Namecoin as a Decentralized Alternative to Certificate Authorities for TLS

Jeremy Rand
Lead Application Engineer, The Namecoin Project
https://www.namecoin.org/

OpenPGP: 5174 0B7C 732D 572A 3140 4010 6605 55E1 F8F7 BF85

Presented at Grayhat 2020 Monero Village
A brief introduction to Namecoin

- Like the DNS, but secured by a blockchain.
- Uses the “.bit” top-level domain (TLD).
- Names are represented by special coins.
- First project forked from Bitcoin (in 2011; Bitcoin was created in 2009).
- Original focus of developers was on censorship-resistance.
  - We later became interested in PKI use cases (e.g. for TLS) as well.
The Threat of Certificate Authorities (CA’s) in TLS: Censoring Content

- CA’s can censor websites by revoking their certificates.
- Censorship of scientific knowledge: Comodo revoked the certificate of Sci-Hub in order to comply with a copyright enforcement court order obtained by research paywall vampires (American Chemical Society).
- Geopolitical censorship: Let’s Encrypt revoked the certificates of all entities residing in the People’s Republic of Donetsk in order to comply with OFAC sanctions.
- Censorship of journalism: Let’s Encrypt revoked the certificate of the (allegedly Russian-funded) media outlet USA Really in retaliation for the site “posting content focused on divisive political issues” and “attempting to hold a political rally”.
The Threat of Certificate Authorities (CA’s) in TLS: Intercepting Traffic

- TLS trusts over 1000 certificate authorities.
- CA’s get compromised, enabling man-in-the-middle (MITM) attacks.
  - DigiNotar was allegedly compromised by Iranian intelligence.
- CA’s don’t perform due diligence.
  - WoSign handed out a TLS certificate for github.com to a random guy because... he proved he had an account on GitHub.
- CA’s achieve Too Big To Fail status.
  - StartCom (AKA the CA version of Martin Shkreli) held large parts of the Internet for ransom during the Heartbleed incident; they were never punished.
These two threats have an inverse correlation.

- Too many CA’s? Easy to find one who can be compromised. Vulnerable to MITM.
- Not enough CA’s? Hard to find one who will do business with you. Vulnerable to censorship.
The DNS community long ago realized that a secure version of DNS could be used instead of CA's.
- Website owner puts a TLS certificate fingerprint in their DNS record.
- End user’s browser makes sure that the certificate matches the fingerprint from DNS.
- Standardized by IETF as DANE.
- If we assume that the DNS is secure (e.g. via DNSSEC), this should be secure.

We don't trust the DNS, but maybe we do trust Namecoin to do what the DNS is supposed to do.
Adapting DANE to Namecoin?

- Since Namecoin is interoperable with DNS, we can put TLS certificate fingerprints in Namecoin according to the DANE spec.
- A Namecoin-DNS bridge (running on localhost) signs the records with a bridge-generated DNSSEC key.
- User configures Unbound to use the bridge’s DNSSEC key for the .bit zone.
- Should be as simple as that, right?
Web browsers don’t support DANE

• No major web browsers do DNS lookups for DANE records.
• Some browsers briefly experimented with stapling of DANE records in the TLS handshake.
  – Useless for Namecoin, since for Namecoin the DNSSEC trust root is different per user.
  – Useless for preventing MITM’s, since this only expands the set of accepted certificates.
• Chromium and Firefox devs have rejected DANE (even the stapled variant).
Design goals for Namecoin TLS interoperability

- Support “positive overrides”: a certificate that matches the Namecoin blockchain must be accepted. (Prevents censorship.)
- Support “negative overrides”: a certificate that doesn’t match the Namecoin blockchain must be rejected. (Prevents MITM.)
- Interoperable with standard, unpatched TLS implementations. (We don’t want to fork Firefox.)
- Low attack surface. (Intercepting proxies are not okay; we want to re-implement as little TLS logic as possible. Preferably sandboxable.)
- Minimal on-chain data size. (Blockchains don’t scale well; we don’t want to make this worse.)
- Restricting to special certificate forms is okay if it helps us achieve these goals. (It’s okay to not support the entire DANE specification.)
Existing (Non-)Solutions

- **Intercepting / MITM proxy?**
  - Re-implements entire TLS protocol.
  - Tends to break client certs and cert pinning.
  - Way too much attack surface.
  - Remember Lenovo SuperFish?

- **Shared library hooking (LD_PRELOAD)?**
  - Re-implements entire certificate verifier.
  - Unstable C structures; might corrupt memory if a library gets upgraded.
  - Better, but still too much attack surface.
Could a browser extension work?

- Nope.
- All major browsers removed the needed API’s years ago.
- Due to concerns about malware abusing the API’s.
- Even the old API’s would often leak login cookies.
- Chromium and Firefox devs have actively refused to support our use case when we asked.
Targeted TLS Implementations

- Microsoft CryptoAPI
  - Used in most Windows software.

- Mozilla NSS
  - Used in Firefox (cross-platform).
  - Used in a lot of GNU/Linux software (e.g. Chromium).
Positive Overrides in Microsoft CryptoAPI

• If you manually add a self-signed website certificate to the CryptoAPI root CA store, it will be accepted in any subsequent TLS handshakes.

• But this is a horrible idea for many reasons.
  – What if the certificate is also valid as a CA? Now it can impersonate other websites!
  – What if the certificate has multiple hostnames? Ditto!
  – Requires us to know the full certificate contents before we start the TLS handshake. Violates “Minimal on-chain data size” design goal.
• <ryan-c> how small can we actually make a self-signed ecdsa cert?
• <Jeremy_Rand> Probably not small enough to fit in a Namecoin name
• <ryan-c> maybe not
• <ryan-c> er maybe it is
• <ryan-c> one sec
• <ryan-c> let me do some wizarding
• * Jeremy_Rand loves it when ryan-c puts on his wizard hat
• <ryan-c> Jeremy_Rand: the cert may too big, but we should consider cheating
• <ryan-c> Jeremy_Rand: yes, we can fit a self-signed ecdsa cert by cheating
Dehydrated Certificates

- Ryan's solution: starting with only a public key, validity period, signature, and hostname (called a **dehydrated certificate**), you can deterministically construct a valid certificate by filling a template (**rehydrating** the certificate).
  - Pubkey, validity period, and signature go in the Namecoin value.
  - Hostname determined by what Namecoin name is being looked up.
  - Use ECDSA instead of RSA – much smaller keys and signatures.
Efficiency Advantages of Dehydrated Certificates

- In theory: 104 bytes per certificate.
- In practice: 255 bytes.
  - Due to JSON/base64 encoding, no compressed pubkeys, other compromises.
- Before dehydration: 464 bytes binary, 620 bytes base64.
- A Namecoin name can hold 520 bytes (which also needs to include IP addresses and other DNS records).
Security Advantage of Dehydrated Certificates

- All of the potentially dangerous X.509 fields (e.g. the CA bit) are controlled by the template, not the attacker.
- The only fields the attacker controls are the public key, the validity period, and the signature.
  - Attacker-controlled public keys are already standard in the TLS ecosystem – clearly safe.
  - Validity period's only potentially harmful effect is disincentivizing key rotation – only impacts the hostname who chose that validity period.
  - The signature check normally passes, and the only thing an attacker-controlled signature can change is making the signature check not pass – doesn't accomplish anything useful attack-wise.
Rehydrating and injecting via DNS hook

• When a DNS request for a Namecoin domain name is received by the Namecoin-DNS bridge on localhost, the dehydrated certificate is rehydrated into DER format, and injected into the CryptoAPI root CA store.

• Once injection has happened, the Namecoin-DNS bridge replies with the IP address, and the connection proceeds as usual.
The standard Windows API’s for adding trusted certificates require Administrator privileges.

But, the certificate store actually lives inside the Windows Registry.

The Registry has an ACL permission scheme, just like the filesystem.

So we create a sandboxed service user, and grant it write privileges to the specific Registry key that contains the root CA store.

Run the Namecoin-DNS bridge under this account, and it can now add trusted certificates via standard Registry API’s, without any other privileges.
Demo of Positive Overrides in Microsoft CryptoAPI
Positive Overrides in Mozilla NSS

- NSS doesn't always honor self-signed website certificates from NSS's trust store.
  - The Mozilla people believe that supporting this would be a footgun.
  - So we need to find another approach.
Name Constraints

• Name constraints restrict the set of domain names that a TLS CA can issue certs for.

• Supported by virtually all TLS implementations.
  – Last major stragglers were Apple (implemented in 2018), Java (still not supported as of 2017), and Node.js (still not supported as of 2017).

• In theory: you can buy a name-constrained CA from a public CA, and then you can issue as many certs as you want within your domain name without bothering the public CA.
  – Not used in practice because of regulatory capture.
  – Public CA’s would rather make you pay for multiple certs.
Real-World Usage of Name Constraints

- A corporate intranet CA can be constrained to only issue certs within a corporate intranet TLD.
  - Used by Netflix’s intranet CA’s.

- Public CA’s can be constrained to never issue certs for TLD’s with unusual regulatory requirements.
  - Used by Let’s Encrypt to blacklist the .mil TLD.
Storing Name-Constrained CA's in the Blockchain

- We can construct a name-constrained intermediate CA from a public key + domain name.
- Validity period and signature don’t need to be deterministic, so we can omit them from the blockchain and generate them locally.
- The name constraints RFC says that name constraints are ignored for root CA's.
  - No idea if implementations follow the spec on this, but doesn’t affect us since we sign the blockchain’s name-constrained CA with a locally generated root CA.
Efficiency Advantages of Name-Constrained Certificates

- In theory: 34 bytes per certificate.
- In practice: 134 bytes.
  - Due to JSON/base64 encoding, no compressed pubkeys, other compromises.
- For comparison: dehydrated was: 104 bytes in theory, 255 bytes in practice.
- A Namecoin name can hold 520 bytes. This easily fits with room to spare.
Name-Constrained Certificates are Layer 2

- The blockchain only commits to the name-constrained CA’s public key.
- You can issue new website certificates with that CA (using new keys) as often as you like.
- This doesn’t require updating the blockchain.
- You can have scalability **and** key hygiene.
Should we inject via DNS hook for Mozilla NSS?

- We **could** inject the name-constrained CA’s into the NSS certificate store.
- But… there are some issues with that.
  - NSS’s cert store uses sqlite. Very slow to inject.
  - Leaves your browsing history in the NSS cert store. Privacy issue.
  - Tor Browser disables the sqlite cert store completely.
  - Confuses key pinning (sqlite-stored certs get privileges to bypass key pins). Attack surface we don’t want.
How does Mozilla NSS’s cert store actually work?

- The NSS sqlite-based cert store is actually a PKCS#11 module called Softoken.
  - Yes, this is the spec that’s usually used by HSM’s.

- So we wrote our own PKCS#11 module that feeds NSS the name-constrained CA certs from Namecoin, without the Softoken/sqlite middleman.

- It’s called ncp11. Written in Go.
  - We also made a Go library for writing your own PKCS#11 modules.
Sandboxing

- The name-constrained CA’s from the blockchain can be signed with a root CA that has its own name constraint… restricting it to only Namecoin domains.
- The root CA can be imported as a trust anchor separately from ncp11.
- NSS can then be configured to use ncp11 only to look up intermediate certs, not root CA’s.
- Result: Any exploit of ncp11 stays confined to Namecoin domains; it can’t harm DNS-based domains.
Name-Constrained Blockchain CA’s with Microsoft CryptoAPI

• We plan to port the name-constrained CA design back to CryptoAPI.
  – Stay tuned for progress on this.

• The Windows Registry sandboxing and NSS intermediate-only sandboxing tricks can be combined.
  – So we can inject name-constrained intermediate CA’s to the Registry, and any exploit against Namecoin can’t metastasize to DNS domains.
Negative Overrides in Mozilla NSS

- We experimented with using key pinning.
  - Pin the local Namecoin root CA for the .bit TLD.
  - Alas, key pinning API’s are being phased out by major browsers.

- Is there another way to prevent all public CA’s from issuing .bit certs?
  - We could politely ask them to put a name constraint in their cert, like Let’s Encrypt did for .mil.
  - But… they’d probably say no.
Rewriting the Public CA’s’ Certificates

- We don’t actually need the public CA’s’ permission to add a name constraint to their certs.
- We can simply convert their root CA certs to intermediate CA certs, and sign them with a locally generated CA that blacklists the .bit TLD.
- This is actually something that CA’s do for each other all the time, it’s called cross-signing.
Adding Cross-Signing to ncp11

- The list of built-in NSS trusted certificates is... you guessed it, another PKCS#11 module (called “CKBI”).
- So we rigged ncp11 to act as a PKCS#11 proxy to CKBI.
- ncp11 cross-signs all of CKBI’s certificates to add a name constraint blacklisting the .bit TLD.
- It also marks the original CKBI certificates as “prohibited”.
- Result: built-in root CA’s cannot sign .bit certificates.
Negative Overrides in Microsoft CryptoAPI

- Cross-signing as a negative override mechanism has some unpleasant side effects.
- It converts root CA’s to intermediate CA’s, and changes their fingerprints.
- This breaks some assumptions by (poorly designed) software.
  - Extended Validation certificates can break.
  - So can certificate pinning.
- But there is a better way in CryptoAPI.
Certificate Properties in Microsoft CryptoAPI

- CryptoAPI, like NSS, stores a bunch of metadata for each certificate.

- In CryptoAPI, the metadata is in the form of “Properties”.
  - Stored in the Windows Registry, along with the cert itself.

- Hmm… the wincrypt.h file in Windows has a #define called CERT_ROOT_PROGRAM_NAME_CONSTRAINTS_PROP_ID.

- Zero hits for this #define on DuckDuckGo, other than the header file itself, and a Microsoft docs page that just says “Reserved.”
Undocumented Windows CryptoAPI Feature: External Name Constraints

- Some experimentation revealed that if you set this Property’s value to an ASN.1-encoded X.509 name constraints extension, the name constraint will be applied to the corresponding root CA.
- The Property name (“ROOT_PROGRAM”) insinuates that Microsoft intended to use it for their root CA list.
  - According to Wine’s Git history, it was added to Windows before 2007.
- But Microsoft apparently never used it for anything.
Injecting Name Constraints to the Windows Registry

- Remember our positive override tool that injects certificates to the Windows Registry based on a DNS hook?
- We’ve extended that codebase to support injecting the name constraints Property into all of the Registry keys that store the built-in CryptoAPI CA’s.
- Result: all of the default CA’s in the CryptoAPI root CA store cannot issue .bit certs.
But there’s a catch!

- Most of the root CA’s that CryptoAPI trusts *aren’t in the root CA store!*
- Microsoft ships a “certificate trust list” (AuthRoot.stl) as part of Windows Update, which contains hashes of all the trusted root CA’s (currently 420 root CA’s).
- At most 24 of them are actually included in the root CA store shipped with a default Windows install.
Where are the other 396 certs, then?

- When CryptoAPI verifies a certificate, it downloads and installs any needed root CA's listed in AuthRoot.stl from Windows Update on the fly.
  - Supposedly this is a performance optimization.
  - Seems dubious, since it actually adds network latency.

- So we can’t apply a name constraint Property to all the trusted root CA’s, because they don’t exist in the Registry until they’re actually used to verify something.

- How can we work around this?
Pre-Downloading All the AuthRoot.stl Certificates

- There’s a Windows certutil command that fetches all the AuthRoot.stl certificates and saves them as .crt files.
  - Intended for enterprise environments where IT needs to vet each CA.
- If you then ask certutil to “verify” each of those .crt files, this triggers the code path in CryptoAPI that imports the certs from Windows Update in order to verify them.
- Result: all 420 root CA’s end up in the Windows Registry.
  - And now we can apply the name constraint Property.
Sandboxing

- Note that importing all 420 root CA’s can be done by a sandboxed user with no interesting privileges.
- Because all we’re doing is asking CryptoAPI to verify certificates.
  - This is an unprivileged operation, naturally.
- CryptoAPI does all the importing of certificates to the Registry for us.
An Even Simpler Way

- There’s a “Verify Certificate Trust List” command in Windows certutil, which downloads all the root CA’s, and verifies them for us, in a single step.
- We’ve integrated all of this into the Namecoin Windows installer.
- When you install Namecoin, the installer makes certutil import all 420 root CA’s, and sets a name constraint on all of them.
Demo of Negative Overrides in Microsoft CryptoAPI
So, What Have We Achieved?

- Positive certificate overrides (censorship-resistant TLS).
- Negative certificate overrides (interception-resistant TLS).
- Works with Microsoft CryptoAPI (most Windows applications).
- Works with Mozilla NSS (Firefox and most GNU/Linux applications).
- Minimal attack surface (sandboxing-friendly).
- Minimal blockchain storage usage (uses Layer 2).
Credits

- R&D and coding by:
  - Jeremy Rand (me)
  - Hugo Landau
  - Aerth
  - Ryan Castellucci

- Funded by NLnet Foundation's Internet Hardening Fund.
  - Funding sourced from the Netherlands Ministry of Economic Affairs.
Contact Me At...

- https://www.namecoin.org/
- OpenPGP: 5174 0B7C 732D 572A 3140 4010 6605 55E1 F8F7 BF85
- jeremy@namecoin.org

- Questions? Ask me on #namecoin on Freenode IRC.
- Thanks to the Monero Village for inviting me here!