Mixers: Analysis and Implementation

CS6760 Project Proposal, Jack Robertson

October 11, 2020

Proposal

EXSUM

Our company, JackPay, implements peer-to-peer payments on our centralized platform (the App). JackPay uses a centralized ledger (the Ledger), a centralized network where users are nodes, and all payments pass through a central router.

For this project, we will focus on hardening the App against user privacy concerns presented when payment records are passively monitored, or the Ledger's confidentiality is lost.

We propose implementing a "mixer" to obfuscate the source from the destination of payments. Mixers, in broad terms, aggregate inputs from multiple source nodes. Then, mixer nodes output the inputs to intended destination nodes in a manner that obfuscates the source node when analyzing the destination node, and vice versa.

We will build a product demo of a network-level mixer, **NetMix**, and a ledger-level mixer, **PayMix**. We will conduct theoretical before and after privacy engineering analysis of JackPay: do we more robustly preserve user autonomy and liberty, help prevent discrimination against users and loss of trust in JackPay, and so on.

Current Territory

System

JackPay is an payments application, best conceptualized as a network implemented at the application layer. Users, enterprises, and similar entities are nodes ("Jill," "Kantipur Restaurant"), and they are connected by a centralized

application feature that effectively serves as single router. The router's logic determines how to connect payment source and destinations. A payment is like a single network packet, containing source/destination/amount payment headers. It routed on a direct link from payer to router, and then router to receiver. Payment routing is done in real time, using a user lookup mechanism. We only log payment amounts and time, not source/destination.

JackPay obviously produces numerous threat vectors - do we encrypt data at rest and in transit, do we use conduct phishing training on our employees, is MFA implemented across the org, is... **For this project, we are exclusively focused on analyzing the technical implementation of mixers into our network and ledger, and if/how it improves user payment privacy.**

Assets

Our assets are our users and our network. We seek to protect the privacy of the link between the payer and the payee, in this use case.

JackPay is agnostic towards who can create accounts. As such, user Janet can donate to a variety of political causes that have JackPay accounts. User Robert can pay for products that his culture finds immoral.

Attackers

Multi-national law enforcement may seek to monitor the payments of our users. Know Your Customer/Anti-Money Laundering laws (KYC/AML) operate in this space. We cooperate with warrants as technically and legally able.

Cultural forces in users' communities, families, and nation states may seek to monitor the implied behavior presented by payments, as to infringe on the free practice of those behaviors.

Business and political intelligence analysts may be interested in our payment records, due to implied political and consumer trends based on payment behavior.

Vulnerabilities

Single Router: Our centralized router offers a single point of failure in the network, and aggregator of payment data

Unencrypted Network: Connections from nodes to the router, although within the application, are not encrypted. As of now, we rely on security at the edge, versus zero trust.

Ledger Accuracy: The events on our ledger are currently "as-is:" if A pays B $100 at 10pm ET, $100 at 10pm ET is reflected on the ledger. We currently preserve this state for internal business intelligence reasons and possible sale of anonymized data, but doing so presents trade offs.

Threats

Single Router: Our single router application can be compromised to monitor payment routing, or it presents a single point of network failure.

Unencrypted Network: Attackers can passively monitor the payments network and sniff unencrypted traffic, thus identifying source/destination of payments.

Ledger Accuracy: Users eventually "on-ramp" and "off-ramp" between JackPay and the outside financial system. Using forensic accounting and TTPs such as "chainalysis tools," those two events can lead to de-anonymization of payments through induction when our Ledger is a accurate record of all payments.

Risks:

Importance/Likelihood of Exploit

Single Router: HIGH/MED, failure of router destroys network, compromise of router exposes single aggregation point of payments, and router security relies on secure software engineering capabilities versus more "simple" network security on direct links. However, router is at center of application's network, and locating it takes reconnaissance and time.

Unencrypted Network: MED/HIGH, individual exposure of user payments via individual payment interceptions does not compromise our entire network of payments. Each user's payment uses a dynamic IP address, and it is technically challenging to monitor our entire IP space due to its size (we rival cloud providers). However, if the target user payment is monitored, due to the plaintext payment object, exploit is guaranteed.

Ledger Accuracy: LOW/LOW, relative to other risks. The Ledger currently doesn't serve much business purpose for us other than short term payment disputes, and we could safely dispose of it after 24 hours post-payment. We do not yet sell it for profit. Exploiting the ledger is also lower likelihood. Chainalysis, our term for the procedure previously mentioned, is a complex undertaking, and we currently encrypt the Ledger at rest, whenever it is in transit, and all update communications to it. Applying PayMix is possibly a premium feature we can

offer for JackPay, and a hedge against advances in data science that may harm our customer's privacy.

Proposed Implementation

Theoretical Analysis of Current State

We will conduct a theoretical analysis of our current application's privacy engineering approach, using the many frameworks offered. Our legal team has assured us we are on the right side of all laws in this area. Therefore, we seek to analyze what privacy concerns will successful exploit the previously described risks present? Within the defined project scope of network, router and ledger use, are we conducting any of the 7 problematic data actions? Are we a privacy-friendly system? What key outcomes does JackPay need to achieve?

Theoretical Analysis of Proposed State

We will define the feature set of the two mixers, and conduct the same analysis as before. How would JackPay improve or change, in terms of privacy engineering definitions? We will then implement a product demo of NetMix and PayMix using Python 3 and basic local host routing.

Concepts Involved

NetMix derives technical inspiration from David Chaum, who originated the concept in 1981's Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms, and other follow-on implementations such as Tor. Source nodes send messages into a chain of proxy servers, "mixer nodes." The mixers will send the messages in a different order than received. Therefore, it is more challenging to deduce that since A, B, and C sent messages, and then D, E, and F received messages, therefore A->D, B->E, C->F.

PayMix derives technical inspiration from cryptocurrency tumblers. A conceptual term is "CoinJoin" which appears to be first described in white papers by Gregory Maxwell, a core developer for the Bitcoin protocol. From this source and also David Chaum's writings, many implementations of payment mixers are developed. First, assume each "unit" of currency is discrete. From a source of 1.0 units, .3 and .7 units can be traced back to that source 1.0. Multiple users send payments into a mixer transaction. A needs to send 1 unit to B, and C needs to send 2.4 units to D. So, A will send 1 unit and C will send 2.4 units to Mixer Z: Input user. Mixer Z internally transfers from its Input user to its Output user (different "accounts"). This severs the direct link from A to B, and C to D. Then, Mixer Z can take a variety of approaches. To send the 1 and 2.4 units to B and D, Mixer Z can "blend" A's units with C's units (assuming the units' parts are traceable), or send .5 and then .5 to B and 2.0 and 0.4 to D, and so on.

Privacy engineering theoretical analysis will draw from the various frameworks and schools of thought in the field. Will draw casual observations on JackPay, within the scope of the project, using OECD Guidelines, FIPPs, ISO 2700x, GAPP, Warren and Brandeis (1890), Bambaur (2012), etc. We will commit to analyzing how JackPay measures up to the "Solove Six" definitions of privacy from Daniel Solove (2008), before and after the mixers.

Cryptography and network protocols TCP/IP protocols are also inherently involved, although we will not seek to improve upon the two in this project and describe only as needed for understanding the technical concepts, i.e. "here, cryptography obscures packet contents on the network, although if it was weaker cryptography...," or "we use IP addresses to designate host X, following this format..."

Expected Benefits

Conducting a theoretical analysis of JackPay before and after mixer implementation will ensure JackPay's fidelity to privacy engineering concepts is analyzed, and show if the proposed mixer aids or hinders those efforts. If the mixer hinders the efforts, we will know not to pursue the concept in the near future. The analysis, if made available to customers, will have trust and safety benefits as well as it shows we continually analyze ways to harden our application and protect users' privacy.

The technical demos of NetMix and PayMix, although almost certainly lacking in the security necessary for production deployment, nevertheless will help model the basis and benefits for such use. By expanding our payment to involve multiple mixer routers between the source and destination hosts and using a mixing protocol we will obscure the links between senders and receivers of payments. Encrypting the payment objects over the network, assuming properly strong encryption, will obfuscate payments over the network from passive network monitoring. By altering the payment records stored on the ledger through the use of a mixer entity, ability to forensically analyze our ledger is decreased.

Expected Drawbacks

The mixer network will almost certainly introduce latency to our payment network. Analysis of privacy state may produce unfavorable results that we would like to otherwise not have on record both internally or externally. PayMix may attract regulatory attention over KYC/AML concerns if we chose not to keep mixer logs. NetMix's encryption across multiple nodes presents key derivation and key rotation technical complexity. Multiple mixer nodes likely requires increased technical overhead in the form of a software defined network, in order to manage it at scale. Mixer nodes must all carry the same application routing logic, and introduces DevOps complexity. Our on-staff mathematicians may not

be up to the task to safely determine the privacy vs. latency trade offs based on factors such as the relationships between number of users using PayMix, number of nodes in NetMix, and difficulty in tracing by induction source/destination payment links.

Technical Implementation of Mixer

NetMix will at a minimum demonstrate 3 source nodes sending encrypted messages across 3 mixer nodes to 3 destination nodes, following the "mixing protocol," and recorded on a temporal ledger. This will be done on localhost first (multiple terminals), with pre-loaded encryption keys and existing Python encryption libraries. Possible improvements we will attempt if time and logic allows: Diffie Hellman key derivation and distribution, fail-over routing methods for node failures.

PayMix will at a minimum demonstrate a single transaction recorded on a temporal ledger (Python list of transaction objects), involving 3 senders, 3 receivers, Mixer:In, and Mixer:Out entities. The mixer will hold custom application logic that implements the various methodologies a mixer can employ (time staggering, 1 transaction morphs multiple smaller transactions, etc.)

We will not do the following: mathematical analysis to determine the relationship between the minimum number of mixer nodes, payment inputs, etc. required to truly obscure sources from destinations. If during literature review we find solutions for the mathematics, we will reference them. We will not blend NetMix and PayMix into a single implementation. We will not seek to move beyond a localhost implementation. We make no guarantees about secure software development approaches in this demo. This is not an inclusive list.

Sources

Understanding Privacy, Daniel Solove (2008)

The Right to Privacy, Samuel Warren and Louis Brandeis (1890)

The New Intrusion, Jane Bambauer (2012)

State of CoinJoin, Gregory Maxwell https://en.bitcoin.it/wiki/User:Gmaxwell/state_of_coinjoin

Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms, David Chaum (1981)

The Privacy Engineer's Manifesto, Dennedy, Fox, Finneran (2014)

Sprint Planning

Product Backlog: All Tasks

Lit Review

- Solove: deep read
- Warren/Brandeis: deep read
- Bambauer: deep read
- Maxwell: skim
- Chaum: deep read
- Dennedy/Fox/Finneran: skim
- NIST etc: survey the related NIST docs but don't dive too deeply

App Basics

- Detail list of system specs: what does the app specifically do and how does it do it. Make sure to manage scale here, this isn't the point of the project necessarily. Don't write any code here

NetMix

- create ledger entity (source, destination, amount sent, time)
- create user nodes (serves as source and destination)
- create very basic node<>node socket connection logic
- Create basic routing logic (have a roster of users, read packet, send to destination user, ensure the ledger reflects it)
- Add mixing logic (queue of packs, reorder, send, ensure ledger reflects)
- add encryption (hard code keys into router application, each mixer node holds this, ID crypto lib to use)
- tests

PayMix

- create ledger entity
- create mixer application (split big payment into smaller payments)
- mixer application: time stagger (split big payments into smaller payments, stagger time)
- tests

Theory

- conduct JackPay privacy engineering analysis: pre-mixer
- conduct JackPay privacy engineering analysis: post-mixer

Admin

- Make repo on Github
- Make Readme
- Add proposal to repo
- push final code to repo
- try some limited threat modeling on the app I made - how can I break them?

Sprint Backlog: 4 Weeks

Week 1

- Make repo on Github
- Make Readme
- Add proposal to repo
- Solove: deep read
- Warren/Brandeis: deep read
- Bambauer: deep read
- Maxwell: skim
- Chaum: deep read
- Dennedy/Fox/Finneran: skim

- NIST etc: survey the related NIST docs but don't dive too deeply
- Detail list of system specs: what does the app specifically do and how does it do it. Make sure to manage scale here, this isn't the point of the project necessarily. Don't write any code here

Week 2

- create ledger entity (source, destination, amount sent, time)
- create user nodes (serves as source and destination)
- create very basic node<>node socket connection logic
- Create basic routing logic (have a roster of users, read packet, send to destination user, ensure the ledger reflects it)
- create basic mixer application (split big payment into smaller payments)

Week 3

- Add mixing logic (queue of packs, reorder, send, ensure ledger reflects)
- add encryption (hard code keys into router application, each mixer node holds this, ID crypto lib to use)
- tests
- PLACEHOLDER: further improvements on Week 2 work

Week 4

- conduct JackPay privacy engineering analysis: pre-mixer
- conduct JackPay privacy engineering analysis: post-mixer
- Attempt DH key exchange on network
- final cleanup