
# Modeling Systemic Dependencies through Attack Surface Analysis with DANE

Eric Osterweil

Danny McPherson

Lixia Zhang

## What's so important about *systemic dependencies* and *attack surfaces?*

- Well, do we know what our Internet services rely on?
  - Are their foundations secure and reliable?

- We've become very good at using the Internet to make our lives easier
  - I can use one tweet to tell 50 million people that I just spilled my coffee on Emma Stone at Starbucks

- Our Internet services depend on networked systems, they are becoming increasingly layered and complex
- Ultimately, what are we paying for these conveniences?

# What networked systems do I need to tweet to 50 million followers?

# Systemic Dependencies

- What is the relationship between dependent systems?
  - We aim to map out what we gain, and what we *pay*
- But the application is so much broader: what systems do national critical infrastructures rely on?
  - DNS resolution for the Estonian government? Routing for Cairo? Cross modal threats from China? SCADA systems?
- Our engineering precepts tell us to build systems on top of other systems
  - But, does it make us more vulnerable to attacks, failures, etc.?
- Is there a difference between *vulnerability* and *availability*?

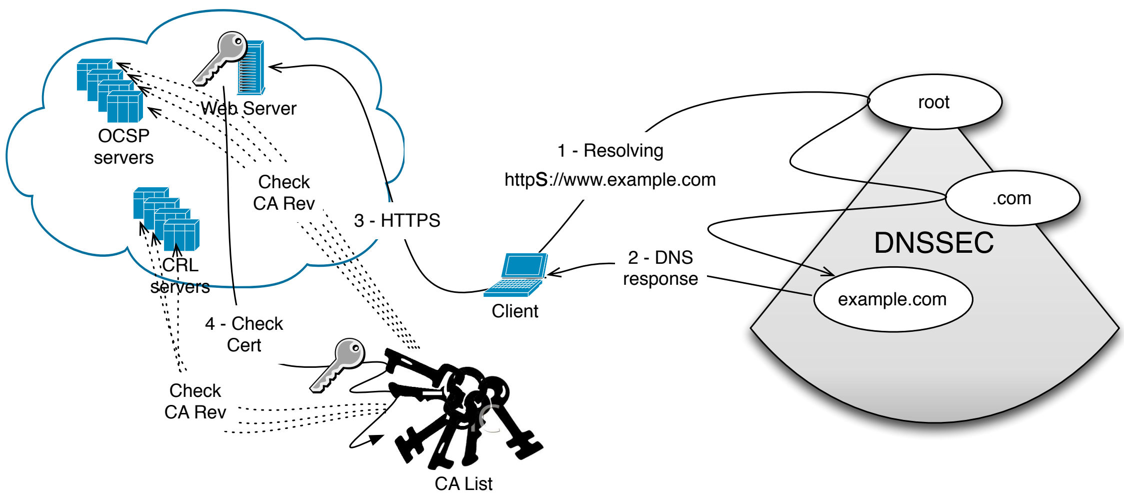
# Measuring Vulnerability

- Lots of ways to describe vulnerabilities, taxonomize attack vectors, etc.
- We can't always enumerate *all* of the attacks a priori
  - They're often only discovered reactively
- We turn to *attack surface* ::= what could *feasibly* be used to attack *a system's correct operation*
  - If I keep my attack surface small, I reduce my exposure
  - Quantifying targets and their values are different problems
  - But, a *quantifiable* definition is elusive
- We will *quantify* the attack surface of HTTPS using X.509 CA verification and compare it to HTTPS+DANE

# Outline

- X.509 CA verification and DANE verification
- Attack surface methodology
- Measuring the Alexa top 1,000
- Discussion and future work

# Certification Authority (CA) Verification

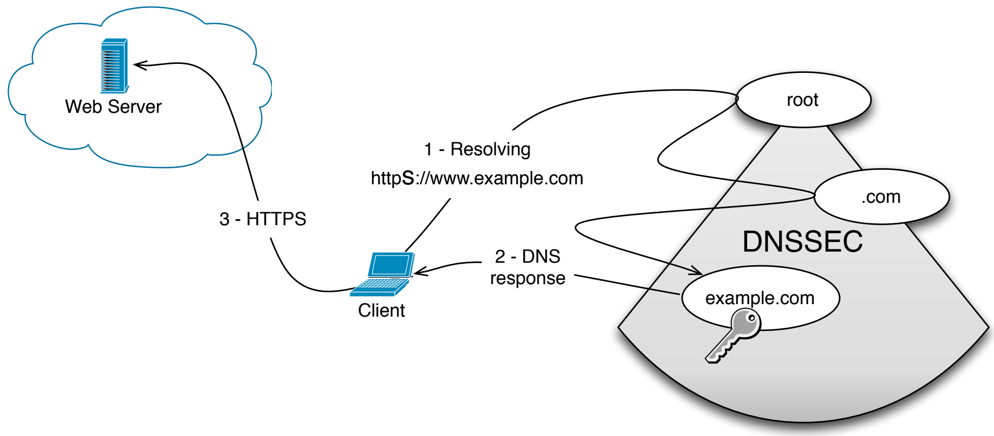



- Protocols like Transport Layer Security (TLS) need to be bootstrapped by cryptographic keys
  - Servers offer certificates and clients must verify authenticity
- CA verification uses a set of globally trusted authorities who can *each* vouch for *any* certificate's authenticity
  - Certificates represent previous verification: contain signatures from CAs, and point to revocation points for status checks

# So, when we go to Facebook…

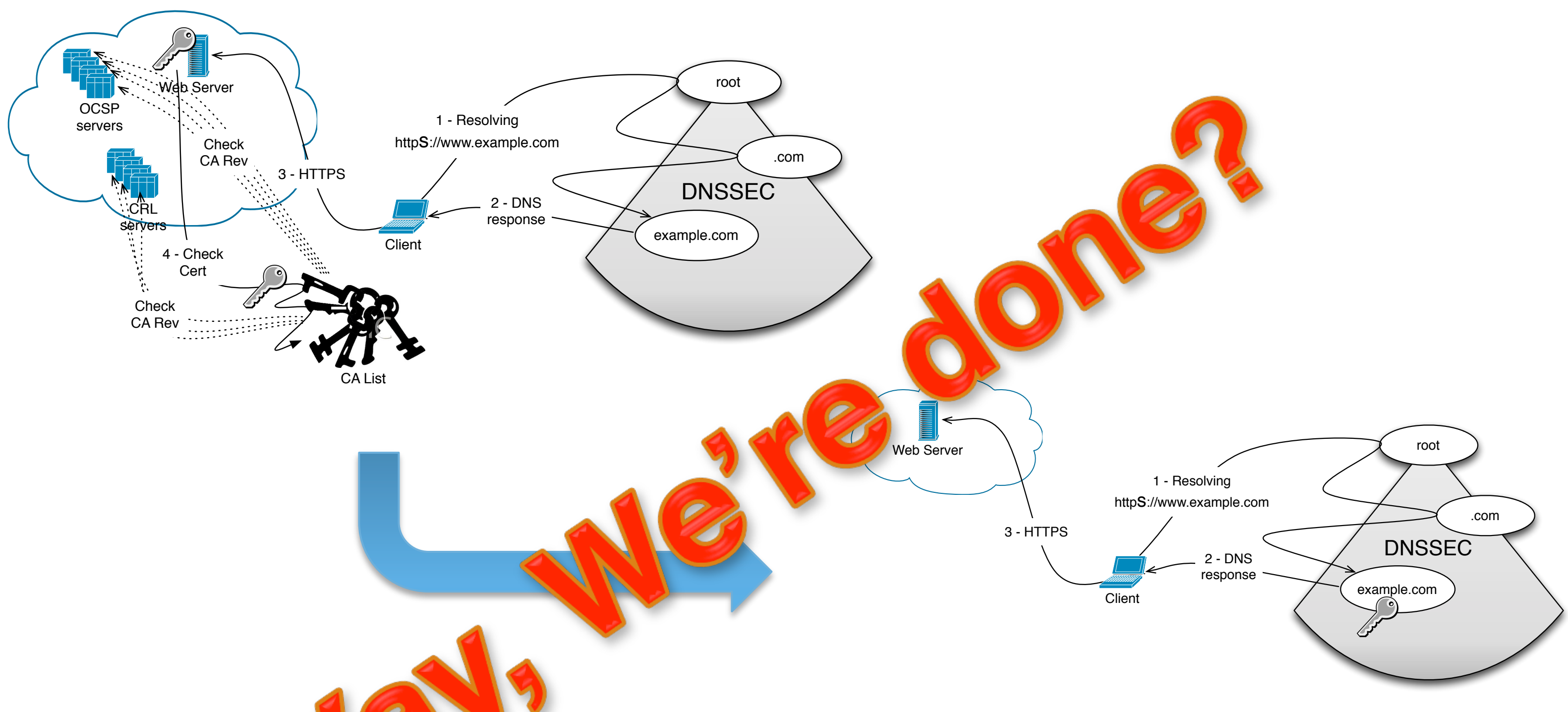
- The systemic dependencies are:
  - We first need to resolve its domain name (i.e. use DNS)
  - Next we connect to a webserver that we found from DNS
  - Then we pull an X.509 certificate down from that webserver
  - With that, we use a list of pre-bundled CA certificates that our browser has pre-configured to verify the webserver's certificate
  - Then check that the cert was not revoked since being issued
  - Finally, we exchange a TLS session key with the webserver, and start our online experience
- During this process, the attack surface includes any resource that could feasibly enable an attack
  - Possibly, a DNS secondary could give us a bogus answer that directed us to a malicious web site, or DigiNotar could have verified a bogus certificate, etc.

# DANE verification process




- DNS-Based Authentication of Named Entities (DANE)
  - IETF working group, and standards track RFC for TLS
- Observation: since even CA verification is frontloaded by DNS, why not do verification there too?
  - Rather than the multi-rooted X.509 hierarchies, DANE uses DNSSEC for verification
- DNS zones have TLSA record(s) that uniquely authorize cert used by web servers

# Look at what we just cut out…




- Qualitatively, a picture is worth 1,000 words: we can *see* that the attack surface is reduced
  - Recall: we're using Usage/Selector/Matching: 3/1/1
- By cutting out our CA check and revocation checks, we removed a lot of moving parts

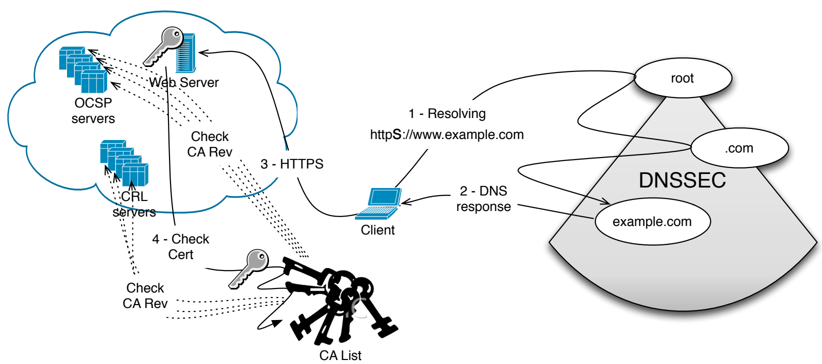
# Attack Surface Methodology

- Qualitatively, we can see that DANE's attack surface is smaller, but…
  - Talk is cheap, how can we quantify it?
- We use the measure of systems' systemic dependencies to quantify their attack surfaces
  - Added complexity and moving parts, increases attack surface
  - If a system needs *n* resources for ``*correct operation,*'' and its complexity increases this to *n+m*, its attack surface is greater
- But, we need a way to systematically decompose systems into the resources they need…

- Our methodology starts by identifying the logical procedural steps (processes) needed in *Functional Process Digraphs (FPDs)*
- Then, we map each process to the resources it needs, and *those* resources are the actual attack surface

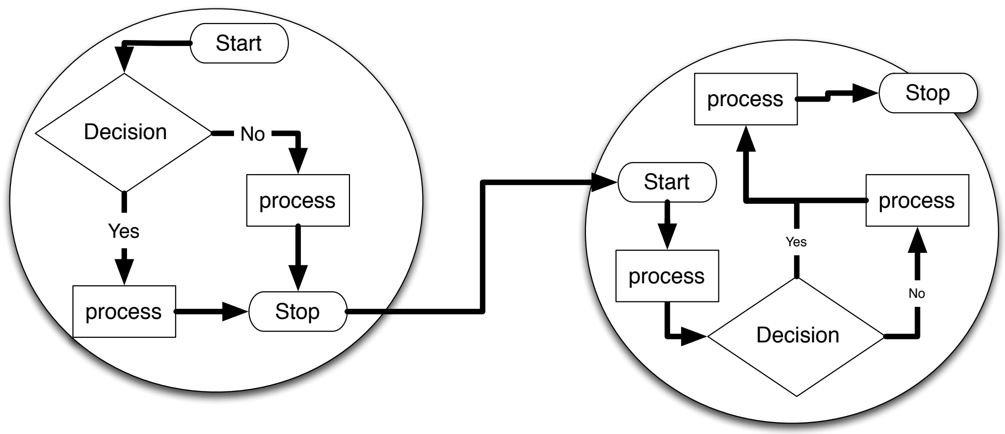
# Functional Process Digraphs (FPDs)

- Identifying all of the resources needed by networked systems can be non-trivial
- To identify *what* resources a networked system uses, we start by identifying the logical processes it uses
- FPDs are a way to codify networked systems
  - Every logical step/procedure becomes a process in the FPD
- Then, we look deeper at each process node and decompose *it* into a high-level workflow
  - Workflows don't fully describe all of the processes' inner-workings, but focus on how they access resources



# Example FPD

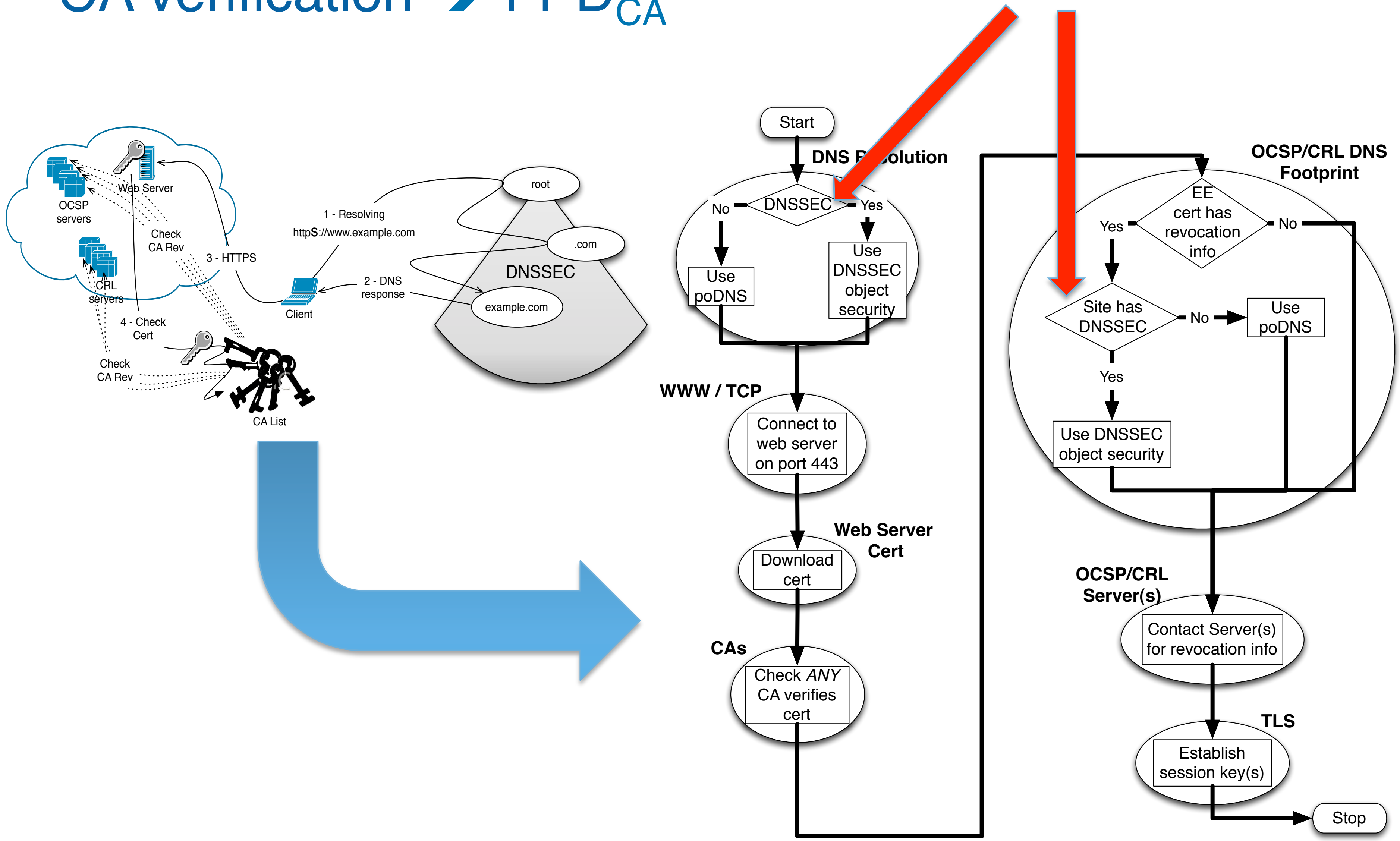
- Imagine a networked system that first required (say) a domain name resolution, and then connected to a remote system to search for a file
- Below we might see the DNS process checking and processing, followed by a server process searching for a file




- In creating an FPD, we must avoid the temptation to recursively codify processes that invoke processes beyond the semantic scope of the networked system
  - For example, it may not be semantically useful to identify the process of electrons interacting with copper atoms

# CA verification → $FPD_{CA}$

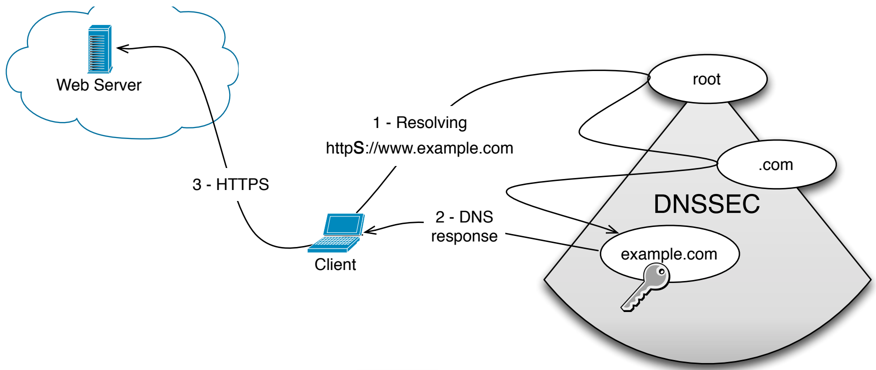
Note: DNSSEC is optional



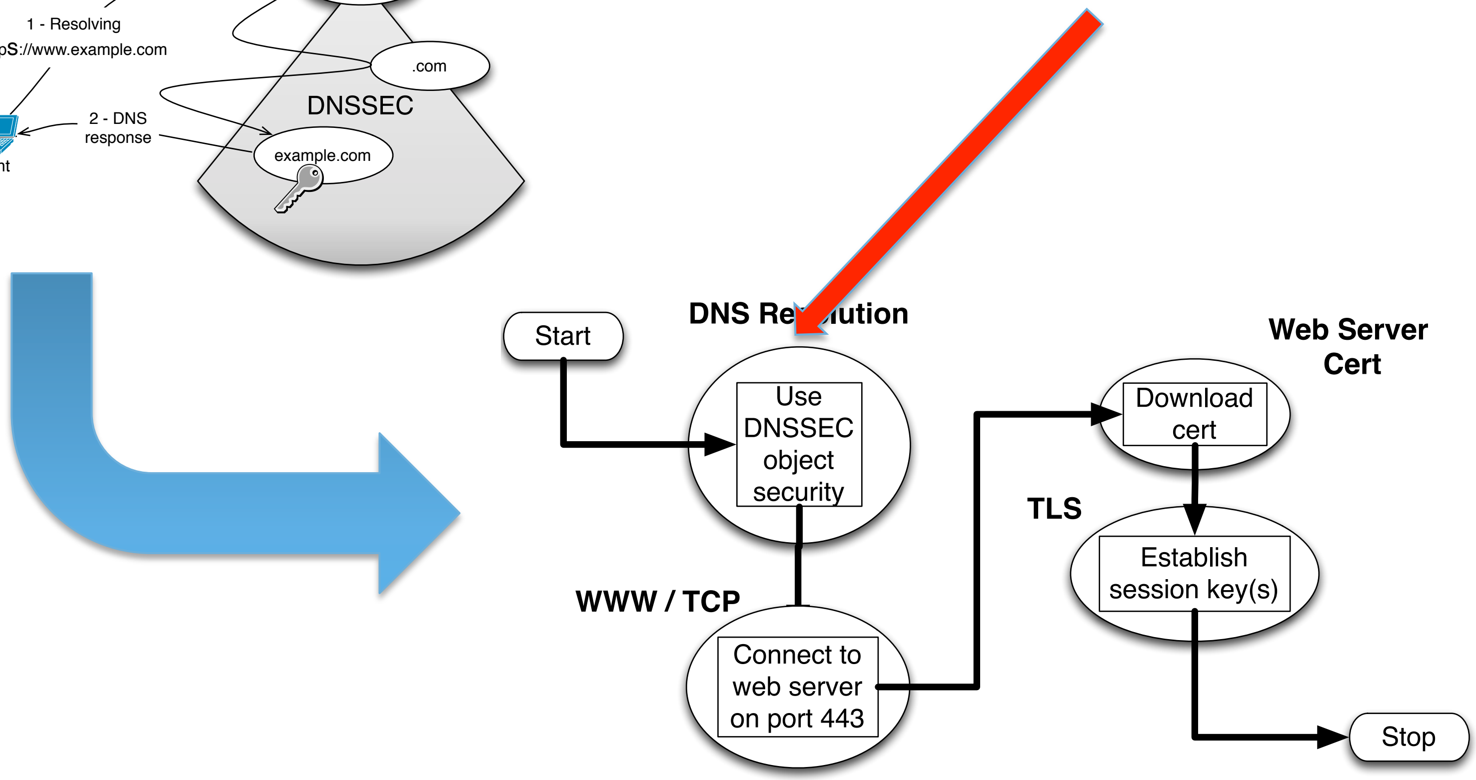
# CA verification requires…

- Domain name lookups
  - This could be plain old DNS (poDNS) or DNSSEC
- Connection over a network path to a webserver
- An X.509 certificate (from that webserver)
  - Could require a chain of certificates
- The list of CA certificates in the client browser
- Domain name lookups for each revocation point
  - CRL domain names and/or OCSP domain names
- Connection over a network path to a revocation point
- A TLS session key

# DANE verification → $FPD_{DANE}$




Note: DNSSEC
is mandatory



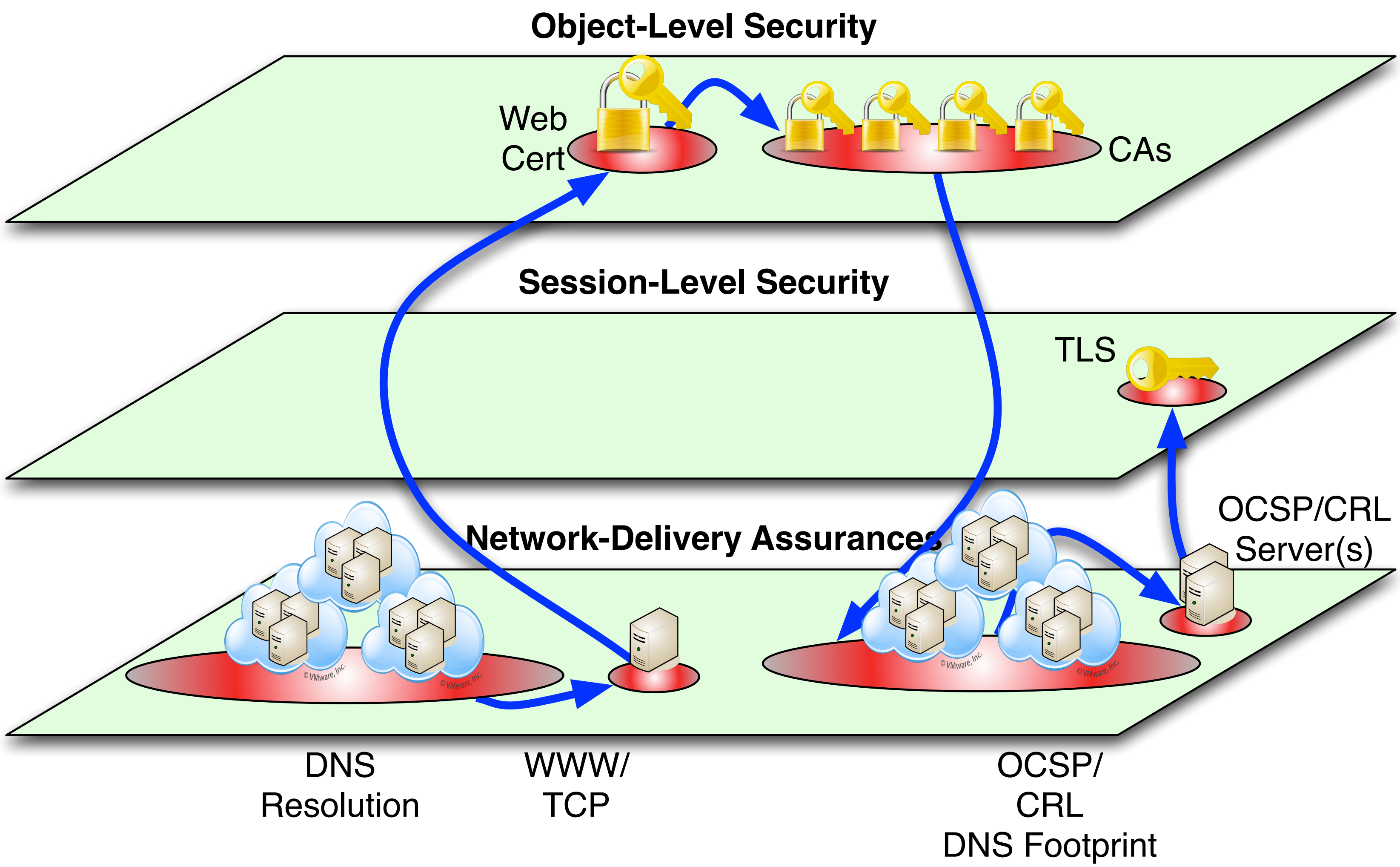
# DANE verification requires…

- Domain name lookups
  - DANE requires DNSSEC
- Connection over a network path to a webserver
- An X.509 certificate (from that webserver)
- A TLS session key

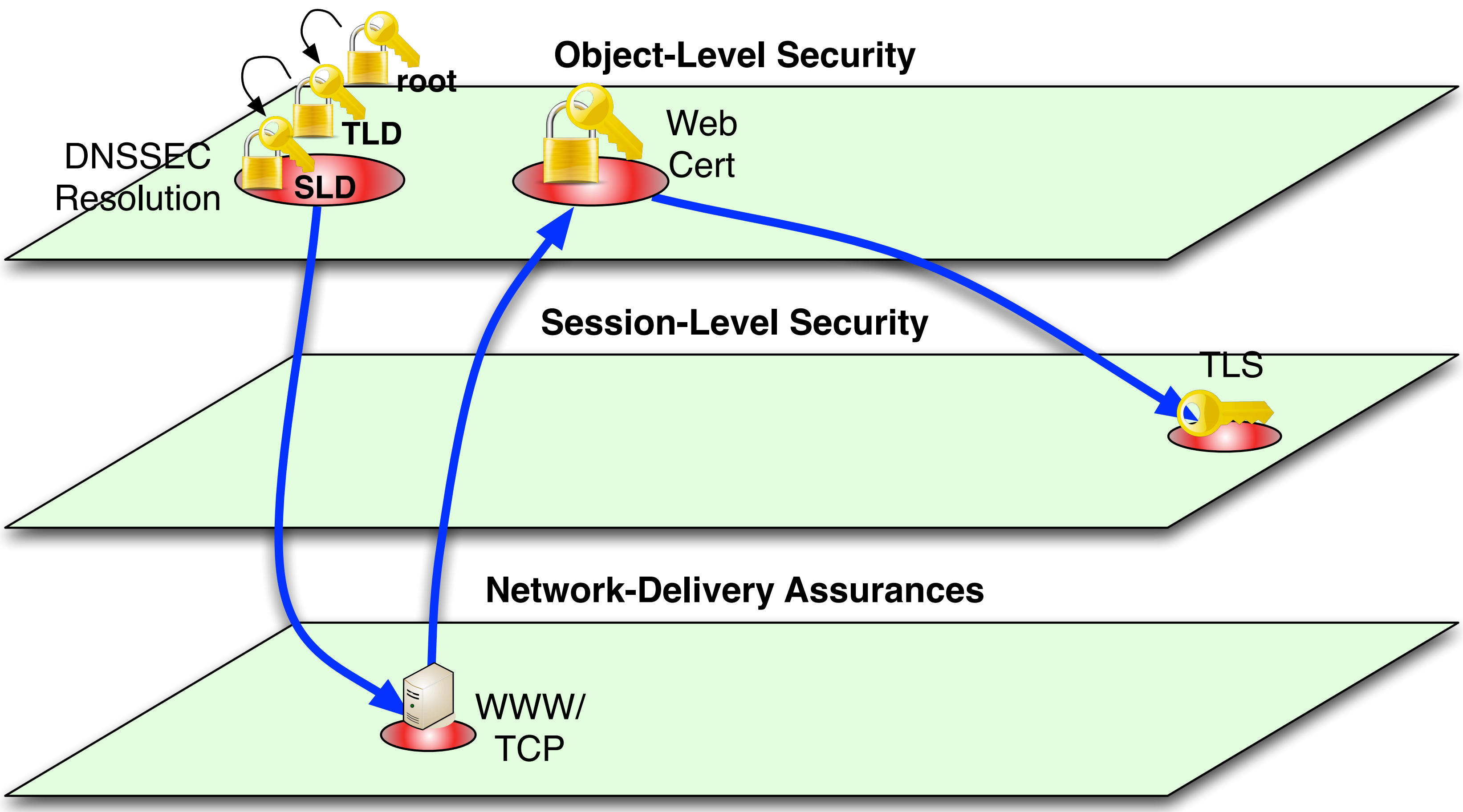
# Turning FPDs into something more meaningful

- We can visibly see one system's FPD is different than the other, but this is not quantitatively measurable
  - FPDs help us codify the control flow of networked systems
  - This helps us isolate their decision processes
  - *This* helps us focus on what resources are used
  - *That* helps us transform FPDs into resource graphs
  - Finally, *these* graphs form our attack surfaces
- The resources that each process uses inherit the same graphical relationships that they have in the FPD
- We chose a candidate set of classification *types* of resources that processes use
  - Network-Delivery Assurances, Session-Level Security, and Object-Level Security

# CA Verification's Attack Surface



# DANE's Attack Surface

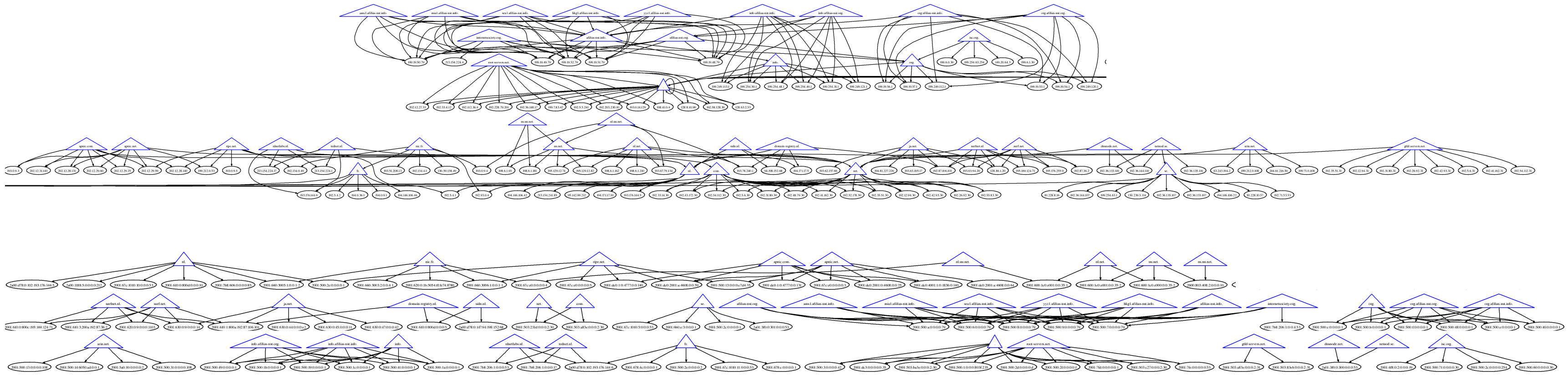


# Modeling resource element graphs

- These graphs offer us several novel facilities
  - We can quantify the sizes of each resource node
  - We have a formal dependency/relationship between fundamentally different types of resources (certs can be related to IP servers)
  - We can observe resource footprint changes from protocol decisions (re: DNSSEC)
- Consider this last point: note that DANE's requirement of DNSSEC moves the DNS portion of its attack surface out of the Network-Delivery Assurances tier, and into the Object-Level Security tier
  - This is a fundamental change!
  - In DNSSEC, secondary servers cannot successfully lie!

# Transitive trust graph sizes

- In DNS, zones depend on name servers, and those name servers can depend on other zones
  - This recursive dependency can lead to large graphs
  - The transitive trust graph of the IPv4 and IPv6 name servers of (for example) internetsociety.org are 119 and 85 nodes, respectively



- But, because DANE requires DNSSEC, these graphs are reduced to just the DNSKEYs in the chain of trust

# Evaluation

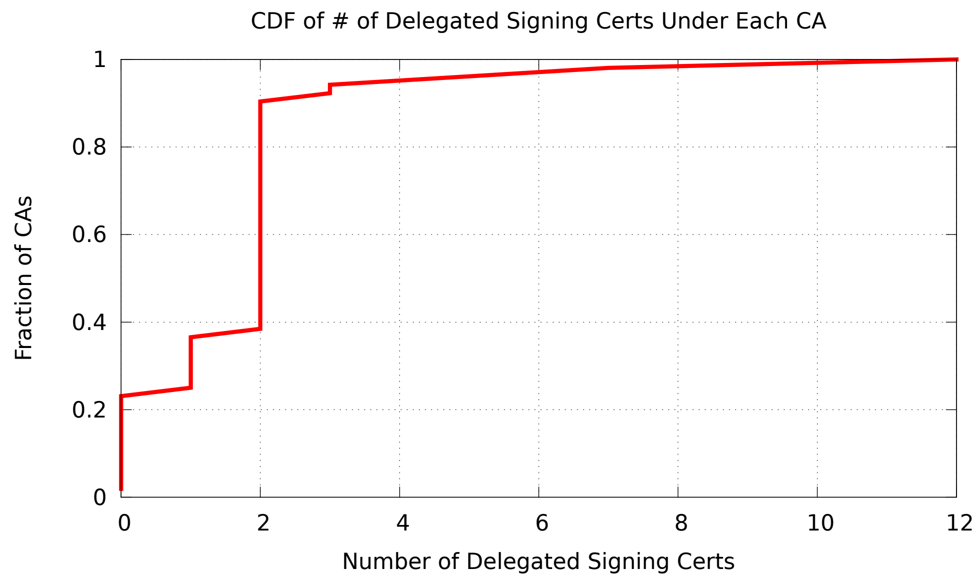
- We crawled the Alexa top 1,000 sites
- We examined:
  - How many ran HTTPS on their main site
  - How many ran HTTPS on www.<their-site>
  - How many had different certificates on <their-site> and www.<their-site>
- From this, and the certificates, we calculated the attack surface for each of them
- Then, we calculated what their attack surfaces would be if:
  - They deployed DNSSEC
  - They deployed DNSSEC + DANE

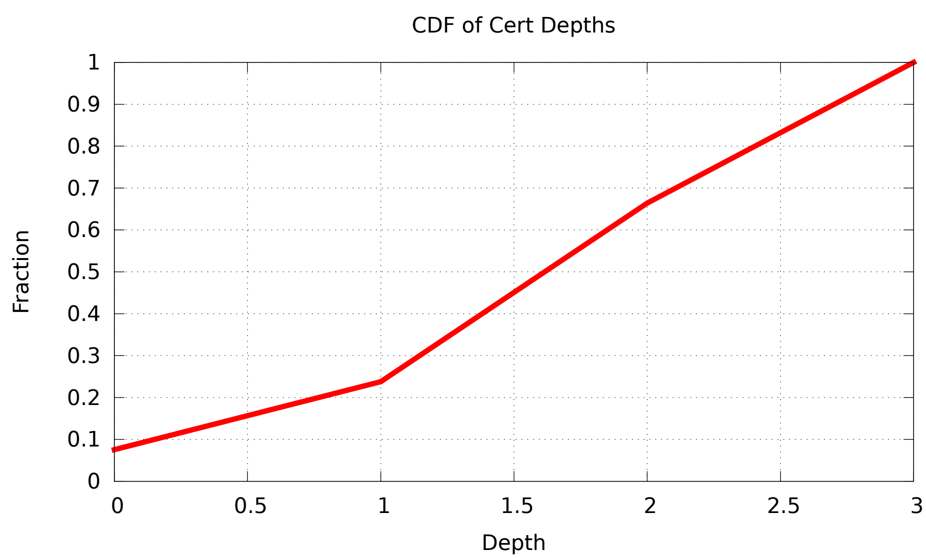
# CA list size

- The CA verification process requires us to calculate how large the CA set is
- However, this is client-vendor specific
- We can see that the list size can vary, so we evaluated using Mozilla's size: 169

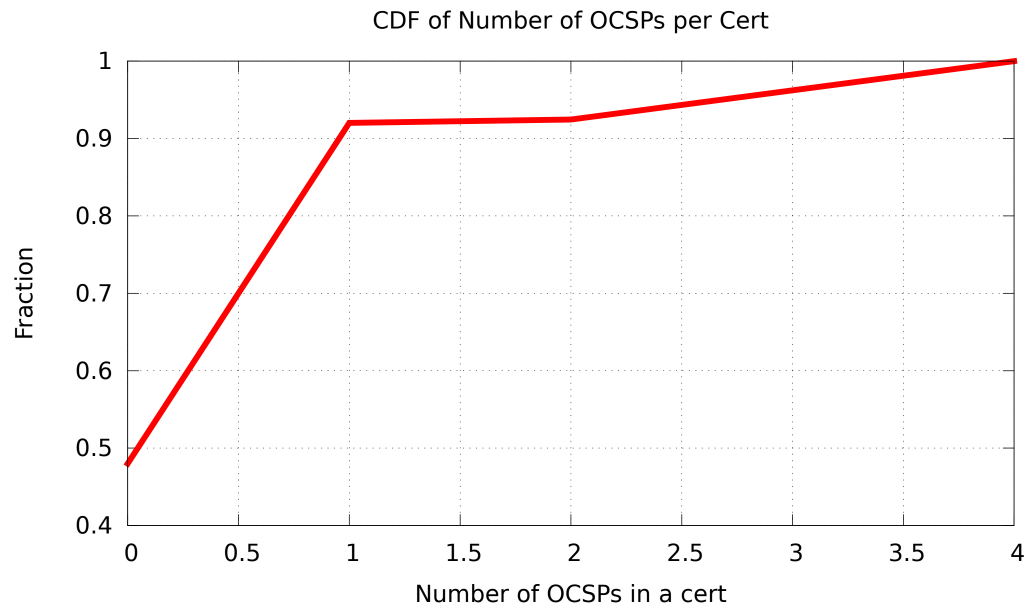
| Number of CAs | RP software |
|---|---|
| 169 | Mozilla |
| 167 | Windows |
| 167 | Apple's iOS 5 |
| 92 | Apple's iOS 3 |

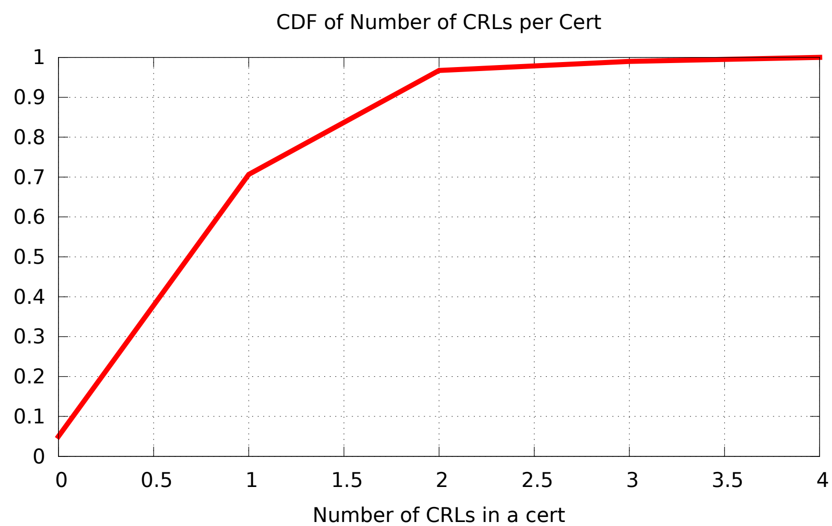
# CA delegation chains



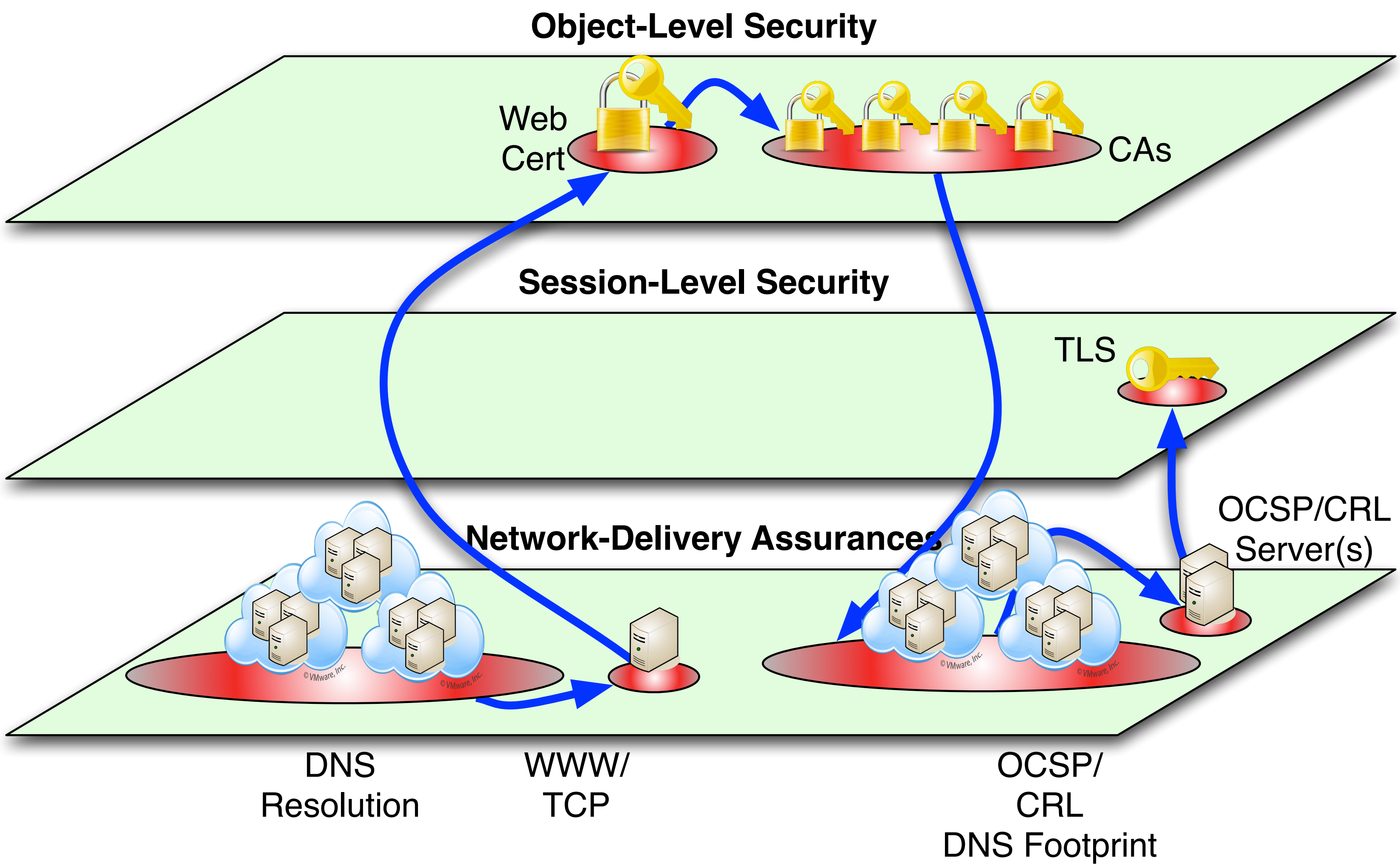



# Revocation details (from CRL/OCSP URIs)

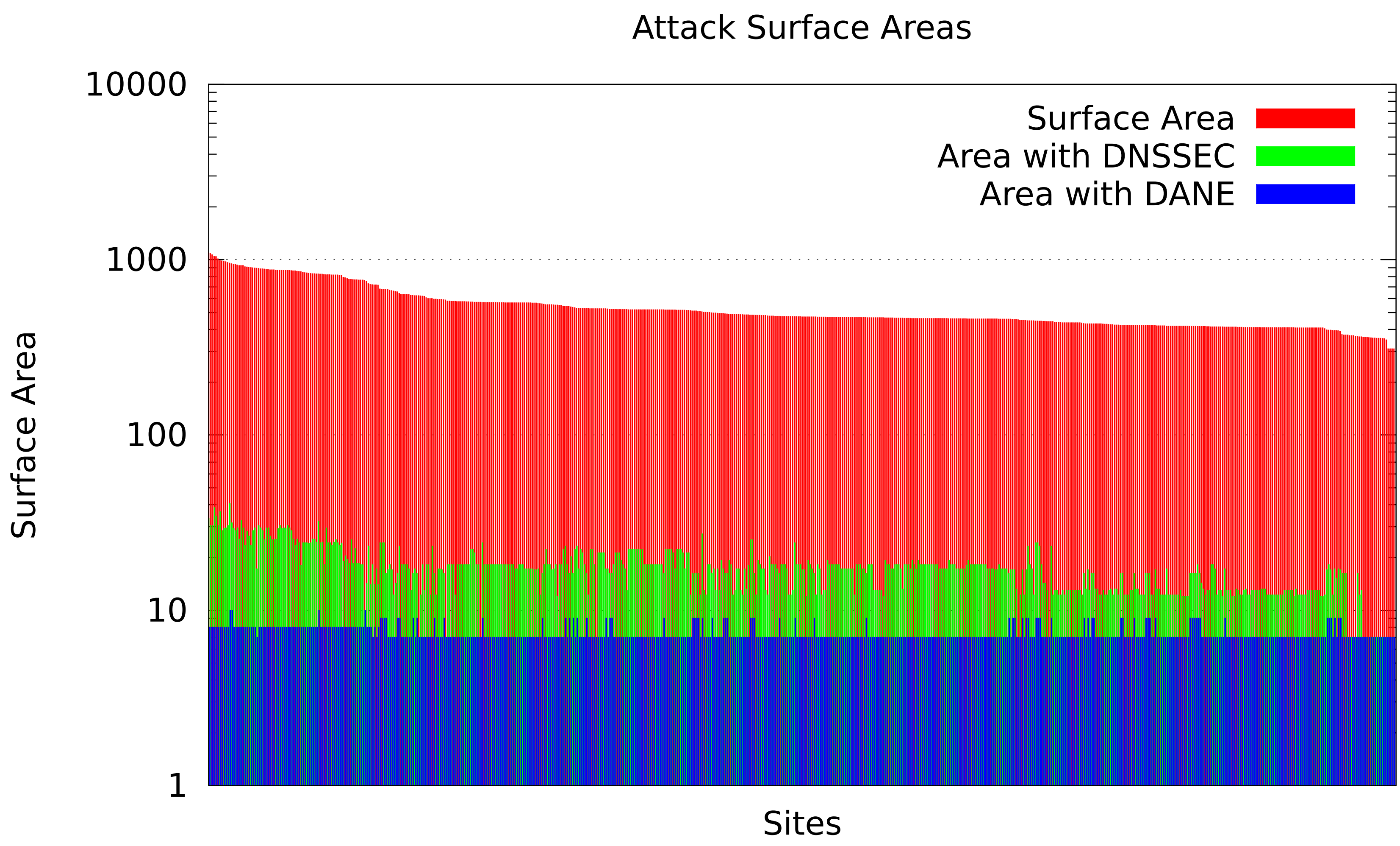


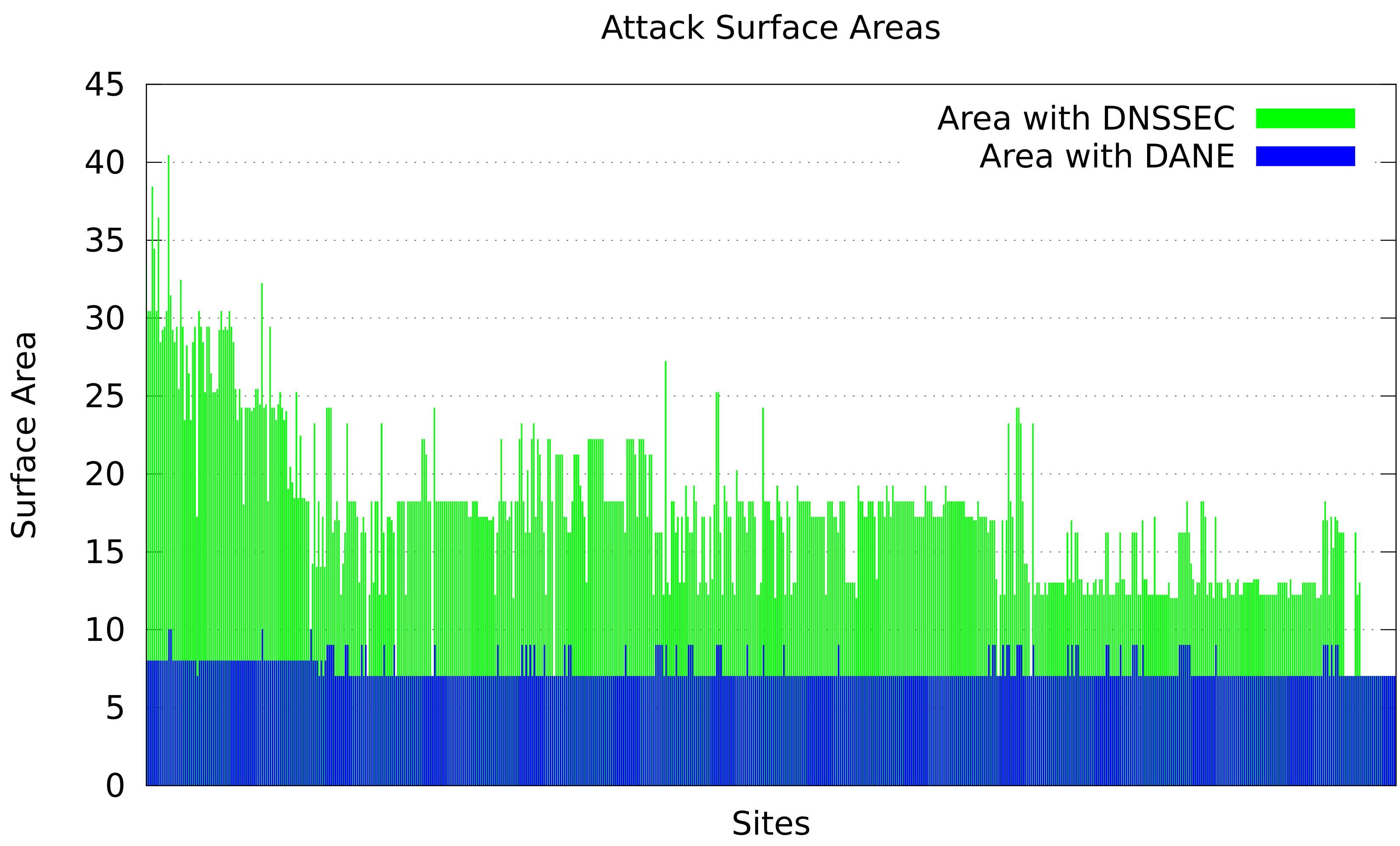
# CA Verification's Attack Surface



# Quantitatively comparing these two systems



# Non-log scale (just DNSSEC and DANE)



# DNSSEC (alone) makes a huge difference

- Almost no sites in the Alexa top 1,000 had DNSSEC deployed
  - Just deploying DNSSEC reduced attack surfaces by up to a factor of $10^2$
- Deploying DANE reduced attack surfaces by as much as a factor of $10^3$

| Max | Average | Min | Type |
|---|---|---|---|
| 1,104.92 | 531.68 | 309 | Actual measured attack surface |
| 40.42 | 17.29 | 7 | Attack surface if sites deployed DNSSEC |
| 10 | 7.38 | 7 | Attack surface if sites deployed DANE |

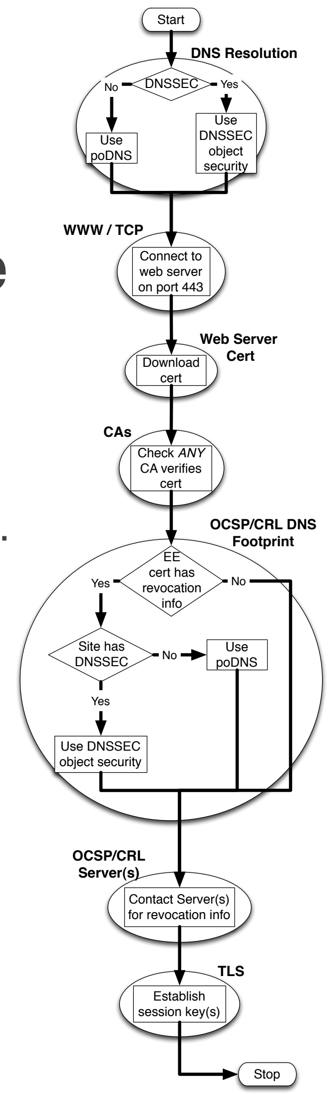
# Attack surface vs. Availability

- Attack surface and Availability can be seen as orthogonal concepts

- One might add additional servers to their deployment, but does this increase their attack surface?
  - It might: if any of these servers is able to compromise the *correct operation of the system*
  - This is why DNSSEC reduces the attack surface over DNS: secondaries cannot lie
- Conversely, one might find that higher redundancy does not add to attack surface if increasing resources is independent of correctness

# Using FPDs for kill chain analysis

- FPDs may lend nicely to kill chain-analysis
- Disrupting a step in an FPD can render the process moot
- Example: failing in the DNS stage renders · processing useless



# Thoughts going forward…

- There are lots of systems and protocols that have dependencies
  - Many times, these dependencies are non-obvious

- Nation states need ways to evaluate who and what they depend on so they know their vulnerability

- Enterprises need ways to know which of their systems have cyclic dependencies

- With this methodology we hope to offer a tool that lets people start evaluating what systems depend on

# Thanks!

powered by

VERISIGN™