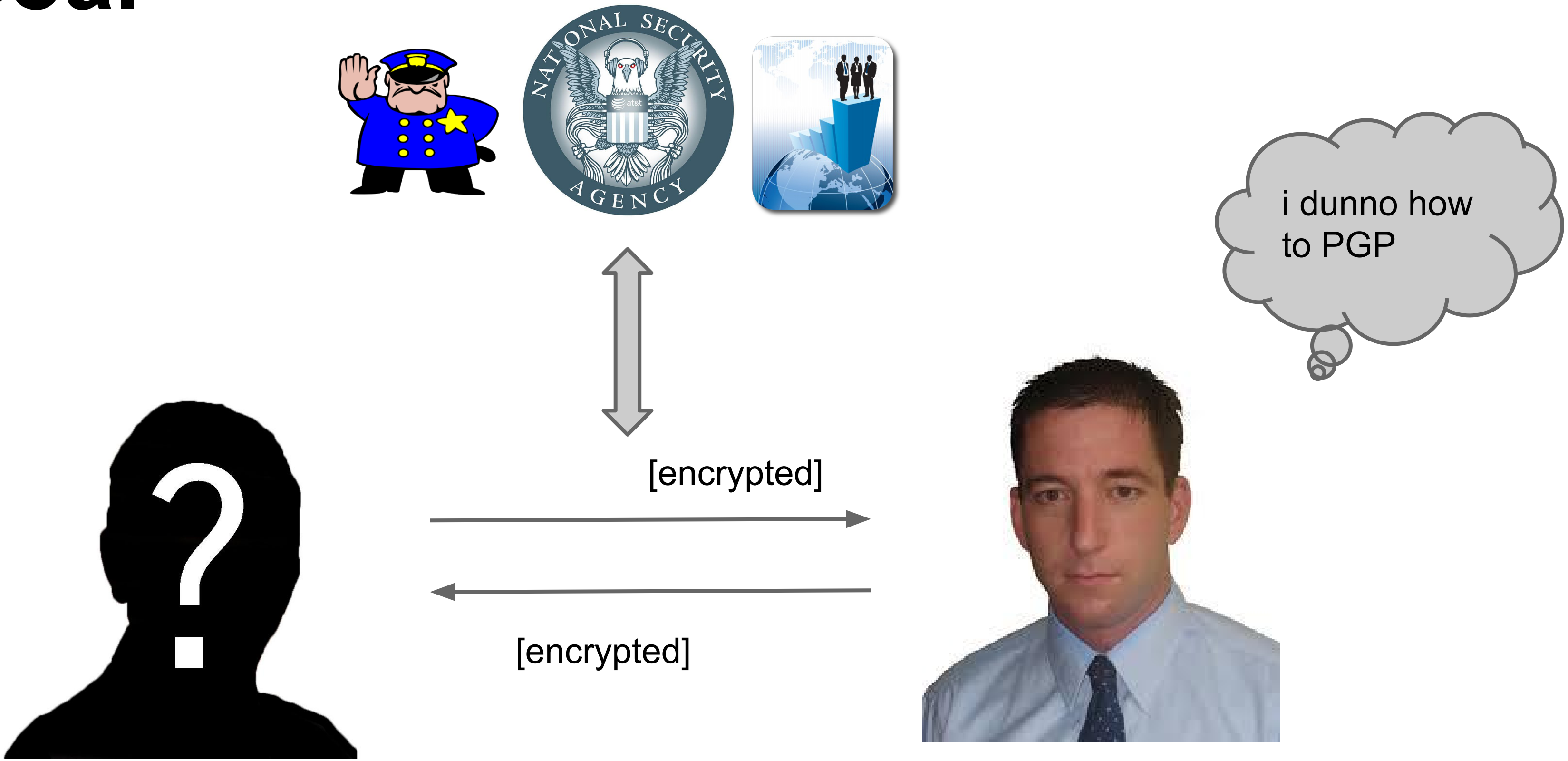
SECUREDROP and beyond

Garrett Robinson and Yan Zhu

Real World Crypto 2015

Goal


NATIONAL SECURITY AGENCY
i dunno how to PGP
[encrypted]
[encrypted]

Who uses it?

THE // INTERCEPT

theguardian

THE NEW YORKER

GREENPEACE

balkanleaks
The Balkans are not keeping secrets anymore
Powered by SECUREDROP

NRK beta

EXPOSE FACTS

The Washington Post

PROPUBLICA

THE CENTURY FOUNDATION

POGO PROJECT ON GOVERNMENT OVERSIGHT

Radio24syv

https://freedom.press/securedrop/directory

Threat Model

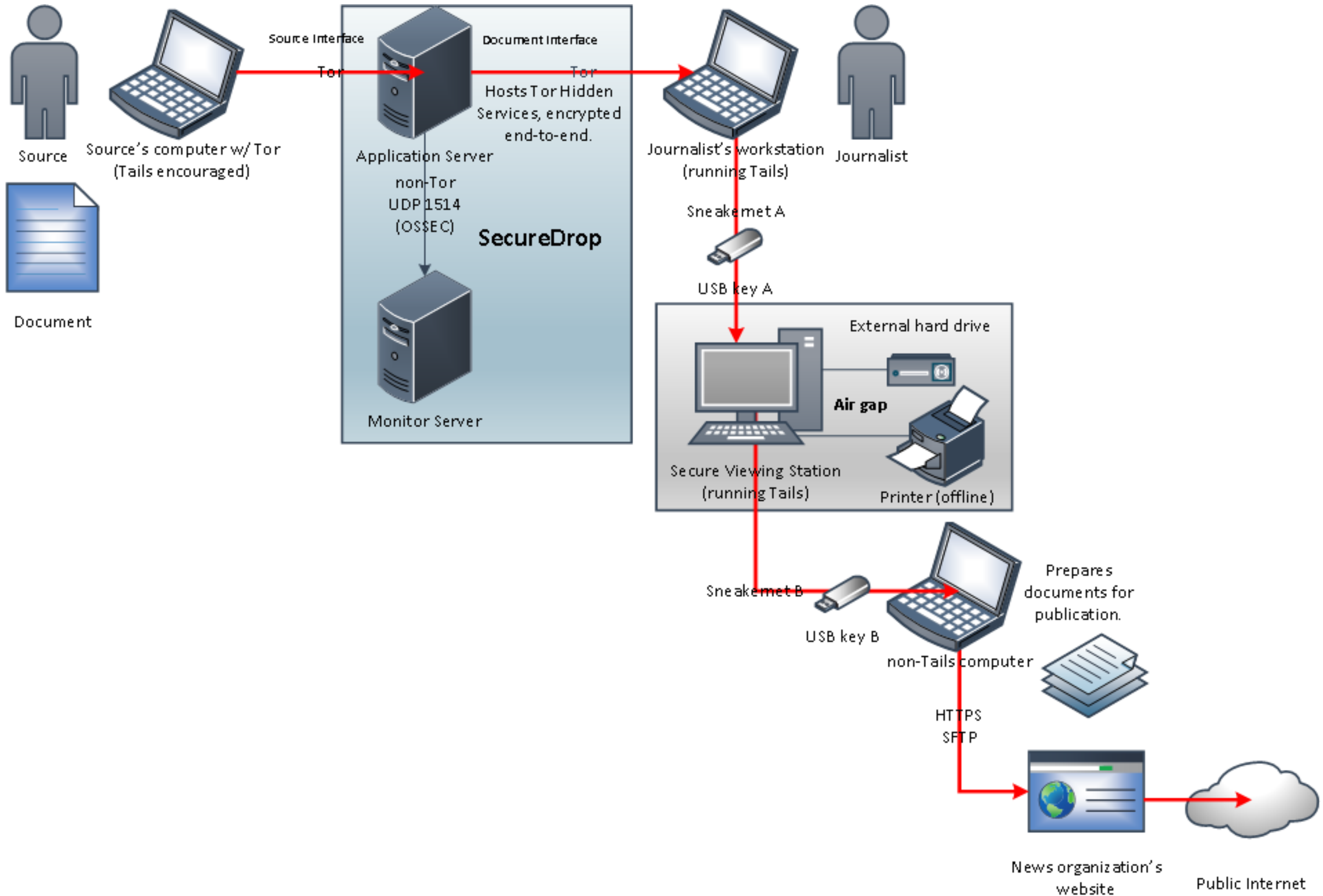
Assets

- Source's identity
- Confidentiality & integrity of submissions
- Confidentiality, authenticity, & integrity of messages between source and journalists

Threat Model

Adversary

- Active network attacker
- Could seize server
- Could seize and search devices of suspected sources


Source Interface
Tor
Document Interface
Tor
Hosts Tor Hidden Services, encrypted end-to-end.
Source
Source's computer w/ Tor (Tails encouraged)
Application Server
Journalist's workstation (running Tails)
Journalist
non-Tor UDP 1514 (OSSEC)
SecureDrop
Document
Sneakernet A
USB key A
Monitor Server
External hard drive
Air gap
Secure Viewing Station (running Tails)
Printer (offline)
Sneakernet B
USB key B
Prepares documents for publication.
non-Tails computer
HTTPS
SFTP
News organization's website
Public Internet

Demo

Desired properties for SD 1.0

- Forensic deniability for sources
- Resilience against SD server compromise
- Flexible client model
- Usability for everyone
- Leverage existing tools


CHALLENGES
AHEAD

1. End-to-end encryption

Why end-to-end?

- Reduce potential harm of server compromise
- Simplifies server implementation, reducing attack surface
- Defense in depth

Challenge

- Inherent conflict with **forensic deniability**
- Where do we store the key?

Solution #1

- Generate key in the client
- Encrypt the key with a secret (e.g. passphrase)
- Store the encrypted key on the server
- **Problem**: adversary who gains copy of encrypted private key can try to guess the passphrase

Improvements

- Idea: require “strong” passphrases
  - Use entropy estimator such as Dropbox’s zxcvbn
  - h/t to Minilock
- Idea: auto-generate strong passphrase
  - e.g. Diceware passphrases
  - 8-word Diceware: 104 bits of entropy
- Idea: increase resistance to guessing
  - scrypt

Tradeoff

- Want to minimize cognitive load on sources
- We reuse a single token, the *codename*, as a username and a passphrase
- Makes salting tricky
    - Want to salt to prevent precomputation
    - Need salt to hash (scrypt) to create authenticator
    - Need authenticator to know which salt to use (in a typical random-salt-per-user system)

Proposal

- Use a unique per-instance salt, or a *pepper*
- All SecureDrop instances are independent and noninteroperable
- Server must be compromised to even start precomputing
- Effort must be repeated for each server

Setup flow

1. Generate keypair (sk, pk) on <u>client</u>
2. Fetch salt s from <u>server</u>
3. Stretch human-memorable passphrase (codename) with function S: $S(p, s) \rightarrow p'$
4. Create authenticator a with any secure hash function H: $H(p') = a$
5. Encrypt private key pk with stretched passphrase p': $E(p', pk) = c$
6. Store on server: c, a, pk

Signing in

1. Authenticate
   a. Fetch salt s from server
   b. Derive authenticator: $H(S(s, p)) = a$
   c. Send a to server. If a matches, server returns c.
2. Decrypt private key on <u>client</u>
   a. $D(S(s, p), c) = sk$
3. Can now decrypt messages on client, sign submissions, etc.

Solution #2

- Derive the key from the passphrase
- Inspired by Nadim Kobeissi's minilock

Solution #2

human-memorable secret *s*

↓

scrypt(*s*)

↓

sha256

↓

twiddle bits

↓

curve25519_public

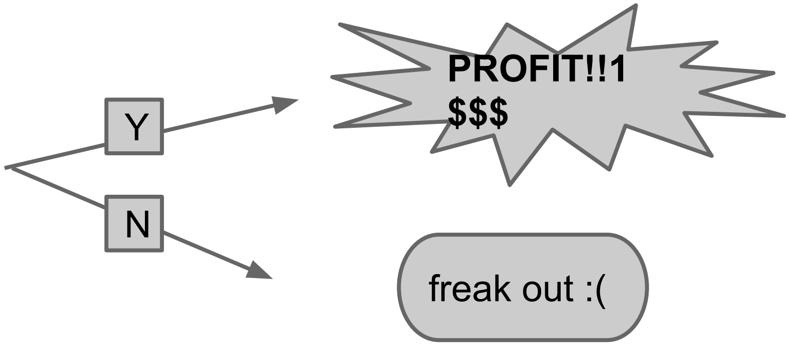
Pros/Cons

- Similar security properties
  - Both require adversary to compromise server
  - Very similar difficulty in guessing passphrase
- Solution #1 **pro**: can use any public key cryptosystem
- Solution #2 **pro**: neat!

2. Secure code delivery

Server code delivery

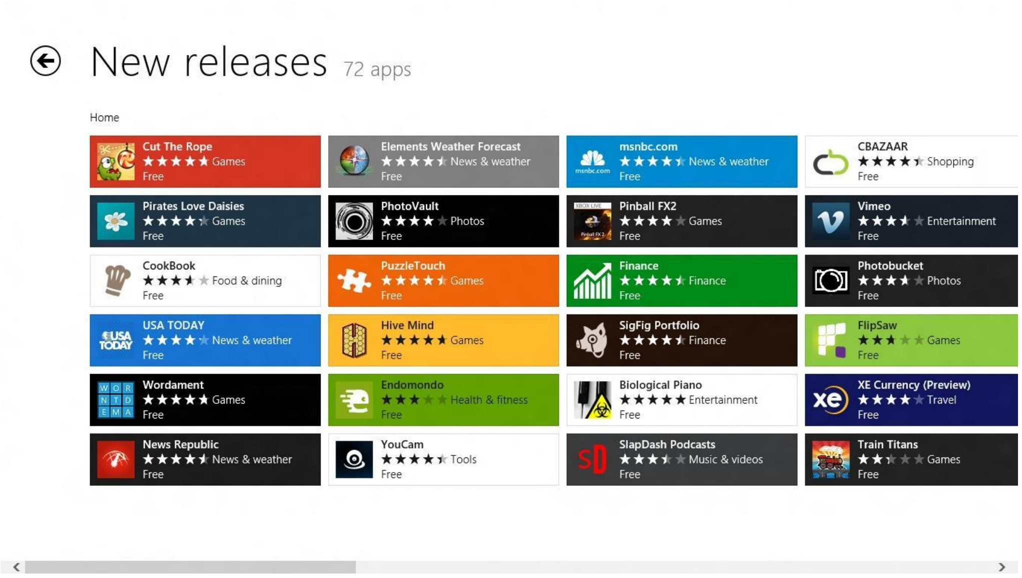
```
$ gpg --keyserver pool.sks-keyservers.net --recv-key
B89A29DB2128160B8E4B1B4CBADDE0C7FC9F6818
$ gpg --fingerprint
B89A29DB2128160B8E4B1B4CBADDE0C7FC9F6818

$ git clone https://github.com/freedomofpress/securedrop.git

$ git checkout 0.3

$ git tag -v 0.3
```



Client code delivery

not as easy . . .

HTTPS:// (or Tor Hidden Service)



Option 1: regular Tor Hidden Service website (strawman)

- No software installation beyond TBB
- Good forensic deniability
- Poor sandboxing
- No code signing
- Hard to detect backdoors
- **Grade: D**

Why not make the web platform safe for crypto?

- Lots of recent progress here (Content Security Policy, WebCrypto API)
- Example: use Service Workers to “trust” code on first use
- Limitations: slow standardization process + TBB is ESR Firefox :(

Option 2a: TBB extension for secure messaging in general

- Good forensic deniability, especially if included in TBB by default
- Better sandboxing than a normal web page
- “prollyfill” for future web standards
- Can be compromised by another malicious installed extension
- Need to support many use cases
- **Grade: A**

Option 2b: TBB extension for SecureDrop only

- Only support one use case
- Better sandboxing than a normal web page
- Can be compromised by another malicious installed extension
- Low chance of getting into TBB by default; otherwise poor forensic deniability
- **Grade: C**

Option 3: Native desktop client

- Much smaller attack surface than a browser
- Poor forensic deniability (unless included in TAILS, etc.)
- Need to support multiple platforms
- Loneliness
- **Grade: B**

Package managers protect us from:

- MITMs
- Malware pretending to be legit . . . or not

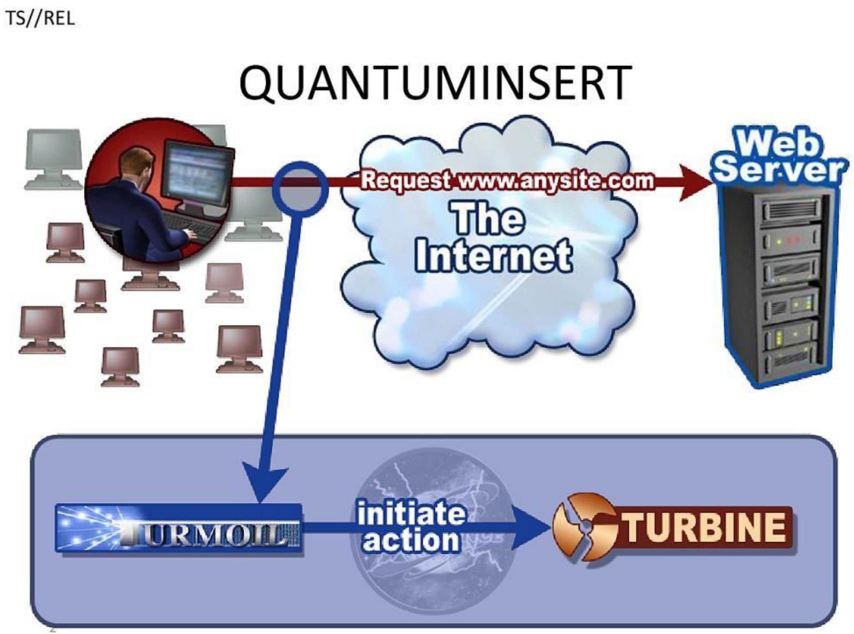
CNET › Tech Industry › Researchers slip malware onto Apple's App Store, again

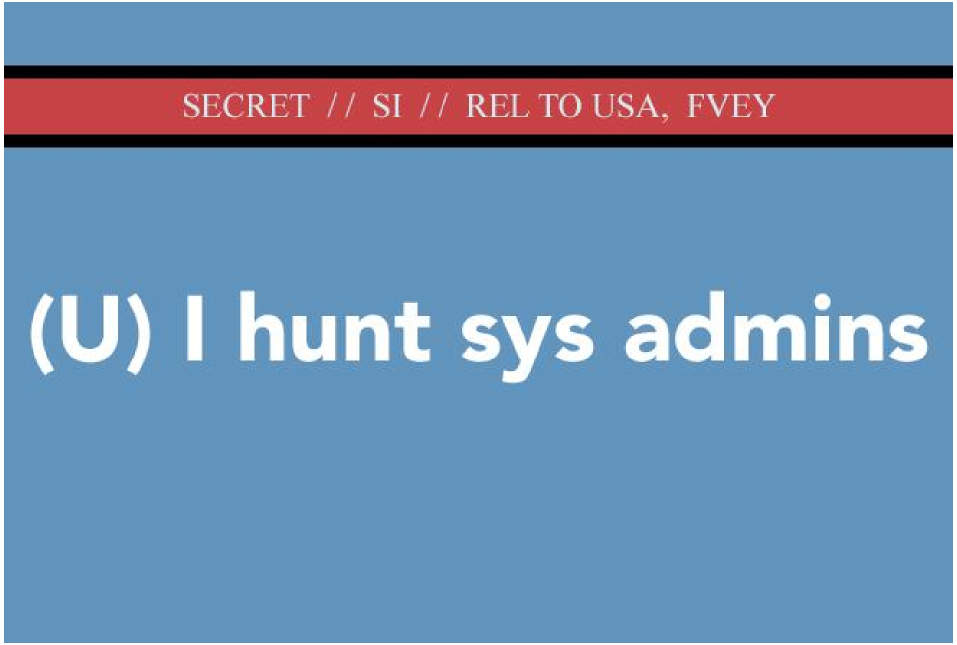
Researchers slip malware onto Apple's App Store, again

Georgia Tech security researchers this week noted they managed to successfully slip some malware onto the App Store in May.

by Josh Lowensohn @Josh / August 16, 2013 2:56 PM PDT

Package managers should protect us from:





Package managers need code transparency

Two guarantees against backdoors:

1. Package that Alice installs is the same as package that everyone else installs.
2. Code that Alice runs corresponds to the publicly available source code.

Solutions

- Put all package hashes into a public append-only log, which client checks before installing. (“Binary transparency”)
- Implement reproducible build process
- [Your ideas here]

Questions?

- https://freedom.press/securedrop
- Twitter:
  - @FreedomofPress
  - @bcrypt
  - @garrettr_
- We’re hiring! https://freedom.press/jobs