Onion services

Philipp Winter
pwinter@cs.princeton.edu

Nov 30, 2015

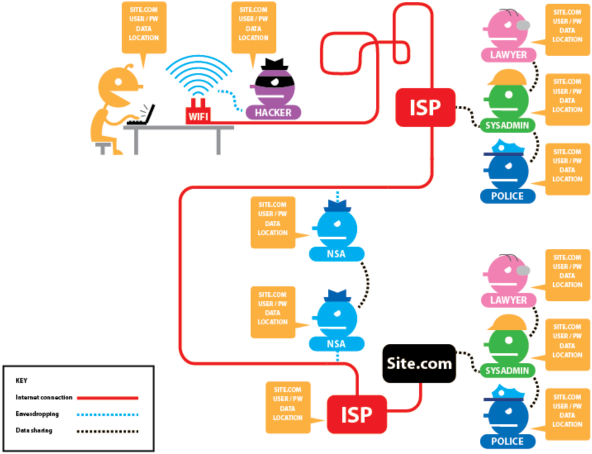
Quick introduction to Tor

TorProject.org

An overview of Tor

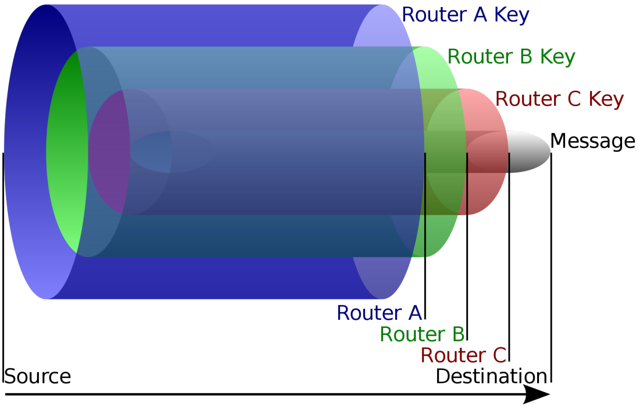
- Tor is a **low-latency** anonymity network
  - Based on Syverson's onion routing...
  - ...which is based on Chaum's mix nets
  - Network consists of $\sim$7,000 relays
- Transports TCP streams (and DNS A records)
  - Any TCP-based protocol can be run over Tor
  - Not always a good idea (e.g., Bitcoin, BitTorrent)
  - TCP over TCP bad for performance
- Decouples **who you are** from **what you do**
  - Entry guard knows who you are
  - Exit relays knows what you do

What does an attacker see?



Source: https://www.eff.org/pages/tor-and-https

The idea behind onion routing



Source: https://en.wikipedia.org/wiki/Onion_routing

Bootstrapping

- Relays are listed in **consensus**, which is published by **directory authorities**
  - Currently ∼7,000 relays in consensus
  - Eight directory authorities
  - Consensus voted on and signed by authorities
- Directory authorities and their keys are **hard-coded** in code
  - Directory authorities rarely change
  - Operators well known to Tor developers

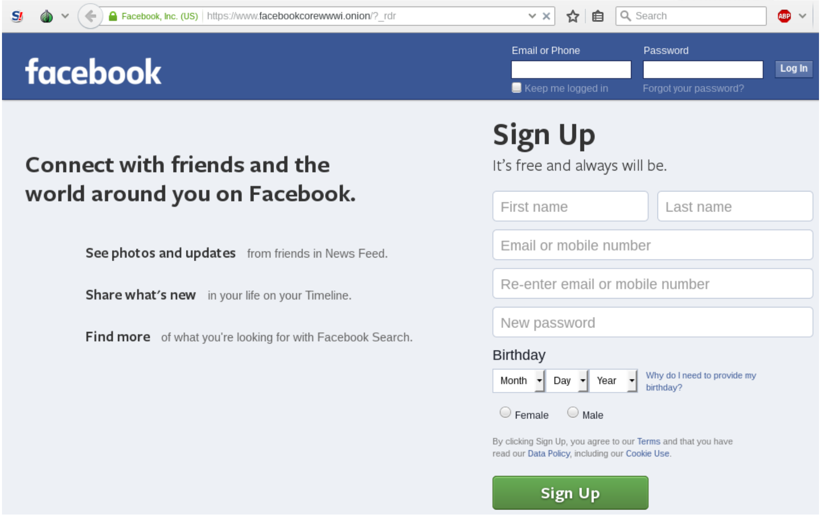
```
r Karlstad0 m5TNC3uAV+ryG6fwI7ehyMqc5kU OgDQHa7kI07jhA/6wtD8gOnZw+4 \
  2015-11-29 19:03:19 193.11.166.194 9000 80
s Fast Guard HSDir Running Stable V2Dir Valid
v Tor 0.2.6.10
w Bandwidth=4270
p reject 1-65535
```

Onion services

In a nutshell

- Most people know Tor for **sender anonymity**
  - Server doesn't know client's IP address
- Onion services add **responder anonymity**
  - Client doesn't know server's IP address
  - Run arbitrary TCP service without revealing location
  - Sender and responder anonymity can be coupled
- **Anonymous** clients can communicate with **anonymous** servers without ever leaving the Tor network
- In addition: limited **DoS** and **censorship** protection
- No protection against deanonymisation on the application layer

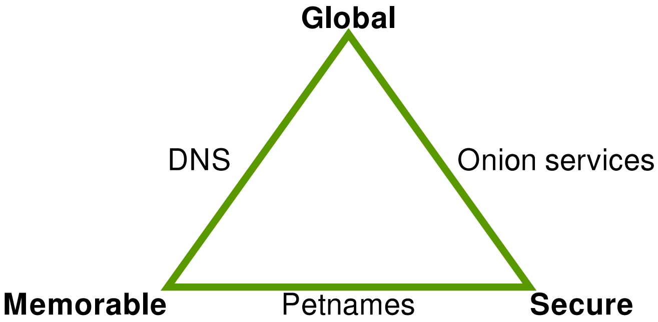
Onion services based on `.onion` pseudo TLD



How are onion services used in practice?

- Many providers now offer it as **alternative**
  - Facebook
  - DuckDuckGo search
  - Many Bitcoin sites
- Metadata-free chat services built on top (Ricochet, pond)
- According to statistics, $\sim$30,000 deployed services [1]
- Details about content not known because of **crawling-resistance**

Zooko's triangle



Source: http://zooko.com/distnames.html

Onion services by example: Bob

- Bob is a **journalist** who wants to publish **sensitive information**
- He wants to publish his articles **anonymously** and without getting **censored**
  - His adversaries shouldn't be able to take offline his server
- So Bob decides to set up a **onion service** (OS) in the Tor network
- There are six steps, from announcing the OS to using it

Step 0: Installation and configuration

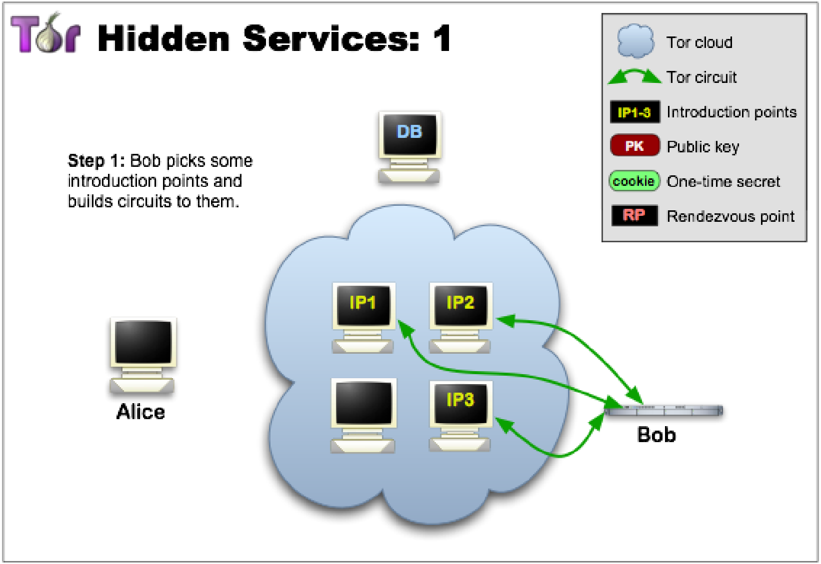
- Before Bob starts using Tor, he has to **install** the service
- So Bob sets up his own lighttpd **web server** which is **not** accessible over the Internet, i.e., it is bound to `127.0.0.1:80` instead of `0.0.0.0:80`
- Also, Bob downloads the Tor binary and **configures** the onion service:

```
HiddenServiceDir /path/to/directory/
HiddenServicePort 80
```

Step 1: Announcing existence

- Bob's OS needs to **advertise** its existence in the Tor network
- The OS randomly picks **relays**, so called **introduction points**, in the network and establishes **circuits** to them
- Then, the OS asks these relays to act as introduction points by giving them its **public key**

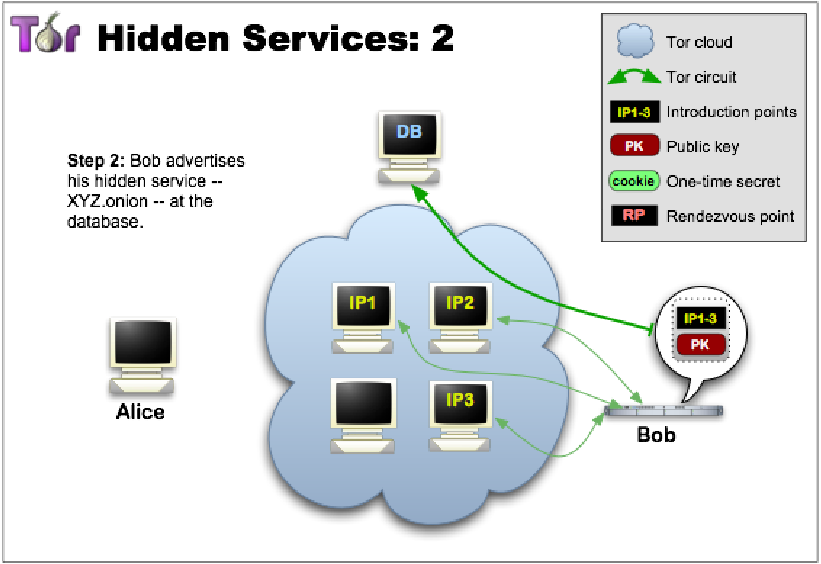
Step 1: Announcing existence



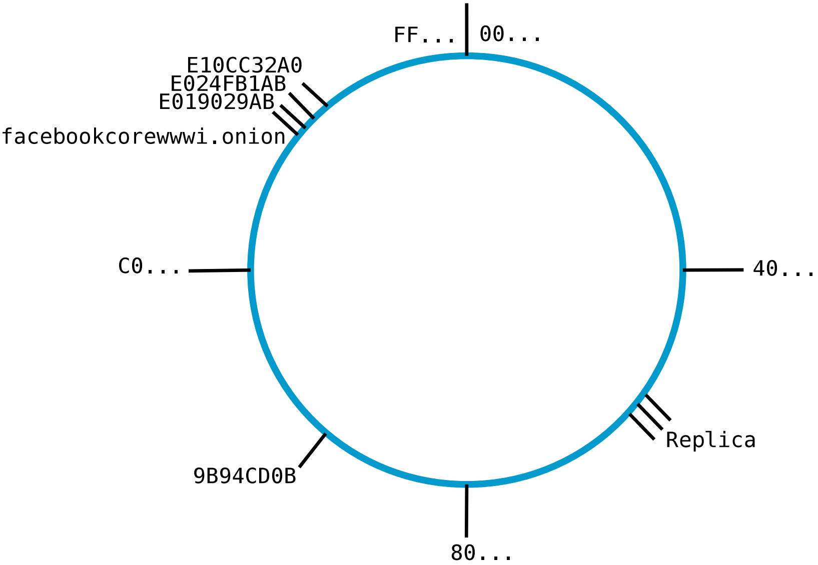
Step 2: Upload of onion service descriptor

- Now, an **onion service descriptor** must be built
  - $\text{descriptor} \mapsto (\text{PK}_{hs}, \text{IP}_1, \text{IP}_2, ..., \text{IP}_n)_{\text{Sig}_{\text{PK}_{hs}}}$
- The descriptor maps the **name** of an OS to its **reachability** information
- It is uploaded to six Tor relays that serve as **onion service directories**
- Clients reach the OS by accessing KEY.onion; KEY is derived from the OS' public key
  - $\text{Base32}(\text{SHA-1}(\text{public key})[:10])$
- Now, the OS is **set up** and ready to receive connections!

Step 2: Upload of onion service descriptor



Tor's distributed hash table

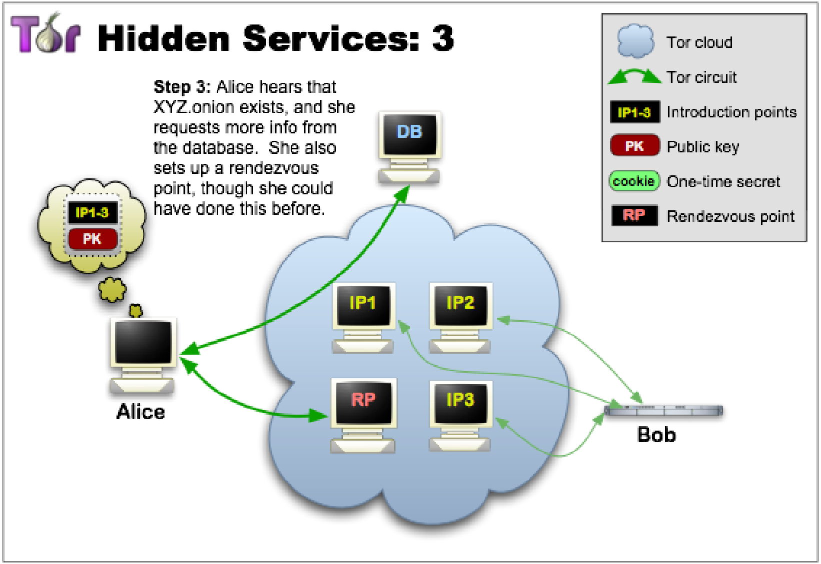


As of Nov 28: 2,884 out of 6,773 relays (43%) are HSDirs

Step 3: Alice prepares a connection

- Alice now wants to **connect** to Bob's OS to read his **articles**
- Alice somehow learns about the onion address `http://bjt5zk37w27c6fy2.onion` out-of-band since there is no complete central directory by design.
- Alice's client **downloads the service descriptor** from the onion service directory
  - SHA-1(permanent-id|SHA-1(time-period|descriptor-cookie|replica))
- That way she obtained the **public key** and the **introductory points**!
- Finally, Alice randomly picks a **rendezvous point**

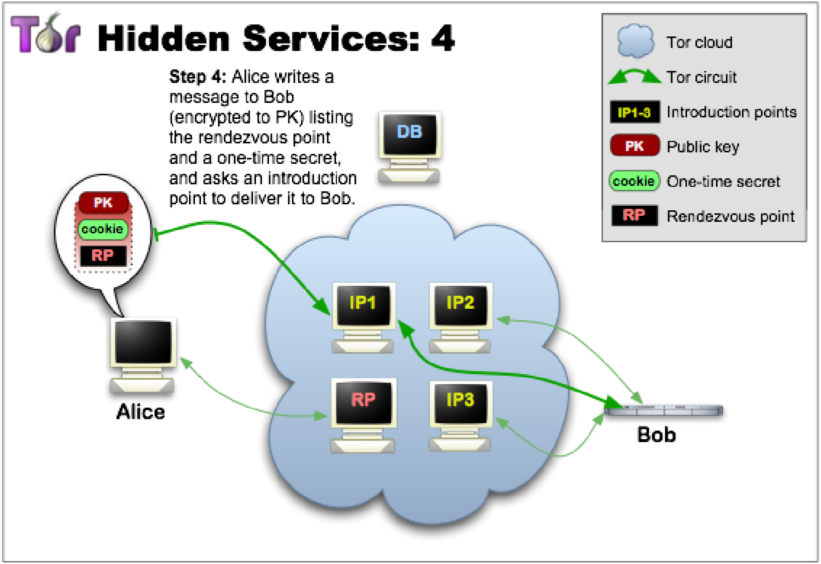
Step 3: Alice prepares a connection



Step 4: Alice informs the onion service

- Now Alice's client prepares an **introduce** message encrypted with OS' public key
- The message contains the **address** of the **rendezvous point** and a **one-time secret**
- Alice sends this message to one of OS' **introductory points** and they **forward** it to the OS
- Alice does all this over a Tor circuit so she remains **anonymous**

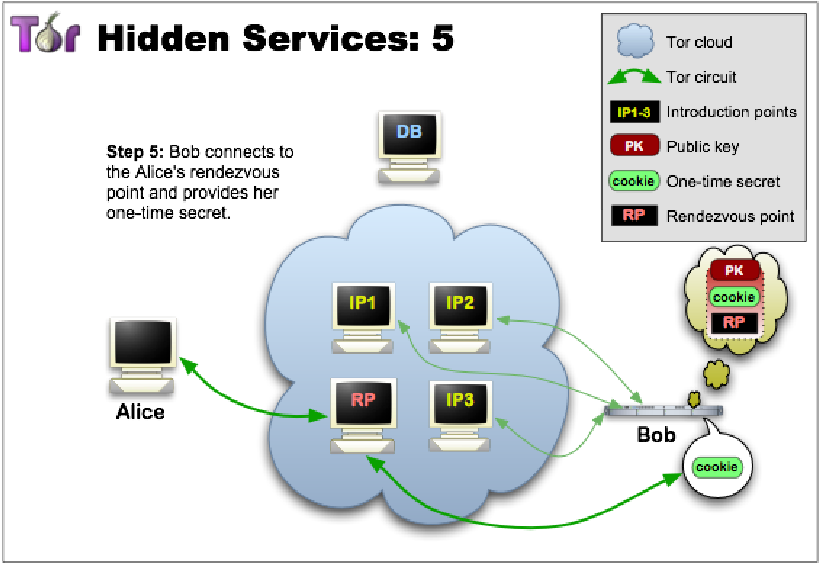
Step 4: Alice informs the onion service



Step 5: The onion service prepares a connection

- The OS decrypts Alice's introduce message and obtains the **rendezvous point's address** as well as the **one-time secret**
- The OS creates a **circuit** to the rendezvous point and sends the **secret** to it

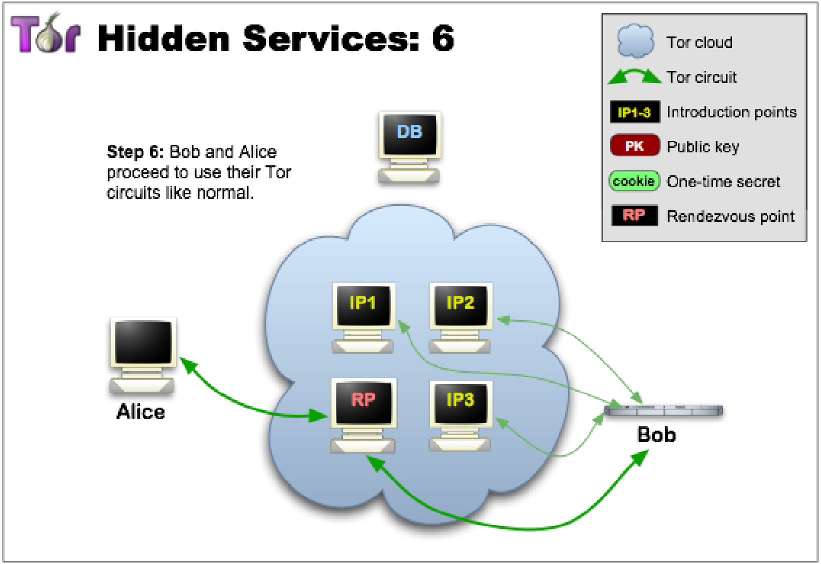
Step 5: The onion service prepares a connection



Step 6: The connection is established

- Finally, the rendezvous point **notifies** Alice of the successful connection
- The rendezvous point now simply **forwards** end-to-end encrypted data between Alice and the OS

Step 6: The connection is established



Why rendezvous points?

- Introduction points only forward **connection information** and no actual traffic
- So they don't seem to be “responsible” for a onion service
- Also, the traffic load could become **too high** if they would also forward traffic

What the involved parties know

The Client...

- Does not know the location of the OS
- Knows the location of the rendezvous point

The rendezvous point...

- Does not know the location of both, the OS and the client
- Knows nothing about the nature of the OS or the data being transferred, other than its volume

The onion service...

- Does not know the location of the client
- Knows the location of the rendezvous point

The onion service directories...

- Knows the name of the onion service
- Knows how often (anonymous) clients request the onion service

A more practical point of view

How Bob operates his OS...

- Bob runs lighttpd which is listening to localhost:80 and is hence **unreachable** to the wide Internet
- lighttpd is **not aware** of the fact that it is used as Onion service!
- The Tor process running on the same machine is accepting connections to the OS and **forwards** them to localhost:80
- The client application can **also** be **unaware** of Tor if it is used together with torsocks (e.g. `torsocks ssh u73zzkakuscok7zq.onion`)
- So client and server could be communicating completely **anonymous** over Tor without even **knowing**

Attacks on onion services

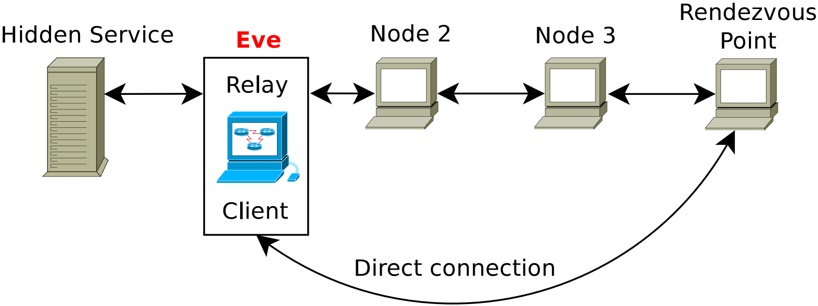
First attack: Øverlier & Syverson

- In 2006, Øverlier and Syverson demonstrated how the **location** (i.e. IP address) of an OS can be **revealed**
- Attacker only needed a Tor **client** and a **relay** (trivial requirements) and the attack could work within minutes
- **Core vulnerability**: OS chose relays for its circuit at **random**
- **Goal of attacker**: Get chosen by OS as the **first hop** in the circuit

Øverlier & Syverson: How it works in practice

- Eve uses her Tor **client** to connect to the OS and she also runs a **relay**
- Eve continuously establishes connections to the OS and checks every time whether her relay was selected as first hop in the circuit $OS \rightarrow RP$
- As soon as her relay was chosen by the OS as first hop, she has the IP address!
- She can confirm whether her relay was selected by doing **traffic pattern analysis** using statistics
- **Solution**: Guard nodes for OSes were proposed and implemented

Øverlier & Syverson: Visualized



Second attack: Murdoch

First we have to know...

- Computing devices have a so called **clock skew**, the ratio between the computer's actual and the nominal clock frequency
- So after x days, a computer's clock drifted off by y milliseconds
- Clock skew is a **very small** value but can even be **measured** over a network
- Computer's (even identical models) have **different** clock skews because the manufactory process is not perfectly accurate $\rightarrow$ the clock skew can be seen as a **hardware fingerprint**

Second attack: Murdoch

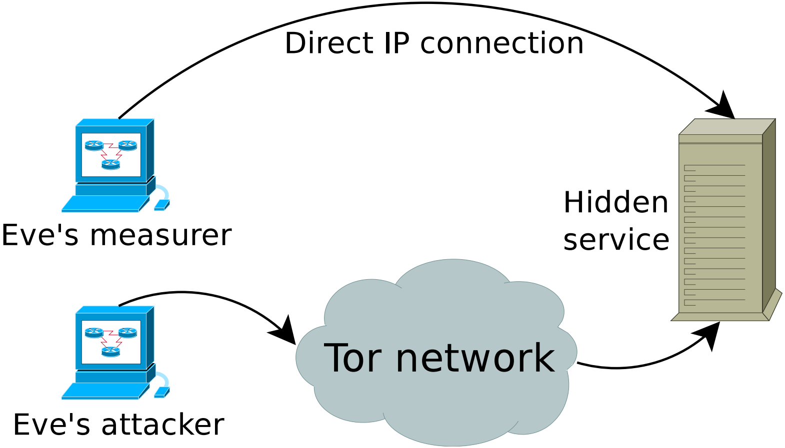
Clock skew and CPU load...

- Clock skew **changes** with temperature of the CPU (differences of 1–1.5°C or 1.8–2.7°F are measurable)
- The CPU's temperature can be influenced by controlling the **load**
- High load can be induced remotely by making the OS busy (e.g. fetching many websites)

Murdoch: How it works in practice

- Eve **suspects** several IP addresses to be the OS she wants to deanonymize. This “closed-world model” is a practical limitation for attackers.
- She sends alternating traffic bursts through Tor to the OS and **measures** the clock skew of the suspected IPs (directly and not over Tor)
- Using **correlation techniques**, she can identify the OS if the IP addresses was in the set of suspects

Murdoch: Visualized



Conclusions

What you should keep in mind

- OSes provide **responder anonymity** as well as **DoS** and **censorship protection**
- OSes are fairly **flexible** and do not require modifications of the underlying service (e.g. apache or sshd)
- OS anonymity weaker than Tor client anonymity because attackers can always make them “talk”

Literature I

Unique .onion addresses. URL: https://metrics.torproject.org/hidserv-dir-onions-seen.html.

Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. "Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization". In: *Security & Privacy*. IEEE, 2013. URL: http://www.ieee-security.org/TC/SP2013/papers/4977a080.pdf.

Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *USENIX Security*. USENIX, 2004. URL: https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf.

Steven J. Murdoch. "Hot or Not: Revealing Hidden Services by their Clock Skew". In: *Computer and Communications Security*. ACM, 2006. URL: http://www.cl.cam.ac.uk/~sjm217/papers/ccs06hotornot.pdf.

Lasse Øverlier and Paul Syverson. "Locating Hidden Servers". In: *Security & Privacy*. IEEE, 2006. URL: http://www.onion-router.net/Publications/locating-hidden-servers.pdf.

Literature II

The Tor Project. *Tor: Hidden Service Protocol*. URL: https://www.torproject.org/docs/hidden-services.html.en.

The Tor Project. *Tor Rendezvous Specification*. URL: https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=rend-spec.txt.