PKI Layer Cake: New Collision Attacks Against the Global X.509 Infrastructure

Dan Kaminsky[1], Meredith L. Patterson, and Len Sassaman[2]

[1] IOActive, Inc.
[2] Katholieke Universiteit Leuven

1 Introduction

Research unveiled in December of 2008 [15] showed how MD5's long-known flaws could be actively exploited to attack the real-world Certification Authority infrastructure. In this paper, we demonstrate two new classes of collision, which will be somewhat trickier to address than previous attacks against X.509: the applicability of MD2 preimage attacks against the primary root certificate for Verisign, and the difficulty of validating X.509 Names contained within PKCS#10 Certificate Requests. We also draw particular attention to two possibly unrecognized vectors for implementation flaws that have been problematic in the past: the ASN.1 BER decoder required to parse PKCS#10, and the potential for SQL injection from text contained within its requests. Finally, we explore why the implications of these attacks are broader than some have realized — first, because *Client* Authentication is sometimes tied to X.509, and second, because Extended Validation certificates were only intended to stop phishing attacks from names similar to trusted brands. As per the work of Adam Barth and Collin Jackson [4], EV does not prevent an attacker who can synthesize or acquire a "low assurance" certificate for a given name from acquiring the "green bar" EV experience.

The attacks we will discuss in this paper fall into the following categories:

1. **MD2RSA Signature Transfer:** Verisign's MD2 root can be exploited by creating a malicious intermediate with the same MD2 hash as its parent and transfering the signature from the root to the malicious intermediate.
2. **Subject Name Confusion:** Inconsistent interpretation of the Subject X.509 Name in a PKCS#10 request can cause a CA to emit a certificate for an unauthorized Common Name. Existing PKCS APIs vary in their handling of Common Names and Subject Names, and these differences can be exploited in a number of ways which we will explore in detail.
3. **PKCS#10-Tunneled SQL Injection:** Certificate Authorities inserting PKCS#10 Subject Names into a database do not necessarily employ comprehensive string validation for the BMPString, UTF8String and UniversalString types, which allows SQL injection attacks. Given the special trust that a CA's database backend presupposes for the rest of the Internet, SQL injection is especially problematic.

4. **PKCS#10-Tunneled ASN.1 Attacks:** Certificate Authorities exposing a PKCS#10 receiver may be exposing unhardened ASN.1 BER listeners. ASN.1 BER is tricky to parse, with many possibilities for consistent and predictably exploitable attack surfaces. The PROTOS project [12] found a large number of vulnerabilities via the SNMP consumer, but it is possible that some of the ASN.1 BER parsers found in commercial CA implementations were not covered in the 2002 PROTOS lockdown and thus are still vulnerable.
5. **Generic SSL Client Authentication Bypass:** The MD2 attacks in this paper may have larger implications in certain deployments. An attacker with the ability to directly issue certificates — rather than just the ability to get an arbitrary X.509 Subject Name past a validator — gets access to the "Client Authentication" EKU (Extended Key Usage) attribute that controls whether a certificate allows for authenticating a client to a server. Since Root CAs do not normally issue certificates with "Client Authentication" set, some systems may not test for what would happen if such a certificate arrived. This may create a generic authentication bypass in some systems. A similar bypass may be extended from Stevens and Sotirov's MD5 collisions, in situations where the Client Authentication EKU (which is not present in the root certificate they attacked) is insufficiently validated.
6. **EV Hijacking:** EV certs were designed to address phishing attacks where a bank at `www.bankoffoo.com` is suffering attacks from the owner of `www.bank-of-foo.com` or `www.bankofoo.com`. They were specifically not designed to deal with the case where an attacker has a certificate, even a low assurance certificate, for `www.bankoffoo.com`, and the attacker has a DNS or other route manipulation attack, e.g. DNS cache poisoning [19]. Barth and Jackson have shown that browsers do not enforce a scripting barrier between `https://www.bankoffoo.com` (EV certified) and `https://www.bankoffoo.com` (Low Assurance certified). Thus, an attacker need simply proxy enough of an SSL session to get the main HTML of a page loaded in EV (thus causing the green bar), then kill the TCP session. After that, the attacker can host any script from the Low Assurance cert, and that script will inevitably be merged with the real site with no negative impact on the EV experience.

We will summarize the recent history of attacks against the CA infrastructure; describe the methodology used to discover the attacks listed above; investigate these attacks in greater detail; outline a principled approach for remediating these issues and what steps browser manufacturers, cryptographic API manufacturers and certificate authorities need to take; and finally, identify directions for future work.

2 Background

The SSL protocol [2] is used for encrypting reliable data flows from one endpoint to another. But encryption without authentication is worthless: one can easily end up encrypting information with the key of an attacker! SSL manages authentication via *certificates* — assertions of identity that are cryptographically signed by mutually trusted third parties known as Certificate Authorities, or CAs. Verisign is probably the Internet's most well known CA, but the CA infrastructure includes over 200 issuers, all of whom handle edge cases in slightly different ways [3]. During 2008, the CA system weathered a series of shocks. Mike Zusman of Intrepidus Research was able to bypass WHOIS validation at one CA by claiming his desired certificate — for Microsoft's `www.live.com` — was to be used "for internal servers only". The CA Startcom also discovered a competing CA that entirely failed to check whether a certificate requester was a legitimate representative for the domain in question.

Beyond these implementation flaws, Kaminsky [19] exposed the basic design of CA validation via both WHOIS email and HTTPS-via-IP-in-DNS as faulty by means of DNS cache poisoning attacks. If DNS is compromised at the CA, both the email and the HTTPS connection can easily be subverted. While DNS has been remediated at all known CAs, other route manipulation mechanisms, such as Pilosov's BGP attacks [11], create some continuing exposure (though the BGP stream is small enough, and logged enough, for firms such as Renesys to know immediately if such an attack took place).

The most widely publicized attack against CAs in some time occurred in December 2008, with Stevens and Sotirov's applied work against CAs that still used MD5 as their hash algorithm for certificate signing. MD5 had been known to be insecure since at least 1996, with a regular stream of findings against the algorithm, punctuated in particular by the generation of MD5 collisions in 2004 [20] and the extension of these attacks to chosen prefix attacks in 2007 [14]. Stevens and Sotirov demonstrated a real-world application of the chosen-prefix attack by finding a CA, RapidSSL, that used MD5 and generated entirely predictable certificates (in particular, the Serial Number and Signing/Expiration time fields) and giving it a PKCS#10 request that forced it to generate a certificate that had the same MD5 hash as an intermediate certificate they had already generated. They then transferred RapidSSL's signature to their intermediate certificate, creating a forged certificate that could issue certificates for `www.bank.com`.

Luckily, very few CAs used MD5 at that time, and since then they have switched to SHA-1. However, the use of SHA-1 does not prevent the attacks we describe below.

2.1 Current status of these vulnerabilities

In this paper, we discuss the vulnerabilities we have discovered as they pertain to versions of X.509 certificate vendors and browser software in early 2009, prior to our disclosure of these attacks to the affected parties. Many of the attacks we describe have been patched or mitigated through our collaboration with vendors, prior to our public disclosure in August 2009. In particular, the Verisign MD2 root certificate has been superseded in Internet Explorer, Mozilla, Chrome, Android, Safari, Opera, and NSS; Postfix has remediated the early null termination issue; and Microsoft's CryptoAPI has remediated early null termination and the integer-overflow inefficient encoding issue.

However, due to the issue of legacy code persisting long after security updates have become available, the potential for attacks executed using malicious X.509 certificates issued prior to the discovery of these flaws, and the impossibility of evaluating every X.509 infrastructure in use (especially those used internally to organizations whose CAs are not exposed to the greater Internet), these attacks remain relevant today. We refer the reader to the individual vendors for comprehensive attack resistance information.

3 Methodology

Although ASN.1 is a well-established standard, not all ASN.1 parsers are created equal. It is a complicated format, requiring a context-sensitive parser. Context-free grammars can easily be converted to parsers using a parser generator such as `yacc` or `bison`, but generating a context-sensitive parser is difficult in mainstream (i.e., strictly evaluated) languages [8]. Moreover, the ASN.1 specification is not written in a fashion conducive to implementing an ASN.1 parser with a parser generator. Thus, in practice, the ASN.1 parsers that X.509 implementations rely on are handwritten, and the likelihood that the parse trees generated[3] by two separate implementations will vary (in other words, that they implement slightly different grammars) is high. The context-free equivalence problem — "given two CFGs, F and G, determine whether $L(F) = L(G)$" — is known to be undecidable, and thus the context-sensitive equivalence problem is as well.

Therefore, since there can be no guarantee that two ASN.1 parsers that were not generated from a CSG specification actually parse the exact same language, we examined subtle differences in the ways that different ASN.1 parsers handle X.509 certificates. We also deliberately focused on unusual representations of key components of an X.509 certificate, such as OIDs and Common Names: if one implementation can be tricked into misinterpreting a sequence, S, as a desired sequence, S', we can get a CA using an implementation which does *not* misinterpret S to sign a certificate containing S, and any browser using the first implementation will treat the certificate as a valid, signed certificate containing S'. All of our Subject Name confusion attacks rely on this strategy, and until ASN.1 implementations can agree on a consistent, well-defined grammar from which to generate their parsers, it is certain that similar attacks will emerge.

4 Attacks

4.1 MD2RSA Signature Transfer

As late as 1998, Verisign was still issuing certificates using a predecessor of MD5, the MD2 algorithm [3]. Historically, in the choice between MD2, MD4,

[3] This is a simplification; most ASN.1 parsers do not explicitly generate parse trees that can be recovered, but mathematically a tree structure exists.

and MD5, MD2 offered the highest security level at the expense of speed [6]. However, ten years after RFC 2313 advocated MD2, a 2^{73} preimage attack was published [18]. Although 2^{73} is well outside the bounds of trivial computation, given that the previous attack was on the order of 2^{97} [9], this can be considered one mathematical advance away from a distributed-computation work effort.

Given that MD2RSA has not been used to sign certificates for over a decade, it is reasonable to ask whether it would matter if MD2 fell. Unfortunately, the answer is yes. Verisign's primary root certificate — which is trusted by all browsers, and required to validate certificates from sites such as `https://www.amazon.com` — is signed with MD2RSA. Anything that this certificate signs is fully trusted. However, signatures are only valid across hashes, and Verisign has signed its own root certificate's MD2 hash. Thus, if we can generate an intermediate CA certificate with the same MD2 hash as Verisign's root, we can transfer the RSA signature from the root to the intermediate, and the signature will still be valid. Like Stevens and Sotirov, this attack transfers a signature from a legitimate certificate to a forged one, using the preimage to keep the signature valid. However, this attack can be performed entirely offline: there is no need to trick a CA server into signing something it ought not to.

As of August 2009, Verisign has reissued its root certificate using SHA-1 as the signature hash. The MD2-signed certificate cannot be revoked, but as the new certificate replaces the old one in shipped browsers, the number of browsers still using the MD2 certificate will dwindle, rendering this attack obsolete. We informed Verisign of our discovery early in our work, and their SHA-1 reissue was a direct response to this.

4.2 Subject Name Confusion

Acquiring a certificate involves submitting a public key and a claimed identity to a certification authority, generally via an ASN.1 BER-encoded PKCS#10 request[4] submitted through a web form. RFC 2986 describes the full schema of PKCS#10 requests; however, due to the nature of ASN.1, the encoding reflects as little of the schema as possible, instead trusting that a decoder will have the schema compiled into it.

We focus on the Subject X.509 Name because it is at the heart of the trust model in certificates. An X.509 Name is an ASN.1 Sequence of Sets of Sequences of OID/String pairs. These pairs can represent many descriptors, including Country, Organization, and Organizational Unit, but in the context of web browsers, the only name that matters is the Common Name, since the name of the website being secured is compared against the Common Name. The

[4] In an ideal world, the ASN.1-based protocols mentioned in this paper would use DER rather than BER. Unfortunately, thanks to Postel's robustness principle — "be conservative in what you send, be liberal in what you accept" — real-world encoders are willing to accept loosely-encoded BER bytestreams when called upon to parse. As we shall see, this practice paves the way for subtle variations between what different CAs will accept, and therein lies substantial danger.

Common Name is thus the one element that a CA must validate correctly, or else it will issue a certificate granting rights for names that the user does not legitimately represent.

There are thus two classes of consumer for the same sequence of bytes: CAs and browsers. If a CA and a browser parse the same sequence differently, a CA may grant rights incorrectly, or a browser may misinterpret what entity a certificate represents. We now examine several real-world cases of this problem.

Multiple Common Names in one X.509 Name are handled differently by different APIs Consider an X.509 Name where 2.5.4.3 is an OID paired with a String, and this pair constitutes a Sequence (embedded in a Set) representing the Common Name. If the Name contains more than one Common Name Sequence, and each Sequence has the OID 2.5.4.3, which one will be interpreted as the Common Name? Unfortunately, this behavior turns out to be implementation-dependent. We identified four possible policies:

1. First: The Sets comprising the Name are scanned for Sequences with an OID of 2.5.4.3. The first one that qualifies returns the associated String.
2. All-Inclusive: Each Sequence that matches the OID has its associated String added to a list, which is returned to the caller.
3. Last: The Sets of the Sequence are scanned, and whenever a Sequence is found that matches the desired OID, the planned response is updated to contain only the associated String. The last Sequence to match has its String returned.
4. Subject: No filtering is done. The entire X.509 subject is returned, either as a string or as a list, and the caller must extract the CNs in which it is interested. In other words, this is a client-side policy.

OpenSSL's command-line tools use the Subject policy, and require callers to implement text parsers, which must themselves implement one of the above policies. The OpenSSL API provides functions which can be used to extract Common Names; in a few lines of code, one can retrieve a list and iterate through it. However, we discovered that many open-source projects only retrieve the first matching CN — in fact, we could find *no* examples of open-source projects that process this list properly [17, 1, 21, 16, 10].

We next consider browsers. Internet Explorer, which uses Microsoft's CryptoAPI, follows the All-Inclusive policy: if *any* CN in the X.509 Subject Name matches the domain being browsed, then IE assumes that that CN has been validated by the issuing CA. This technically allows an attacker to shoehorn as many CNs into a Subject Name as he wants — the specification never mentions a limit — though in practice CAs limit the size of certificates they will generate. Still, this allows for a degree of parallelization in attack generation against IE.

NSS, the cryptographic library behind Mozilla Firefox, respects only the *last* CN in the Subject Name. This would limit an attacker to one malicious name per certificate — except that the name may be a wildcard, which NSS does not restrict against. As such, one successful breach against one CA will allow SSL bypass against all names under Firefox.

Inefficient BER encodings of OIDs can lead to some APIs recognizing the OID of Common Names Validating the X.509 Subject Name in a certificate request against the (somehow) validated identity of the user requesting a certificate is the task of the CA. In practice, we found that validation is limited to the Common Name; all other fields, e.g., Country, Organization Name, etc., are ignored. This leads to an even more hazardous source of disagreement between a browser and a CA: what happens when they disagree on what constitutes a Common Name in the first place?

As we know, a CN is an ASN.1 BER Sequence consisting of the OID 2.5.4.3 followed by a String containing the name of the website being authorized. However, BER's flexibility with respect to byte-level encoding means that more than one possible encoding can be interpreted as 2.5.4.3, whether that behavior is desired or not. We have identified two ambiguities in the ASN.1 Basic Encoding Rules which can lead to this condition.

Leading-Zero Padding. An OID is not encoded using the textual representation of its digits and '.' separating nodes; rather, the encoding uses base-128. Extra 0x80 bytes can be introduced to add leading zeroes to a node, e.g. 2.5.4.0003. OpenSSL's OID resolver catches leading-zero padding — it does not interpret 2.5.4.03 and the like as the Common Name OID — but its textual representation of 2.5.4.03 is 2.5.4.3. Any implementation which mistakenly operates on this representation instead of the parsed OID is in for a nasty surprise. Worse, however, is CryptoAPI, whose OID parser happily strips off leading-zero padding, interprets 2.5.4.$\{0\}*3$ as 2.5.4.3, and resolves it to Common Name. Assuming the CA passes 2.5.4.3 (the textual form) into the final certificate as yet another unrecognized element of the X.509 Subject Name, IE will allow an attacker full access to any name he wants.

Integer Overflow. Since an OID is encoded as a base-128 integer, which is then converted to a native form, an ASN.1 parser which fails to take into account the fact that these integers are unbounded may fall victim to integer-overflow attacks. This is as simple as passing 2.5.4.18446744073709551619 as an OID, since $18446744073709551619 = 2^{64} + 3$. OpenSSL wisely uses a bignum library, and is not susceptible, but until recently, CryptoAPI expected integers to be no larger than a 64-bit unsigned long, and mistakenly recognized anything congruent to n in a field modulo 2^{64} as n. Thus, IE was easily tricked into accepting *anything* as a CN, simply by passing an OID that overflows.

Early null terminators in an X.509 Name can cause some APIs to recognize different Common Name values [5] Having explored the semantics of the CN field itself and how OIDs are recognized, we now turn to the parsing of the CN string. Since ASN.1 BER encodes its strings "Pascal-style", with an explicit length field, rather than "C-style", with a string ending at the first 0x00 value, a string with one or more NULL values is still valid BER as long as its length field is correct. Two problems arise from this: first, an incorrect length field can

[5] This attack was independently and simultaneously discovered by Moxie Marlinspike, who presented it at Black Hat 2009 the same day that we did.

force reading or writing of data outside the blob being parsed, and second, once the binary data has been resolved to a string, C-style interpretation can cause unexpected behavior.

Consider an X.509 SN containing `CN=www.bank.com[NULL].badguy.com`. OpenSSL parses this as `CN=www.bank.com\x00.badguy.com`. Perl's Crypt::OpenSSL::X509 module goes even further, eliding the NULL to read `CN=www.bank.com.badguy.com`. And OpenSSL's own *X509_NAME_get_text_by_NID* function terminates on the NULL. However, before we get ahead of ourselves, what do the CAs attempt to validate?

Validation typically occurs either by checking the WHOIS for the domain in question or by attempting to retrieve a selected file from the server identified by DNS. In both cases, with our example, the CA is being asked to validate a strangely named server under `badguy.com`. The technical contact listed in the WHOIS for `badguy.com` will presumably approve any request sent by the CA. But what if a DNS query is actually issued for `www.bank.com\x00.badguy.com`? If the client resolver strips the slash, the query becomes a lookup for `www.bank.comx00.badguy.com`, which the attacker simply hosts. If the DNS query contains a NULL byte, it will likely be rejected by the nameserver, since NULL is an invalid character in DNS. If the slash is not stripped, the query propagates to the attacker, who can then reply with an IP address.

Unfortunately, neither Firefox nor IE handle NULLs in the CN either; both interpret our malicious example above as `CN=www.bank.com`.

OpenSSL's mechanisms for emitting X.509 Subject Names are vulnerable to injection attacks Although OpenSSL provides many scriptable command-line operations which can automate many aspects of PKI, it cannot automate the process of validating an identity. However, it can and does emit the X.509 Subject Name at various places from the command line, specifically to make it possible to audit the name as necessary. The CA need only write code to parse the text from OpenSSL's command line, rather than linking to OpenSSL's function calls or having to implement its own ASN.1 parser. While this approach is much easier, it does beg the question: Will the X.509 Subject Name parsed by the CA's text parser, after OpenSSL has munged it through its text filters, match the X.509 Subject Name ultimately contained with the PKCS#10 request, embedded within the generated X.509 Certificate, and delivered to the user's browser for validation?

If OpenSSL's default "compat" mode is used to emit X.509 Subject Names, not necessarily. (Three other modes, not enabled by default, are safe against the following attack.)

There are three points at which output from OpenSSL's command line interface might be parsed by a CA, looking to validate an X.509 Subject Name before certificate delivery to a client. The first, and easiest, is while signing a PKCS#10 Certificate Request, as the CN is emitted in a line that begins with "`subject=`". The client could also dump the PKCS#10 request to text and parse that. Or, the CA might sign the certificate no matter what, but suppress returning it to the user unless it is successfully validated. In this case, the generated certificate can be dumped and the Subject Name extracted.

In all three cases, however, the CA is parsing ASCII characters, rather than the actual ASN.1 tree the browser will ultimately validate. What the CA knows of that structure, it extracts from the ASCII, by splitting on the presence of commas, slashes, and other so-called “escape characters” in the text. But what if the value of one of the non-validated elements in the X.509 Subject Name — OrganizationName (O), perhaps — itself contained escape characters? This constitutes yet another form of injection attack, directly akin to SQL injection (causing a variable in a SQL query to appear to be something more) or cross-site scripting (causing a variable in an HTML page to appear to be something more).

And, indeed, using an organizationName of `Badguy Inc/ CN=www.badguy.com` with an actual CN of `www.bank.com` results in a situation where the request and generated certificate appear, to a simple regular-expression match on the emitted ASCII, to have multiple CNs, with the first being `www.badguy.com` — and no way to tell that the *actual* CN, as denoted by OID, is `www.bank.com`. The CA’s business logic is the only line of defense, and as we have noted, many CAs do not employ manual review.

Textual CN injection is, however, probably the simplest of all the vulnerabilities listed in this paper to ameliorate. OpenSSL’s `nameopt` command-line flag prepends fields with their field names, which disambiguates the situation for both human readers and scripts. This option should be used in any automated CA system which relies on the OpenSSL command line.

A non-exploitable flaw exists in all of the filtering modes for OpenSSL $<$ 0.9.8a, when a two– or four–byte–wide character set is filtered. The flaw is in the *do_buf* handler in *A_strex.c*, and involves the assumption that ASN.1 strings that contain 2– or 4–byte characters will be a multiple of 2 or 4 bytes. This is true for legitimate strings, but we can craft malicious ones for which it is not true; providing the handler with a string that thwarts its assumption causes the handler to fail. However, the pointer that OpenSSL uses to keep track of its place in the string is never actually written to; thus, what in any other codebase might be a trivial exploit merely becomes a denial of service.

4.3 PKCS#10-Tunneled SQL Injection

As mentioned earlier, ASN.1 allows many string types, with BMPString (UTF-16, supposedly minus certain characters) and UTF8String being the most flexible, but UniversalString is also worthy of analysis. This is a problem of insecurity through obscurity: since these encodings are rather abstruse, strings which use them may be injected into backend CA databases without sufficient validation. Unicode-based database injection attacks also come into play here. SQL injection into a CA’s database backend would be distinctly problematic, due to the special trust this particular data store has to the rest of the Internet.

4.4 PKCS#10-Tunneled ASN.1 Attacks

ASN.1 BER is tricky to parse, with many, many possibilities for consistent and predictably exploitable attack surfaces. The PROTOS project found a large number of vulnerabilities, via the SNMP consumer, but it is possible that some of the ASN.1 BER parsers found in commercial CA implementations were not covered in the 2002 PROTOS lockdown and thus are still vulnerable.

4.5 SSL Client Authentication Bypass

Many of the attacks in this paper have centered on vagaries with X.509 Subject Name validation. Bypassing the checks yields a certificate for somebody else's name. But what does it mean to have a certificate? For what purposes can it be used? In practice, most X.509 implementations support checking of a field called "Extended Key Usage", or EKU. EKUs come from two different sources. First, an EKU can show up in a leaf node as an explicit X.509 extension. In this context, a CA asserts the trustworthiness of a certificate. Second, an EKU can be applied out of band to a CA's root certificate, when the CA's root certificate is added to the browser's trust store. In this context, the browser manufacturer is limiting the trust semantics that a particular CA is allowed to express.

The most commonly used EKU is "Server Authentication", which states that a certificate may be used to validate a server to a client. But there are others, as we see specifically in the MD2 certificate we discussed transferring the self-signature: specifically, we are more interested in Client Authentication.

Most web sites use SSL certificates to authenticate the server to the client, followed by passwords to authenticate the client to the server. It is possible to use certificates to authenticate the client to the server as well, but this has a significant deployment and usability cost and is avoided by all but the most security-sensitive implementations. For these implementations, the user goes through the same CA experience as the server — except the X.509 Subject Name refers to *him*, not a website. A certificate-bearing SSL client, after authenticating the server, can then present his own certificate. If the certificate validates on the server — meaning that it chains back to a root trusted for Client Authentication — then some mapping will occur between the X.509 Subject Name and the application's own user database, and the user will be logged in.

These systems can fail in several major ways. First, one of the root certificates, used by the server to identify the client, might have its certificate compromised. For example, the MD2 attack discussed earlier would yield access to a VeriSign root cert with the Client Authentication EKU set in most trust stores. In the real world circa 2009, most systems are not intended to accept Client Authentication as asserted by a public CA. Instead, private CAs issue certificates to internal X.509 Subject Names, and those certs are accepted by servers in the infrastructure. However, this was not how SSL or X.509 was supposed to work. What was supposed to happen was that every user of the Internet would acquire strong cryptographic credentials from global CAs, which could be presented on demand in lieu of passwords. Through this path, a user at Microsoft could log into a server at Yahoo, and neither Microsoft nor Yahoo would have to interact with each other's private CAs. To this day, there are systems that accept not only certificates from their own private CA, but any CA in their certificate store with the Client Authentication EKU. They even broadcast their list of accepted CAs, as part of the SSL client certificate exchange.

Thus, if we compromise this VeriSign cert, we may end up with an authentication bypass for some systems. (Since SSL client certificate use is generally limited to extremely secure systems, this is of particular concern, since the limited exposure to the vulnerability is in exactly the most sensitive systems.) There is solid evidence that this bug is exposed in the field, as per a quirk of SSL. The SSL protocol, in order to support client authentication, will not simply accept whatever client certificate a client intends to transmit. It instead provides extensive hints — on Windows, by emitting the list of all root certificate X.509 Issuer Names that have the "Client Authentication" bit set in the trust store. This CTL, or Certificate Trust List, appears to be extensive, as per the "global PKI" model that was originally hoped for. However, the server set up as described at [13] to allow one private key to enter, actually allows by default may CAs to express arbitrary X.509 Subject Names and thus gain access to the server.

Theoretically, Stevens and Sotirov, in their 2008 attack against MD5, were not actually capable of similar damage; the root certificate they compromised did not have the Client Authentication root set on it. Supposedly, this should mean that any certificate signed by Stevens and Sotirov's intermediate certificate should be unable to operate as a client certificate. In practice, actual behavior is subject to implementation quirks. The actual certificate they signed includes an EKU of "Digital Signature, Non-Repudiation, Certificate Sign, CRL Sign". As per specification, this should not matter, as in-band EKU's are only supposed to be respected on leaf certificates that authenticate a given node, not on intermediates. What actually happens is murkier.

In order to prevent their malicious certificate from being respected, Stevens and Sotirov set the expiration date on their cert to late 2004. While this worked for many browsers, they found applications (a chat client, in their case) that did not realize it was important to check the expiration date. For these applications, their false intermediate worked perfectly. EKUs are an even more obscure part of the X.509 system. It is likely there are systems that ignore them too.

4.6 EV Hijacking

One of the more pernicious problems the web faces is the rise of phishing attacks. Put simply, which is the real Bank of America? Is it `www.bankofamerica.com`? `www.bofa.com`? `www.bofabank.com`? `www.bank-of-america.com`? Banks, in the real world, use physical trappings of wealth and police authority to regulate the abuse of their brand. Regulating the international DNS upon which the web is built is a much trickier problem. To deal with this pressing issue — widely exploited by phishers impersonating major banking establishments — Extended Validation certificates were developed. It was intended that EV certificates would simply not be issued without a thorough, manual validation of the IP behind the claimed name. In return, the browser UI would be updated to herald the fully validated identity of the brand. Thus, even an attacker who could legitimately obtain a certificate for `https://www.bank-of-america.com` (since he was the owner, as per DNS), would not be able to emit the same trustworthy user interface, and his phishing attack would be foiled. This is what EV was designed to do, and it succeeds reasonably well.

When Stevens and Sotirov presented their research on using MD5 to generate certificate collisions, they admitted that the EV program was not vulnerable to their attacks. As they pointed out, Extended Validation certificates must, at all points in their validation chain, avoid MD5. Since Stevens and Sotirov were only able to generate attacks against certificate signatures executed with MD5 — and since their attack depended on automatic issuance of certificates, something that EV is specifically designed to avoid — they were technically correct in their assertion. The intermediate certificate they generated could not actually expose the correct bits to force the address bar green for an arbitrary domain.

However, EV was never actually designed to stop their attack, or any of the attacks described throughout this paper. The threat being mitigated was the $12 registration of `www.bank-of-america.com` combined with the $20 certificate, not the comparatively exotic MD5 or ASN.1 collision attack combined with the once-obscure DNS cache poisoning attack. EV offers no such defense. As Adam Barth and Collin Jackson wrote in "Beware of Finer Grained Origins":

> The browser's scripting policy does not distinguish between HTTPS connections that use an Extended Validation (EV) certificate from those that use non-EV certificates. For example, PayPal serves `https://www.paypal.com/` using an EV certificate, but a principal who has a non-EV certificate for `www.paypal.com` can inject script into the PayPal login page without disrupting the browser's Extended Validation security indicators.

An attacker who can synthesize a DV certificate for `www.bank.com` and DNS cache poison `www.bank.com` can act as a man in the middle, using port forwarding to negotiate EV certification with the actual server while allowing the attacker to both read the traffic flowing between a user and `www.bank.com` and to inject arbitrary data into that stream. This will work on effectively all browsers that have implemented EV SSL. It is difficult to impossible to imagine a defense that would not involve breaking the limited number of EV sites out there. EV was designed to stop phishing attacks, not failure of DV certificates.

5 Remediation

5.1 Immediate Steps

The following table summarizes immediate steps which browser manufacturers, cryptographic API maintainers, and certificate authorities should take to address the issues we have raised:

	Browser Manufacturers	Cryptographic API Manufacturers	Certificate Authorities
MD2RSA	Possibly, to support Cryptographic API changes	Yes, to change validation rules	Yes, to agree to resolution plan
Multiple Common Names	Possibly, to determine policy and measure exposure	Yes	Possibly, to determine policy and measure exposure
Inefficient ASN.1 bypass	Possibly, to determine policy and measure exposure	Yes	Possibly, to determine policy and measure exposure
Null terminator bypass	Possibly, to determine policy and measure exposure	Yes	Possibly, to determine policy and measure exposure
OpenSSL "compat" bypass	No	Yes, definitely for SSL, possibly for others	Yes, to determine if commercial CA implementations have similar string parsing layers
PKCS#10 SQL injection	No	Possibly, to add support for filtering at the API layer	Yes
PKCS#10 ASN.1 exploitation	No for the major browsers, since presumably they've already had to lock down their ASN.1 engine	Possibly, to make sure that PKCS#10 is being parsed with a post-PROTOS hardened library	Possibly, to make sure that PKCS#10 is being parsed with a post-PROTOS hardened library
Client certificate bypass	No	Yes, to control the list of certificates that a web server will insert into the CTL	No
EV bypass	Yes, to manage PR/understanding around the purpose of EV	No	Possibly, to manage PR, and to perhaps create a "blacklist" of EV certified names for which CAs will not issue a certificate

5.2 EV Remediation

Although the EV attack we describe is particularly pernicious, there are two defenses that might be worth considering. A "`httpev://`" scheme could be developed, which would force content to only be loaded from an EV certificate.

However, this would require modifications at the crypto layer to support a new X.509 element, declaring that a certificate could only emit the "green bar" positive feedback experience when the `httpev://`method was used. Otherwise, an attacker could simply use Moxie Marlinspike's method of forwarding a user from `http://` to `https://` instead of `httpev://` and acquire 99% of the positive feedback while still being able to use his compromised DV certificate [7].

Another defense that might be interesting to explore would be a blacklist of names that, once issued via EV, should never be issued via DV. This paper describes many ways around the CA system. It might be interesting to have an emergency check, just for EV, before a certificate is sent to a user that might have the same X.509 Subject Name as an issued EV certificate.

6 Future Work

We continue to investigate certificate chain validation. It has not gone unnoticed that X.509, which was supposed to be a fully delegatable system, never actually found a safe way to delegate signing authority across chunks of DNS namespace. However, we believe we can still attack chain validation. There are multiple ways to find an issuer in X.509, and we have ways of creating valid certificates with near-arbitrary subject names. If we can find a validation path that confuses our certificate (which we do have the key for, but does not have any special capabilities) with another certificate (which we do not have the key for, but does have special capabilities) due to them sharing the exact same X.509 Subject Name, then we believe we can generate fully Root CA equivalent certificates without the still-temporarily-impractical MD2 attacks.

Another area we are investigating are the three Issuer paths: Authority Information Access, Authority Key Identifier, and the actual X.509 Issuer Name. Any and all of these can be used to generate collisions.

The wide array of string and length encodings available in ASN.1 also provides a rich attack surface. We suspect that it is possible to cause two encoders to read two entirely different ASN.1 trees by cleverly manipulating length fields, but have not yet developed a proof of concept.

Finally, there may be interesting attacks down the path of Internationalized Domain Names. For the most part, IDNs are blocked at major CAs due to the homograph attacks of Eric Johanson and the Shmoo Group in 2005 [5]. However, Moxie Marlinspike showed in early 2009 that wildcards in certificates allow IDN characters to pass validation [7]. We must consider new attacks down this path, particularly with alternate representations that may collapse back to wildcards.

References

1. Open1x IEEE 802.1x open source implementation. http://open1x.sourceforge.net/.
2. T. Dierks and E. Rescorla. The transport layer security (tls) protocol, August 2008. http://tools.ietf.org/html/rfc5246.

3. Peter Gutmann. X.509 style guide, October 2000. http://www.cs.auckland.ac.nz/ pgut001/pubs/x509guide.txt.
4. Collin Jackson and Adam Barth. Beware of finer-grained origins. In *In Web 2.0 Security and Privacy (W2SP 2008)*, 2008.
5. Eric Johanson. The state of homograph attacks, 2005. http://www.shmoo.com/idn/homograph.txt.
6. B. Kaliski. Pkcs #1: Rsa encryption, March 1998. http://tools.ietf.org/html/rfc2313.
7. Moxie Marlinspike. New tricks for defeating ssl in practice, July 2009. http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf.
8. Simon Marlow. Happy user guide, 2001. http://www.haskell.org/happy/doc/html/sec-AttributeGrammar.html.
9. F. Muller. The md2 hash function is not one-way. In P.J. Lee, editor, *Advances in Cryptology — ASIACRYPT 2004*, volume 3329, pages 214–229. Springer, 2004.
10. neon HTTP and WebDAV client library. http://www.webdav.org/neon/.
11. Alex Pilosov and Tony Kapela. Stealing the internet: An internet-scale man-in-the-middle attack. DEFCON 16, August 2008.
12. J. Rning, M. Laakso, A. Takanen, and R. Kaksonen. Protos - systematic approach to eliminate software vulnerabilities, 2002.
13. Saurabh Singh. Certificate trust list not being honored by iis 5.0/6.0/7.0, December 2007. http://blogs.msdn.com/saurabh_singh/archive/2007/12/07/certificate-trust-list-not-being-honored-by-iis-5-0-6-0-7-0.aspx.
14. Marc Stevens, Arjen Lenstra, and Benne Weger. Chosen-prefix collisions for md5 and colliding x.509 certificates for different identities. In *EUROCRYPT '07: Proceedings of the 26th annual international conference on Advances in Cryptology*, pages 1–22, Berlin, Heidelberg, 2007. Springer-Verlag.
15. Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate. Cryptology ePrint Archive, Report 2009/111, 2009. http://eprint.iacr.org/.
16. Bacula the open source network backup software solution. http://www.bacula.org/en/.
17. Claws Mail: the user-friendly lightweight and fast email client. http://www.claws-mail.org/.
18. Søren S. Thomsen. An improved preimage attack on md2. Cryptology ePrint Archive, Report 2008/089, 2008. http://eprint.iacr.org/.
19. US-CERT. Vulnerability note vu#800113: Multiple dns implementations vulnerable to cache poisoning. US-CERT Vulnerability Notes Database, 2008. http://www.kb.cert.org/vuls/id/800113.
20. Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions md4, md5, haval-128 and ripemd. Cryptology ePrint Archive, Report 2004/199, 2004. http://eprint.iacr.org/.
21. GNU Wget. http://www.gnu.org/software/wget/.