The EFF SSL Observatory

Peter Eckersley
@ EFF

Jesse Burns
@ iSec Partners

An EFF mission

Turn the unencrypted web of 2009

into the encrypted web of ~2012-13

Our contributions

Hassle sites to support https://

HTTPS Everywhere

SSL Observatory

Decentralized Observatory

Other stuff

Our contributions

Hassle sites to support https://

HTTPS Everywhere

SSL Observatory ← This talk

Decentralized Observatory

Other stuff

So, HTTPS will save the web

but...

encryption security

≤

ability to identify the other party

HTTPS uses certificates

Certificate Authorities (CAs) say

”this key belongs to mail.google.com”

(browsers trust the CAs)

We are afraid of CAs because:

2009: 3 vulnerabilities due to CA mistakes

2010: evidence of governments compelling CAs

2011: more exploits against CAs

Generally: too many trusted parties!

Also afraid of X.509

Designed in 1980s
By the ITU (!), before HTTP (!!!)

+ extremely flexible & general

- extremely flexible & general
- extremely ugly
- history of implementation vulnerabilities

X.509: Security via digital paperwork

X.509 certs can (and do) contain just about anything

What to do about it?

1. Write alternative browser code?

2. Study CA behaviour and detect problems

1 is hard → let's do 2 first

EFF SSL Observatory

Scanned all allocated IPv4 space
(port 443)

Built a system for analysing the data

Various results presented at DEFCON 2010, 27C3

This talk:

Brief overview of previously reported results

Hints on using our datasets

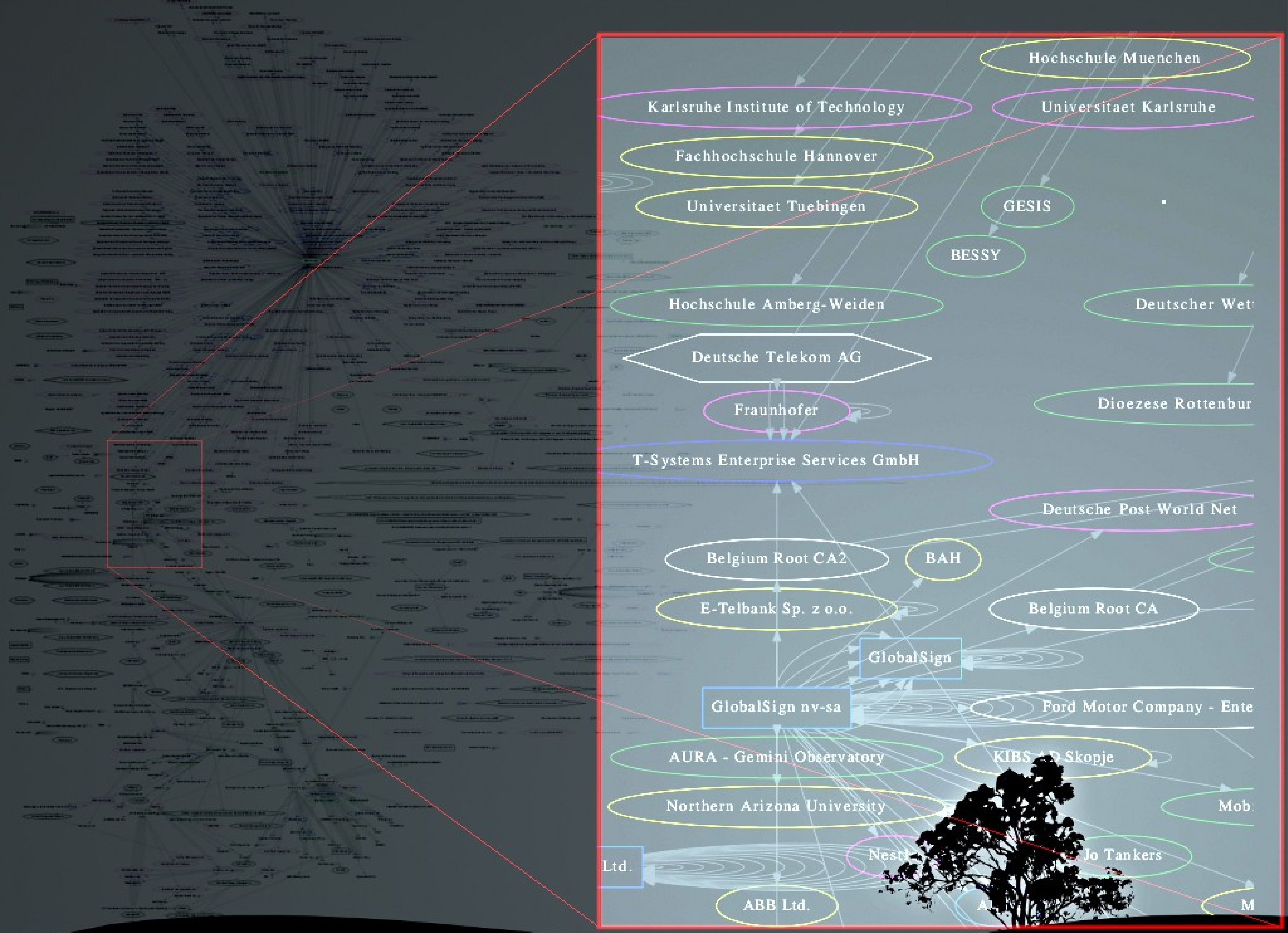
Details on forthcoming Decentralised Observatory

Size of the SSLiverse

16.2M IPs were listening on port 443
11.3M started an SSL handshake
4.3+M used valid cert chains
1.5+M distinct valid leaves

Lots of CAs!

1,482 CAs trustable by Microsoft or Mozilla
1,167 disinct Issuer strings
651 organisations


Hochschule Muenchen
Karlsruhe Institute of Technology
Universitaet Karlsruhe
Fachhochschule Hannover
Universitaet Tuebingen
GESIS
BESSY
Hochschule Amberg-Weiden
Deutscher Wet
Deutsche Telekom AG
Fraunhofer
Dioezese Rottenbur
T-Systems Enterprise Services GmbH
Deutsche Post World Net
Belgium Root CA2
BAH
E-Telbank Sp. z o.o.
Belgium Root CA
GlobalSign
GlobalSign nv-sa
Ford Motor Company - Ente
AURA - Gemini Observatory
KIBS AD Skopje
Northern Arizona University
Mob
Ltd.
Nestl
Jo Tankers
ABB Ltd.
M

Noteworthy subordinate CAs

U.S. Department of Homeland Security

U.S. Defence Contractors

CNNIC, 2007 (why debate their root CA?)

Etisalat

Gemini Observatory

Exposure to *many* jurisdictions

CAs are located in these ~52 countries:

['AE', 'AT', 'AU', 'BE', 'BG', 'BM', 'BR', 'CA', 'CH', 'CL', 'CN', 'CO', 'CZ', 'DE', 'DK', 'EE', 'ES', 'EU', 'FI', 'FR', 'GB', 'HK', 'HU', 'IE', 'IL', 'IN', 'IS', 'IT', 'JP', 'KR', 'LT', 'LV', 'MK', 'MO', 'MX', 'MY', 'NL', 'NO', 'PL', 'PT', 'RO', 'RU', 'SE', 'SG', 'SI', 'SK', 'TN', 'TR', 'TW', 'UK', 'US', 'UY', 'WW', 'ZA']

Vulnerabilities

~30,000 servers use broken keys

~500 had valid CA signatures, including:

diplomatie.be
yandex.ru
lawwebmail.uchicago.edu

(now fixed/expired)

Other whackiness

Certificates that were and were not CA certs

Lots of certs for ”localhost”, ”mail” and various IPs

Violations of Extended Validation rules

Also, we've published the data, so you can do further research on it

The data

Info at
https://www.eff.org/observatory

Available in an Amazon EC2 snapshot

(or on your own machine, but...
4GB download / 12 GB MySQL DB
~10 hours to import on a fast PC)

The database schema is fairly baroque.

In part: blame X.509
In part: only 2.5 of us

But let's show you how to use it!

Hard way to get the data:

get the torrent file from https://www.eff.org/observatory

```
bittorrent ssl-database-paths-fixed-ext.sql.lzma.torrent
```

```
mysqladmin -u root -p create observatory
```

```
unlzma -c ssl-database-paths-fixed-ext.sql.lzma | mysql -u root -p
```

(~ 10 hours later)

now you have a database of certs

Easy way to get the data:

Use Amazon EC2

https://www.eff.org/observatory/cloud

Main db tables

valid_certs } indexed by certid or
all_certs } fingerprint (SHA1)

names } Common Names + Subject
anames } Alternative Names -> certids

certs_seen: maps (time, IP) -> fingerprint
(also stores chain order)

Some simple examples:

```
SELECT RSA_Modulus_Bits, count(*)
FROM valid_certs
GROUP BY RSA_Modulus_Bits
ORDER BY cast(RSA_Modulus_Bits as decimal);
```

```
+-------------------+-----------+
| RSA_Modulus_Bits  | count(*)  |
+-------------------+-----------+
| 511               |         3 |
| 512               |      3977 |
| 730               |         1 |
| 767               |         1 |
| 768               |        34 |
| 1023              |       968 |
| 1024              |    821900 |
| ...               |       ... |
+-------------------+-----------+
SELECT `Signature Algorithm`, count(*)
FROM valid_certs
WHERE startdate > ”2010”
GROUP BY `Signature Algorithm`;

+--------------------------+----------+
| Signature Algorithm      | count(*) |
+--------------------------+----------+
|  md5WithRSAEncryption    |        3 |
|  sha1WithRSAEncryption   |   455511 |
|  sha256WithRSAEncryption |       17 |
|  sha512WithRSAEncryption |        1 |
+--------------------------+----------+
SELECT distinct issuer
FROM valid_certs
WHERE stardate > ”2010” AND
  `Signature Algorithm`= " md5WithRSAEncryption";

+-------------------------------------------------------------------------+
| issuer                                                                  |
+-------------------------------------------------------------------------+
|  O=Ministere de la Justice, CN=Autorite de Certification Serveurs       |
|  C=US, O=Anthem Inc, OU=Ecommerce, CN=Anthem Inc Certificate Authority |
+-------------------------------------------------------------------------+
```

(fortunately, these CAs don't robo sign)

Caveats...

Some fields (name, IP) in the _certs tables are correct but not comprehensive

```
SELECT count(distinct ip) FROM all_certs     --  5,536,773
SELECT count(distinct ip) FROM seen          -- 11,373,755
```

(the former undercounts due to certs seen on multiple IPs)

some columns have unintuitive semantics; moz_valid, ms_valid are the outputs of:

```
openssl verify -CApath <roots> -untrusted <rest of chain> cert ; eg:
```

```
Yes
Yes
self-signed: OK
self-signed:   /CN=sw-mhs-ser-3750-1./unstructuredName=sw-mhs-ser-3750-1. error 10
               at 0 depth lookup:certificate has expired OK
Yes
Yes
self-signed: in certificate chain
self-signed: OK
No:        'stdin: /C=US/ST=Default State/L=Default Locality/O=American Power
            Conversion Corp/OU=Internally Generated Certificate/CN=ZA0535013730\\n
           error 20 at 0 depth lookup:unable to get local issuer certificate\\n'None
```

So:

```
select count(*) from valid_certs where moz_valid=”Yes”      →1,359,292
select count(*) from valid_certs where not moz_valid=”Yes” →  174,067
select count(*) from valid_certs where not ms_valid=”Yes”  →  213,401
```

Even worse...

Firefox and IE cache intermediate CA certificates...

So OpenSSL can't necessarily say whether a cert is valid in these browsers (!!!)

”Transvalidity”

valid, but only if the browser cached the right intermediate CA certs first

→

we catch all / almost all transvalid certs

explaining transvalidity.py

First, find invalid certs where a plausible, valid intermediate cert was seen somewhere in the SSLiverse:

```
SELECT certs1.path, certs1.id, valid_certs.path, certs1.fingerprint,
       certs1.fetchtime
FROM certs1 join valid_certs
ON certs1.issuer = valid_certs.subject and (
        (certs1.`Authority Key Identifier:keyid` is null and
         valid_certs.`Subject Key Identifier` is null)
    or
       certs1.`Authority Key Identifier:keyid` =
       valid_certs.`Subject Key Identifier`
)
WHERE not certs1.valid and
    (locate("unable to get local issuer certificate", certs1.moz_valid) or
     locate("unable to get local issuer certificate", certs1.ms_valid) )
GROUP BY certs1.fingerprint, valid_certs.path
```

Note: some variable names were simplified in this query:
certs1 is an example raw input certs table, Authority Key IDs have longer column names

transvalidity.py (ct'd)

Once we have some missing, valid, possibly determinative CA certs, we re-run OpenSSL:

```
openssl verify -CApath <all roots> -untrusted <rest of chain + query results> cert
```

Results go in the ”transvalid” column

```
select count(*) from valid_certs where transvalid="Yes"
```

→ 97,676 tranvalid certs

Validity in general

```
boolean valid = (      moz_valid == ”Yes”
                   or   ms_valid == ”Yes”
                   or  transvalid == ”Yes”)
```

More examples of the dataset at work...

Which root CAs created the most subordinate CAs? SubordinateTracking.py

For each root cert:

```
SELECT certid, subject, issuer, `Subject Key Idenfier`
FROM valid_certs where issuer = <root CA's subject>
    and locate(”true”, `X509v3 Basic Constraints:CA`)
    and `X509v3 Authority Key Identifier:keyid` = <root CA's SKID>
                                                      (which may be NULL)
```

(and recurse)

Results: top roots by CA proliferation

1. C=DE, CN=Deutsche Telekom Root CA 2	252 sub-CAs (4,164 leaves)
2. C=US, CN=GTE CyberTrust Global Root	93 sub-CAs (20,937 leaves)
3. C=SE, CN=AddTrust External CA Root	72 sub-CAs (384,481 leaves)
4. C=BE, CN=GlobalSign Root CA	63 sub-CAs (140,176 leaves)
5. C=US, CN=Entrust.net Secure Server Certification Authority	33 sub-CAs (91,203 leaves)
6. C=FR, O=PM/SGDN, OU=DCSSI, CN=IGC/A...	24 sub-CAs (448 leaves)
7. OU=ValiCert Class 3 Policy Validation Authority	20 sub-CAs (1,273 leaves)
8. O=VeriSign, Inc, OU=Class 3 Public Primary Certification Authority	18 sub-CAs (312,627 leaves)

Extended Validation

Great idea: Certs become reliable again

http://cabforum.org/EV_Certificate_Guidelines.pdf

Stricter rules like:

Owners exclusively own domains

Use relatively strong keys

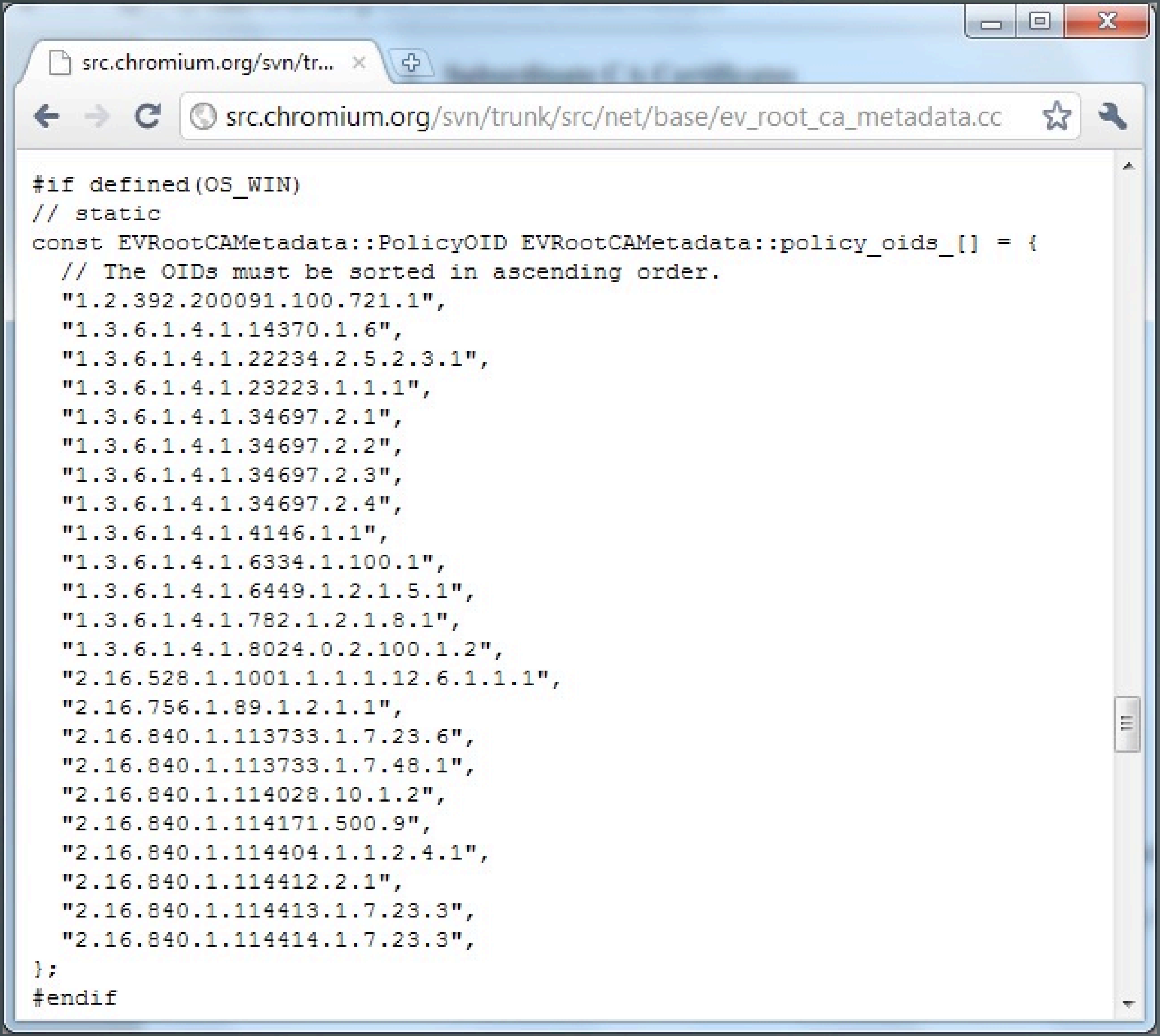
Identifiable Owners

Audits

Extended Validation

Special OID per CA
Chromium Source documents:
ev_root_ca_metadata.cc

EV's Per CA OIDs


src.chromium.org/svn/tr...
src.chromium.org/svn/trunk/src/net/base/ev_root_ca_metadata.cc

```
#if defined(OS_WIN)
// static
const EVRootCAMetadata::PolicyOID EVRootCAMetadata::policy_oids_[] = {
  // The OIDs must be sorted in ascending order.
  "1.2.392.200091.100.721.1",
  "1.3.6.1.4.1.14370.1.6",
  "1.3.6.1.4.1.22234.2.5.2.3.1",
  "1.3.6.1.4.1.23223.1.1.1",
  "1.3.6.1.4.1.34697.2.1",
  "1.3.6.1.4.1.34697.2.2",
  "1.3.6.1.4.1.34697.2.3",
  "1.3.6.1.4.1.34697.2.4",
  "1.3.6.1.4.1.4146.1.1",
  "1.3.6.1.4.1.6334.1.100.1",
  "1.3.6.1.4.1.6449.1.2.1.5.1",
  "1.3.6.1.4.1.782.1.2.1.8.1",
  "1.3.6.1.4.1.8024.0.2.100.1.2",
  "2.16.528.1.1001.1.1.1.12.6.1.1.1",
  "2.16.756.1.89.1.2.1.1",
  "2.16.840.1.113733.1.7.23.6",
  "2.16.840.1.113733.1.7.48.1",
  "2.16.840.1.114028.10.1.2",
  "2.16.840.1.114171.500.9",
  "2.16.840.1.114404.1.1.2.4.1",
  "2.16.840.1.114412.2.1",
  "2.16.840.1.114413.1.7.23.3",
  "2.16.840.1.114414.1.7.23.3",
};
#endif
```

EV hints via ugly where clause

```
`X509v3 Authority Key Identifier` is null and
    (locate("1.2.392.200091.100.721.1:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.14370.1.6:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.22234.2.5.2.3.1:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.23223.1.1.1:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.34697.2.1:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.34697.2.2:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.34697.2.3:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.34697.2.4:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.4146.1.1:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.6334.1.100.1:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.6449.1.2.1.5.1:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.782.1.2.1.8.1:", `X509v3 Certificate Policies:Policy`) or
    locate("1.3.6.1.4.1.8024.0.2.100.1.2:", `X509v3 Certificate Policies:Policy`) or
    locate("2.16.528.1.1001.1.1.1.12.6.1.1.1:",`X509v3 Certificate Policies:Policy`)or
    locate("2.16.756.1.89.1.2.1.1:", `X509v3 Certificate Policies:Policy`) or
    locate("2.16.840.1.113733.1.7.23.6:", `X509v3 Certificate Policies:Policy`) or
    locate("2.16.840.1.113733.1.7.48.1:", `X509v3 Certificate Policies:Policy`) or
    locate("2.16.840.1.114028.10.1.2:", `X509v3 Certificate Policies:Policy`) or
    locate("2.16.840.1.114171.500.9:", `X509v3 Certificate Policies:Policy`) or
    locate("2.16.840.1.114404.1.1.2.4.1:", `X509v3 Certificate Policies:Policy`) or
    locate("2.16.840.1.114412.2.1:", `X509v3 Certificate Policies:Policy`) or
    locate("2.16.840.1.114413.1.7.23.3:", `X509v3 Certificate Policies:Policy`) or
    locate("2.16.840.1.114414.1.7.23.3:", `X509v3 Certificate Policies:Policy`))
```

Finding EV problems with the Observatory

About 33,916 EV certs this time with 38 issuers

Not all unique, not all really used.

Extended Validation problems found by the Observatory

RFC-1918 Addreses
Unqualified Names...
Localhost?!?
Weak (512 bit) keys
Long expiration

Future Work

1. A decentralised observatory

2. The question of how to reinforce the CA system more generally

Decentralised Observatory Objectives

1. Detect MITM attacks
   - even if only the victim gets the cert
2. Protect user privacy
   - never know who looks at which site

Decentralised Observatory Design

1. User has Tor running

2. Send raw certs to Observatory
   - asynchronosly
   - via Tor for anonymity, w/ exit enclave

3. Warn users about phishy CA signatures?
   - yes
   - not until a few seconds later :(
   - better late than never

Decentralised Observatory

the code is close to ready

Conclusion

join us

eff.org/observatory

questions: ssl-survey@eff.org