https://doi.org/10.51885/3107-2755_JISC_2026_2_1
SRSTI 81.93.29

Developing a REST and gRPC API Security Testing Framework Based on OWASP Standards

Разработка фреймворка тестирования безопасности REST и gRPC API на основе стандартов OWASP

OWASP стандарттары негізінде REST және gRPC API қауіпсіздігін тестілеу фреймворкін әзірлеу

A. Yelzhanov[1*]
[1]Astana IT University, Astana, Kazakhstan
* Corresponding author: Assanali Yelzhanov, e-mail: asanelzhanov2@gmail.com

Keywords:
API security, REST, gRPC, OWASP API Security Top 10, security oracle, stateful fuzzing, authorization testing CI/CD integration.

ABSTRACT
REST and gRPC application programming interfaces expose different interface, transport, state and message models, which complicates unified security assessment. This review-based design study examines standards and automated testing techniques for specification-based test generation, stateful fuzzing, authorization testing, resource-consumption testing and continuous assessment, and on that basis proposes an OWASP-aligned security testing framework covering both protocols. Eight families of existing approaches were classified along nine dimensions using a targeted corpus of primary sources. The review shows that current tools provide effective individual components, including dependency inference, model feedback, search-based generation and vulnerability-specific oracles, but that these components remain fragmented across protocols and testing objectives. The proposed framework combines interface discovery, OpenAPI and protocol-buffer parsing, identity management, risk mapping, stateful test generation, fuzzing, specialized security oracles, evidence-based reporting and continuous delivery integration. Its principal methodological contribution is that the security oracle of every covered OWASP risk family receives an explicit logical predicate over identities, objects, responses and persistent state, while streaming obligations are expressed in linear temporal logic, so that a finding is defined by a mechanically checkable condition rather than by an informal description. Test state is modelled as the product of an operation dependency graph, a per-resource lifecycle automaton and the normative HTTP/2 stream automaton, closed under a provenance environment recording the identity and timestamp of every harvested identifier. An evaluation protocol based on paired vulnerable and patched services, seeded ground truth, baseline tools and nine metrics is also defined. The results are conceptual and specification results: the practical effectiveness of the framework is not evaluated here and is deferred to a separate experimental study.

Түйінді сөздер:
API қауіпсіздігі, REST, gRPC, OWASP API

ТҮЙІНДЕМЕ
REST және gRPC қолданбалы бағдарламалау интерфейстері интерфейс, тасымал, күй және хабарлама модельдері бойынша бір-

Security Top 10, қауіпсіздік оракулы, күй негізіндегі фаззинг, авторизацияны тестілеу, CI/CD интеграциясы.

бірінен ерекшеленеді, бұл олардың қауіпсіздігін біртұтас бағалауды қиындатады. Шолу-жобалау сипатындағы бұл мақалада спецификацияға негізделген тест генерациясы, күйді ескеретін фаззинг, авторизацияны тексеру, ресурстарды тұтынуды бағалау және үздіксіз тестілеу әдістері қарастырылып, осы талдау негізінде екі хаттаманы да қамтитын, OWASP стандарттарына сәйкестендірілген қауіпсіздікті тестілеу фреймворкі ұсынылады. Қолданыстағы тәсілдердің сегіз тобы тоғыз өлшем бойынша жіктелді. Талдау нәтижесі көрсеткендей, бүгінгі құралдар тиімді жекелеген компоненттер ұсынады, бірақ олар хаттамалар мен тестілеу мақсаттары бойынша бытыраңқы күйде қалып отыр. Ұсынылған фреймворкте әрбір оракул отбасына тұлғалар, нысандар, жауаптар мен тұрақты күй үстінде анықталған айқын логикалық предикат беріледі, ал ағындық міндеттемелер сызықтық уақыттық логикада өрнектеледі. Тест күйі тәуелділік графының, ресурс өмірлік циклі автоматының және HTTP/2 ағын автоматының көбейтіндісі ретінде модельденеді. Сонымен қатар осал және түзетілген қызметтердің жұптарына негізделген бағалау хаттамасы ұсынылады. Нәтижелер тұжырымдамалық және спецификациялық сипатта: фреймворктің практикалық тиімділігі бұл мақалада бағаланбайды және жеке эксперименттік зерттеуде қарастырылады.

Ключевые слова:
безопасность API, REST, gRPC, OWASP API Security Top 10, оракул безопасности, фаззинг с учётом состояния, тестирование авторизации, интеграция CI/CD.

АННОТАЦИЯ

Прикладные программные интерфейсы REST и gRPC различаются моделями интерфейса, транспорта, состояния и сообщений, что затрудняет их единообразную оценку безопасности. Статья является обзорно-проектным исследованием: в ней рассматриваются стандарты и автоматизированные методы тестирования: генерация тестов на основе спецификаций, фаззинг с учётом состояния, проверка авторизации, тестирование потребления ресурсов и непрерывная оценка. На этой основе предлагается фреймворк тестирования безопасности, согласованный со стандартами OWASP и охватывающий оба протокола. Восемь групп существующих подходов классифицированы по девяти измерениям. Анализ показал, что современные инструменты предоставляют эффективные отдельные компоненты, однако эти компоненты остаются фрагментированными по протоколам и целям тестирования. Основной методологический вклад состоит в том, что каждому семейству оракулов сопоставлен явный логический предикат над идентичностями, объектами, ответами и устойчивым состоянием, а потоковые обязательства выражены средствами линейной темпоральной логики. Состояние теста моделируется как произведение графа зависимостей операций, автомата жизненного цикла ресурса и автомата потока HTTP/2. Предложен также протокол оценки на основе парных уязвимых и исправленных сервисов. Полученные результаты носят концептуальный и спецификационный характер: практическая эффективность фреймворка в статье не оценивается и будет исследована в отдельной экспериментальной работе.

INTRODUCTION

Application programming interfaces (APIs) are a principal integration boundary in modern distributed and microservice systems. Because they expose data and business functions to remote identities, security testing of an API must evaluate authentication, authorization, input

handling, resource controls and configuration across sequences of interactions, rather than only syntactic conformance or server stability. National and international guidance treats technical security assessment in the same way, namely as a planned process of preparation, execution, analysis and mitigation rather than as a single scanning activity (Scarfone et al., 2008). The practical consequence for API testing is that a test is meaningful only when the identity that issues it, the state in which it is issued and the evidence retained after it are all defined.

REST commonly represents resources through HTTP methods, identifiers, headers and structured bodies. Although HTTP itself is stateless (Fielding et al., 2022), applications built on it require stateful sequences such as create–read–update–delete, and access decisions frequently depend on the position of a request within such a sequence. REST testing therefore uses specification-based, dependency-aware, model-based and search-based generation (Arcuri, 2019; Atlidakis et al., 2019; Liu et al., 2022). gRPC, in contrast, exposes typed remote procedure call methods over HTTP/2, whose streams, multiplexing and flow control introduce concurrency and resource concerns that have no direct equivalent in the REST request–response model (Thomson & Benfield, 2022). Unary and streaming calls additionally require protocol-specific tests for messages, metadata, deadlines and stream lifecycle (Zhang et al., 2023).

The risk families of the API Security Top 10 of the Open Worldwide Application Security Project (OWASP) (OWASP Foundation, 2023) are useful traceability labels, because they convert general security concerns into concrete preconditions, identities, mutations, expected behaviour and evidence. However, the tools reviewed in this article mainly optimize operation coverage, fault discovery or a selected class of vulnerabilities. They do not jointly provide OWASP-oriented tests for REST and gRPC, stateful identity handling, specialized security oracles and evidence suitable for continuous integration and continuous delivery (CI/CD) gating. The resulting gap is not a gap in individual techniques but a gap in composition: the necessary components exist, yet no published framework binds them to an explicit security decision procedure that is valid for both protocols.

The aim of this study is to develop, at the level of architecture and formal specification, a protocol-independent, OWASP-aligned framework for security testing of REST and gRPC APIs in which every OWASP risk family is associated with a mechanically checkable decision rule, and to define a reproducible protocol for its future empirical evaluation. To achieve this aim, the following objectives were set: (1) to review standards and automated testing techniques relevant to REST and RPC API security and to classify existing approaches along comparable dimensions; (2) to construct a cross-protocol security taxonomy in which each OWASP risk family is realized as a paired REST and gRPC test with a defined evidence type; (3) to express the security oracle of each OWASP risk family as an explicit logical predicate, and to specify streaming obligations that no state-based predicate can capture by means of temporal logic; (4) to define the state model and the modular architecture of the framework; and (5) to design an evaluation protocol with subjects, seeded ground truth, baselines, metrics and ablation configurations.

The object of the research is the process of automated security testing of web and RPC application programming interfaces. The subject of the research is the set of models, decision rules and architectural components that make such testing aligned with the OWASP risk families and reproducible across both protocols. The scientific novelty of the work consists in the formalization of security oracles as logical and temporal predicates over an explicitly defined state model, which replaces the informal textual descriptions that currently dominate this line of work, and in the unification of REST and gRPC testing under a single security-requirement model that retains protocol-specific execution.

By type, this article is a review-based design study rather than an experimental one. In accordance with the title, “developing” refers here to the design and specification stage of framework development: the contribution is a framework design, its formal model and a

reproducible evaluation plan, and the article does not claim that the framework has been fully implemented or experimentally validated. No implementation results and no quantitative measurements of detection effectiveness are reported, because the reference implementation is incomplete at the time of writing. The remainder of the article is organized as follows. The literature review summarizes the relevant standards, protocol properties and existing testing approaches. The materials and methods section describes the review design, the classification procedure, the formal notation and the planned evaluation protocol. The results and discussion section presents the taxonomy, the oracle predicates, the architecture and the state model of the framework, and discusses their limitations. The conclusion summarizes the findings and outlines the next stage of the work.

LITERATURE REVIEW

REST, OpenAPI and contract-based testing. A REST request combines an HTTP method, a target URI, a set of fields and an optional representation (Fielding et al., 2022). OpenAPI contracts describe paths, parameters, request and response bodies and security schemes, which enables the automatic generation of structurally valid requests. RESTest combines such descriptions with inter-parameter constraints and configurable oracles (Martin-Lopez et al., 2021), while RESTTESTGEN derives both nominal and error-oriented black-box requests and sequences directly from the specification (Viglianisi et al., 2020). Contracts, however, define syntax and rarely encode ownership, roles, abuse limits or business invariants. Conformance to a contract is therefore not equivalent to security: a request may be perfectly valid with respect to the schema and still constitute an authorization violation.

gRPC, protocol buffers and transport security. gRPC defines services and typed protocol-buffer messages and commonly uses HTTP/2 as its transport. Testing must therefore handle scalar, nested, repeated and returned fields, as well as unary, client-streaming, server-streaming and bidirectional streaming methods. Fuzzing of RPC interfaces consequently requires schema-aware and grammar-aware generation (Veldkamp et al., 2023; Zhang et al., 2023). Transport Layer Security (TLS) protects channel confidentiality, integrity and server identity, and mutual TLS can additionally authenticate clients. Transport identity, however, does not replace object-level, field-level or function-level authorization, so both the channel configuration and the application policy require separate testing.

Security of microservice architectures. Microservices exchange data across independent services and trust boundaries. Authentication establishes a user, client or workload identity, whereas authorization determines the objects, fields and functions that this identity may reach. Stateful dependency models improve the reachability of deep application logic (Atlidakis et al., 2019; Lin et al., 2023), but security oracles must preserve identity and ownership information across the whole sequence, otherwise the verdict cannot be attributed to a specific identity. Structured payload mutation can reach deeper processing stages while remaining schema-valid (Godefroid et al., 2020), which is the property that distinguishes security-relevant fuzzing from generic malformed-input testing.

Existing REST and gRPC API testing approaches. RESTler infers producer–consumer dependencies between operations and generates stateful request sequences, which allows it to reach states that single-request testing cannot reach (Atlidakis et al., 2019). RESTest supports fuzzing, constraint-based generation, configurable oracles and continuous execution (Martin-Lopez et al., 2021). RESTTESTGEN generates nominal and error-oriented black-box tests from REST descriptions (Viglianisi et al., 2020). MOREST maintains and updates a property graph of the API using execution feedback, thereby adapting the model during the campaign (Liu et al., 2022). These systems substantially improve reachability, but none of them is a complete verifier of security controls, because their oracles are primarily oriented towards unexpected status codes

and server faults.

VOAPI2 adopts an explicitly vulnerability-oriented approach to REST testing, associating interface functionality with likely vulnerability classes and confirming findings through feedback-based testing (Du et al., 2024). EvoMaster applies search-based black-box and white-box generation to both REST and RPC APIs and is, among the reviewed systems, the one that treats both protocol families as first-class targets (Arcuri, 2019; Arcuri et al., 2025; Zhang et al., 2023). Complementary work targets intelligent payload mutation (Godefroid et al., 2020), tree-based dependency modelling (Lin et al., 2023), grammar-based fuzzing of JSON-RPC interfaces (Veldkamp et al., 2023) and the detection of mass-assignment vulnerabilities, which demonstrates that field-level authorization requires specialized sequences and state-based evidence (Corradini et al., 2023). Dynamic application security testing (DAST) in CI/CD pipelines, finally, contributes the reproducibility and gating perspective, but leaves state management, cost control and oracle configuration to the user. Table 1 contrasts the resulting eight families of approaches.

Table 1. Comparison of existing API testing approaches

Approach	Input	Strategy	REST / RPC	OWASP	Strength	Limi-tation	Auto.
RESTler (Atlidakis et al., 2019)	Contract, dictionary	Stateful dependency fuzzing	Yes / No	Indirect	Valid sequences, value reuse	Limited security oracles	High
RESTest (Martin-Lopez et al., 2021)	Contract, constraints, oracles	Black-box mixed generation	Yes / No	Partial	Extensible, CI-oriented	Depends on supplied rules	High
RESTTESTGEN (Viglianisi et al., 2020)	Contract, endpoints	Nominal and error black-box tests	Yes / No	Indirect	Automatic requests and sequences	Mainly functional	High
MOREST (Liu et al., 2022)	Contract, responses, feedback	Dynamic model-based testing	Yes / No	Indirect	Adaptive dependencies	No security model	High
VOAPI2 (Du et al., 2024)	Operations, vulnerability knowledge	Vulnerability-oriented testing	Yes / No	Partial	Security-focused selection	REST- and pattern-specific	High
EvoMaster (Arcuri, 2019; Zhang et al., 2023)	Schema, endpoints, optional instrumentation	Search-based fuzzing	Yes / Yes	Partial	Feedback and RPC support	Instrumentation cost; partial OWASP mapping	High
RPC fuzzing (Veldkamp et al., 2023)	RPC schema, grammar, or source	Grammar and search-based fuzzing	Limited / Yes	Indirect	Typed method exploration	Limited stream and authorization evidence	Med.–High
DAST in CI	Service, identities, attacks, policy	Runtime regression and campaigns	Yes / Possible	Suite-based	Reproducible security gates	State, cost, and oracle setup	Med.–High

Note: compiled by the author on the basis of the reviewed sources.

The comparison makes the research gap explicit. What is missing is not another generation strategy, but a unified framework that maps OWASP objectives onto reproducible REST and gRPC tests while combining contract parsing, identity management, stateful generation, fuzzing, security-specific oracles with explicit decision predicates, retained evidence and CI/CD execution. The present article addresses this gap at the level of design and specification.

MATERIALS AND METHODS

Design of the review and composition of the corpus. The study is a review-based design study without an experimental part; its review component was carried out as a targeted structured review based on official or publisher-controlled records obtained from the National Institute of Standards and Technology, the RFC Editor, IEEE Xplore, the ACM Digital Library, SpringerLink, USENIX and arXiv. Search terms combined the keys REST, gRPC, RPC, OpenAPI, stateful testing, fuzzing and authorization with the names of known tools. Primary sources were included when they defined relevant protocol behaviour or introduced or evaluated a REST or RPC testing technique. General web application testing, non-API scanning, duplicate versions of the same study and records that could not be verified were excluded. Where both a preprint and a peer-reviewed version of the same work exist, the peer-reviewed version is cited. Metadata of digital object identifiers were verified through Crossref or DataCite, and every cited record in this article resolves to a registered identifier, with the exception of the USENIX proceedings entry, for which the publisher does not assign such identifiers and the canonical publisher URL is given instead. The review is deliberately targeted rather than exhaustive; it does not claim the status of a systematic meta-analysis.

Classification procedure and mapping of risk families. The selected approaches were classified along nine dimensions: (1) input, (2) access mode, (3) generation strategy, (4) state handling, (5) use of feedback, (6) oracle type, (7) protocol support, (8) security focus and (9) degree of automation. For compactness, Table 1 reports input, generation strategy, protocol support, security focus (as OWASP alignment) and degree of automation as separate columns, while access mode, state handling, use of feedback and oracle type are summarized in the strength and limitation columns. These dimensions were chosen because they determine what a tool can observe and therefore what it can decide. The mapping onto the OWASP risk families was qualitative and was performed in a fixed order: each risk family was first translated into a protected asset, then into a precondition, a mutation, an expected secure response and a required piece of evidence, and only then realized separately for REST and for gRPC. Generic crashes were never equated with exploitability, since an unhandled server error is a reliability observation and becomes a security finding only when the security consequence is demonstrated. The limitations of this procedure are the heterogeneity of the experimental designs in the underlying studies and the stronger representation of REST than of gRPC in the available literature.

Terminology. To avoid ambiguity, the following terms are used consistently throughout the article. A REST API is a resource-oriented interface over HTTP, described, where available, by an OpenAPI contract. RPC denotes the general class of remote procedure call interfaces (including JSON-RPC and Thrift-based services), whereas gRPC denotes the specific RPC framework over HTTP/2 with protocol-buffer contracts that is the second target protocol of the framework; the term RPC is used only when a statement concerns the general class. An OWASP risk family is one of the ten entries API1:2023–API10:2023 of the OWASP API Security Top 10 (OWASP Foundation, 2023), abbreviated API1–API10 in the tables. A security oracle is a decision procedure that maps the retained evidence of one or more executions to the verdict “violation” or “no violation” for a given risk family. The framework is the set of models, decision rules and modules proposed in this article, as opposed to an individual testing tool. Stateful test generation

produces request or call sequences in which values produced by earlier operations are consumed by later ones and the state of the subject is tracked between steps; stateful fuzzing is its mutation-based form. CI/CD denotes continuous integration and continuous delivery pipelines in which security tests are executed and used as gating conditions.

Notation and execution model. Informal descriptions of oracles leave the decision procedure implicit, which is the main source of disagreement about what constitutes a finding. The framework therefore uses a common notation that a test harness can evaluate mechanically. Let I be the set of test identities, O the set of objects with the owner function $\omega : O \to I$, P the set of protocol operations (for REST, a method together with a path template; for gRPC, a service together with a method), V the space of parameter and field assignments, and Σ the space of observable service states. Execution is then the partial function.

$$\delta: \mathcal{I} \times \mathcal{P} \times V \times \Sigma \to \mathcal{R} \times \Sigma \tag{1}$$

written as $\delta(i, p, v, \sigma) = (r, \sigma')$, where R is the space of observable responses (status, headers or metadata, and body or message) and $r \in R$ is the response to a single execution. The predicate ok(r) holds when the response r carries a success status, that is an HTTP 2xx code or the gRPC status OK; obs(r, f) holds when the field f is observable in the response; $\Delta(\sigma, \sigma')$ denotes the difference in persistent state; and auth(i, p, o) denotes the ground-truth policy that permits the identity i to invoke the operation p on the object o. Read and write permissions on a field f are written auth(i, f, o) and wauth(i, f, o) respectively, and $F_{restr}(i, o)$ denotes the set of fields of o that i may not read. This notation is protocol-independent: the same symbols are instantiated by an HTTP request in the REST case and by a typed remote procedure call in the gRPC case. The set of fields of an object o is written attr(o), σ(o).f denotes the value of the field f of the object o in the state σ, and $\Delta \neq \emptyset$ abbreviates $\Delta(\sigma, \sigma') \neq \emptyset$. Each security oracle is written as a violation predicate V_k, whose index k is the number of the corresponding OWASP risk family APIk:2023 (for example, V_1 for API1:2023); the superscripts r and w of V_3 distinguish the read and write variants, and the families API6, API8 and API10 have no closed predicate for the reasons given below.

Design of the evaluation protocol. Although no experiment is reported here, the evaluation is specified in advance so that it is reproducible by third parties. Evaluation should use paired vulnerable and patched REST and gRPC services that implement equivalent user, document, order or URL-retrieval workflows. Subjects should include multiple roles, owner and non-owner objects, REST contracts, protobuf descriptors and both unary and streaming methods, so that the protocol dimension can be isolated from the application dimension.

Seeded defects should be small, reviewed changes covering authorization, authentication, sensitive fields, resource limits, business-flow abuse, server-side request forgery, unsafe consumption of downstream data and misconfiguration. Each seed requires an identifier, a trigger, a precondition, an expected behaviour and a ground truth expressed as the corresponding predicate defined below, so that detection is scored against a formal condition rather than against a textual description. This is the point at which the formalization becomes practically relevant: the same predicate serves as the specification of the oracle and as the definition of the ground truth.

REST baselines should include RESTler, RESTest, RESTTESTGEN, MOREST and EvoMaster, while RPC baselines should include EvoMaster RPC and grammar-based RPC fuzzing; tools are scored only on the modes they support. The metrics are OWASP risk-family coverage, vulnerability detection rate, false-positive rate FP/(FP + TN), false-negative rate FN/(FN + TP), endpoint and method coverage, stateful dependency and sequence coverage, time to first finding, total execution time and CI/CD integration effort. Each trial should use a clean containerized deployment, fixed credentials and resources, equivalent specifications, full state

reset between runs and recorded tool versions. Stochastic tools require repeated runs with recorded seeds and equal time or request budgets, and patched subjects are used to measure false positives.

Four ablation configurations are foreseen: disabling the risk mapping, disabling stateful generation, disabling execution feedback and disabling the specialized oracles. Because the state model described below is a product of three components, the lifecycle layer and the stream layer can be ablated independently in order to isolate their individual contribution. Retained evidence includes requests or calls, identities, sequence steps, responses, statuses and timing. Construct validity is threatened when risk-family coverage or crashes are mistaken for vulnerabilities; internal validity is affected by configuration, randomness and residual state; external validity is limited by the range of languages, frameworks and subjects; and conclusion validity is affected by small or correlated seed sets. Ground truth, repeated trials, state reset, transparent configurations and deliberately bounded conclusions are the corresponding mitigations.

RESULTS AND DISCUSSION

A cross-protocol security taxonomy. The risk families of the OWASP API Security Top 10 are broken object-level authorization, broken authentication, broken object-property-level authorization, unrestricted resource consumption, broken function-level authorization, unrestricted access to sensitive business flows, server-side request forgery (SSRF), security misconfiguration, improper inventory management and unsafe consumption of APIs (OWASP Foundation, 2023). In the proposed framework they organize tests rather than certify coverage. A test is represented by a protocol operation, an identity, a required state, a mutation, an oracle and the evidence that must be retained.

REST realizations mutate methods, identifiers, parameters, headers, cookies and body fields, whereas gRPC realizations mutate methods, protobuf fields, metadata, deadlines and stream actions. Authorization testing requires cross-role and cross-owner comparisons in both cases. Research on mass assignment shows that field-level authorization additionally needs specialized sequences and state-based evidence (Corradini et al., 2023). Injection tests retain enough schema validity to reach application logic, while SSRF requires controlled outbound evidence. Resource tests vary rate, concurrency, size and duration, with gRPC additionally testing concurrent streams and message frequency. Misconfiguration tests examine TLS and mutual TLS settings, verbose errors, exposed descriptions or reflection interfaces, dangerous methods and inconsistent policy. Table 2 summarizes the realization of each family in both protocols and names the evidence that decides the verdict.

Table 2. OWASP risk-to-test mapping for REST and gRPC

Risk family	REST test	gRPC test	Oracle / evidence	Formula
Object authorization	Substitute object identifiers across identities	Substitute identifier fields and results	Unauthorized read or state change, with owner control	(2)
Authentication	Omit, alter, expire, or replay cookies, keys, tokens	Alter metadata or channel credentials	Protected result returned without valid identity	(3)
Property authorization	Add or read restricted JSON fields	Set or read restricted message fields	Unauthorized disclosure, or change confirmed by owner read	(4), (5)
Resource consumption	Vary rate, concurrency, size, pagination	Vary calls, messages, streams, size, duration	Missing limits or uncontrolled degradation	(6)

Risk family	REST test	gRPC test	Oracle / evidence	Formula
Function authorization	Invoke privileged routes under weak roles	Invoke privileged RPC methods	Successful protected state transition	(7)
Sensitive flows	Replay or reorder protected workflows	Repeat or reorder unary or streaming workflows	Policy or quota bypass beyond Q	—
SSRF	Mutate URL, callback, webhook, import fields	Mutate URL or host message fields	Nonce received at controlled collector	(8)
Misconfiguration	Check TLS, CORS, errors, methods, exposed contracts	Check TLS/mTLS, reflection, errors, limits	Deviation from declared policy	—
Inventory	Compare discovered routes and versions with contract	Compare services and methods with descriptors	Undeclared reachable interface	(9)
Unsafe consumption	Supply malicious downstream response data	Return malicious messages or metadata from dependency	Injection, bypass, or unsafe persisted state	—

Note: compiled by the author. Formula numbers refer to the predicates defined below.

Formal oracle predicates. Broken object-level authorization. A violation requires that the policy forbids the access, that the call nevertheless succeeded, that something was actually disclosed or changed, and that a positive control executed by the legitimate owner also succeeds:

$$V_1 \equiv \neg\mathsf{auth}(i,p,o) \wedge \mathsf{ok}(r) \wedge \mathsf{ok}(r_{\mathsf{ctrl}}) \wedge \left[\mathsf{obs}\big(r,\mathsf{attr}(o)\big) \vee \Delta(\sigma,\sigma') \neq \varnothing\right] \quad (2)$$

where r_{ctrl} is the response of $\delta(\omega(o), p, o, \sigma)$. The control term is not decorative: without it, an empty successful response and a correctly denied request are indistinguishable, and a deleted or non-existent object is reported as a false positive.

Broken authentication. For a degraded credential $i_\perp \in$ {anonymous, expired, revoked, replayed, forged} and a valid credential i_v applied to the same operation, the condition is

$$V_2 \equiv \mathsf{ok}\big(\delta(i_\perp,p,v,\sigma)\big) \wedge \mathsf{ok}\big(\delta(i_v,p,v,\sigma)\big) \quad (3)$$

The second conjunct guarantees that the operation is genuinely available and that the first conjunct is not satisfied by a service that accepts every request.

Broken object-property-level authorization. Reading and writing are distinct conditions. Unauthorized disclosure is expressed as

$$V_3^r \equiv \exists f \in F_{\mathsf{restr}}(i,o):\mathsf{obs}(r,f) \quad (4)$$

while unauthorized assignment is expressed as

$$V_3^w \equiv \neg\mathsf{wauth}(i,f,o) \wedge \sigma'\big|_{o.f} = v \wedge \sigma\big|_{o.f} \neq v \quad (5)$$

Here v denotes the value that the mutation assigns to the field f, and σ' is the state after the mutation. Condition (5) must be confirmed by a subsequent read issued under $\omega(o)$, because the response to a mutation frequently echoes the submitted value without persisting it. This is precisely the state-based evidence requirement identified in mass-assignment testing (Corradini et al., 2023).

Unrestricted resource consumption. Let L be the declared limit on successful requests or calls within an observation window and K = |{k : ok(r_k)}| the number of successful calls observed in that window, where r_k is the response to the k-th call. Let lat(σ) be the median latency of a fixed probe of legitimate requests in the state σ, where σ_{base} and σ_{after} denote the states before the load and at its end, and let λ = lat(σ_{after}) / lat(σ_{base}) be the latency ratio with threshold τ. Server-side consumption is described by the set of monitored indicators M = {cpu, mem, net, str}, namely CPU utilization, memory (RAM) utilization, network bandwidth and, for gRPC, the number of concurrently active HTTP/2 streams; $u_m(\sigma)$ is the value of the indicator m in the state σ, κ_m = $u_m(\sigma_{after})$ / $u_m(\sigma_{base})$ is its ratio and τ_m the admissible ratio. Finally, t_{rec} is the recovery time, i.e. the time after the end of the load until λ and every κ_m return below their thresholds, with bound T_{max}, and s_{leg} is the success rate of the legitimate probe during the load, which measures the impact on legitimate users, with lower bound s_{min}. The condition is

$$V_4 \equiv (K > L) \vee (\lambda > \tau) \vee (\exists m \in M{:}\, \kappa_m > \tau_m) \vee (t_{\mathrm{rec}} > T_{\max}) \vee \left(s_{\mathrm{leg}} < s_{\min}\right) \qquad (6)$$

The first two disjuncts form the minimal black-box variant of the oracle, which requires only responses and is applicable to any subject; when only these disjuncts can be evaluated, (6) is to be understood as a minimal example of a security oracle rather than as a complete model of resource exhaustion. The third disjunct requires server-side monitoring of the subject, the fourth requires observation continued after the load, and the fifth requires a concurrent legitimate probe; each is enabled when the corresponding observation is available. For gRPC, the number of active streams is additionally compared with the advertised SETTINGS_MAX_CONCURRENT_STREAMS value, because the acceptance of streams beyond that value without a stream error indicates a missing transport-level limit (Thomson & Benfield, 2022). All thresholds L, τ, τ_m, T_{max} and s_{min} are policy parameters supplied as configuration, because safe values depend on the capacity of the subject under test.

Broken function-level authorization. For the set of privileged operations P_{priv} and the role assignment π(i), the condition is

$$V_5 \equiv p \in P_{\mathrm{priv}} \wedge \pi(i) \cap \mathrm{allowed}(p) = \varnothing \wedge \mathrm{ok}(r) \wedge \Delta \neq \varnothing \qquad (7)$$

where allowed(p) is the set of roles permitted to invoke p, so that π(i) ∩ allowed(p) = ∅ states that no role of i grants access to p.

Unrestricted access to sensitive business flows. For a protected workflow ϱ = ⟨p_1 … p_n⟩ with quota Q, a violation is a repetition or a reordering ϱ′ that reaches the same terminal state more than Q times. This family is deliberately left without a closed formula, because the quota and the terminal state are application-specific and must be supplied as configuration.

Server-side request forgery. Let n be a unique nonce embedded in a candidate URL, callback, webhook or import field, and let cb(n) denote the receipt of n at a controlled collector within the window Δt:

$$V_7 \equiv \exists n{:}\mathrm{cb}(n) \qquad (8)$$

This predicate is deliberately independent of r. Out-of-band evidence, and not the status code, decides the finding, which illustrates the general limitation of response-based oracles discussed below.

Improper inventory management. For a declared contract C, a reachable but undeclared operation is a finding:

$$V_9 \equiv \mathrm{reachable}(p) \wedge p \notin \mathrm{declared}(C) \qquad (9)$$

Streaming lifecycle. Response-based and state-based predicates cannot express gRPC stream obligations, because these are properties of an event trace $t = \langle e_1 \dots e_n \rangle$ rather than of a single observation. Using linear temporal logic, a compliant server must stop emitting messages once authorization is lost or the stream is terminated:

$$\mathbf{G}(\text{revoked} \vee \text{deadline} \vee \text{cancel} \rightarrow \mathbf{X}\,\mathbf{G}\,\neg\text{ServerMsg}) \tag{10}$$

Here G ("globally") and X ("next") are the standard operators of linear temporal logic (LTL); revoked, deadline and cancel are atomic propositions that hold at the event at which the credential of the stream is revoked, its deadline expires (DEADLINE_EXCEEDED) or the stream is cancelled (CANCELLED); and ServerMsg holds at every event at which the server emits a message on that stream. A violation is any index $j > k$ at which ServerMsg holds while the triggering event occurred at k. Backpressure and deadline obligations are expressed analogously, which yields the temporal oracles required by the streaming rows of the module table below.

Architecture of the proposed framework. The framework separates protocol-independent security intent from protocol-specific discovery, construction and observation. Discovery feeds the OpenAPI and proto/reflection parsers, which produce a normalized operation model. Identity management supplies role-specific cookies, tokens, keys, metadata and certificates. The mapping engine assigns risks, mutations, identities and oracles to operations; generation then combines valid baselines, stateful dependencies, boundary values and security payloads. The architecture is shown in Figure 1, and the responsibilities of the individual modules are given in Table 3.

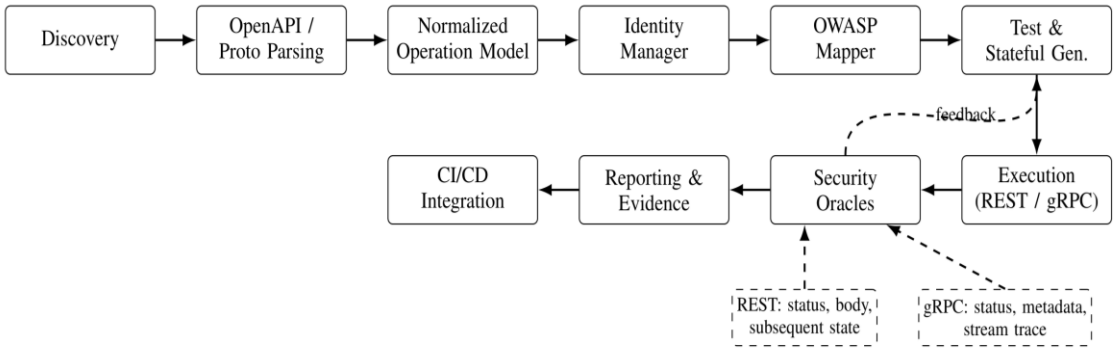


Figure 1. Architecture of the proposed framework

Note: compiled by the author. Solid arrows show the primary pipeline from interface discovery to CI/CD gating; the dashed arrow shows execution feedback refining stateful generation. Oracle inputs differ by protocol: REST oracles consume status, body and subsequent resource state, while gRPC oracles additionally consume metadata and the stream event trace required by predicate (10).

For REST, the workflow parses paths, parameters, bodies, responses and security schemes; creates role-specific baselines; reuses response identifiers; and mutates methods, fields and the order of a sequence. The oracles compare HTTP responses with the subsequent state of the affected resource. For gRPC, descriptors identify typed messages and unary or streaming methods. Tests mutate fields, metadata, deadlines, message order, stream duration, cancellation and concurrency, and the oracles evaluate gRPC statuses, messages, metadata, persistent state and stream termination (Thomson & Benfield, 2022; Zhang et al., 2023).

Table 3. Framework modules, responsibilities and protocol-specific behaviour

Module(s)	Purpose	Input / output	Risks	REST behaviour	gRPC behaviour
Discovery; OpenAPI parser; proto/reflection parser	Inventory and normalize interfaces	Targets, contracts → API model	API8–10	Paths, methods, schemas	Services, methods, messages, streams
Authentication and token manager	Maintain identities and sessions	Credentials → execution contexts	API1–3, 5–6	Headers, cookies, keys	Metadata, tokens, channel credentials
OWASP mapper; test generator	Derive executable security tests	Model, rules → test definitions	API1–10	HTTP operations and mutations	Typed calls and stream actions
Stateful generator; fuzzing engine	Explore D × Lρ × S and input space	Tests, feedback → executions	API1, 3–8, 10	Resource sequences and payloads	Method dependencies, fields, stream events
Authorization oracle	Decide predicates (2)–(7)	Multi-role traces → findings	API1, 3, 5–6	Object, property, function comparison	Method, field, owner comparison
Resource, injection, misconfiguration testers	Test limits, unsafe inputs, deployment	Policies, payloads → evidence	API3–4, 7–10	Rate, bodies, TLS, CORS, errors	Streams, fields, TLS/mTLS, reflection
Reporting; CI/CD integration	Reproduce, prioritize, and gate	Findings, policy → reports, build status	API1–10	HTTP traces and regression suites	RPC transcripts and stream suites

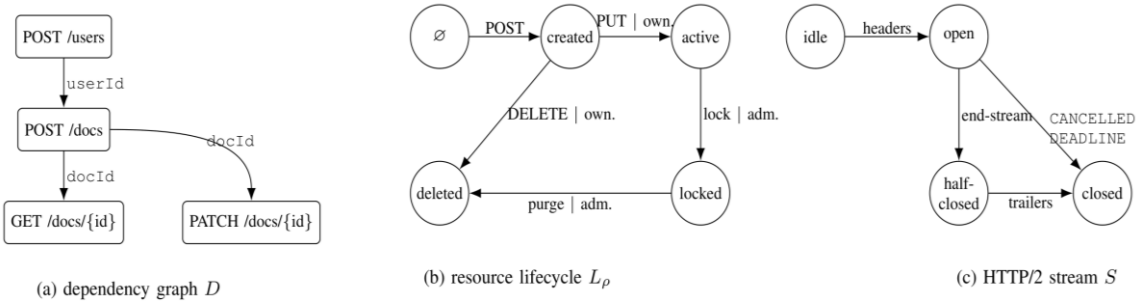
Note: compiled by the author.

State binding and the API state model. The step from single HTTP requests to long-lived gRPC connections is the point at which a purely sequential test model becomes insufficient, so the state model is defined explicitly rather than assumed. The framework represents test state as a product of three components, illustrated in Figure 2.

First, the operation dependency graph $D = (P, E)$ is a directed graph in which an edge $p_a \rightarrow p_b$ exists when p_a produces a value that p_b consumes. This is the producer–consumer abstraction used by stateful REST fuzzers (Atlidakis et al., 2019) and, in a richer form, by property-graph models (Liu et al., 2022).

Second, each resource type ϱ carries a lifecycle automaton $L_\varrho = (Q_\varrho, q_0, \Gamma_\varrho)$ over states such as {∅, created, active, locked, deleted}, whose transitions are labelled by an operation together with a guard encoding the required role and ownership. The lifecycle automaton is what makes an authorization test meaningful: the same call is legitimate in one state and a violation in another, and without the state component this distinction cannot be represented at all. Here Q_ϱ is the finite set of lifecycle states, $q_0 = \emptyset$ is the initial state in which the resource does not yet exist, and Γ_ϱ is the guarded transition relation.

Third, for gRPC the stream automaton S is the HTTP/2 stream state machine, with the states idle, open, half-closed (local), half-closed (remote) and closed, extended with the distinguished events DEADLINE_EXCEEDED and CANCELLED. Because these transitions are normative (Thomson & Benfield, 2022), the automaton is adopted rather than inferred, and multiplexed streams are modelled as independent instances of S sharing a single connection-level flow-control budget.



(a) dependency graph D (b) resource lifecycle L_ρ (c) HTTP/2 stream S

Figure 2. API state model used for stateful test generation

Note: compiled by the author. A concrete test is a path through the product $D \times L_\rho \times S$ together with the provenance environment β of formula (11), which records the identity and the timestamp of every harvested identifier. Guards on the lifecycle transitions carry the role and ownership conditions evaluated by predicates (2)–(7); the stream automaton in part (c) is normative in RFC 9113 and supplies the event trace over which the temporal oracle (10) is checked.

Binding between the three components is provided by an environment

$$\beta : \mathrm{Refs} \to (\mathrm{Value} \times \mathcal{I} \times \mathrm{ts}) \tag{11}$$

where Refs is the set of identifiers harvested from responses, Value the domain of their concrete values and ts the timestamp domain; β records, for every such identifier, the value, the identity under which it was created and a timestamp. A concrete test is then a path through the product $D \times L_\varrho \times S$ together with β. The provenance component is what allows the owner term ω(o) in predicates (2) and (5) to be resolved at execution time instead of being supplied manually, and the timestamp supports the handling of expiry and interference discussed below. For streaming methods the path is additionally annotated with the event trace required by predicate (10).

Oracles, reporting and the status of the prototype. Authorization oracles evaluate predicates (2)–(7) by comparing roles, owners, fields and functions. Resource testers instantiate (6) by varying rate, concurrency, size and duration. Injection testers preserve structural validity where this is necessary to reach application logic, and SSRF tests instantiate (8) with controlled destinations. Misconfiguration checks cover transport settings, reflection or exposed contracts, error verbosity and limits. Reports bind each finding to a risk family, an identity, a sequence, the retained evidence, a severity and a reproduction procedure. CI/CD integration separates fast regression tests from scheduled fuzzing campaigns and retains artifacts for later inspection.

A reference implementation of the framework is under active development. At the time of writing, the contract-parsing, identity-management and mapping modules exist as a working prototype, while the streaming oracles of (10) are only partially implemented; no evaluation results are reported from this prototype in the present article. The prototype, the seeded benchmark services and the ground-truth mappings will be released as an open repository so that the evaluation plan described above is reproducible by third parties.

Discussion

The results reported above establish conceptual feasibility rather than practical effectiveness, and the two should be kept apart. Conceptual feasibility means that every OWASP risk family has a paired REST and gRPC realization, that seven families have a decision predicate

over observable quantities, and that the state model supplies every quantity these predicates require: ownership through β, lifecycle state through L_{ϱ} and stream events through S. Practical effectiveness, that is the detection rate, the false-positive and false-negative rates, coverage and cost on concrete services, can only be established empirically; it lies outside the scope of this review-based design study and is the object of the evaluation protocol defined in the materials and methods section. The article is therefore intended to be used as a specification against which implementations can be built and compared, and not as evidence that a particular implementation detects vulnerabilities.

Standards are necessary but insufficient, because they do not encode application ownership, roles, business invariants or safe thresholds. Authorization is the most difficult family in this respect: an identical response may be legitimate for one identity and a violation for another, which is precisely why predicate (2) is stated over a pair of executions rather than over a single response. Stateful tests improve reachability, but they also enlarge the search space, introduce expiring values and interference between concurrent tests, and complicate reproduction; the timestamp component of β in (11) exists in order to make expiry explicit rather than incidental. gRPC adds typed binary payloads and streaming lifecycles, so naive byte-level mutations may fail before reaching application logic, while message order, cancellation, deadlines and flow control require the temporal oracles of (10).

Oracles remain the main epistemic limit of the approach. A server error need not be exploitable, whereas unauthorized access may return an entirely normal status. SSRF requires controlled callback evidence, which is the reason why (8) ignores the response entirely, and resource tests need safe thresholds and recovery observations in order not to damage the subject under test. Adoption in CI/CD additionally requires identity provisioning, state reset, credential protection, artifact retention and a clear separation of fast regression suites from expensive campaigns. White-box search can improve guidance, but it increases instrumentation and maintenance cost (Arcuri et al., 2025). Normalization cannot remove the semantic differences between resources and long-lived gRPC streams, and the mapping onto OWASP risk families supplies traceability rather than a proof of complete coverage. Further limitations include incomplete contracts, unavailable role knowledge, undocumented business rules, cross-service side effects and the restricted visibility of black-box testing.

Ground truth beyond seeded subjects. Seeded defects make the ground truth exact by construction, because each seed is defined by the same predicate that its oracle checks. For real-world and industrial APIs this is no longer the case: the true policy auth(i, p, o) is not available in closed form, role knowledge may be incomplete, and a satisfied predicate is only a candidate finding. For such subjects the framework therefore treats ground truth as the outcome of a confirmation procedure with graded evidence rather than as an input. Four complementary mechanisms are foreseen.

First, expert validation: each candidate finding is reviewed independently by at least two security analysts against the documented access-control policy and business rules, disagreements are resolved by discussion, and inter-rater agreement is reported. Second, server log analysis: a unique correlation identifier is injected into every request header or gRPC metadata entry, so that application and access logs can confirm that the request reached the handler and that the authorization decision recorded by the service contradicts the policy. Third, distributed tracing: trace context propagated with the test shows which downstream services, queries and state changes a request caused, which is required for cross-service effects, SSRF chains and unsafe consumption of APIs. Fourth, white-box instrumentation: where source code or bytecode access is available, instrumentation of authorization checks and data-access calls, as used in white-box search-based testing (Zhang et al., 2023; Arcuri et al., 2025), shows directly whether an access

check was executed and what it returned.

A finding on a real-world API is reported as confirmed only when its predicate holds and at least one independent mechanism corroborates it; otherwise it is reported as unconfirmed together with its evidence. In this way detection-rate and false-positive figures obtained on real APIs remain distinguishable from those obtained on seeded subjects, and the cost of confirmation, which grows with the number of candidate findings, becomes an explicit part of the evaluation.

Directions for further research. Future work should evaluate constrained assistance by large language models for the generation of semantic values, candidate business invariants and adversarial sequences; the outputs of such assistance must remain schema-valid and must be verified by the oracles rather than trusted. Research on gRPC should improve stream-state fuzzing with respect to ordering, concurrency, cancellation, backpressure and deadlines, extending the temporal specification of (10) from safety obligations to liveness obligations as well. Service-mesh adapters could compare workload identity, mutual TLS configuration, routing and authorization policy with the observed behaviour. Runtime traces, logs, resource signals and outbound connections could strengthen multi-service oracles. Maintained benchmarks should provide paired REST and gRPC services in vulnerable and patched variants, together with ownership models, stream scenarios and ground-truth OWASP mappings. The immediate next step, however, is an initial experimental validation on a limited set of intentionally vulnerable and patched REST and gRPC services covering several OWASP risk families, which will separate the practical effectiveness of the framework from the conceptual feasibility established in this article, followed by the completion of the prototype and the execution of the evaluation protocol described in the materials and methods section.

CONCLUSION

This article reviewed standards and automated REST and RPC testing approaches and identified useful but fragmented support for contracts, dependencies, execution feedback, search-based generation, payload mutation and vulnerability-specific testing. Eight families of approaches were classified along nine dimensions; none of them provides OWASP-aligned coverage for REST and RPC together with identity-aware stateful generation and specialized security oracles, and only one treats both protocols as first-class targets. The objectives formulated in the introduction were therefore addressed as follows.

All ten families of the OWASP API Security Top 10 were mapped onto paired REST and gRPC realizations with a defined evidence type; seven of them were additionally given an executable decision predicate, and the streaming obligations that no state-based predicate can express were specified in linear temporal logic. State binding, which in this line of work had previously been described only informally, was defined as the product $D \times L_{\varrho} \times S$ of a dependency graph, a five-state resource lifecycle and the normative HTTP/2 stream automaton, closed under a provenance environment β that resolves object ownership at execution time. The framework was decomposed into seven modules with declared inputs, outputs and OWASP responsibilities, and the evaluation protocol fixed nine metrics, four ablation configurations and a seeded ground-truth format tied to the predicates above.

The theoretical significance of the work lies in replacing informal oracle descriptions with checkable conditions, which makes disagreements about what constitutes a finding resolvable by inspection of a formula rather than by interpretation of prose. Its practical significance lies in the fact that the same predicates serve simultaneously as the specification of the oracle, as the definition of the seeded ground truth and as the gating condition in a delivery pipeline, which is what makes security results reproducible across teams and tools.

The article is a review-based design study, and these are design and specification results,

not empirical ones: the framework is presented at the level of architecture, formal model and evaluation protocol, and its practical effectiveness remains to be established. The article reports no detection rates, no coverage measurements and no performance figures, because the reference implementation is incomplete and the protocol described above must first be executed against paired vulnerable and patched subjects. Establishing detection capability, false-positive behaviour, cost and generalizability is therefore the explicit object of the next stage of this work.

CONFLICT OF INTEREST: The author declares no conflict of interest.

FUNDING: This research received no external funding.

INSTITUTIONAL REVIEW BOARD STATEMENT: Not applicable. The study did not involve human participants or animals.

INFORMED CONSENT STATEMENT: Not applicable.

DATA AVAILABILITY STATEMENT: No new empirical data were created or analysed in this study. The reference prototype, the seeded benchmark services and the ground-truth mappings described in the article will be made openly available in a public repository once the implementation is completed.

ACKNOWLEDGEMENTS: The author thanks the colleagues of the School of Cybersecurity of Astana IT University for their comments on earlier versions of this manuscript.

STATEMENT ON THE USE OF ARTIFICIAL INTELLIGENCE TECHNOLOGIES: A large language model was used for language editing and for formatting the manuscript in accordance with the journal template. It was not used to generate research results, formal definitions or conclusions. The author reviewed and verified all content and bears full responsibility for it.

REFERENCES

Arcuri, A. (2019). RESTful API automated test case generation with EvoMaster. ACM Transactions on Software Engineering and Methodology, 28(1), Article 3, 1–37. https://doi.org/10.1145/3293455

Arcuri, A., Zhang, M., Seran, S., Galeotti, J. P., Golmohammadi, A., Duman, O., Aldasoro, A., & Ghianni, H. (2025). Tool report: EvoMaster — black and white box search-based fuzzing for REST, GraphQL and RPC APIs. Automated Software Engineering, 32(1), Article 4. https://doi.org/10.1007/s10515-024-00478-1

Atlidakis, V., Godefroid, P., & Polishchuk, M. (2019). RESTler: Stateful REST API fuzzing. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (pp. 748–758). IEEE. https://doi.org/10.1109/ICSE.2019.00083

Corradini, D., Pasqua, M., & Ceccato, M. (2023). Automated black-box testing of mass assignment vulnerabilities in RESTful APIs. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (pp. 2553–2564). IEEE. https://doi.org/10.1109/ICSE48619.2023.00213

Du, W., Li, J., Wang, Y., Chen, L., Zhao, R., Zhu, J., Han, Z., Wang, Y., & Xue, Z. (2024). Vulnerability-oriented testing for RESTful APIs. In Proceedings of the 33rd USENIX Security Symposium (pp. 739–755). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/du

Fielding, R., Nottingham, M., & Reschke, J. (Eds.). (2022). HTTP semantics (RFC 9110). RFC Editor. https://doi.org/10.17487/RFC9110

Godefroid, P., Huang, B.-Y., & Polishchuk, M. (2020). Intelligent REST API data fuzzing. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) (pp. 725–736). ACM. https://doi.org/10.1145/3368089.3409719

Lin, J., Li, T., Chen, Y., Wei, G., Lin, J., Zhang, S., & Xu, H. (2023). foREST: A tree-based black-box

fuzzing approach for RESTful APIs. In Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE) (pp. 695–705). IEEE. https://doi.org/10.1109/ISSRE59848.2023.00023

Liu, Y., Li, Y., Deng, G., Liu, Y., Wan, R., Wu, R., Ji, D., Xu, S., & Bao, M. (2022). Morest: Model-based RESTful API testing with execution feedback. In Proceedings of the 44th International Conference on Software Engineering (ICSE) (pp. 1406–1417). ACM. https://doi.org/10.1145/3510003.3510133

Martin-Lopez, A., Segura, S., & Ruiz-Cortés, A. (2021). RESTest: Automated black-box testing of RESTful web APIs. In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA) (pp. 682–685). ACM. https://doi.org/10.1145/3460319.3469082

OWASP Foundation. (2023). OWASP API Security Top 10 – 2023. https://api-security.owasp.org/editions/2023/en/0x11-t10/

Scarfone, K. A., Souppaya, M. P., Cody, A., & Orebaugh, A. D. (2008). Technical guide to information security testing and assessment (NIST Special Publication 800-115). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-115

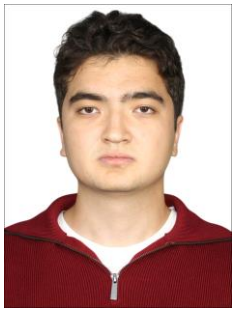
Thomson, M., & Benfield, C. (Eds.). (2022). HTTP/2 (RFC 9113). RFC Editor. https://doi.org/10.17487/RFC9113

Veldkamp, L. S., Olsthoorn, M., & Panichella, A. (2023). Grammar-based evolutionary fuzzing for JSON-RPC APIs. In Proceedings of the 2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT) (pp. 33–36). IEEE. https://doi.org/10.1109/SBFT59156.2023.00026

Viglianisi, E., Dallago, M., & Ceccato, M. (2020). RESTTESTGEN: Automated black-box testing of RESTful APIs. In Proceedings of the 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST) (pp. 142–152). IEEE. https://doi.org/10.1109/ICST46399.2020.00024

Zhang, M., Arcuri, A., Li, Y., Liu, Y., & Xue, K. (2023). White-box fuzzing RPC-based APIs with EvoMaster: An industrial case study. ACM Transactions on Software Engineering and Methodology, 32(5), Article 122, 1–38. https://doi.org/10.1145/3585009

Авторлар туралы мәліметтер
Информация об авторах
Information about authors



Ельжанов Асанали - ақпараттық қауіпсіздік магистратурасының білім алушысы, Astana IT University, Астана, Қазақстан

Ельжанов Асанали — магистрант по направлению информационная безопасность, Astana IT University, Астана, Казахстан

Yelzhanov Assanali — Master's student in Information Security, Astana IT University, Astana, Kazakhstan

e-mail: 255305@astanait.edu.kz, ORCID: https://orcid.org/0009-0008-4576-8570