ESGALDNS:
TOR-POWERED DISTRIBUTED DNS
FOR TOR HIDDEN SERVICES

by

Jesse Victors

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Dr. Ming Li Major Professor	Dr. Nicholas Flann Committee Member
Dr. Daniel Watson Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2015

ABSTRACT

EsgalDNS:

Tor-powered Distributed DNS

for Tor Hidden Services

by

Jesse Victors, Master of Science

Utah State University, 2015

Major Professor: Dr. Ming Li
Department: Computer Science

The Tor network is a second-generation onion routing system that aims to provide anonymity, privacy, and Internet censorship resistance to its users. In recent years it has grown significantly in response to revelations of national and global electronic surveillance, and remains one of the most popular and secure anonymity network in use today. Tor is also known for its support of anonymous websites within its network. Decentralized and secure, the domain names for these services are tied to public key infrastructure (PKI) but have usability challenges due to their long and technical addresses. In response to this difficulty, I propose and partially implement a decentralized DNS system inside the Tor network. The system provides a secure and verifiable mapping between human-meaningful names and traditional Tor hidden service addresses, is backwards- and forwards-compatible with Tor hidden service infrastructure, and preserves the anonymity of the hidden service and its operator.

(36 pages)

This work is dedicated to the developers and community behind the Tor Project and the Tails OS. These individuals work tirelessly to preserve the privacy and security of everyday citizens, journalists, activists, and others around the globe, providing the much-needed service of private conversations in a very public world.

CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The Tor network is a second-generation onion routing system that aims to provide anonymity, privacy, and Internet censorship protection to its users. The Tor client software multiplexes all end-user TCP traffic through a series of relays on the Tor network, typically a carefully-constructed three-hop path known as a *circuit*. Each relay in the circuit has its own encryption layer, so traffic is encrypted multiple times and then is decrypted in an onion-like fashion as it travels through the Tor circuit. As each relay sees no more than one hop in the circuit, in theory neither an eavesdropper nor a compromised relay can link the connection's source, destination, and content. Tor remains one of the most popular and secure tools to use against network surveillance, traffic analysis, and information censorship.

While the majority of Tor's usage is for traditional access to the Internet, Tor's routing scheme also supports anonymous websites, hidden inside Tor. Unlike the Clearnet, Tor does not contain a traditional DNS system for its websites; instead, hidden services are identified by their public key and can be accessed through Tor circuits. A client and the hidden service can thus communicate anonymously.

CHAPTER 2

BACKGROUND

2.1 Tor

The Tor network is a third-generation onion routing system, originally designed by the U.S. Naval Research Laboratory for protecting sensitive government communication, but today it continues to see global widespread use. Tor refers both to the client-side multiplexing software and to the worldwide volunteer-run network of over six thousand nodes. The Tor software provides an anonymity and privacy layer to end-users by relaying TCP traffic through a series of relays on the Tor network. Tor's encryption and routing protocols are designed to make it very difficult for an adversary to correlate an end user to their traffic. Tor has been recognized by the NSA as the "the king of high secure, low latency Internet anonymity".

2.1.1 Design

Tor routes encrypted TCP/IP user traffic through a worldwide volunteer-run network of over six thousand relays. Typically this route consists of a carefully-constructed three-hop path known as a *circuit*, which changes over time. These nodes in the circuit are commonly referred to as *guard node*, *middle relay*, and the *exit node*, respectively. Only the first node is exposed to the origin of TCP traffic into Tor, and only the exit node can see the destination of traffic out of Tor. The middle router, which passes encrypted traffic between the two, is unaware of either. The client negotiates a separate TLS connection with each node at a time, and traffic through the circuit is decrypted one layer at a time. As such, each node is only aware of the machines it talks to, and only the client knows the identity of all three nodes used in its circuit, making traffic correlation much more difficult

difficult compared to a VPN, proxy, or a direct TLS connection.

The Tor network is maintained by nine authority nodes, who each vote on the status of nodes and together hourly publish a digitally signed consensus document containing IPs, ports, public keys, latest status, and capabilities of all nodes in the network. The document is then redistributed by other Tor nodes to clients, enabling access to the network. The document also allows clients to authenticate Tor nodes when constructing circuits, as well as allowing Tor nodes to authenticate one another. Since all parties have prior knowledge of the public keys of the authority nodes, the consensus document cannot be forged or modified without disrupting the digital signature. [1]

2.1.2 Routing

In traditional Internet connections, the client communicates directly with the server. TLS encryption cannot hide IP and TCP headers, which must be exposed to allow routing. Eeavesdroppers can track end-users by monitoring these headers, easily correlating clients to the their activities. Tor combats this by routing end user traffic through a randomized circuit through the network of relays. The client software first queries an authority node or a known relay for the latest consensus document. Next, the Tor client chooses three unique and geographically diverse nodes to use. It then builds and extends the circuit one node at at time, negotiating respective TLS connections with each node in turn. No single relay knows the complete path, and each relay can only decrypt its layer of decryption. In this way, data is encrypted multiple times and then is decrypted in an onion-like fashion as it passes through the circuit.

The client first establishes a TLS connection with the first relay, R_1, using the relay's public key. The client then performs an ECDHE key exchange to negotiate K_1 which is then used to generate two symmetric session keys: a forward key $K_{1,F}$ and a backwards key $K_{1,B}$. $K_{1,F}$ is used to encrypt all communication from the client to R_1 and $K_{1,B}$ is used for all replies from R_1 to the client. These keys are used conjunction with the symmetric cipher suite negotiated during the TLS handshake, thus forming an encrypted tunnel with perfect forward secrecy. Once this one-hop circuit has been created, the client then sends

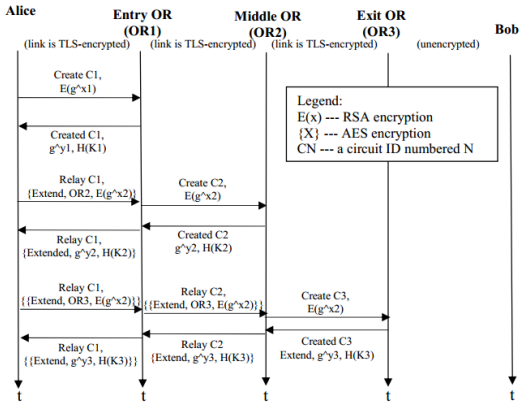

Figure 2.1: Anatomy of the construction of a Tor circuit.

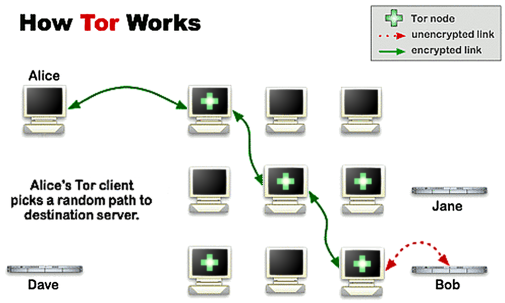

Figure 2.2: A circuit through the Tor network.

R_1 the RELAY_EXTEND command, the address of R_2, and the client's half of the Diffie-Hellman-Merkle protocol using $K_{1,F}$. R_1 performs a TLS handshake with R2 and uses R_2's public key to send this half of the handshake to R_2, who replies with his second half of the handshake and a hash of K_2. R_1 then forwards this to the client under $R_{1,B}$ with the RELAY_EXTENDED command to notify the client. The client generates $K_{1,F}$ and $K_{1,B}$ from K_2, and repeats the process for R_3, [2] as shown in Figure 3. The TCP/IP connections remain open, so the returned information travels back up the circuit to the end user.

Following the complete establishment of a circuit, the Tor client software then offers a Secure Sockets (SOCKS) interface on localhost which multiplexes any TCP traffic through Tor. At the application layer, this data is packed and padded into equally-sized Tor *cells*, transmission units of 512 bytes. As each relay sees no more than one hop in the circuit, in theory neither an eavesdropper nor a compromised relay can link the connection's source, destination, and content. Tor further obfuscates user traffic by changing the circuit path every ten minutes, [3] as shown in Figure 4. A new circuit can also be requested manually by the user.

Tor users typically use the Tor Browser, a custom build of Mozilla Firefox with a focus on security and privacy. The TBB anonymizes and provides privacy to the user in many ways. These include blocking all web scripts not explicitly whitelisted, forcing all traffic including DNS requests through the Tor SOCKS port, mimicking Firefox in Win-

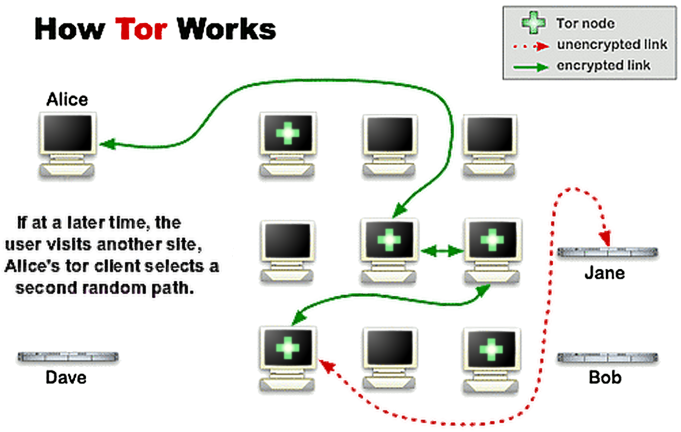

Figure 2.3: A Tor circuit is changed periodically, creating a new user identity.

dows both with a user agent (regardless of the native platform) and SSL cipher suites, and reducing Javascript timer precision to avoid identification through clock skew. Furthermore, the TBB includes the Electronic Frontier Foundation's HTTPS Everywhere extension, which uses regular expressions to rewrite HTTP web requests into HTTPS for domains that are known to support HTTPS. If this is the case, an HTTPS connection will be established with the web server. If this happens, end-to-end encryption is complete and an outsider near the user would be faced with up to four layers of TLS encryption: $K_{1,F}(K_{2,F}(K_{3,F}(K_{server}(\text{client request}))))$ and likewise $K_{1,B}(K_{2,B}(K_{3,B}(K_{server}(\text{server reply}))))$ for the returning traffic, making traffic analysis very difficult.

2.1.3 Hidden Services

Although Tor's primary and most popular use is for secure access to the traditional Internet, Tor also supports anonymous services, such as websites, marketplaces, or chatrooms. These are a part of the Dark Web and cannot be normally accessed outside the context of Tor. In contrast to Tor-anonymized web requests where the client is anonymous but the server is known, Tor hidden services provide bidirectional anonymity where both parties remain anonymous and never directly communicate with one another. This allows for a greater range of communication capabilities. Tor's hidden service protocol was introduced in 2004 and has not seen a major revision since then. [4]

Tor hidden services are known only by their public RSA key. Tor does not contain a DNS system for its websites; instead the domain names of hidden services are an 80-bit truncated SHA-1 hash of its public key, postpended by the .onion Top-Level Domain. Once the hidden service is contacted and its public key obtained, this key can be checked against the requested domain to verify the authenticity of the service server. This process is analogous to SSL certificates in the clearnet, however Tor's authenticity check leaks no identifiable information about the anonymous server. If a client obtains the hash domain name of the hidden service through a backchannel and enters it into the Tor Browser, the hidden service lookup begins.

Preceding any client communication, the hidden server, Bob, first builds Tor circuits to several random relays and enables them to act as *introduction points* by giving them its public key, B_K. The server then uploads its public key and the fingerprint identity of these nodes to a distributed hashtable inside the Tor network, signing the result. When a client, Alice, requests contact with Bob, Alice's Tor software queries this hashtable, obtains B_K and Bob's introduction points, and builds a Tor circuit to one of them, IP_1. Simultaneously, the client also builds a circuit to another relay, RP, which she enables as a rendezvous point by telling it a one-time secret, S.

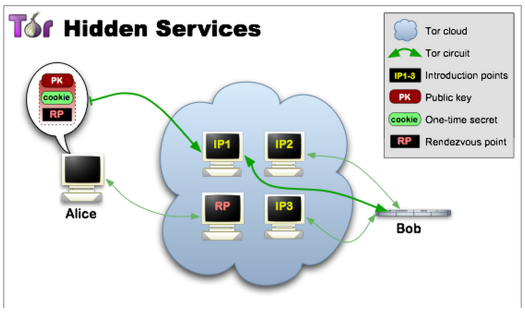

Figure 2.4: Alice uses the encrypted cookie to tell Bob to switch to RP.

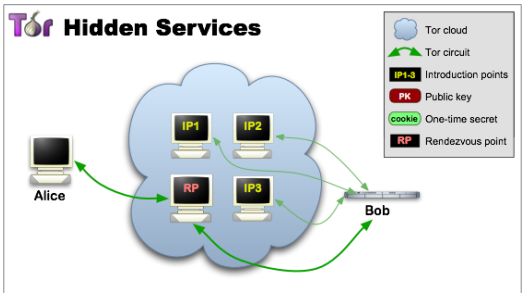

Figure 2.5: Bidirectional communication between Alice and the hidden service.

She then sends to IP_1 an cookie encrypted with B_K, containing RP and S. Bob decrypts this message, builds a circuit to RP, and tells it S_1, enabling Alice and Bob to communicate. Their communication travels through six Tor nodes: three established by

Alice and three by Bob, so both parties remain anonymous. From there traditional HTTP, FTP, SSH, or other protocols can begin, multiplexed over this new channel.

2.2 Motivation

The usability of hidden services is severely challenged by their non-intuitive 16-character hexadecimal domain names: 3g2upl4pq6kufc4m.onion is the address for the DuckDuckGo hidden service, whereas 33y6fjyhs3phzfjj.onion is the Guardian's SecureDrop service for anonymous document submission, and blockchainbdgpzk.onion is the anonymized edition of blockchain.info. It is rarely clear what service a hidden server is providing by its domain name alone without relying on third-party directories for the correlation, directories which must be updated and reliably maintained constantly. These must be then distributed through backchannels such as /r/onions, the-hidden-wiki.com, or through a hidden service that is known in advance.

One attempt at alleviating the issue is to generate hidden service many RSA keys in an attempt to find one whose hash contains or begins with a meaningful name. This is the case with Blockchain.info's hidden service, although such attempts are time-consuming and only partially effective because the size of the domain keyspace is too large to be effectively brute-forced in any reasonable length of time. Tor Proposal 224 makes this solution even worse as it suggests 32-byte domain names which embed the entire hidden service key in base32. Although prefixing the domain name with a meaningful word helps identify a hidden service, it does nothing to alleviate the logistic problems of entering a hidden service domain name manually into the Tor Browser, an issue that is not shared by domains in the clearnet.

It is clear that the usability problem exists and none of the few attempts to solve it have been fully successful. It is for these reasons that I propose EsgalDNS as a full solution.

CHAPTER 3

REQUIREMENTS

3.1 Assumptions and Threat Model

One of the primary assumptions that I make in this system is that not all Tor nodes can be trusted. Some of them may be run by malicious operators, curious researchers, or experimenting developers. They may be wiretapped, the Tor software modified and recompiled, or they may otherwise behave in an abnormal fashion. I assume that adversaries have control over some of the Tor network, they have access to large amounts of computational and financial resources, and that they have access to portions of Internet traffic, including portions of Tor traffic. I also assume that they do not have global and total Internet monitoring capabilities and I make no attempt to defend against such an attacker, although it should be noted that neither does Tor. I work under the believe that attackers are not capable of cryptographically breaking properly-executed TLS connections and their modern components, particularly the AES cipher, ECDHE key exchange, and the SHA2 series of digests, and that they maintain no backdoors in the Botan and OpenSSL implementations of these algorithms. Lastly, I assume that adversaries monitor and may attempt to modify the DNS record databases, but I assume that at least 50 percent of the Tor network is trustworthy and behave normally in accordance with Tor specifications.

3.2 Design Principles

Tor's high security environment is challenging to the inclusion of additional capabilities, even to systems that are backwards compatible to existing infrastructure. Anonymity, privacy, and general security are of paramount importance. We enumerate a short list of requirements for any secure DNS system designed for safe use by Tor clients. We later show

how existing works do not meet these requirements and how we overcome these challenges with EsgalDNS.

1. The registrations must be anonymous; it should be infeasible to identify the registrant from the registration, including over the wire.

2. Lookups must be anonymous; clients must stay anonymous when looking up registrations, otherwise they leak what hidden services they are interested in.

3. Registrations must be publicly confirmable; clients must be able to verify that the registration came from the desired hidden service and that the registration is not a forgery.

4. Registrations must be securely unique, or have an extremely high chance of being securely unique such as when this property relies on the collision-free property of cryptographic hashes.

5. It must be distributed. The Tor community will adamantly reject any centralized solution for Tor hidden services for security reasons, as centralized control makes correlations easy, violating our first two requirements.

6. It must remain simple to use. Usability is key as most Tor users are not security experts. Tor hides non-essential details like routing information behind the scenes, so additional software should follow suite.

7. It must remain backwards compatible; the existing Tor infrastructure must still remain functional.

8. It should not be feasible to maliciously modify or falsify registrations in the database or in transit, even though insider attacks.

The current Tor hidden service protocol meets the first, second, and sevenths requirements though the use of Tor circuits, the third by the nature of the hidden service domain names, the fourth because of the collision-free property of SHA-1, the fifth due to the

hashtable in the Tor network, and the eight requirement because of the verifiability of the domain names to the service's public keys. However, it only partially meets the sixth requirement because although the domain names are entered in the traditional manner into the Tor Browser, the domain names are not entirely human-meaningful or memorable so it suffers from usability problems.

Several additional objectives, although they are not requirements, revolve around performance: it should be assumed that it is impractical for clients to download the entirety or large portions of the domain database in order to verify any of the requirements, a DNS system should take a reasonable amount of time to resolve domain name queries, and that the system should not incur any significant load on client computers.

CHAPTER 4

CHALLENGES

4.1 Zooko's Triangle

One of the largest challenges is inherent to the difficulty of designing a distributed system that maintains a correlation database of human-meaningful names in a one-to-one fashion. The problem is summarized in Zooko's Triangle, an influential conjecture proposed by Zooko Wilcox-O'Hearn in late 2001. The conjecture states that in a persistent naming system, only two out of the three following properties can be established: [5]

- Human meaningfulness: the names have a quality of meaningfulness and memorability to the users.
- Securely one-to-one: each name is unique, corresponds to a unique entity or owner, and cannot be forged.
- Distributed: the naming system lacks a central authority or database for allocating and distributing names.

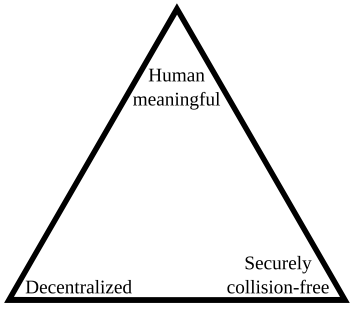

Figure 4.1: Zooko's Triangle.

For example, Tor hidden service .onion addresses and Bitcoin addresses are secure and decentralized but are not human-meaningful. Clearnet domain names are memorable and provably collision-free, but use central database managed DNS under the jurisdiction of ICANN. Finally, human nicknames are meaningful and distributed, but not securely collision-free. [6]

4.2 Communication

4.3 Fault Tolerance

CHAPTER 5

EXISTING WORKS

Existing literature proposing DNS systems for Tor is fairly sparse, though some ideas have been put forward.

5.1 Clearnet DNS

The Internet Domain Name Service (DNS) is a hierarchical distributed naming system for computers connected to the Internet. It links two principal Internet namespaces, Internet Protocol (IP) addresses and domain names, and translates one to the other. IP addresses specify the location of a computer or device on a network and domain names identify that resource. Domain names also serve as an abstraction layer so that devices can be moved to a different physical location or to a different IP address without loss of functionality. In contrast to IP addresses, domain names are human-meaningful and easily memorized, so DNS is a crucial component to the usability of the Internet.

Domain names on the Internet consist of a sequence of labels, delimited by dotes. The right-most label is the top-level domain (TLD) and can be used to classify the Internet resource by country or by organization type, although generic TLDs are more common. One or more subdomains follow the TLD. Each label can consist of up to 63 characters and the domain names can be up to 253 characters.

5.2 Namecoin

Bitcoin is a decentralized peer-to-peer digital cryptocurrency, created by pseudonymous developer Satoshi Nakamoto in 2008. Ownership of Bitcoins consists of holding a private ECDSA key, and a transfer is a transmission of Bitcoins from one key to another. All transactions are recorded on a public ledger, called a blockchain, a data structure whose integrity

is ensured through computational power but publically verifiable. Bitcoins are generated computationally at a fixed rate by *miners* in a process that also secures the blockchain. Although Bitcoin received limited attention in the first two years of its life, it has since grown significantly since then, with approximately 70,000 daily transactions as of the time of this writing. Bitcoin's growth has led to the creation of many alternative cryptocurrencies, and its popularity has influenced financial discussions and legal controversy worldwide.

5.2.1 Architecture

A blockchain is data structure fundamental to Bitcoin, and crucial for its functionality. As a distributed decentralized system, this public ledger is Nakamoto's answer to the problem of ensuring agreement of critical data across all involved parties. The blockchain is a novel structure, and its structure guarantees integrity, chronological ordering of transactions, and the prevention of double-spending of Bitcoins. The blockchain consists of blocks of data that are held together by proof-of-work, a cryptographic puzzle whose solution is provably hard to find but trivial to verify. Bitcoin's proof-of-work is based on Adam Back's Hashcash scheme: that is, find a nonce such that the hash of this nonce and some data produces a result that begins with a certain number of zero bits. In Bitcoin's case this is stated as finding a nonce that when passed through two rounds of SHA256 (SHA256^2) produces a value less than or equal to a target T. This requires a party to perform on average $\frac{1}{Pr[H \leq T]} = \frac{2^{256}}{T}$ amount of computations, but it is easy to verify that $\text{SHA256}^2(\text{msg}||n) \leq T$. Nodes in the Bitcoin network collectively agree to use the blockchain with the highest accumulation of computational effort, so an adversary seeking to modify the structure would need to recompute the proof-of-work for all previous blocks as well as out-perform the network, which is currently considered infeasible. [7]

Each block in the blockchain consists of a header and a payload. The header contains a hash of the previous block's header, the root hash of the Merkle tree built from the transactions in this block, a timestamp, a target T, and a nonce. The block payload consists of a list of transactions. The root node of the Merkle tree ensures the integrity of the transaction vector: verifying that a given transaction is contained in the tree takes

$log(n)$ hashes, and a Merkle tree can be built $n * log(n)$ time, ensuring that all transactions are accounted for. The hash of the previous block in the header ensures that blocks are ordered chronologically, and the Merkle root hash ensures that the transactions contained in each block are order chronologically as well. The target T changes every 2016 blocks in response to the speed at which the proof-of-work is solved such that Bitcoin miners take two weeks to generate 2016 blocks, or one block every 10 minutes. The $SHA256^2$ proof-of-work provides integrity of the data structure, and secp256k1 ECDSA key are used to prove ownership of coins. [7]

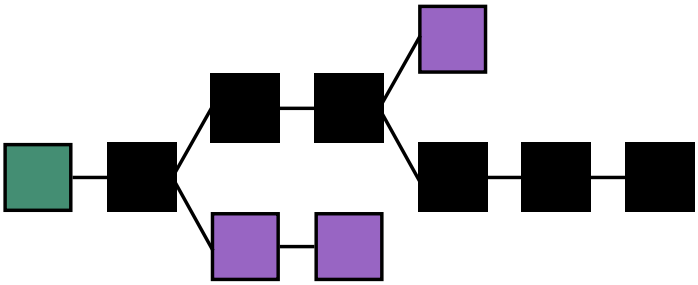

Figure 5.1: A sample blockchain.

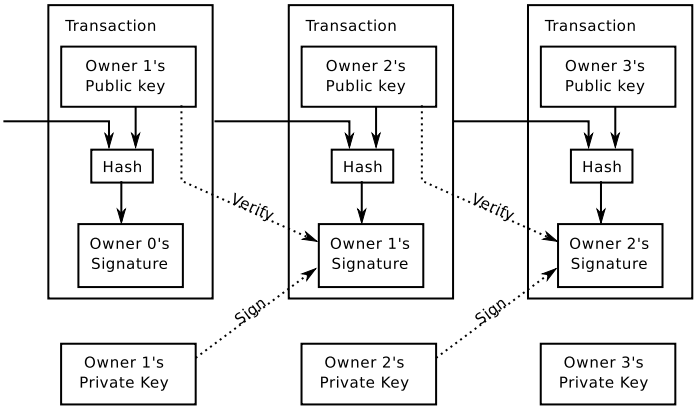

Figure 5.2: Three traditional Bitcoin transactions.

In the possibility that multiple nodes solve the proof-of-work and generate a new block simultaneously, the block becomes orphaned, the transactions recycled, and the blockchain follows the longest path from the genesis node to the latest block. Each transaction contains

the public of the recipient, the ECDSA digital signature of the transaction from the sender, and the hash of the originating transaction. In this way, the digital signatures and proof-of-work in the blockchain can be traced back to the origin and forwards indefinitely.

In recent years, systems have been developed that have shown Zooko's Triangle to be false. One prominent example is Namecoin, a naming system which uses a Bitcoin-like blockchain to store name-value pairs. Human-meaningful names can be embedded in the blockchain, which is distributed by nature. The uniqueness of the names is ensured by the Namecoin network and can be verified with anyone holding the blockchain. However, Tor developers have been wary of using Namecoin to store domain names for Tor hidden services. It is also impractical to require all Tor clients to download the entire blockchain before being able to use a hidden service DNS system, and there are inherent security challenges involved with querying servers or the Namecoin network for a registration without being able to use a complete blockchain to verify it. Therefore, another solution is needed.

Namecoin is a decentralized information registration and transfer system based on Bitcoin. It was the first software fork of Bitcoin and was introduced in April 2011. It uses its own blockchain and can hold name-value pairs in the blockchain attached to coins. While Bitcoin is primarily focused on supporting a currency, Namecoin aims to be a general key-value store, capable of holding cryptographic keys, DNS registrations, or other arbitrary data. It is most commonly used as a secure and censorship-resistance replacement for clearnet DNS. In 2014, Namecoin was recognized by ICANN is the most well-known example of a PKI and DNS system with an emphasis of distributed control and privacy, a growing trend in light of the revelations about the US Government by Edward Snowden.

5.2.2 Names

Although it inheriting Bitcoin's existing infrastructure, Namecoin added several transaction types specifically for registering and processing names, along with two new rules: names in the blockchain expire after 36,000 blocks unless renewed by the owner and no two unexpired names can be identical. These rules are enforced in the blockchain by Namecoin nodes and anyone verifying the Namecoin blockchain. Registering a name consumes 0.01

Namecoin, names can also be transferred to other owners, and they are two types: DNS and personal. The DNS type uses a new Top Level Domain (TLD) not in use by ICANN: .bit, and is used for DNS registrations. The personal name can contain arbitary data, including user information such as cryptographic keys. Like Bitcoin, Namecoin's maximum block size is one megabyte and the difficulty is set such that blocks generate every 10 minutes. Thus names expire every 250 days.

5.3 Different Encoding

One of the most prominent is a 2011 Bachelor's thesis which outlines representing a hidden service's domain name as a series of words, rather than a base58-encoded hash. [4] However, while this scheme would improved recognition and memorability of hidden services, the words would remain random, are not chosen in advance, and do not relate to the hidden service in any meaningful way. Therefore this solution is an improvement but is not a solution. The problem remains open.

CHAPTER 6

SOLUTION

6.1 Overview

I propose a new DNS system for Tor hidden services, which I am calling EsgalDNS. *Esgal* is a Sindarin Elvish word from the works of J.R.R Tolkien, meaning "cloaked" or "hidden". EsgalDNS is a distributed DNS system embedded within the Tor network and powered by existing Tor nodes. EsgalDNS, like other DNS systems, supports a number of command and control operations, including Query, Create, Modify, Move, Renew, and Delete. All commands, aside from Query, are authenticated by digital signatures to ensure that only the operator of the hidden service issued them. These commands are processed by a subset of participating Tor nodes, described below.

6.2 Components

6.2.1 Quorum

EsgalDNS is primarily powered by a special subset of Tor nodes, called a *quorum*. The quorum of M size is derived from the signed consensus document published hourly by the Tor authority nodes; a PRNG such as Merseene Twister is seeded by a SHA-256 hash of the consensus document, and the PRNG then scrambles the list of Tor nodes. The first M nodes become the committee. Since both clients and Tor nodes hold an authenticated and up-to-date copy of the consensus document, all involved parties are aware and agree on the members of the quorum. The consensus document published at 00:00 GMT each day will be used for determining the quorum, giving the quorum nodes a participation lifetime of up to 24 hours. As the health and status of the Tor network cannot be easily determined in advance, the consensus document cannot be known until it is published, thus the quorum

cannot be known until the 0:00 GMT document becomes available. This deterministic but unpredictably random nature makes the system more resilient to attackers attempting to force their node into the quorum for malicious purposes. Furthermore, as the digital signature from the authority nodes is embedded within the consensus document itself, past quorums can be derived from archives of the document.

Nodes currently in the quorum are responsible for handling transactions such as the Create, Modify, Move, Renew, and Delete commands. They can also respond to Query requests, although other nodes can also respond as well. Hidden service operators can use Tor circuits to give one of these former commands to one or more quorum nodes. The transaction then propagate to the remaining quorum nodes when they synchronize their knowledge bases. Each quorum node is responsible for distributing two databases to other quorum nodes: *pages* and *snapshots*. Pages are long-term and stable collections of records, whereas snapshots hold volatile records that may be yet be fully propagate across the entire quorum. Pages reference pages from the previous day's quorum, forming a append-only page-chain that grows forward with time. Snapshots, by contrast, do not reference previous pages but their information is merged into the quorum node's page periodically.

A page on day n belonging to quorum member i can be expressed as $P_{n,i}$. This page contains five distinct elements: *prevN*, *prevI*, *prevHash*, *recordList*, *consensusDoc*, and *digSig*, where *prevN* and *prevI* represents the back-reference to the $P_{prevN,prevI}$ page, *prevHash* is the SHA-256 of $P_{prevN,prevI}$, *recordList* is the list of records that were send the quorum on day n, *consensusDoc* is the consensus document digitally signed by the authority nodes, and finally *digSig* is the digital signature from quorum member i on the preceding fields.

At startup, every Tor node participating in EsgalDNS

One of the central elements in my system is a *quarum*, a set of special decision-making Tor nodes. The set of committee nodes changes every day and the set cannot be known in advance.

At a high level, domain registrations are broadcasted through a Tor circuit to all committee nodes. Every node then analyses the registration as well as its knowledge of

the chain and makes a decision. If the registration is invalid, the node rejects with an appropriate flag. If the domain name is already taken, the node returns a flag along with the pre-existing registration. Otherwise, it digitally signs its approval and the proposal itself and distributes this to the rest of the committee. It then waits for the rest of the committee votes. Since every Tor node has the up-to-date public keys of all other nodes due to the consensus documents, every committee member can verify the votes of all other committee members. Once the node confirms that all committee nodes received the same proposal and that a significant majority indicate that the domain is both valid and available, it adds the registration to its local storage. More importantly, the registration is recorded to an append-only endless scroll distributed within the Tor network. Thus domain names are consumed in a first-come-first-serve basis.

The scroll is a distributed and highly redundant chained data structure that slowly rolls through the Tor network. The scroll is N by M in shape and consists of two primary components: *blocks* and *captures*. Blocks contain one or more captures, and blocks are duplicated across the M committee nodes. Each capture is a collection of information from one day; it contains the consensus document from the previous morning, a list of domain registrations approved that day, the approval sign-off digital signatures from the committee nodes on those registrations, the digital signatures from the committee indicating their approval of the integrity of the scroll, and the hash of the previous four captures. In this way, captures are fully verifiable and contain enough information to link to the previous capture, forming a chain. This chain is not held by any single Tor node, rather it is encapsulated within a rolling window of blocks N days deep. As the days progress, the captures in the oldest block are migrated to the current day's block, rolling the structure forward. Thus the scroll is divided across N blocks, with copies of each block held by M Tor nodes. I consider $N = 16, M = 64$ reasonable values, which would involve 1,024 nodes at any given time, although M can be easily changed even while the scroll is in use.

This distributed system provides the ability to confirm a given domain name is not already in use, without relying on a single central authority. Assuming that the committee

nodes are honest in their vote and trustworthy in their nature, this achieves all three properties of Zooko's Triangle. Even if the committee nodes are malicious, I believe I can introduce sufficient countermeasures to make it infeasible for an attacker to successfully manipulate the system, assuming that the majority of the Tor network is trustworthy. More research, design, and implementation is needed, but this I believe is a very promising approach.

6.2.2 Domain Registration

A domain registration consists of eight components which are tied together by digital signatures and proof-of-work. The components are *nonce*, *consensusHash*, *time*, *domain*, *subdomains*, *contact*, *digSig*, and *pubKey*.

nonce

An eight-byte number that serves as a source of randomness for the proof-of-work.

consensusHash

A 32-byte value containing the SHA256 hash of the consensus document published by the authority nodes. Consensus documents are generated frequently, so *consensusHash* will be based on the document published at 00:00 GMT that day.

time

A four-byte integer holding the number of seconds since January 1st, 2013.

domain

A null-terminated cstring of the human-meaningful domain that will be correlated with the traditional .onion address of the hidden service. This can be up to 32 characters long. The TLD is .tor

subdomains

Up to 255 bytes of subdomain data, preceded by one byte that indicates the byte length. Each subdomain is null-terminated, so with the null characters 15 subdomains are possible when each is 16 characters long.

contact

16 bytes representing the last 32 base64-encoded bytes of the fingerprint of the service operator's PGP key, if they have one. If they do not, these bytes are zeroed. The purpose of this field is to allow the operator to be contacted securely.

digSig

The digital signature of all preceding fields.

pubKey

The public key of the hidden service.

To generate a registration, *domain*, *subdomains*, and *contact* are determined by the operator, while *consensusHash* and *time* are filled in automatically. The hidden service operator then has to find a value for *nonce* such that the proof-of-work is valid, specifically that that the SHA256 of *digSig* is less than a target value T. I plan to set T such that the proof-of-works takes a significant amount of time on a modern CPU. This makes registering a domain expensive, thwarting flooding attacks. If *nonce* is found, the registration is valid and ready for broadcast.

Two common proof-of-work systems are hashing, typically double-SHA256 (SHA256^2) in the case of Bitcoin, and scrypt, a password-based key derivation function used by Litecoin. I chose the latter here; scrypt is a harder proof-of-work system because it requires large quantities of RAM in addition to CPU time, making brute-forcing significantly more challenging. Finding *nonce* is made even more difficult because for every *nonce*, a new digital signature must be made using the service's private key. This slows mining, complicates porting to GPUs and other specialized hardware, and prevents outsourcing to a outside computational resource. The digital signature ensures that all field are authenticated to the key of the hidden service, verifiable by all.

Once created and finalized, domains are broadcasted through Tor circuit(s) to committee nodes, where it can be approved and added into the scroll.

6.2.3 Registration Query

A client requesting *example.tor* can use a Tor circuit to anonymously query committee nodes, or in fact any node holding the scroll, for the domain name. If the domain is taken, the client will receive the full registration (as specified above) as well as the digital signatures of the committee members who voted on that registration. This consensus and the registration itself can both be validated by the client's machine. The client can then extract *pubKey* from the registration, hash it and truncate it, and look up the .onion in the traditional manner.

6.3 Fault Tolerance

Page wrapping stuff...

CHAPTER 7

ANALYSIS

general analysis...

7.1 Performance

bandwidth, CPU, RAM, latency for clients..

7.1.1 Node Load

demand on participating nodes...

7.2 Security

security breakdown...

CHAPTER 8

RESULTS

8.1 Implementation

implementation...

8.2 Discussion

discussions...

8.2.1 Guarantees

Guarantees...

CHAPTER 9

FUTURE WORK

Some open problems that I need to address include:

1. How frequently should domains expire? Are there any security risk in sending a Renew request?

2. How should unreachable or temporarily down nodes be handled? I'd like to know the percentage of the Tor network that is reachable at any given time.

3. How many nodes in the Tor network should be assumed to be actively malicious? What are the implications of increasing this percentage?

4. What attack vectors are there from the committee nodes, and how can I thwart the attacks?

5. What are the implications of a node intentionally voting the opposite way, or ignoring the request altogether?

6. What would happen if a node was too slow or did not have enough storage space to do its job properly?

7. What other open problems are there?

8. What related works are there in the literature that relate to the concepts I have created here?

CHAPTER 10

CONCLUSION

I have presented EsgalDNS, a distributed DNS system inside the Tor network. The system correlates one-to-one human-meaningful domain names and traditional Tor .onion addresses. Tor nodes and clients can both verify the authenticity, integrity, and uniqueness of registrations. Although this design is nowhere near finalized, so far this system seems both promising and novel. I have more work to do, and I am planning to implement and test this system on a simulated Tor network. If accepted by the Tor community, I believe that EsgalDNS will be a valuable infrastructure that will significantly improve the usability and popularity of Tor hidden services.

REFERENCES

[1] L. Xin and W. Neng, "Design improvement for tor against low-cost traffic attack and low-resource routing attack," *2009 International Conference on Communications and Mobile Computing*, 2009.

[2] Z. Ling, J. Luo, W. Yu, X. Fuc, W. Jia, and W. Zhao, "Protocol-level attacks against tor," *Computer Networks*, 2012.

[3] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker, "Shining light in dark places: Understanding the tor network," *Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2969*, 2008.

[4] S. Nicolussi, "Human-readable names for tor hidden services," 2011.

[5] M. S. Ferdous, A. Jsang, K. Singh, and R. Borgaonkar, "Security usability of petname systems," October 2009.

[6] M. Stiegler, "Petname systems," 2005.

[7] K. Okupski, "Bitcoin developer reference," 2014.