Abstract—As more complex security services have been added to today’s Internet, it becomes increasingly difficult to quantify their vulnerability to compromise. The concept of “attack surface” has emerged in recent years as a measure of such vulnerabilities, however systematically quantifying the attack surfaces of networked systems remains an open challenge. In this work we propose a methodology to both quantify the attack surface and visually represent semantically different components (or resources) of such systems by identifying their dependencies. To illustrate the efficacy of our methodology, we examine two real Internet systems by identifying their dependencies. To illustrate the attack surface methodically, we present a visualization technique (called Functional Process Digraphs and resource graphs) to make analysis more intuitive, and iv) we show that leveraging existing services can avoid introducing new attack surface while maintaining usability.

I. INTRODUCTION

As we enhance our online services with increasing numbers of features, we tend to increase the interdependency among individual components. In the case of cryptographic protections, even just adding cryptographic key management can make systems far more complex than they were before. Moreover, failure modes often result in non-intuitive user interactions (such as pop-up boxes in web browsers when HTTPS encounters errors). By trying to ease the user’s life, protocols like HTTPS have inadvertently enabled certain types of attacks that the user cannot even detect (let alone remediate) [24]. We believe that streamlining the cryptographic protections by utilizing existing services can make them more available by end users, operationally usable by administrators, as well as less vulnerable to some forms of compromise.

One aspect of assessing the usability of a security system involves analyzing both the vulnerabilities that it protects against, and the vulnerabilities it has itself. In order to fully understand the vulnerability of the networked systems we depend on, it is important to understand the vulnerabilities of those systems on which they depend. However, it is generally held that even just knowing all possible attacks that can be launched against a given system is difficult, if possible (as evidenced by cybercrime trends [4]). Moreover, as network perimeters and silos continue to be eroded through mobile devices, laptops that move between networks, systems and resources that are deemed “protected” can become vulnerabilities as they become exposed to new threats.

A concept dubbed “attack surface” has emerged in recent years to describe the potential vulnerability that may be exposed by a system through the sum of its nested dependencies on other subsystems [22], [11], [10]. This term is generally used to mean the set of resources that a system relies upon which could feasibly be used to launch an attack against it. Thus, systems with larger attack surfaces are judged as potentially more vulnerable. Yet, the heterogeneity of networked systems, their complex dependencies, and the dynamism of their interactions have made it difficult to quantify or to illustrate the attack surface methodically.

In this work, we present four contributions: i) we believe we are the first to offer a quantitative definition of attack surface for networked systems, ii) we use this definition to codify a repeatable methodology, iii) we present a visualization technique (called Functional Process Digraphs and resource graphs) to make analysis more intuitive, and iv) we show that leveraging existing services can avoid introducing new attack surface while maintaining usability.

Our definition of “attack surface” refers to the specific set of resources that a networked system N uses, such that if any of them were to act incorrectly (by being compromised, subverted, replaced, etc.), N would not be able to discern a problem and could therefore malfunction. We define attack surface by modeling N’s control flow and analyze the way N gets used. That is, if a component of N queries for an answer, but that answer (such as a DNS Resource Record, RR) cannot be verified, then that server (or possibly an element along a network path) can undetectably lie. Alternately, if an answer can be verified by a cryptographic signature that is verifiable by an X.509 certificate (for example), then if that certificate’s private key were to be subverted, any false attestations it made would not be detectable by the networked system’s protocol. This replaces the RR and makes the certificate key-pair part of the attack surface.

To illustrate the efficacy of our approach, we quantified the attack surface of popular deployments of HTTPS, which uses the Transport Layer Security (TLS) [8]. We measured and compared the attack sur-
faces exposed by operational websites under two possible standards-based solutions employed by TLS: the X.509 Certification Authority (CA) verification procedure used by the “Web PKI” model [2], and the DNS-based Authentication of Named Entities (DANE) [1] protocol for TLSA [9]. Our results show that one of the fundamental protocol semantics of DNSSEC (its use of object-level security) can quantifiably reduce the attack surface of systems by up to two orders of magnitude, and that using DANE can reduce the attack surface of systems by leveraging existing DNS service. Our methodology also exposes a fundamental orthogonality between attack surface and availability, which we discuss further in Section VI.

II. COMPARING CA AND DANE VERIFICATION

CA Verification: The goal of the CA verification system [7] is twofold: it verifies the authenticity of the binding between a certificate and a domain name, and it allows the CA organization to vouch for the “trustworthiness” of the entity that is managing the domain name in question (and by extension, the certificate). That is, when a Relying Party (RP, an entity that is trying to validate the authenticity of a remote service), such as a web browser, wants to make a secure HTTPS connection to a website, it wants to be sure it has received the right certificate, and (in some cases) that the certificate holder (the domain name authority) is “trustworthy.” This simple operation implies that RP software must be able to verify certificates from the multitudes of websites it may visit, without knowing them ahead of time.

Today’s RP software uses a set of trusted root certificates that are issued by Certification Authorities (CAs). RP software vendors determine which CA organizations they trust, and pre-install certificates from these CAs in their software. Typically, whenever this set needs to be refreshed or changed, the vendor updates the list out of band via code updates. When an RP receives a certificate, it checks to see if that certificate has a chain of signatures that leads to a trusted CA. Then it performs various sanity checks on the certificate (specified in [7]: has it expired, is it formatted properly, etc.), it must also see if the certificate itself has been revoked. Each certificates can include URL pointers to where its revocation status can be checked. When certificate owners want to revoke their certificates, they instruct the issuing CA to put them in Certificate Revocation Lists (CRLs), or advertise them on Online Certificate Status Protocol (OCSP) servers, and RP software has to check them over the network.

The CA verification model involves numerous systemic dependencies on other systems: i) a recursive set of DNS lookups to find a webserver’s IP(s), ii) a webserver connection to get a certificate, iii) consultation of/verification with root CA list that is maintained out of band, iv) external check to see if certificates have been revoked, and then v) the TLS session generates session keys and begins communication. This model is vulnerable to an attack dubbed “attack on one, defeats all” [17], in which any CA can issue a certificate for any domain name; asserting validity without authorization.

As a result, suppose a website www is serving a certificate CA from CA A. If an adversary has compromised a different CA Comp, and uses it to issue a forged certificate CComp that purports to belong to www, then the RPs have no way to programmatically detect or determine that CComp is not an authorized certificate for www. This attack was used when DigiNotar was compromised and issued CComp certs for Google [24]; several other instances of this attack are well known too [5], [13], [24], [16]. These sorts of problems have led to alternative verification methods.

DANE’s TLSA: The IETF has standardized a new approach, called the DNS-based Authentication of Named Entities (DANE) [1], as an alternative to the CA model. At its core, DANE is an architectural suite for publishing the cryptographic keys in DNS for various protocols and is composed of a growing set of standards. Its development is driven by two simple observations: i) any transaction that uses a DNS domain name (as opposed to a hard coded IP address) begins with a DNS look up (often for an IP address), and ii) DNS transactions can be authenticated by DNSSEC, which is now operational and has a growing deployment [20] and set of tools [12].

In this work, we focus on the use of DANE to distribute TLS certificates (TLSA) [9] to TLS clients. DANE’s TLSA allows domain owners to control and manage their own verification, thus removing the systemic dependencies needed by the CA verification model. DANE’s efficient design also lets DNS administrators manage the life cycle and operations of their own keys, and makes authorization more transparent by signaling explicit domain-to-certificate bindings to RPs.

The TLSA standard uses a naming syntax to encode which port and protocol an X.509 certificate is authorized for into the DNS name. Figure 1 shows how DANE’s TLSA RR is used. In the remainder of this paper, we focus our analysis on TLSA records whose Usage is DANE EE certificates.

Previously, RPs had no way to determine if a certificate they encounter is authentic, or merely certified by a compromised CA [24]. DANE eliminates this vulnerability by a technique called stapling that explicitly maps a domain name to a certificate (or a Trust Anchor). RPs can then verify what certificate they should see when they contact a corresponding web server, and DNSSEC RR sets include the necessary cryptographic signatures to allow RPs to verify the authenticity of the stapling. Also, the complicated systemic dependencies needed for
revocation are obviated by DANE’s design. In DANE, deauthorization of a certificate is implicitly handled by simply removing the TLSA record from a zone. After that, the certificate will only be authorized for as long as it exists in DNS caches.

Consider the following illustrative example, when an RP wants to connect to www.example.com, it queries for both the IP address records for that domain name, and the TLSA record type at _443._tcp.www.example.com (and uses DNSSEC to verify all returned values). If it gets back a TLSA record for that name, it knows that there is an HTTPS server running there (so attacks like sslstrip will not work), and exactly what certificate to expect from that web server (the web server’s certificate must match that TLSA); if that certificate comes from a CA, only that CA can attest to the certificate (stapling). After that, no other verification or revocation needs to be done, and the RP can query the specified web server.

III. AN ATTACK SURFACE METHODOLOGY

We define attack surface as a measurable set of active elements that might be used in some way by potential attack vectors and adversaries, if they were to act incorrectly. This allows us to concisely quantify the attack surface of CA verification and DANE. While our methodology is unambiguous about which elements contribute to the attack surface, we submit this as just one candidate way to systematically define the attack surface for these specific systems, others may define attack surfaces differently.

We quantify attack surface by modeling the way a networked system operates, by tracing its control flow, and enumerating the resources it uses to verify the correctness of its transactions. We begin by modeling a networked system as a set of protocols, systems, procedures, etc. We collectively call these a logical set of processes P. Each process pi ∈ P is composed of both a set of functional logic (the things it does, the decisions of which data it uses and trusts, etc.), and the set of resource elements that it uses (the data it uses, the network elements it needs to contact, etc.). Each process becomes a two-tuple: pi = {Wi, Ri}, where we define Wi as a workflow in a system that describes how a process acquires and determines the trustworthiness of data (or network resources, etc.), and where we define Ri as a resource graph of the elements Wi.

Our goal is to use the set of all Ri graphs by all processes in P as our attack surface. To this end, we break our methodology down into three general phases. We first identify the processes involved in P, by creating a Functional Processes Digraph (FPD), and denote it GFPD = (P, E). This FPD identifies the set and functional interactions (adjacencies) between all the processes involved. Next, we determine the internal workflow Wi for each pi ∈ P. Then, in the final phase, these graphs identify the resource elements in each process’ Ri graph.

Processes in the FPD: In the process digraph GFPD = (P, E), we represent each separate protocol and system as a separate process pi ∈ P, and for each process pi that leads to an invocation of process pj (where {pi, pj} ∈ P), we add a directed edge e(i,j) ∈ E. So, for example, if a set of DNS queries prompts a web browser to create a TCP connection to a web server, then an edge will connect the DNS process with the web server’s TCP process in the FPD.

We focus our attack surface analysis on those processes that are visible to, directly involved in, invoked by, and involved in protocol actions and directly specified at the same logical layer(s) of those actions. Rather than recursing to resource elements all the way down to the physical layer, we represent just a single level of abstraction (i.e., a network path instead of elements at all layers 1-4). We make a simplifying assumption that components that are out of the view of our verification process (while clearly candidate attack elements that could influence our attack surface) are better described in another system’s view of its attack surface.

CA Functional Process Digraph: Since the CA verification model for TLS is different than the DANE model, we create separate process graphs for each of them. GC,A:

DNS → Web connection → web cert → CA list → Check revocation → TLS

First, web clients go to DNS to look for a domain name-to-IP mapping for a web site. If DNSSEC is properly deployed, DNS data can be verified. Then they initiate a TCP connection to the returned web server, which returns an X.509 certificate. The client then validates the certificate by checking that the certificate’s reported CA is in the client’s list of trusted CAs, and that each link in the cryptographic chain from that CA.
verifiable. Then, the client examines the certificate to see if it specifies any CRLs or OCSP servers. If these exist, it queries DNS, then it queries the revocation site(s). Though, recent work suggests these mechanisms can be unreliable [3]. Finally, if the above is successful, the client creates a TLS session.

**DANE Function Process Digraph:** \((G_{DANE})\):  
DNSSEC \(\rightarrow\) Web connection \(\rightarrow\) web cert \(\rightarrow\) TLS  
This verification FPD is a logical subset of the CA validation FPD \(G_{CA}\), except in \(G_{DANE}\) DNSSEC is mandatory. That the simple addition of DANE’s TLSA record overloads the DNS dependency, and removes an RP’s need for all of the related processes, so we can say that \(G_{DANE} \subset G_{CA}\).\(^1\)

**Creating Resource Element Graphs:** The verification workflows defined in our FPDs provide the direction needed to identify the resource element graphs \((R)\) which collectively define our actual attack surface. For example, we can see that the DNS and a web server’s X.509 certificate are needed by both CA and DANE verification. However (as we noted earlier), these systems and protocols are not all specified in the same logical way: DNS is concerned with name resolutions, DNS server IP addresses, communication with name servers, and more. By contrast, X.509 is a cryptographic certificate suite, and even the semantics of its protections are described in a fundamentally different way than a network protocol like DNS; it specifies a notion of object-level security that is session agnostic.

In order to quantify the attack surface that spans these conceptually different resource types, we represent the set of \(R_i\) graphs in a way that focuses on the type and number of security elements each process uses. We do this by creating a model where different types of resources exist as elements in different tiers of our resource graph. In other words, we create a set of tiers in our graph where each tier models a separate (but somewhat general) type of element in the spectrum of security. We seed our model with three general tiers: i) object-level security, ii) session-level security, and iii) Network-Delivery Assurances.\(^2\)

**The CA Resource Tiers:** Here we project the process from \(G_{CA}\) into Figure 2 and then discuss why the placement of each process is in a specific tier.

**Modeling DNS:** In order to quantify the elements involved in the DNS process \(p_{DNS}\), we leverage the prior work [19], [21]. These works describe the notion of Transitive Trust in the DNS, and illustrate that simple DNS lookups often involve recursive (not strictly hierarchical) dependencies, and (without DNSSEC) any

---

\(^1\)Process workflow details are omitted, due to space restrictions.

\(^2\)While this may not be an optimal or complete set of the tiers that others may use, we use these as general examples of types of security in which many different protocols and systems can be aggregated.
term for X.509 certificates is designed to let RPs determine if a certificate that is about to be used has been revoked (and therefore should not be used). Certificate revocation is specified as CRL URI(s), OCSP URI(s), or sets of both. Regardless, the RP must first use DNS to locate the server(s) responsible for serving the revocation info, and then contact those servers. Thus, for \( p_{rec} \), the resource element graph (\( R_{rec} \)) is computed as we previously did for the initial DNS lookup and for the web server: the Transitive Trust graph and the server IP addresses in the Network-Delivery Assurances tier.

**Modeling the TLS Session:** Finally, the TLS process \( p_{TLS} \) will set up a session key, and that key is in the Session-Level Security tier of \( R_{TLS} \).

**The DANE Resource Tiers:** Because \( G_{DANE} \subset G_{CA} \), the tiered attack surface representation is also a logical subset. Figure 3 shows that DANE cuts out the CA verification hierarchy and the certificate revocation surface. However, the Figure also illustrates the effect due to DANE’s requirement to use DNSSEC: \( R_{DNS} \) is in the Object-Level Security tier.

**Measuring and Comparing the Attack Surface Areas:** In our model, we consider the attack surface \( S \) to be defined as the union of resource element graphs in our \( G_{FPD} \): \( S = \bigcup_{i=0}^{G_{FPD}} R_i \). Each element in each tier of our resource graphs can have different importance in the role of compromising the overall system (some things are more useful in some attacks than others). For example, is compromising a DNS secondary server as easy or useful as compromising a CA’s certificate? This would likely depend on the adversary’s attack, motivation, goals, etc. and a weighting could be different in different circumstances. One of the benefits of the attack surface model is that these orthogonal importances are independently assignable. Our evaluation models each element as having the same weight in the overall \( A = \sum_{i=0}^{G_{FPD}} |R_i| \)

### IV. Evaluation

To calculate \( A \), we used the most popular 1,000 websites described by Alexa [23] to find examples of popular TLS deployments. To examine the transitive trust footprints we implemented the technique described in [19], and ran it on each zone in our site list. Next, we considered the surface area of the web servers for each site. However, queries for address records can sometimes return different answers to different clients (e.g., CDNs). As such, we assign each site’s web server as having area = 1, as a lower bound that, if anything, underestimates the attack surface area. Next we calculate \( R_{cert} \) for our web sites. Interestingly, 18.03% of the sites that offered HTTPS at both domain names (e.g., example.com and www.example.com) used unique certificates for each. Each of these certificates expose a different attack surface. Of the 616 sites that ran HTTPS, we found 702 different certificates. However, recall the \( R_{CA} \) portion of the overall attack surface includes the key pairs (represented by certificates) that root CAs delegate to \((C_{deleg})\), which are also used to sign certificates. From the data in some existing measurement, we calculated the average number of delegated signing certificates from each root to be 1.79. In the case of CA verification, the client’s RP software dictates the list of CAs that are used in verification. At the time of this writing, the size of CA lists in some popular RP software varied between 167 and 169 CAs [6], [14], [15]. Across the certificates seen, 24.9% used only CRL-based revocation, 3.3% used only OCSP, and 40.7% used both. The remainder used neither of these revocation systems. Only 4.6% of the measured sites had DNSSEC deployed. While this is a small fraction, it is larger than earlier reports of DNSSEC’s penetration [18]. However, none of these sites had deployed DANE. Using Mozilla’s CA list to represent RPs, Figure 4 illustrates the attack surfaces \( (A_{CA} \text{ and } A_{DANE}) \) for each site we measured (note the logscale y-axis). This Figure illustrates that the attack surface areas for our measured sites are at least two orders of magnitude larger than what they would be with DNSSEC, and the DNSSEC surface area is one order of magnitude larger than it would be with DANE.

### V. RELATED WORK

Attack surfaces have long been discussed by a variety of communities. For example, the authors of No-Hype [22] use the concept of attack surface, but elided a formal definition by broadly classifying the logical boundary between guest operating systems and their hypervisors as an attack surface. Then, by eliminating the entire attack space between these two components, their attack surface went from loosely specified to none. In contrast, in [11], [10], the authors define ways in which to analyze single-instance software systems and detect how much programmatic attack surface they expose by examining and quantifying them through state
machines and I/O automata. The focus on vulnerabilities conflates the notion of an attack surface with the likelihood of being successfully subverted (i.e. attack surface + how an attack will be actually be realized). We feel that these are both important to model, but also that an attack surface definition is maximally useful when it accounts for resource elements that could feasibly be used in an attack even before the attack is known (i.e. those that can undetectable lie). Also, these works focus only quantifying only a relative notion of attack surface between successive versions of the same system (as opposed to comparing the general surface between different designs). However, this work’s I/O automata seem to be quite complimentary with our methodology.

VI. DISCUSSION AND SUMMARY

DNSSEC Collapses Attack Surface: Section III shows that the plain old DNS (poDNS) has trust relationships that result in each name server becoming a resource in the attack surface. However, our methodology uncovers that because DNSSEC (in general) offers Object-Level Security that is based solely on the DNS hierarchy, it effectively removes the ability of name servers to lie. This allows DNS secondary servers to run as untrusted systems and only authorities for a zone can issue verifiable data. Section IV shows reduction of attack surfaces by two orders of magnitude. DNS’ security model changes from Network-Delivery Assurance into Object-Level Security (DNS RRset become signed and self-verifying).

The Difference Between Attack Surface and Availability: Designers, operators, and other engineers often try to bolster the availability of their systems by augmenting them with multiple redundant resources (such as by increasing the number of secondary DNS servers, etc). Our analysis illustrates that vulnerability and availability are not necessarily competing notions, and can be made independent of each other. One could design a system in which increased availability results in additional attack surface, but this is not necessarily the case. Our definition of attack surface is only composed of resources that can lie, and thereby successfully subvert a system’s correct operation. Availability can be thought of as a measure of the probability that one gets data in the presence of unexpected failures. Higher redundancy can result in higher availability. It is true that higher redundancy can result in larger attack surface, if the correctness of the system depends on the correctness of the individual components. However, bolstering a system’s availability with resources that cannot lie does not increase attack surface with availability. For illustration we compare the attack surface of poDNS’ transitive trust with DNSSEC’s key hierarchy.

Summary: This work represents a first step towards the quantitative measurement of systems’ attack surfaces. Our methodology offers an intuitive way to visualize why the attack surface of DANE precludes the possibility of certain types of attacks, like those exposed by DigiNotar [24]. We also hope this paper will serve as an invitation to the community both to help further improve our methodology, and to develop quantifiable measurements for other networked security systems.

REFERENCES