Privilege-Escalation Vulnerability Discovery for Large-scale RPC Services: Principle, Design, and Deployment

A work based on operational experiences

Zhuotao Liu[1†], Hao Zhao[2†], Sainan Li[1], Qi Li[1], Tao Wei[2], Yu Wang[2]
[1] Tsinghua University, [2] Ant Group

Background and Problem Statement

RPC System Overviews

- One of world's largest RPC deployments at Ant Group
- Hundreds of financial services / One billion users worldwide / hundreds of billions of RPC requests every day

Background

RPC System Overviews

- One of world's largest RPC deployments
- Hundreds of financial services / One billion users worldwide / hundreds of billions of RPC requests every day

RPC security is crucial due to our large user base

Privilege-escalation vulnerabilities = catastrophic sensitive data losses!

Background

RPC System Overviews

- One of world's largest RPC deployments
- Hundreds of financial services / One billion users worldwide / hundreds of billions of RPC requests every day

RPC security is crucial due to our large user base

Privilege-escalation vulnerabilities = catastrophic sensitive data losses!

Academic Proposals

- Request-response analysis
- Probing-based solution to learn the system
- Static and/or dynamic program analysis
- Others

Old-School Engineering Solutions

- A "Software Quality Assurance" (SQA) team to detect the RPC vulnerabilities
- Cultivating test users to mimic the real user base
- Replaying manually engineered RPCs

Why "Reintenting the Wheels"

Characteristics for RPC System at Ant Group

- C1 - Dynamic and Unknown RPC Processing Logic:
  - The desired RPC handling logic is unknown a priori (complexity)
  - The desired handling logic often depends on user attributes (dynamism)

Prior Arts / Solutions	Adoption Challenges for our Systems
Request-reponse analysis	
Probing-based solution to learn the system	C1
Static and/or dynamic program analysis	
Manually Engineered RPC Requests	

Why "Reintenting the Wheels"

Characteristics for RPC System at Ant Group

- C1 – Dynamic and Unknown RPC Processing Logic
- C2 – Private RPC Responses:
  - Not allowed to parse the RPC responses for non-testing users for privacy concerns

Prior Arts / Solutions	Adoption Challenges for our Systems
Request-reponse analysis	C2
Probing-based solution to learn the system	C1
Static and/or dynamic program analysis	
Manually Engineered RPC Requests	

Why "Reintenting the Wheels"

Characteristics for RPC System at Ant Group

- C1 – Dynamic and Unknown RPC Processing Logic
- C2 – Private RPC Responses
- C3 – Deeply customized RPC protocols
  - The constructions of our RPC protocols are highly heterogeneous due to various business cases
  - It becomes increasingly difficult to construct valid RPCs

Prior Arts / Solutions	Adoption Challenges for our Systems
Request-reponse analysis	C2
Probing-based solution to learn the system	C1, C3
Static and/or dynamic program analysis	
Manually Engineered RPC Requests	C3

Why "Reintenting the Wheels"

Characteristics for RPC System at Ant Group

- C1 – Dynamic and Unknown RPC Processing Logic
- C2 – Private RPC Responses
- C3 – Deeply customized RPC protocols
- C4 – Extremely large code footprint
  - Each RPC may involves many "system-services" backed by different code bases maintained by different teams

Prior Arts / Solutions	Adoption Challenges for our Systems
Request-reponse analysis	C2
Probing-based solution to learn the system	C1, C3
Static and/or dynamic program analysis	C4
Manually Engineered RPC Requests	C3

Our Design Principle

The "Live Replay" Principle

- Vulnerability detection should be driven by live and authentic RPC requests in production to fundamentally eliminate the limitations of artificially engineered testing RPCs

Our Design Principle

The "Live Replay" Principle

- Vulnerability detection should be driven by live and authentic RPC requests in production to fundamentally eliminate the limitations of artificially engineered testing RPCs

A Strawman Design

- Step One: Sample a live RPC in production
- Step Two: Replace the user identifier in the original PRC to create a hybrid request
- Step Three: Replay the hybrid request and compare the whether the two requests are handled similarly

Our Design Principle

The "Live Replay" Principle

- Vulnerability detection should be driven by live and authentic RPC calls in production to fundamentally eliminate the limitations of artificially engineered testing RPCs

A Strawman Design

- Step One: Sample a live RPC in production
- Step Two: Replace the user identifier with a *companion user* in the original PRC to create a hybrid request
- Step Three: Replay the hybrid request and compare the whether the two requests are handled similarly

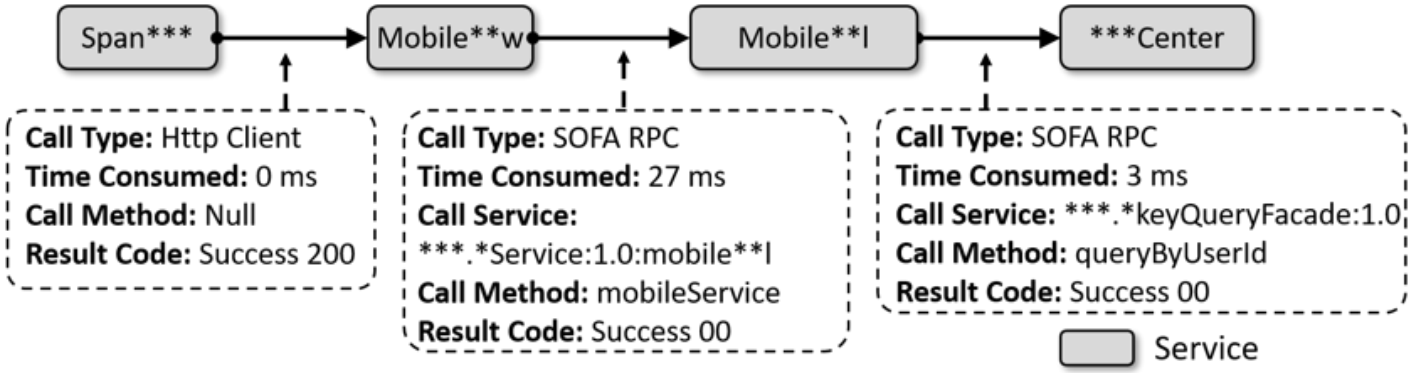
Problems / Challenges

- Lacking the prerequisites of analyzing live RPC requests (recall the privacy requirement)
- RPC is not the right granularity for vulnerability discovery since each RPC has multiple legitimate processing logics (recall the complexity and dynamism of RPC handling)
- It is not possible to find a perfect *companion user* for hybrid request (recall that user attributes impact RPC handling)

PAIR Design

A Privacy-Preserving and Universal Model for RPC Handling

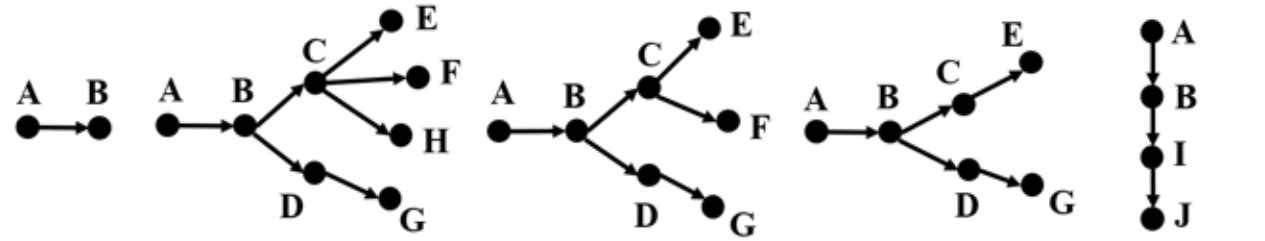
- Modeling the RPC handling logic as a behavioral dependency graph (BDG): a graph of "system-services" invoked when handling a RPC
- Privacy-Preserving (System intrinsic info) + Universal (No Protocol Parsing) + Extensible (System info granularity)
- Build a tracing system to construct BDGs



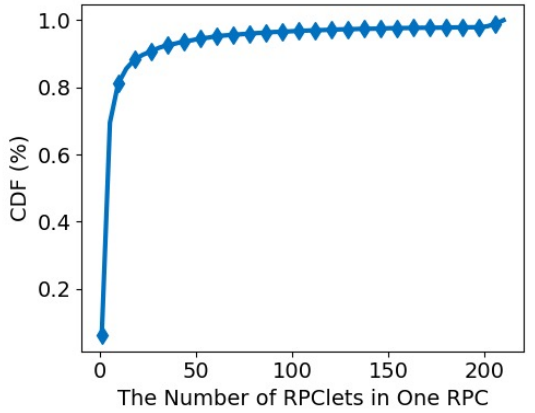
PAIR Design

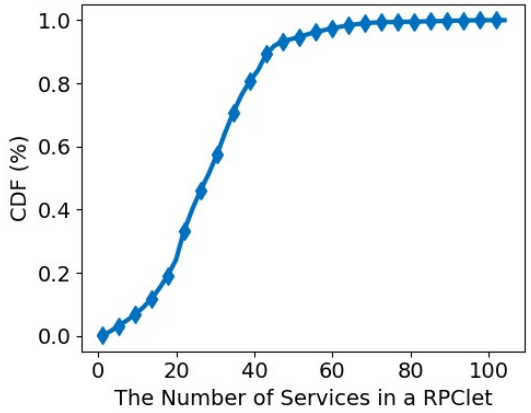
RPClet Construction and Ranking

- Introduce RPClet: one specific handling logic (in form of BDG) for a RPC with certain payloads
  - Learn RPClets: statistical analysis of BDGs of the same RPC to profile the RPClets of the RPC
- RPClet Entropy analysis to decide "High-risk" RPCs



(a) RPClet-1 13.64% (b) RPClet-2 4.09% (c) RPClet-3 39.09% (d) RPClet-4 43.18% (e) Exception

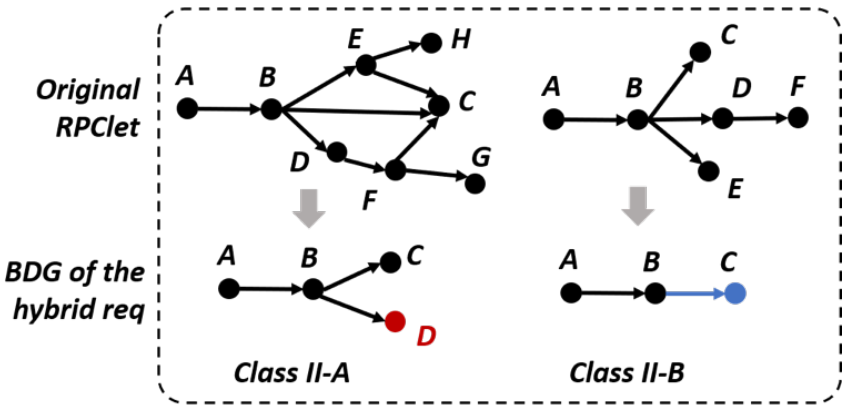




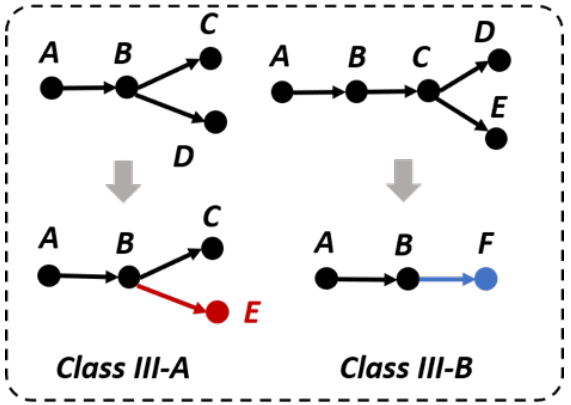
PAIR Design

Empirical Vulnerability Labeling

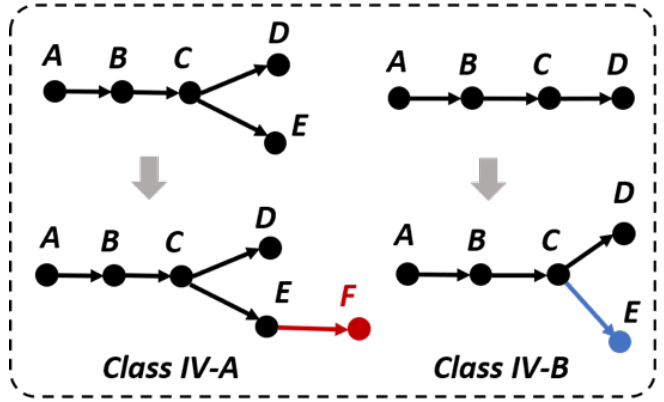
- Absorb the "sad fact" that it is impossible to find a perfect companion user for each sampled RPClet
- Propose to provide data-driven insights into the safety of RPClet by analyzing how our system terminates the hybrid request
  - Intuitively, the BDG of the hybrid request should be terminated by security check nodes



The hybrid request's BDG is a subset of the original RPClet



The hybrid request's BDG has additional termination nodes not appeared in the original RPClet

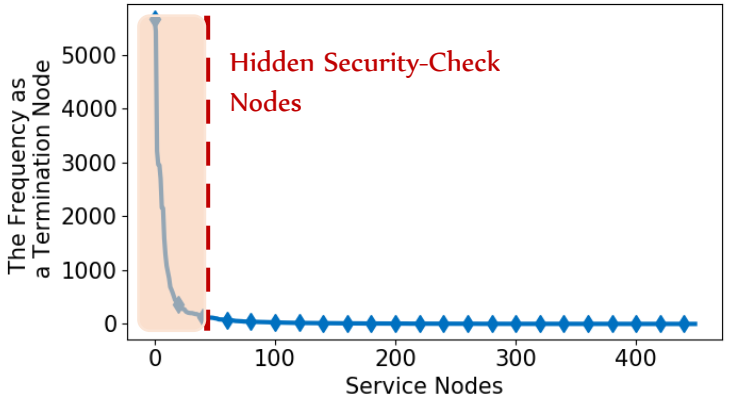


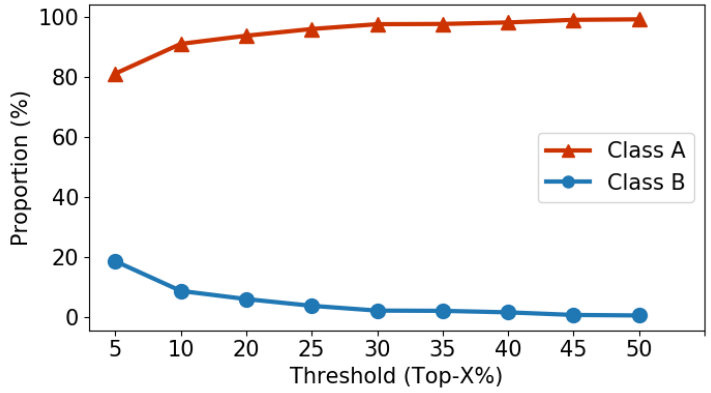
The hybrid request's BDG is a superset of the original RPClet

PAIR Design

Empirical Vulnerability Labeling

- Absorb the "sad fact" that it is impossible to find a perfect companion user for each sampled RPClet
- Propose to provide data-driven insights into the safety of RPClet by analyzing how our system terminates the hybrid request
  - Intuitively, the BDG of the hybrid request should be terminated by security check nodes
  - Problems: only a small number of nodes are explicitly tagged as "security-check" nodes, while many other nodes implement security checks internally





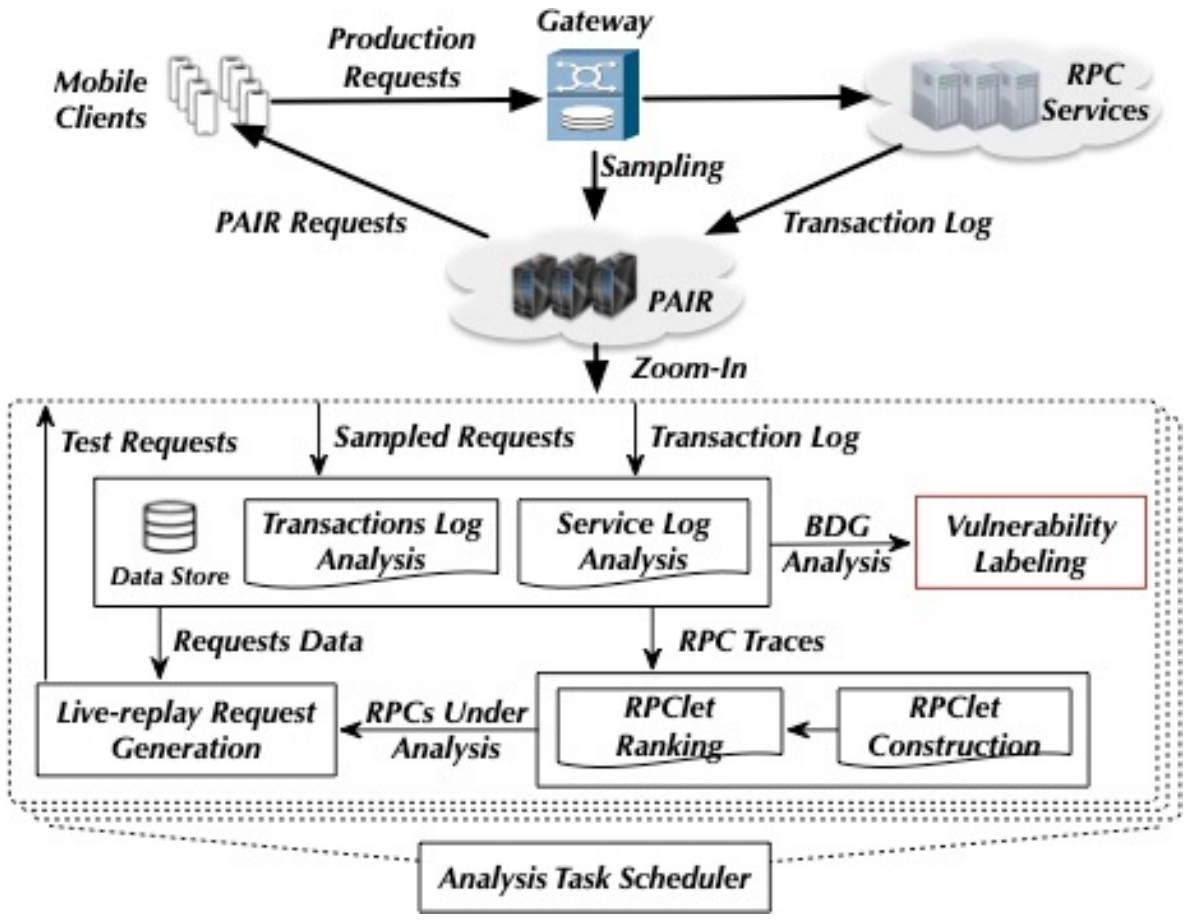
Class A RPCs: at least one of the leaf nodes implement security check logic. Class B RPCs: represents the opposite, meaning high probability of privilege escalation.

PAIR Deployment

System Deployment

- Production deployment for over three years
- We collected data over a course of five days for evaluations

Data Category	Estimated Value
Collected live production RPC requests	500 billion
The number of involved system-services	450
The number of RPC types	10 thousand
Total number of RPClets	600 thousand
Online-analyzed RPClets	110 thousand
Discovered Vulnerabilities	133



Some Results

Total 133 Vulnerabilities

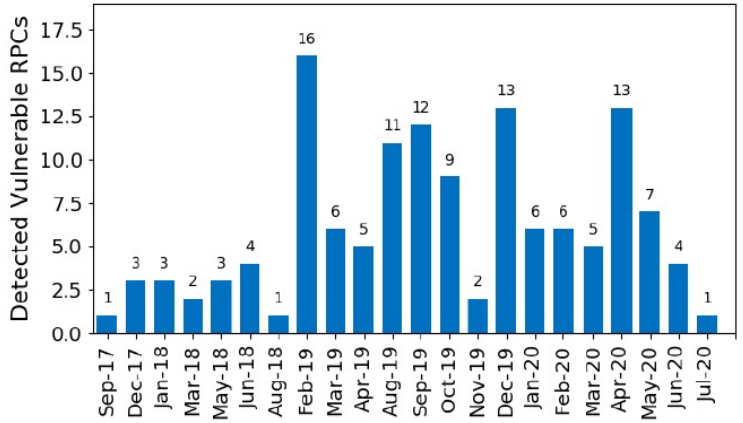


Figure 11: The number of vulnerabilities discovered by PAIR historically.

Table 8: The breakdown of identified vulnerable RPCs by their operation types and the types of targeted information.

Category	①	②	③	④	⑤	⑥	Total
Query	30	7	10	21	15	11	94
Update*	14	2	2	8	10	3	39
Total	44	9	12	29	25	14	133

* Some update operations may also involve querying certain information. We put them in the update category as the major functionality of these RPCs is still updating.

High false positives to trade for zero false negative (by design)!

Query RPCs are more likely to suffer from privilege escalation attacks than write RPCs.

Thank You!

Contact: zhuotaoliu@tsinghua.edu.cn