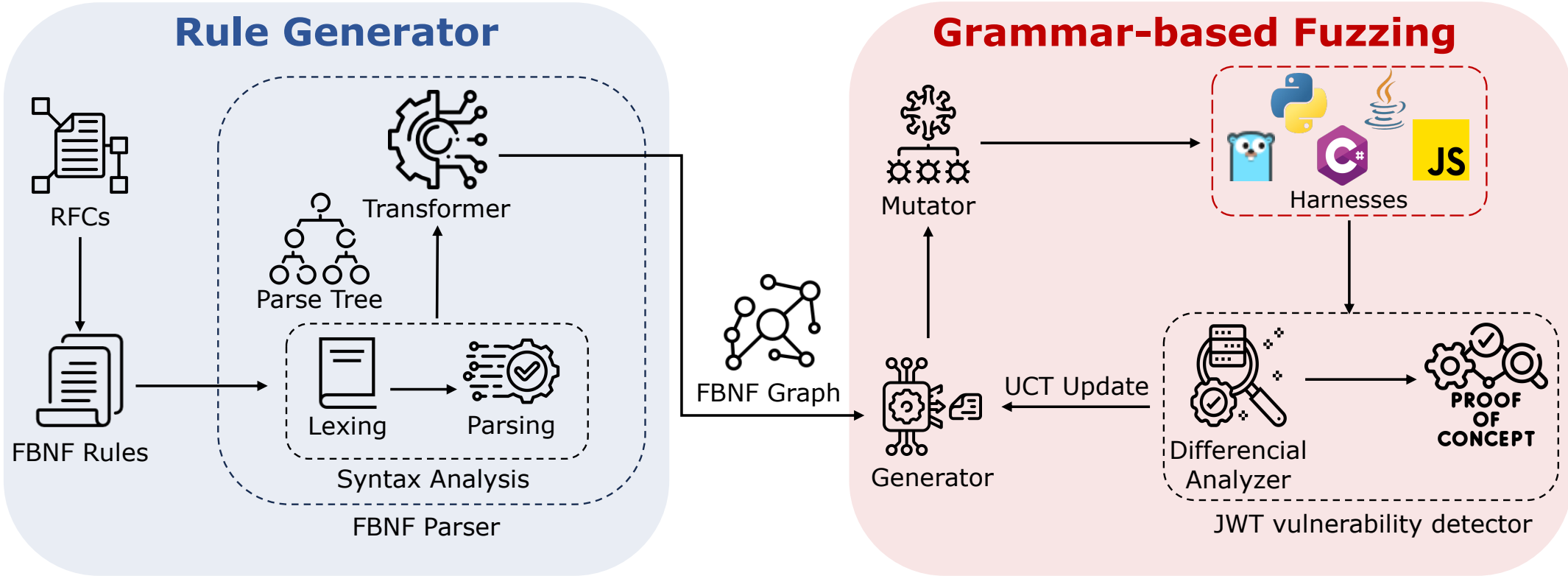
How to systematically and efficiently discovering all vulnerabilities in JWT implementations?

Research Questions

- RQ1: How to generate JWTs to trigger vulnerabilities?
  - Function-extended Backus-Naur Form (FBNF) for defining JWT grammar
  - JWT Generator for producing an initial JWT corpus from the FBNF graph
  - Mutator for expanding input diversity
  - UCT Update for feedback-driven optimization of generation paths
- RQ2: How to detect JWT vulnerabilities automatically?
  - Differential analysis between impls and within the same impl.
- RQ3: What is the prevalence in real-world implementations?
  - 43 libraries across 10 programming languages
  - 31 new vulnerabilities, 20 CVEs assigned

JWTeemo[1]: Overview

- **Rule Generator**
  - **Parsing FBNF rules and generate FBNF graph**
- **Grammar-based Fuzzing**
  - **Utilizing fuzzing with the Mutator and UCT Update to enhance the fuzzing efficiency**



[1] https://github.com/JWTeemo/JWTeemo

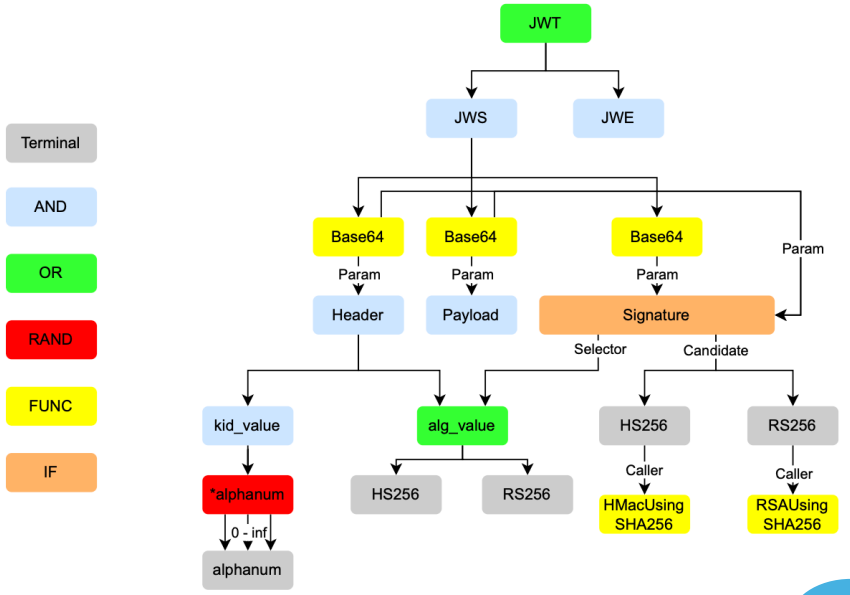
JWTeemo: Rule Generator

- **FBNF Grammar**
  - **FUNC: Call functions within grammar rules**
  - **IF: Select function based on preceding claim values**

Listing 1: An example of JWT's FBNF

```
1 JWT = JWS / JWE
2 JWS = CompactJWS / FlattenJWS
3 CompactJWS = b64header "." b64payload "." base64_encode(signature)
4 signature = if(alg_value,{
5                 "HS256": HMACUsingSHA256(key, b64header "."
                      b64payload),
6                 "RS256": RSAUsingSHA256(key, b64header "."
                      b64payload),
7             }
```

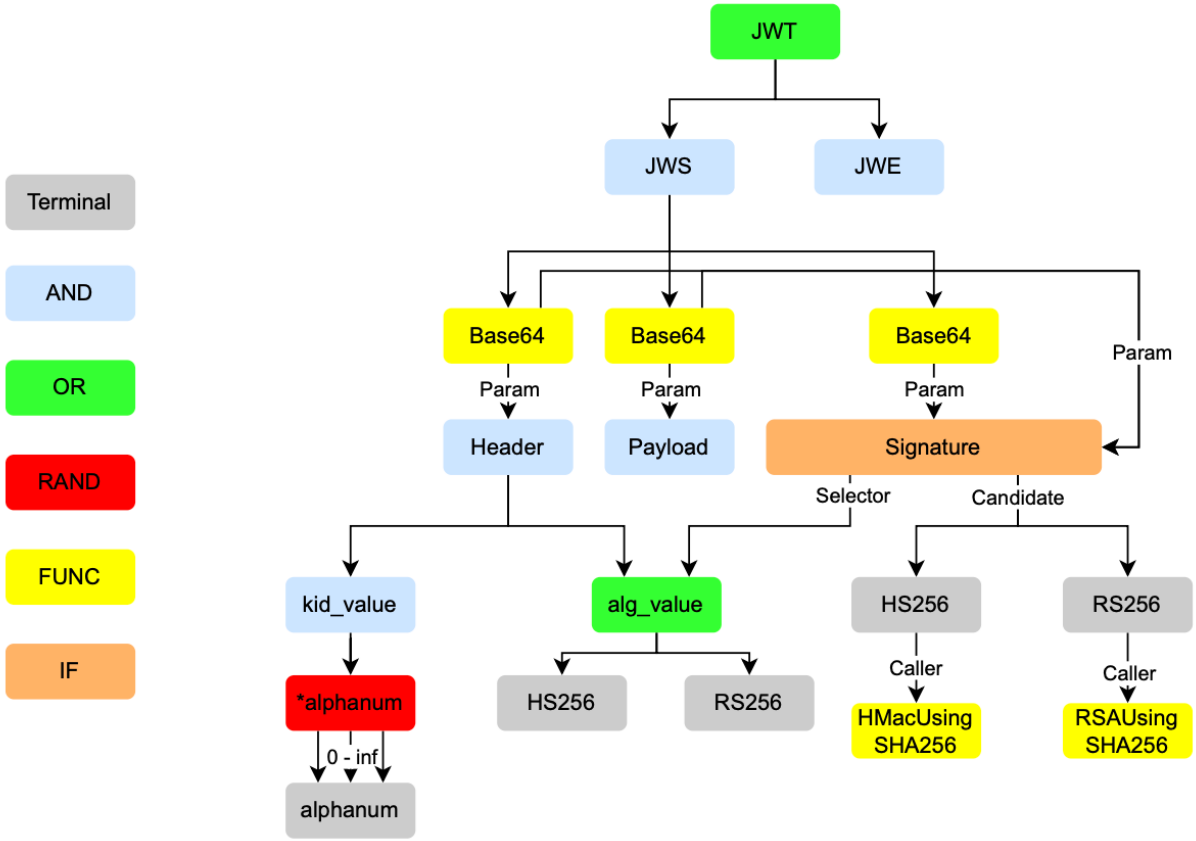
- **FBNF Parser**
  - **Lexing & parsing each FBNF rule into a Concrete Syntax Tree**
  - **Transformer merges all CSTs into a unified directed FBNF Graph**



JWTeemo: Grammar-based Fuzzing

- **JWT Generator**
  - **Depth-first traversal of the FBNF Graph**

NodeType	Indication
AND	Traverse all subtrees
OR	Traverse sub-node selected by UCT-Rand algorithm
RAND	Randomize the number of traversals
FUNC	Traverse all subtrees and call the function
IF	Traverse all subtrees and call the function selected by preceding claim values



$$\pi(v) := \underset{v' \in v.\text{children}}{\text{weighted rand}} \left(Q(v, v') + \sqrt{\frac{2 \ln N(v)}{N(v, v')}} \right)$$

JWTeemo: Grammar-based Fuzzing

- **Mutator: Two-level random mutation**
  - **Structure: delete or replace a non-terminal node in the FBNF graph**
  - **Content: insert or delete a character in a terminal value**

- **UCT Update: Feedback-driven optimization via UCT-Rand algorithm:**
  - **If >50% of implementations accept a JWT, mark the selection as successful**
  - **Increase nodes' weight for successful selections**
  - **Next traversal use the formula and prefers nodes with higher weight**

$$\pi(v) := \underset{v' \in v.\text{children}}{\text{weighted rand}} \left(Q(v, v') + \sqrt{\frac{2 \ln N(v)}{N(v, v')}} \right)$$

JWTeemo: Grammar-based Fuzzing

- **Differential Analyzer**
  - **Differences between implementations**
    - **Comparing Parsing Results Across Implementations**
  - **Differences within the same implementation**
    - **Detecting Abnormal CPU Usage Within an Implementation**
    - **Detecting Abnormal Memory Usage Within an Implementation**
    - **Use Chebyshev's inequality to identify statistically significant deviations**

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

$$R > \mu + k \times \sigma$$

Algorithm 1: Grammar-based Fuzzing Algorithm

Input: G: The initial FBNF graph constructed based on the RFC specification.
Input: H: An array of JWT Implementations, $H = \{H_1, ..., H_n\}$.
Input: k: Resource Usage metric (a constant factor).
Output: *differences*: An array of captured differences.

```
1  μ ← 0; σ ← 0;
2  repeat
3      jwt_seed ← JWTGENERATOR(G);
4      jwt ← MUTATOR(jwt_seed);
5      outputs, RU ← RUNFUZZ(H, jwt);
6      foreach pair (i, j) such that 1 ≤ i < j ≤ n do
7          if outputs[i] ≠ outputs[j] then
               /* Differences between
                  implementations                */
8              differences.APPEND((H_i, H_j, jwt));
9      for i ← 1 to n do
10         if RU[i] > μ[i] + k · σ[i] then
               /* Differences within the same
                  implementation                 */
11             differences.APPEND((H_i, jwt));
12         μ[i], σ[i] ←
             RESOURCEMONITORUPDATE(RU[i]);
13     G ← UCTUPDATE(G, outputs);
14 until ENDCONDITIONS();
```

Evaluation Setup

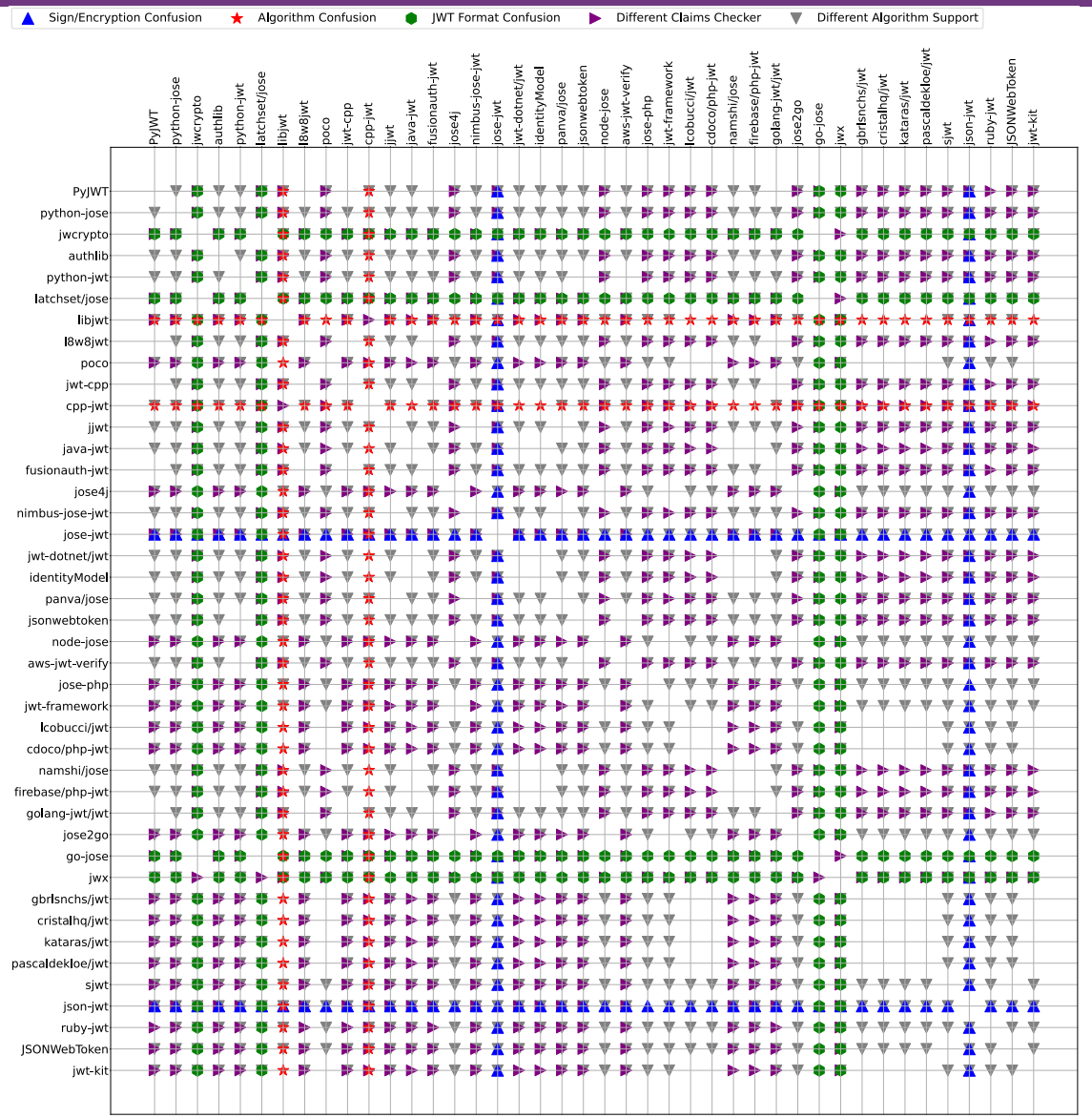
- **Dataset**
  - TIOBE top 16 languages, GitHub stars ≥ 100, from jwt.io
  - 43 JWT libraries across 10 programming languages
- **Setup**
  - Ubuntu server: 4.1GHz 32-core CPU, 512GB RAM
  - Harness for each library to receive and verify JWTs
  - Generated 100,000 JWT test cases

Language	Library	Version	Stars
Python	pyJWT[40]	2.8.0	4.8k
Python	python-jose[41]	3.3.0	1.5k
Python	jwcrypto[38]	1.5.0	430
Python	authlib[42]	1.2.1	4.5k
Python	python-jwt[43]	1.3.1	140
C	jose[39]	11	170
C	libjwt[44]	2.1.0	368
C	l8w8jwt[45]	2.2.1	111
C++	poco[46]	1.12.5p2	7.6k
C++	jwt-cpp[47]	0.7.0	681
C++	cpp-jwt[48]	1.4	387
Java	jjwt[23]	0.12.3	10.1k
Java	java-jwt[49]	4.4.0	5.5k
Java	fusionauth-jwt[50]	5.3.0	153
Java	jose4j[51]	0.9.3	N/A
Java	nimbus-jose-jwt[37]	9.37.1	N/A
C#	jose-jwt[52]	4.1.0	933
C#	jwt[53]	10.1.1	2.1k
C#	System.IdentityModel.Tokens.Jwt[54]	7.2.0	1k
JavaScript	jose[55]	5.1.3	5.3k
JavaScript	jsonwebtoken[56]	9.0.2	17.1k
JavaScript	node-jose[57]	2.2.0	699
JavaScript	aws-jwt-verify[58]	4.0.0	518
PHP	jose-php[59]	2.2.1	138
PHP	jwt-framework[60]	3.2.8	881
PHP	jwt[61]	5.2.0	7.1k
PHP	adhocore/php-jwt[62]	1.1.2	271
PHP	cdoco/php-jwt[63]	1.0.0	232
PHP	jose[64]	7.2.3	1.8k
PHP	firebase/php-jwt[65]	6.10.0	1.4k
Go	golang-jwt/jwt[66]	5.2.0	16.4k
Go	jose2go[67]	1.5.0	186
Go	go-jose[25]	3.0.1	2.3k
Go	jwx[24]	2.0.17	1.9k
Go	gbrlsnchs/jwt[68]	3.0.1	442
Go	cristalhq/jwt[69]	5.4.0	626
Go	kataras/jwt[70]	0.1.12	188
Go	pascaldekloe/jwt[71]	1.0.12	339
Go	sjwt[72]	0.5.1	114
Ruby	json-jwt[73]	1.16.3	297
Ruby	ruby-jwt[74]	2.7.1	3.5k
Swift	JSONWebToken[75]	2.2.0	763
Swift	jwt-kit[76]	4.13.4	180

TABLE IV: The targets evaluated by JWTeemo. N/A indicates the library is not an open source on GitHub.

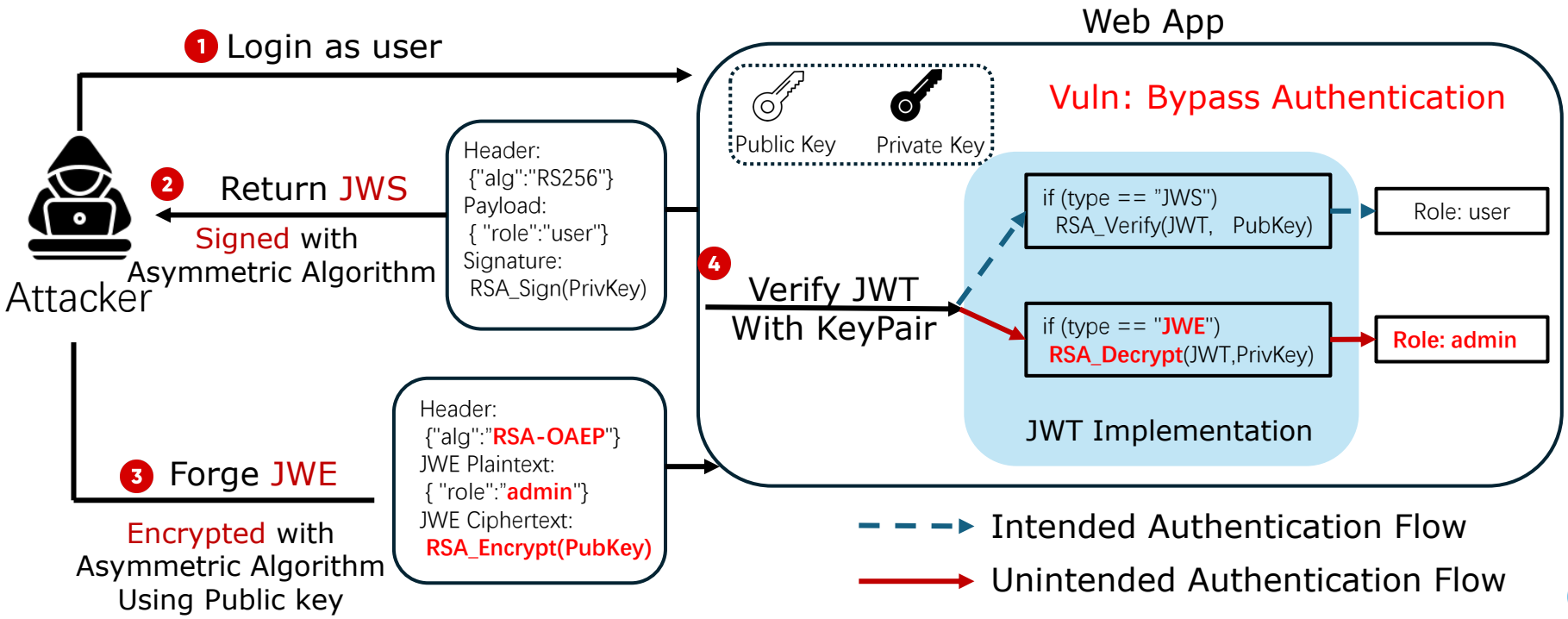
Experimental Results: Differences between implementations

- **1,804** differences
- **5** types difference
  - Sign/Encryption Confusion **VULN**
  - Algorithm Confusion **VULN**
  - JWT Format Confusion **VULN**
  - Different Claims Checker **SAFE**
  - Different Algorithm Support **SAFE**
- False Positive: 635 / 1,804 = 35.1%



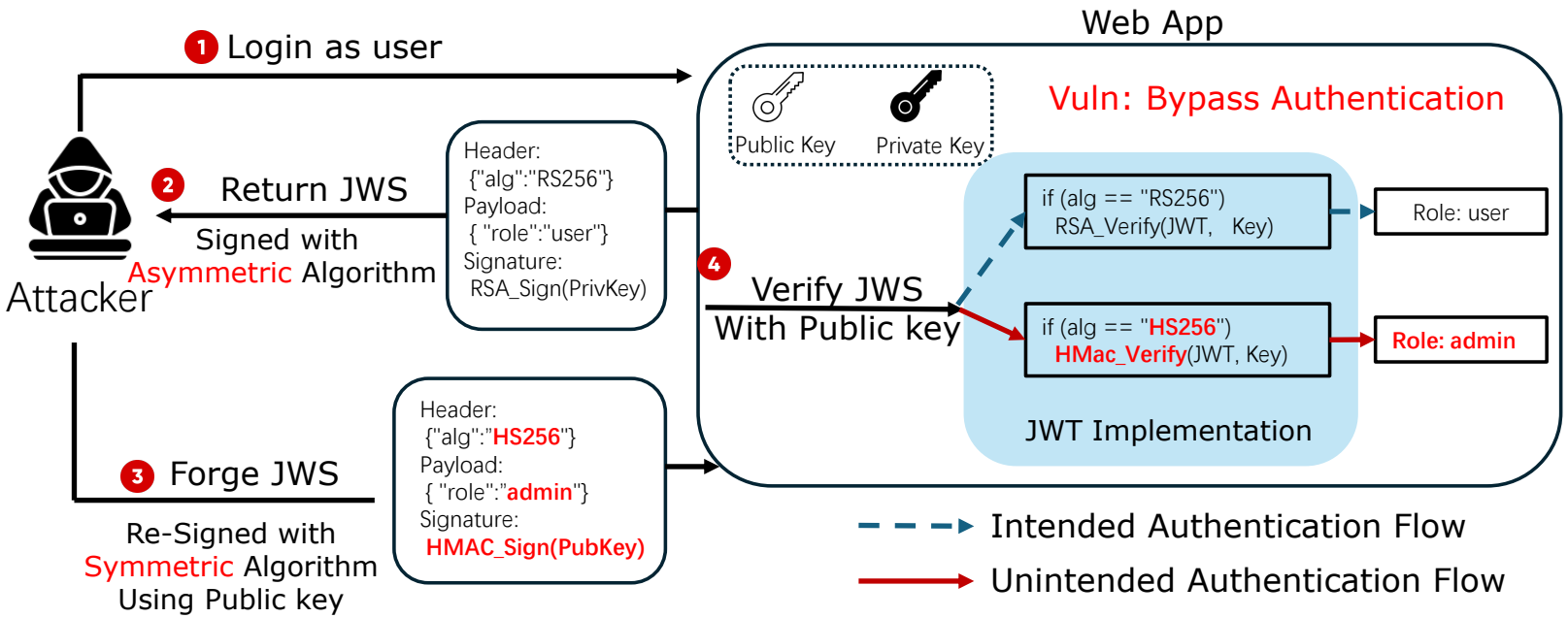
Vulnerability 1: Sign/Encryption Confusion

- **Sign/Encryption Confusion**
  - Attacker obtains the public key used for JWS verification
  - Forges a JWE encrypted with this public key, payload set to admin
  - Vulnerable implementation determines JWT type by counting dots
  - Uses private key to decrypt → attacker's forged payload accepted



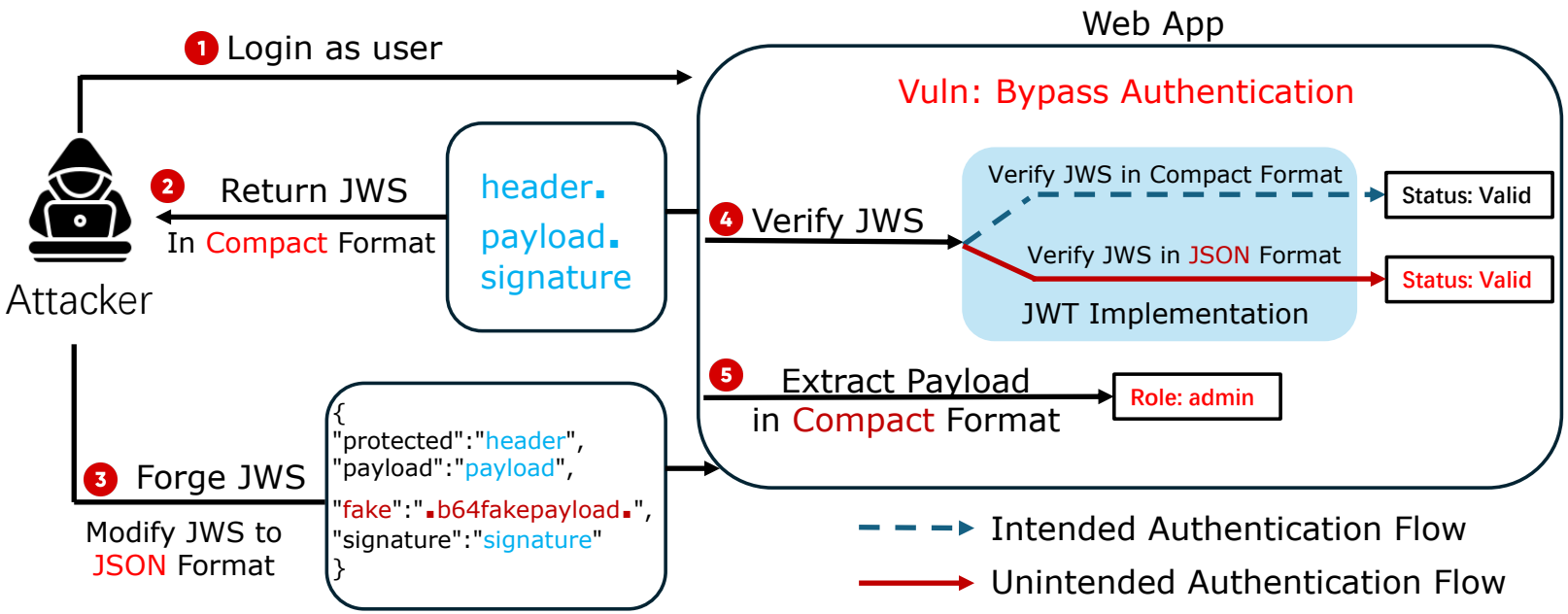
Vulnerability 2: Algorithm Confusion

- **Algorithm Confusion**
  - Attacker obtains the public key used for Asymmetric algorithm verification
  - Attacker changes header alg from RS256 to HS256
  - Re-signs the token using the public key as HMAC secret
  - Vulnerable implementation trusts alg claim, selects HMAC verification
  - Public key matches as HMAC secret → signature accepted



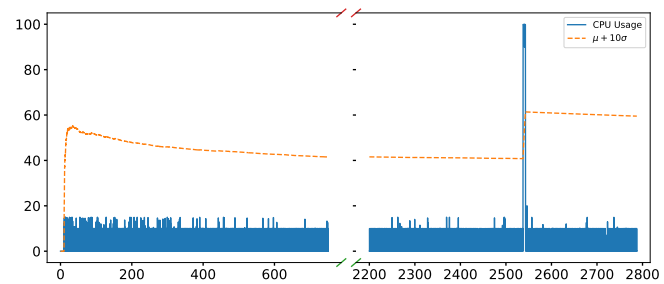
Vulnerability 3: JWT Format Confusion

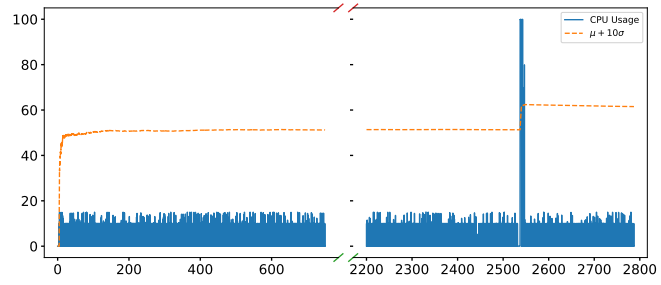
- **JWT Format Confusion**
  - **JWT RFC allows only Compact format; JWS RFC also defines JSON format**
  - **Some implementations accept JSON-format JWS when processing JWT**
  - **Attacker converts JWT to JSON format, inserts forged payload in a custom field**
  - **Implementations verifies signature on JSON format JWT → passes**
  - **App assumes Compact format, extracts claims by dot-splitting → reads forged payload**

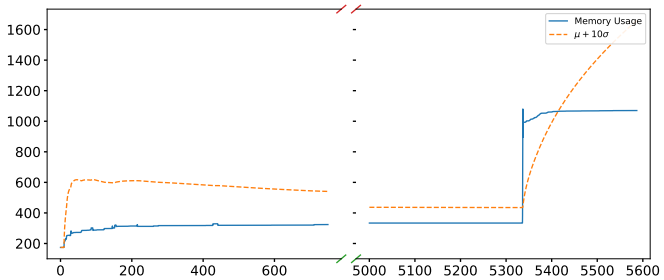


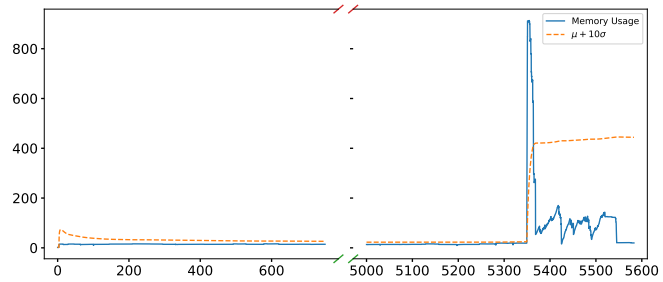
Experimental Results: Differences within the same implementation

- **Detected 2 types of DoS vulnerabilities:**
  - **CPU Exhaustion: Billion Hashes Attack (10 impls)**
  - **Memory Exhaustion: Compression DoS (13 impls)**
- **For example, during fuzzing, JJWT and JWX showed significant resource spikes:**







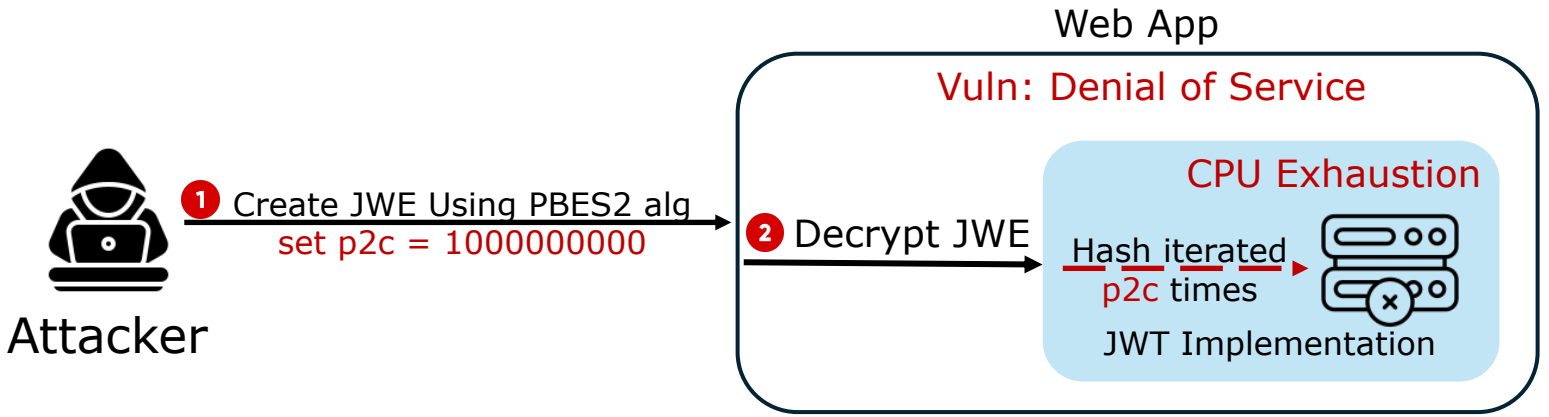


JJWT Library

JWX Library

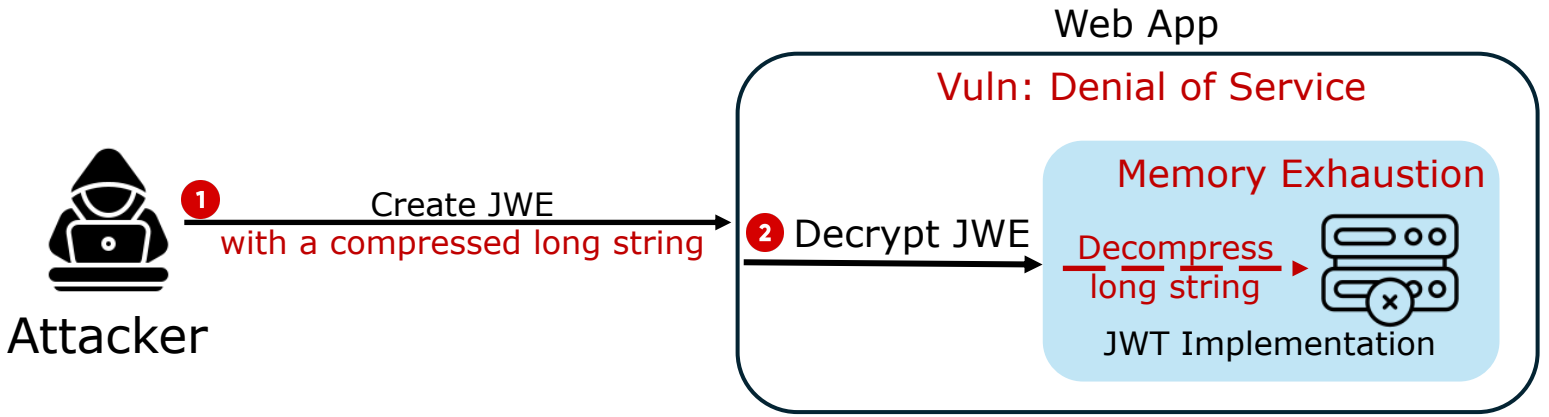
Vulnerability 4: Billion Hashes Attack (CPU Exhaustion)

- **Billion Hashes Attack**
  - PBES2 algorithm uses p2c claim to specify hash iteration count
  - Attacker sets p2c to an extremely large value (e.g., 10^9)
  - Server performs excessive hash computations during key derivation



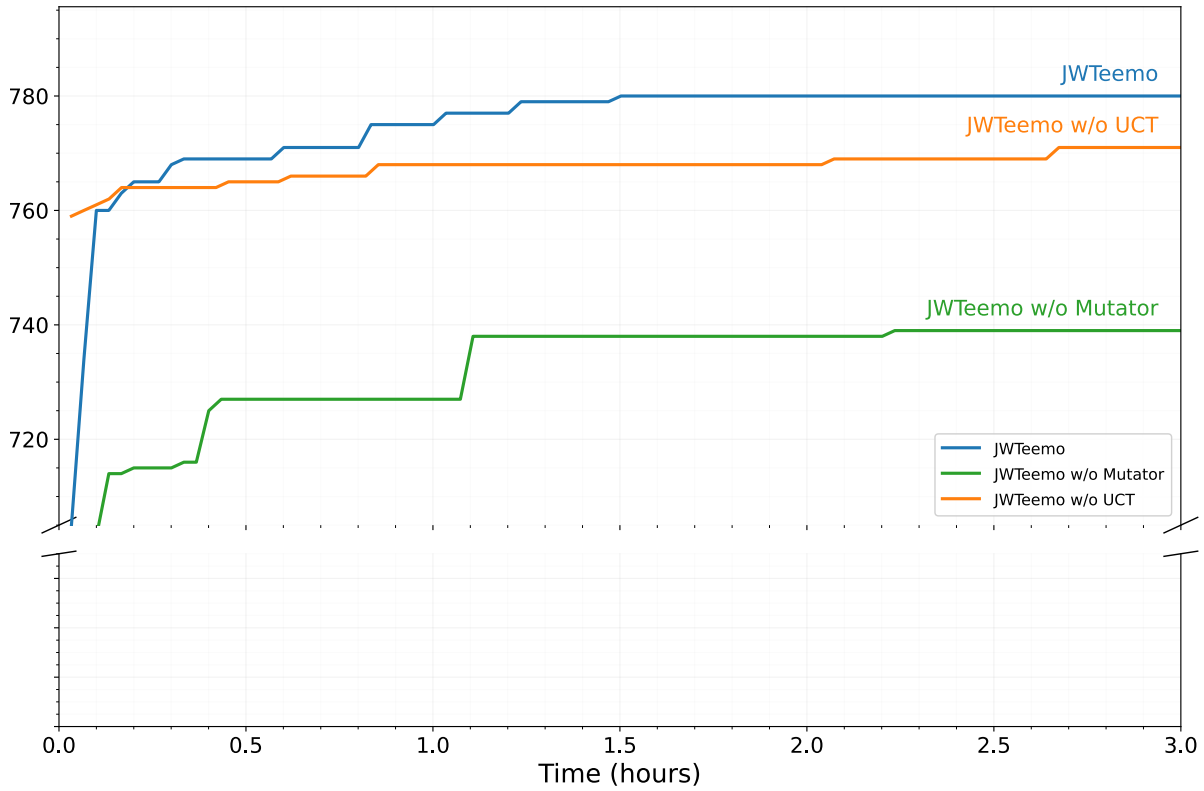
Vulnerability 5: Compression DoS (Memory Exhaustion)

- **Compression DoS**
  - **JWE header zip: DEF indicates payload is compressed**
  - **Attacker crafts a JWE with a highly compressed long string as payload**
  - **Server decompresses payload after decryption → massive memory allocation**



Ablation Study

- **Ablation Study:**
  - **w/o UCT Update: slower coverage growth, longer time to discover vulnerabilities**
  - **w/o Mutator: missed 2 vulnerability types**
  - **Both components are essential for effective fuzzing**



Configuration	Sign/Encrypt Confusion	Algorithm Confusion	JWT Format Confusion	Billion Hashes Attack	Compression Attack
JWTeemo	43.2	8.7	27.2	2535.6	5349.4
JWTeemo w/o UCT Update	60.2	56.3	66.9	3204.4	20106.2
JWTeemo w/o Mutator	46.8	9.7	31.3	N/A	N/A

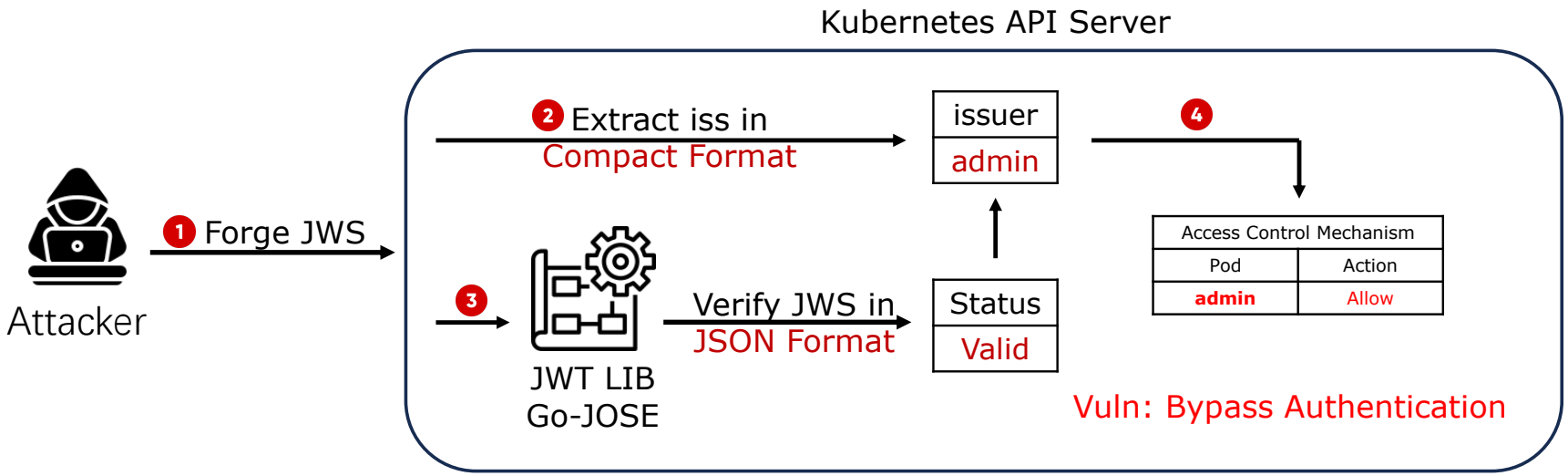
Comparison Study

- **Comparison with Existing Tools:**
  - **JWT Tool & JWT Editor: lower coverage, only detect known vulnerabilities**
  - **JWTeemo: higher coverage, discovers all 5 vulnerability types automatically**
  - **Existing tools rely on predefined payloads; JWTeemo discovers unknown vulnerabilities automatically**

Tools	Time to Discover (s)					Covered Edges
	Sign/Encrypt Confusion	Algorithm Confusion	JWT Format Confusion	Billion Hashes Attack	Compression Attack	
JWTeemo	43.2	8.7	27.2	2535.6	5349.4	780
JWT_Tool	N/A	1.1	N/A	N/A	N/A	473
JWT Editor	N/A	60*	N/A	N/A	N/A	509

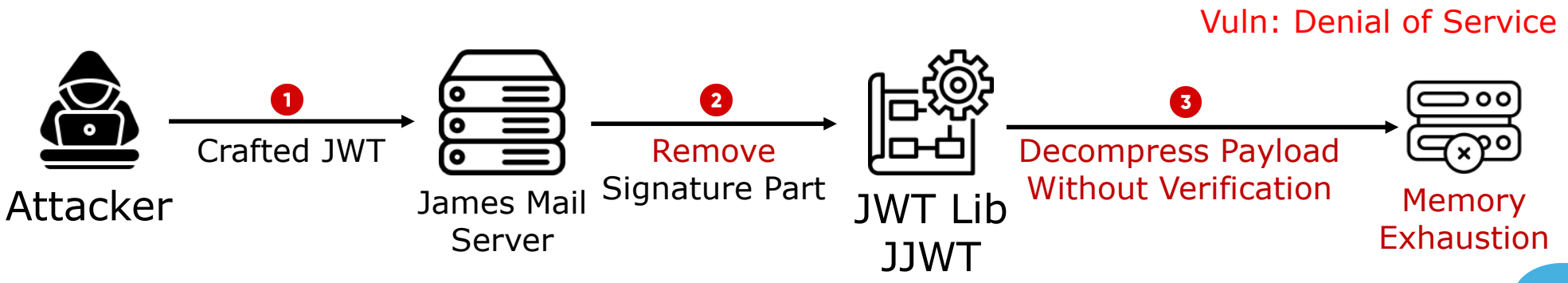
Case Study 1: Authentication Bypass in Kubernetes

- K8s uses JWT-based ServiceAccount tokens for pod authentication
- API server extracts iss by dot-splitting (assumes Compact format), but delegates verification to go-jose which also accepts JSON format
- Attacker crafts JSON JWT with spoofed fakeiss field; go-jose verifies real signature, API server reads forged issuer → auth bypass
- Bug bounty awarded; same vulnerability found in OpenShift Telemeter (CVE-2024-5037)



Case Study 2: Compression DoS in Apache James

- Apache James mail server uses JJWT for SMTP OAuth authentication;
- JJWT accepts zip in JWS header (violates RFC) and decompresses payload before signature verification
- Attacker sends JWT with compressed bomb, and James decompresses before any verification → memory exhaustion, no credentials needed
- Vulnerability fixed by Apache



Root Cause

- **Misunderstanding the Proper Use of JWT Algorithms**
  - **Improper use of public/private keys across algorithms**
  - **Lack of enforcement on algorithm–key compatibility**
- **Non-compliant Implementation of JWT Specifications**
  - **Supporting JWT formats not allowed by the RFC**
  - **Accepting invalid claim usage**
- **Insufficient Security Warnings for Risky JWT Features**
  - **Dangerous claims without explicit limits**
  - **Outdated security guidance**

Mitigation

- For JWT Specification:
  - Limit p2c claim size to prevent Billion Hashes Attack
  - Advise against parsing JSON-type JWS in JWT
  - Suggest upper limit on JWE payload decompression size
  - Recommend enforcing use claim in JWK to distinguish signing/encryption
- For JWT Implementation Developers:
  - Strictly bind keys to allowed algorithms and enforce use/alg constraints in JWKs
  - Avoid supporting excessive or unnecessary features (e.g., JSON format JWS)
- IETF Impact:
  - Reported mitigations to IETF RFC 8725 authors
  - Proposals acknowledged and incorporated into draft-ietf-oauth-rfc8725bis-03[1]

[1] https://datatracker.ietf.org/doc/draft-ietf-oauth-rfc8725bis/

Conclusion

- **New Framework: JWTeemo**
  - **First systematic framework for automated JWT vulnerability discovery**
- **New Findings:**
  - **Evaluated 43 libraries across 10 languages; discovered 31 new vulnerabilities, 20 CVEs assigned**

- **New Mitigations:**
  - **Proposed mitigations adopted by IETF into draft-ietf-oauth-rfc8725bis-03**

Thank you for listening!
Q & A

Jingcheng Yang

Network and Information Security Lab
Tsinghua University
yangjc25@mails.tsinghua.edu.cn