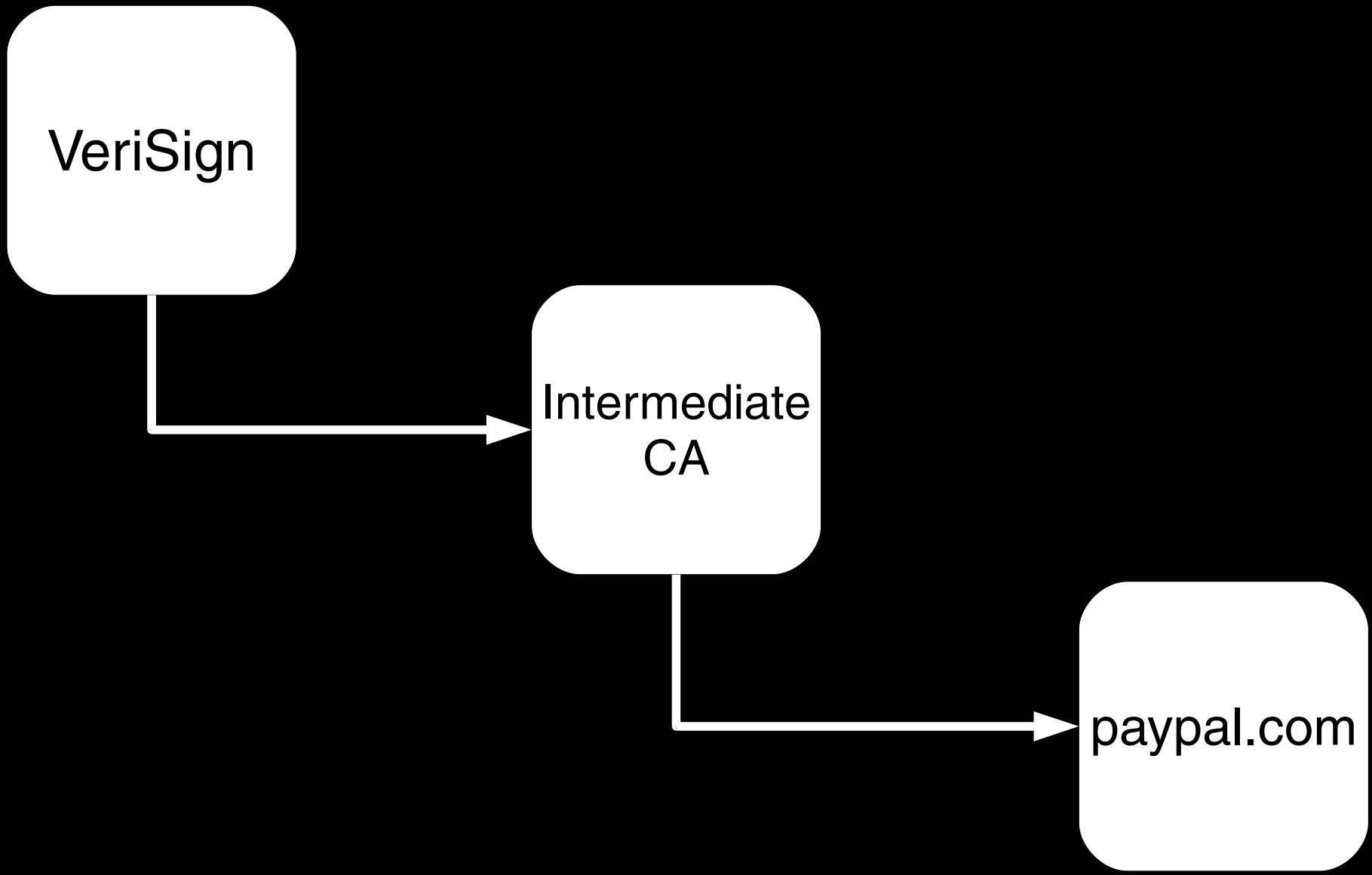
More Tricks For Defeating SSL In Practice

Moxie Marlinspike
moxie@thoughtcrime.org

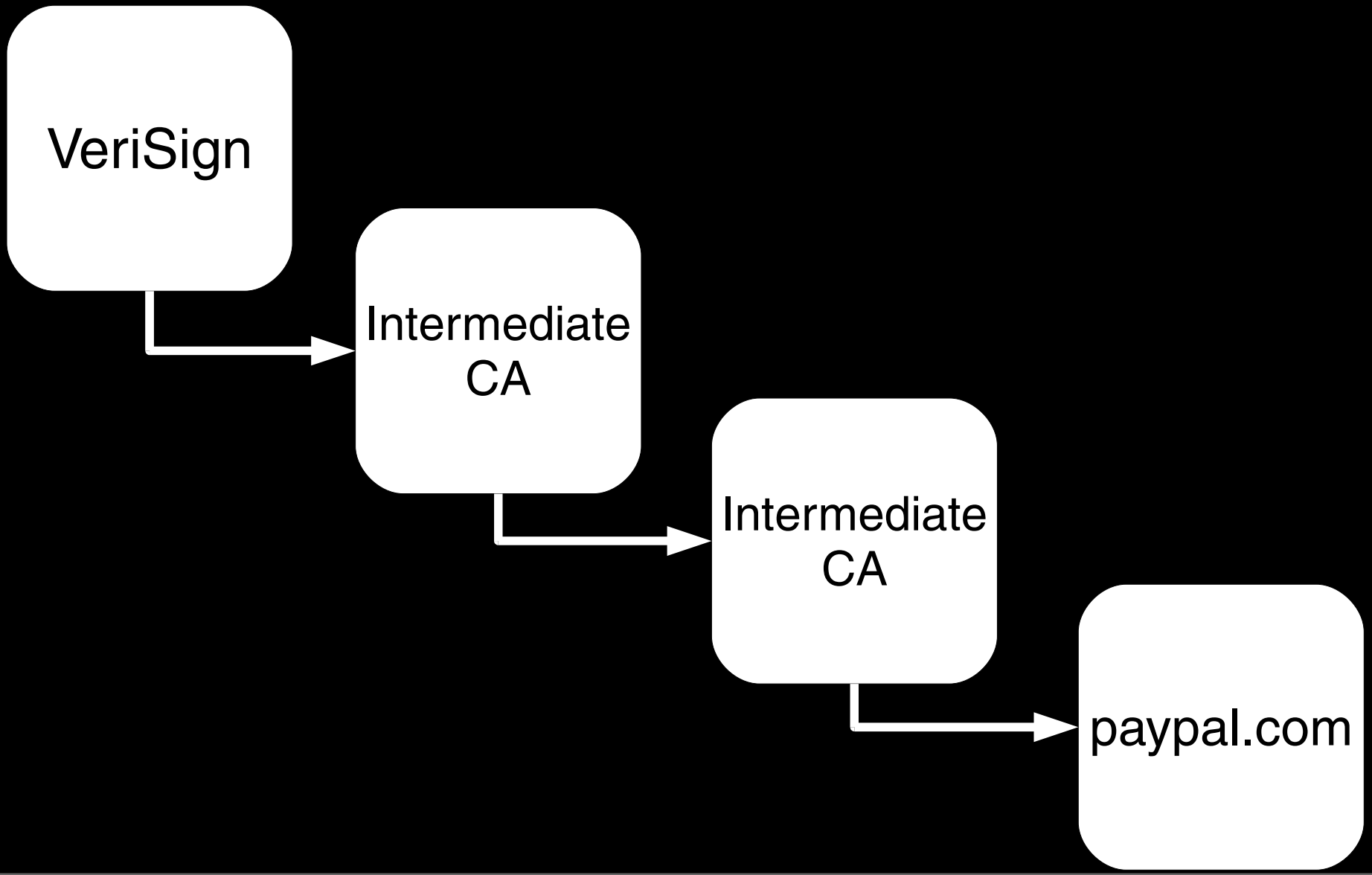
Once Again, The Back Story...

In the past, I've talked about BasicConstraints...

Certificate Chaining



Certificate Chaining

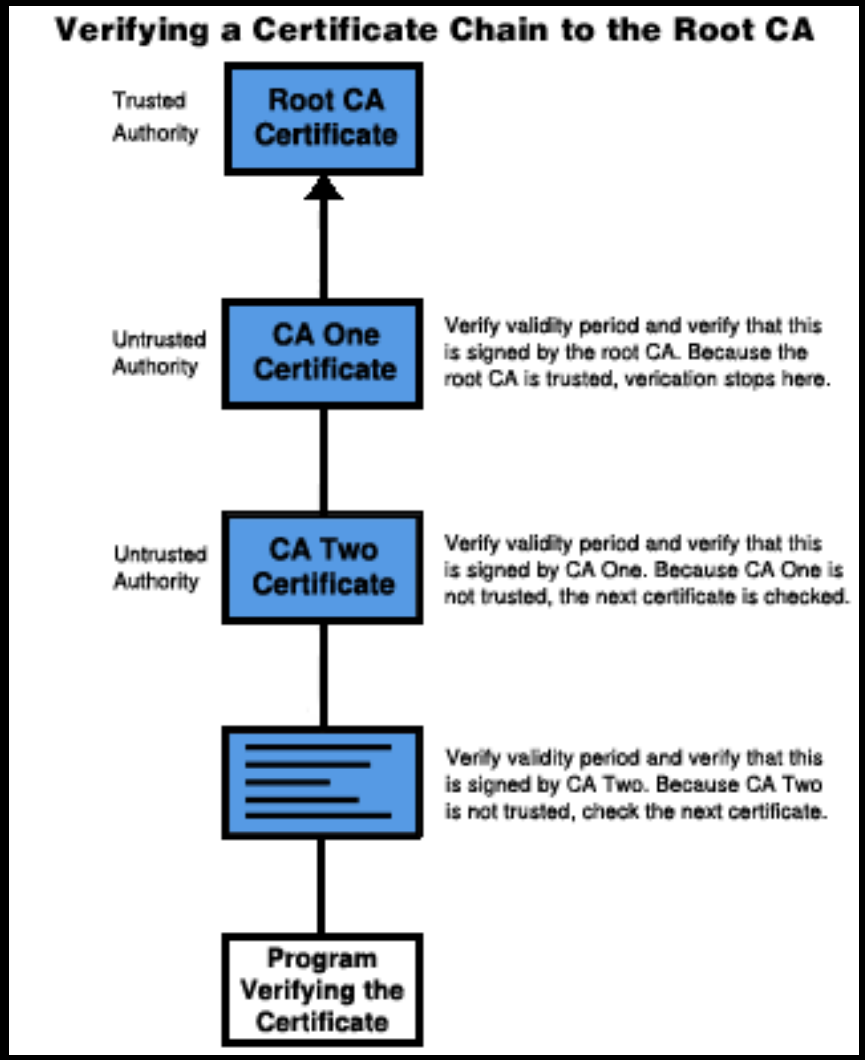


How do we verify these things?

What they say:

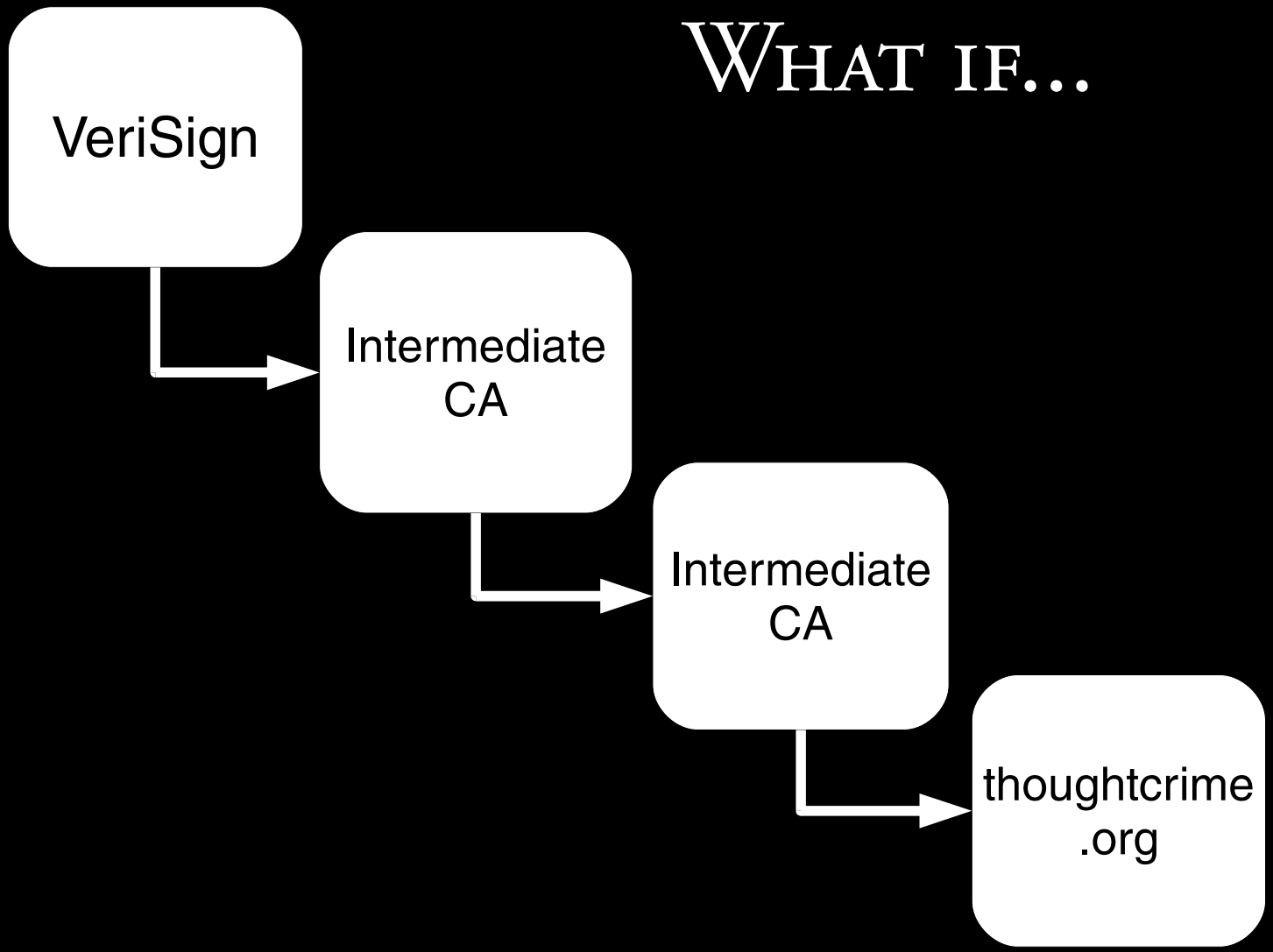
- Verify that the name of the leaf node is the same as the site you're connecting to.
- Verify that the leaf certificate has not expired.
- Check the signature.
- If the signing CA is in our list of trusted root CAs, stop. Otherwise, move one up the chain and repeat.

Here Be Dragons

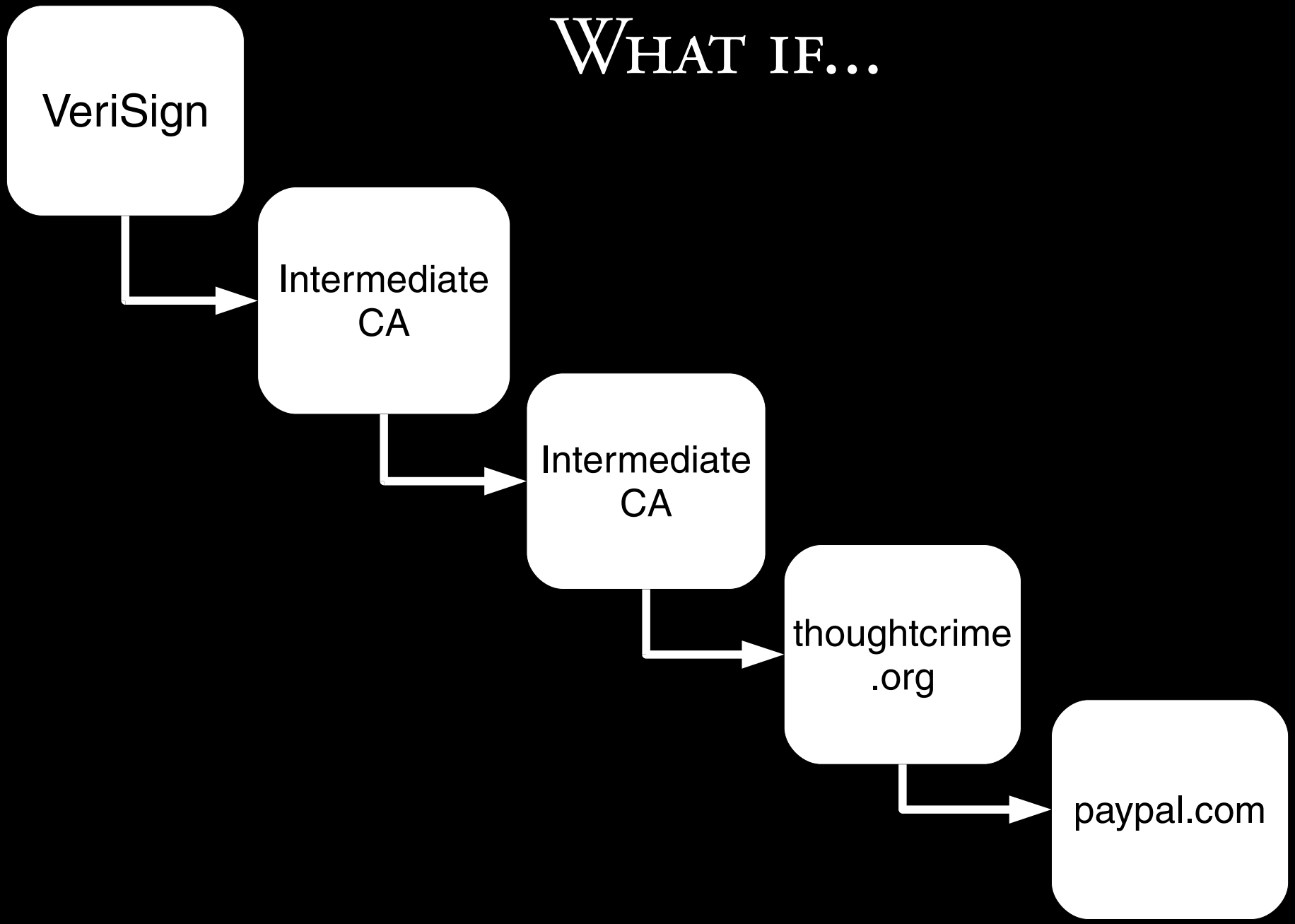


- Very tempting to use a simple recursive function.
- Everyone focuses on the signature validation.
- The result of a naïve attempt at validation is a chain that is complete, but nothing more.

What if...


VeriSign
Intermediate CA
Intermediate CA
thoughtcrime .org

What if...


VeriSign
Intermediate CA
Intermediate CA
thoughtcrime .org
paypal.com

What they say:

- Verify that the name of the leaf node is the same as the site you're connecting to.
- Verify that the leaf certificate has not expired.
- Check the signature.
- If the signing CA is in our list of trusted root CAs, stop. Otherwise, move one up the chain and repeat.

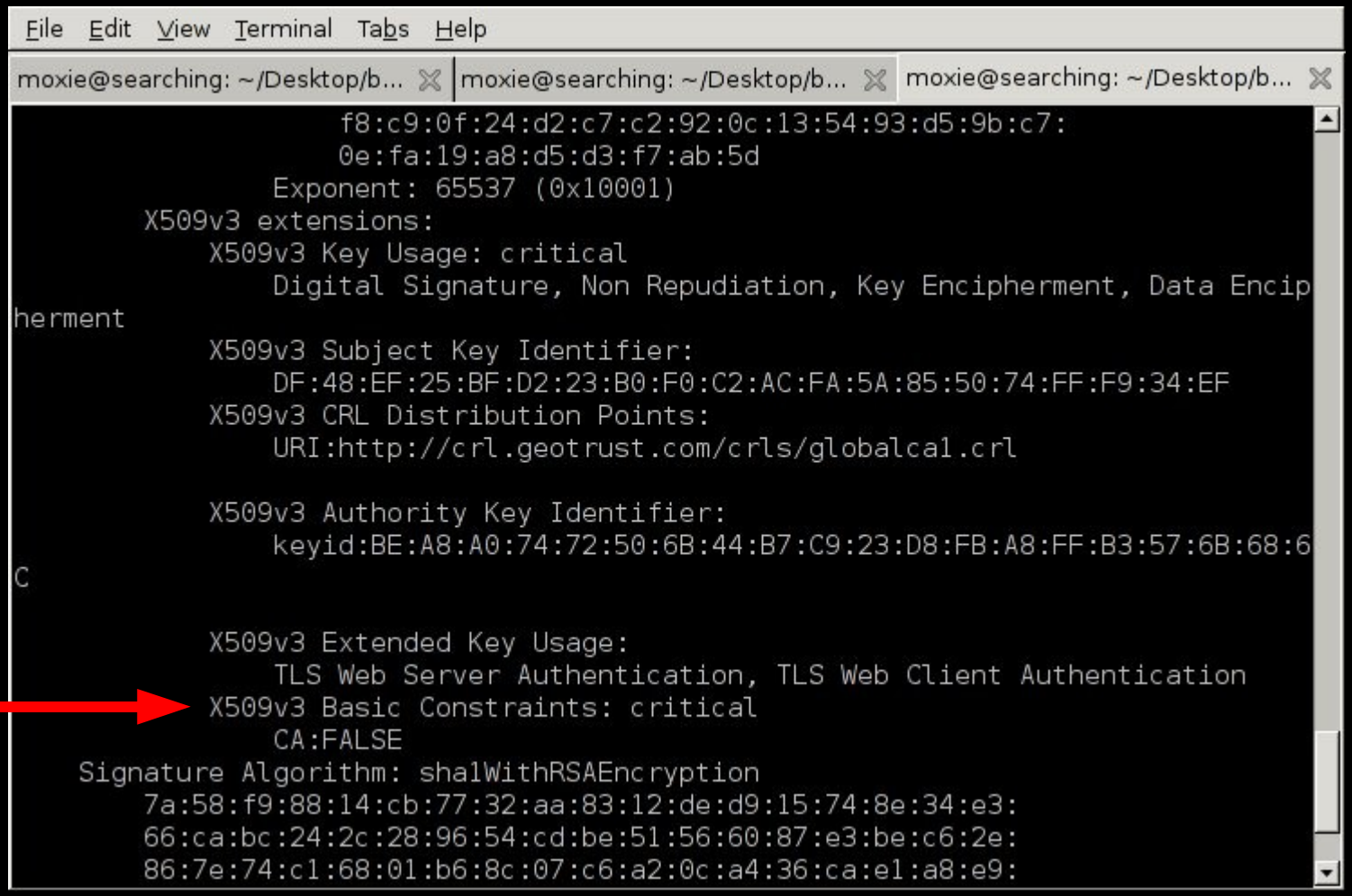
Something must be wrong, but...

- All the signatures are valid.
- Nothing has expired.
- The chain is in tact.
- The root CA is embedded in the browser and trusted.

But we just created a valid certificate for PayPal, and we're not PayPal?

The missing piece...

...IS A SOMEWHAT OBSCURE FIELD.



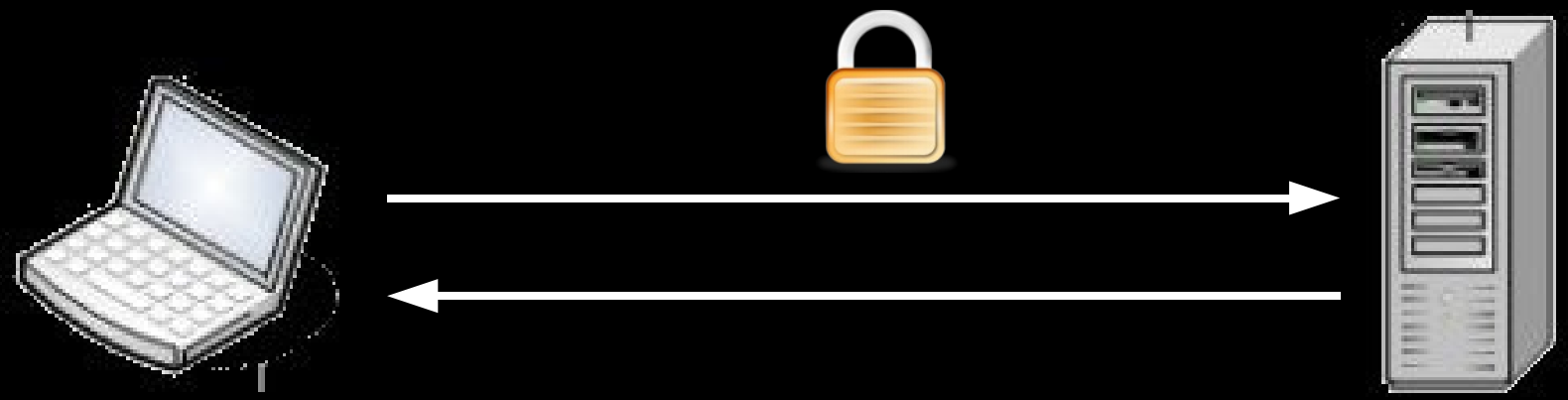
Back In The Day

- Most CAs didn't explicitly set basicConstraints: CA=False
- Whether the field was there or not, most SSL implementations didn't bother to check it.
- *Anyone* with a valid leaf node certificate could create and sign a leaf node certificate for *any other* domain.
- When presented with a complete chain, IE, Outlook, Konqueror, OpenSSL, and others considered it valid...

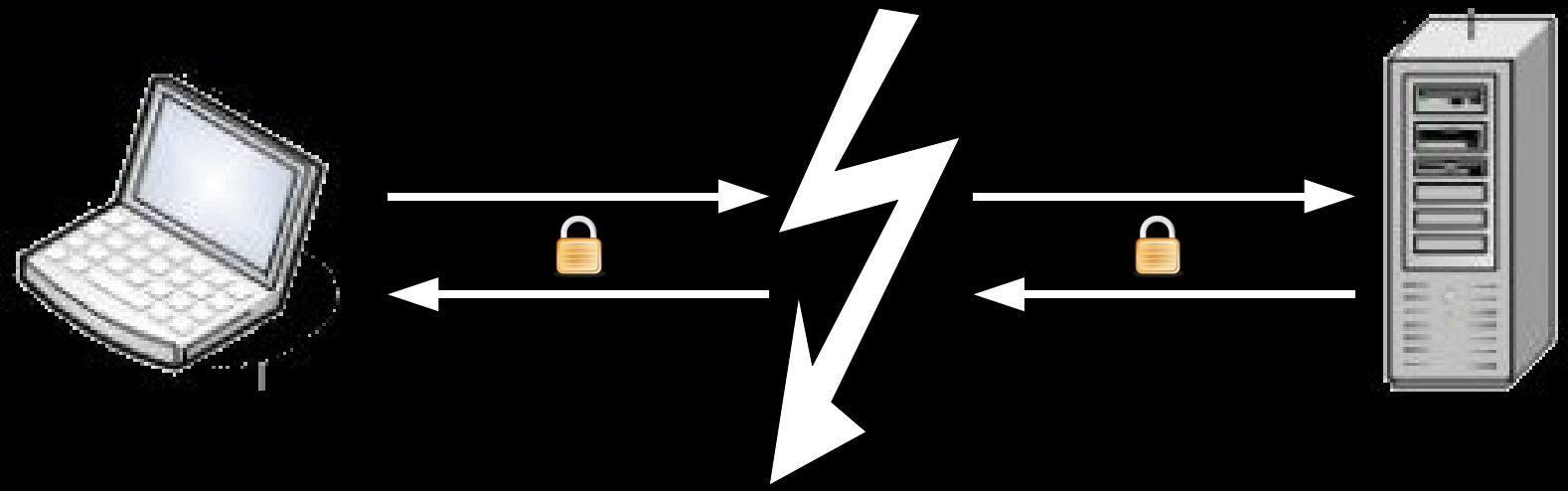
And then in 2002...

- Microsoft did something particularly annoying, so I blew this up by publishing it.
- Microsoft claimed that it was impossible to exploit.
- So I also published the tool that exploits it.

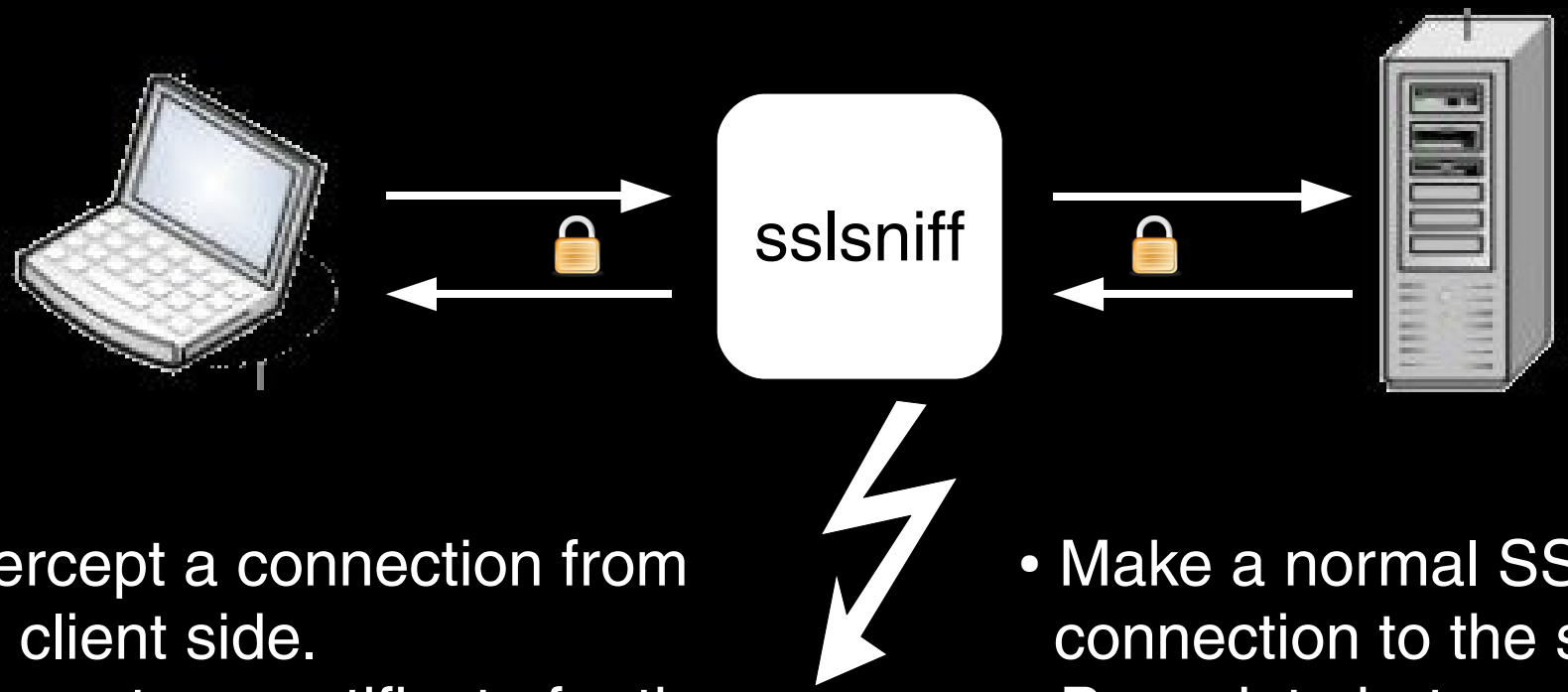
SSLSNIFF



SSLSNIFF



SSLSNIFF



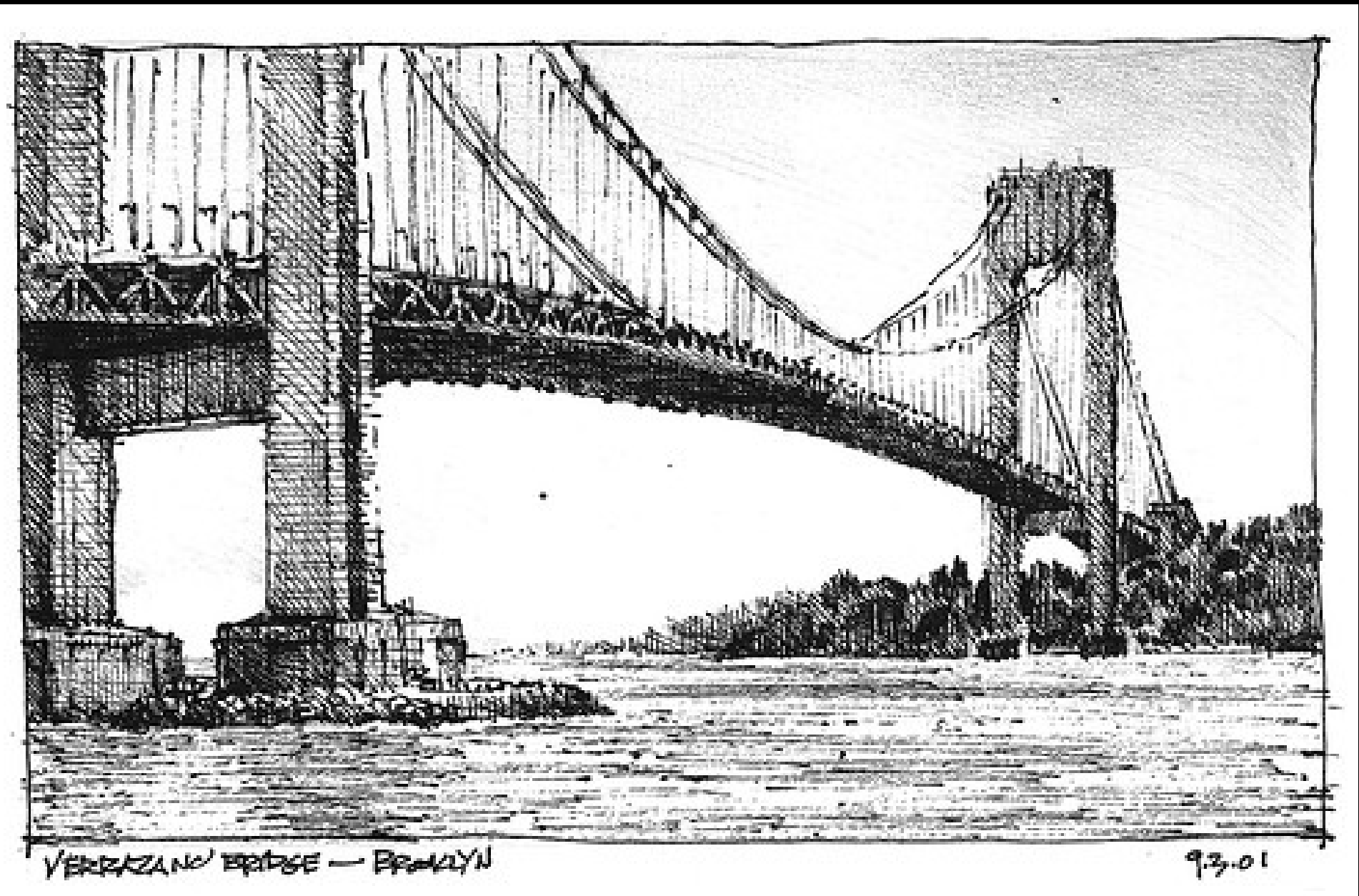
- Intercept a connection from the client side.
- Generate a certificate for the site it is connecting to.
- Sign it with any random valid leaf node certificate.
- Pass that certificate chain to the client.

- Make a normal SSL connection to the server.
- Pass data between client and server, decrypting and encrypting on each end.

Lately, I've been talking about SSL Stripping...

BRIEF

Moxie Marlinspike
Institute For Disruptive Studies



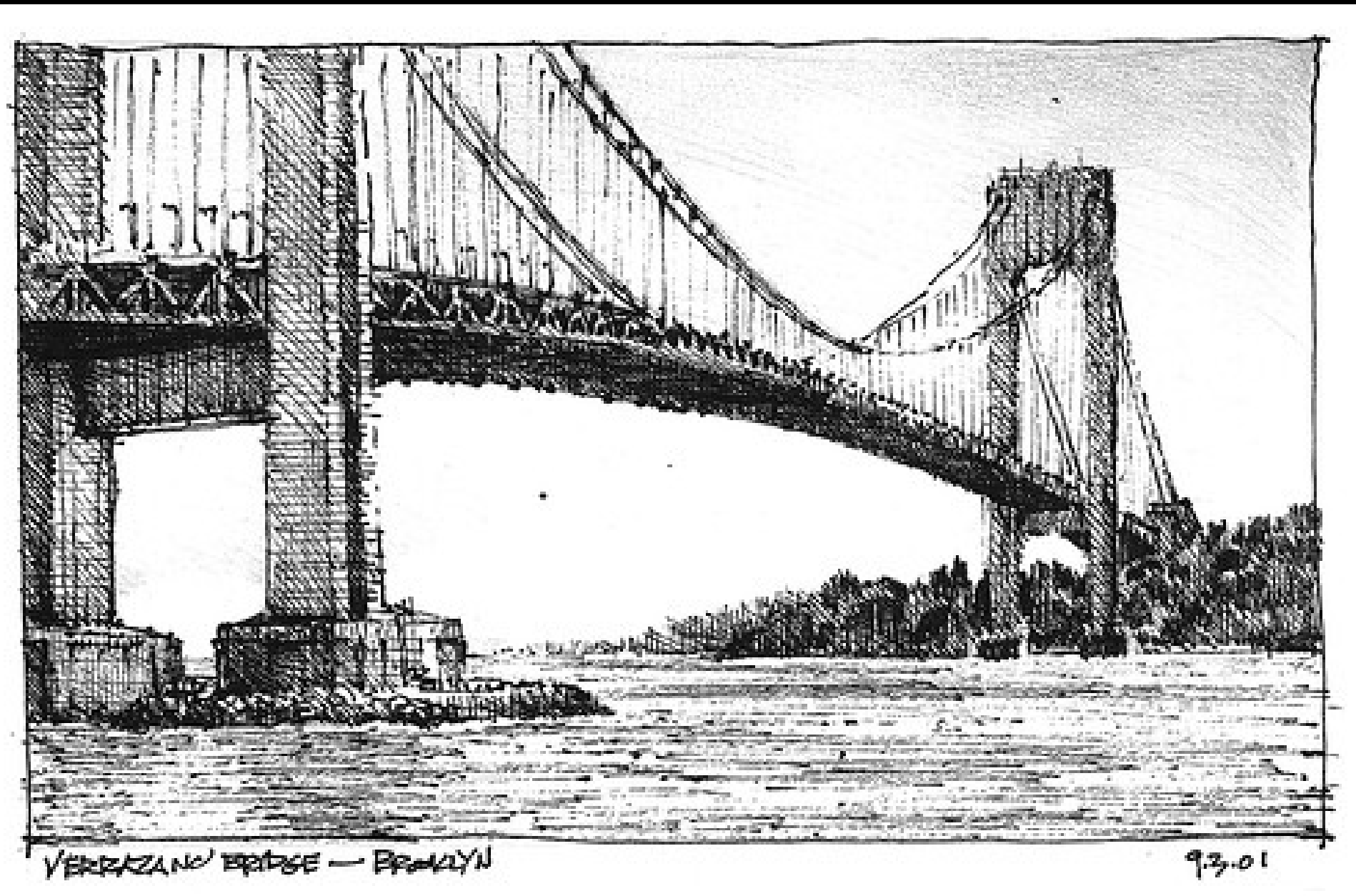
SSL can be useful, but how it's deployed

matters

Moxie Marlinspike
Institute For Disruptive Studies

In the context of web browsing

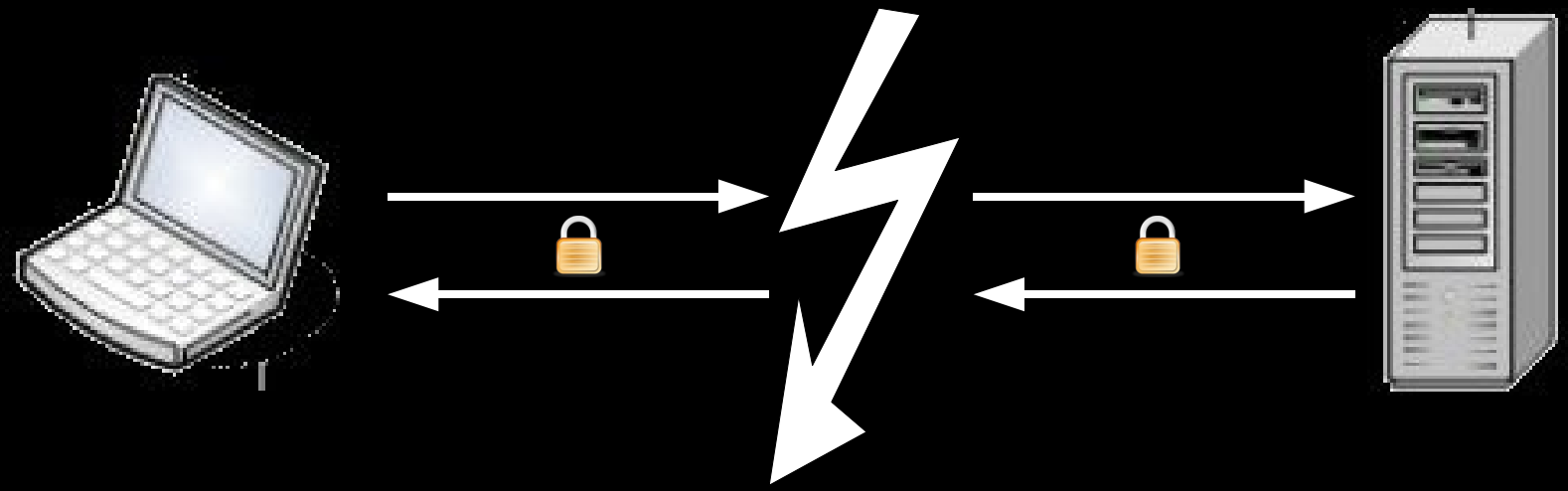
- SSL is almost never encountered directly.
- It is either encountered as a result of:
  - A 302 redirect from an HTTP URL to an HTTPS URL.
  - An HTTPS link that a user clicks on from an HTTP page.
    - (Think, “My Cart,” “Checkout,” “Login,” etc...)



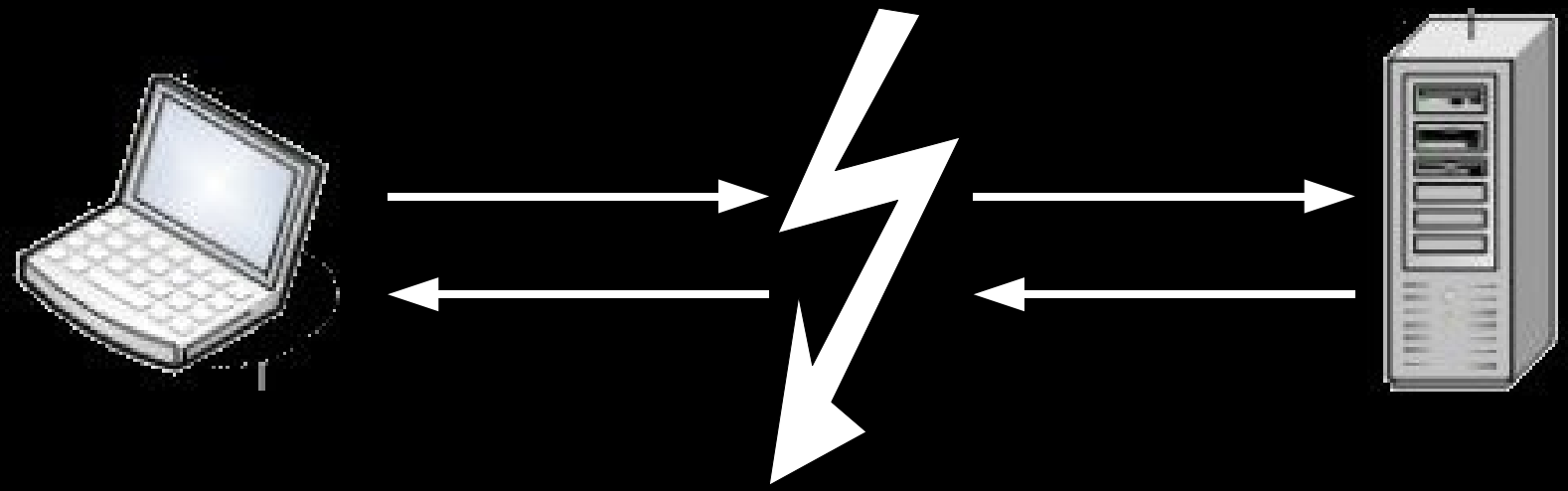
We Can Attack SSL Before We Even Get There

Moxie Marlinspike
Institute For Disruptive Studies

SSLSNIFF

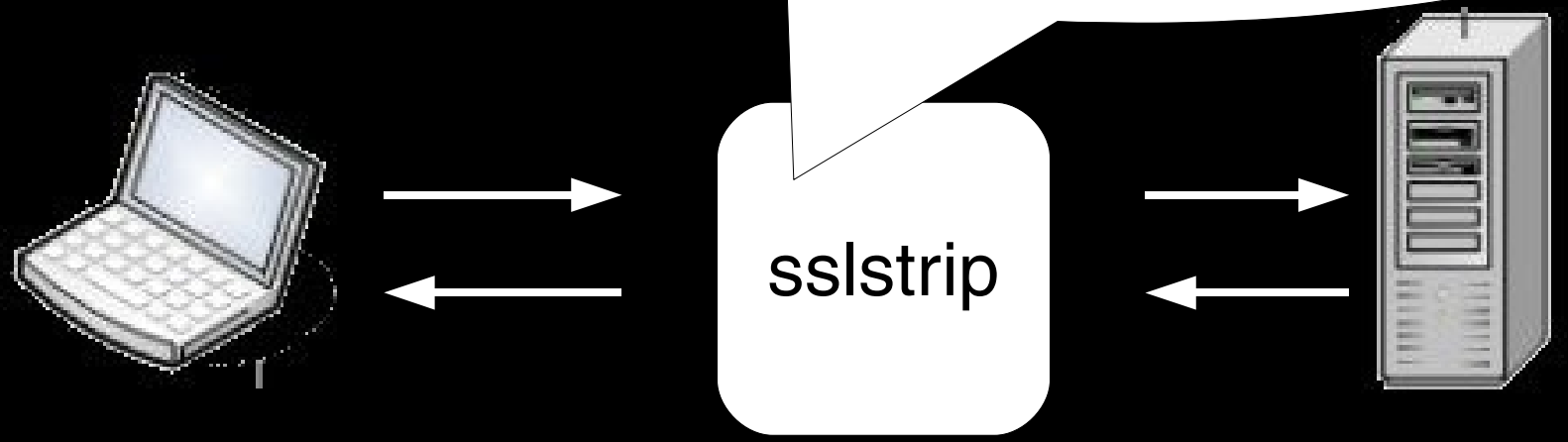


SSLSTRIP



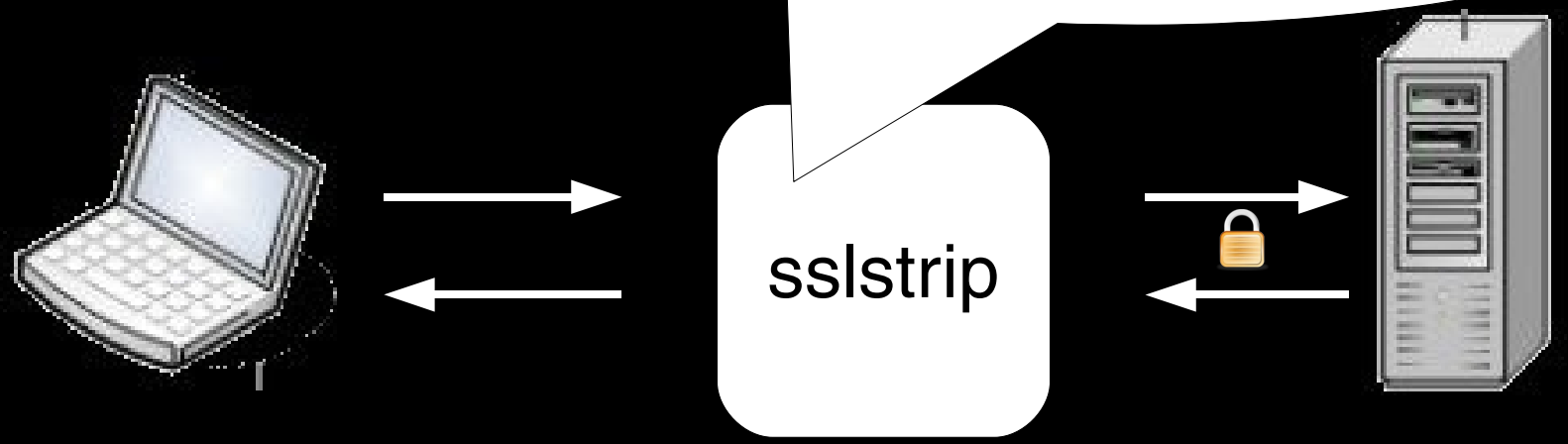
SSLSTRIP

- Watch HTTP traffic go by.
- Switch <a href="https://..."> to <a href="http://..."> and *keep a map of what you've changed*.
- Switch Location: https:// to Location: http:// and *keep a map of what you've changed*.

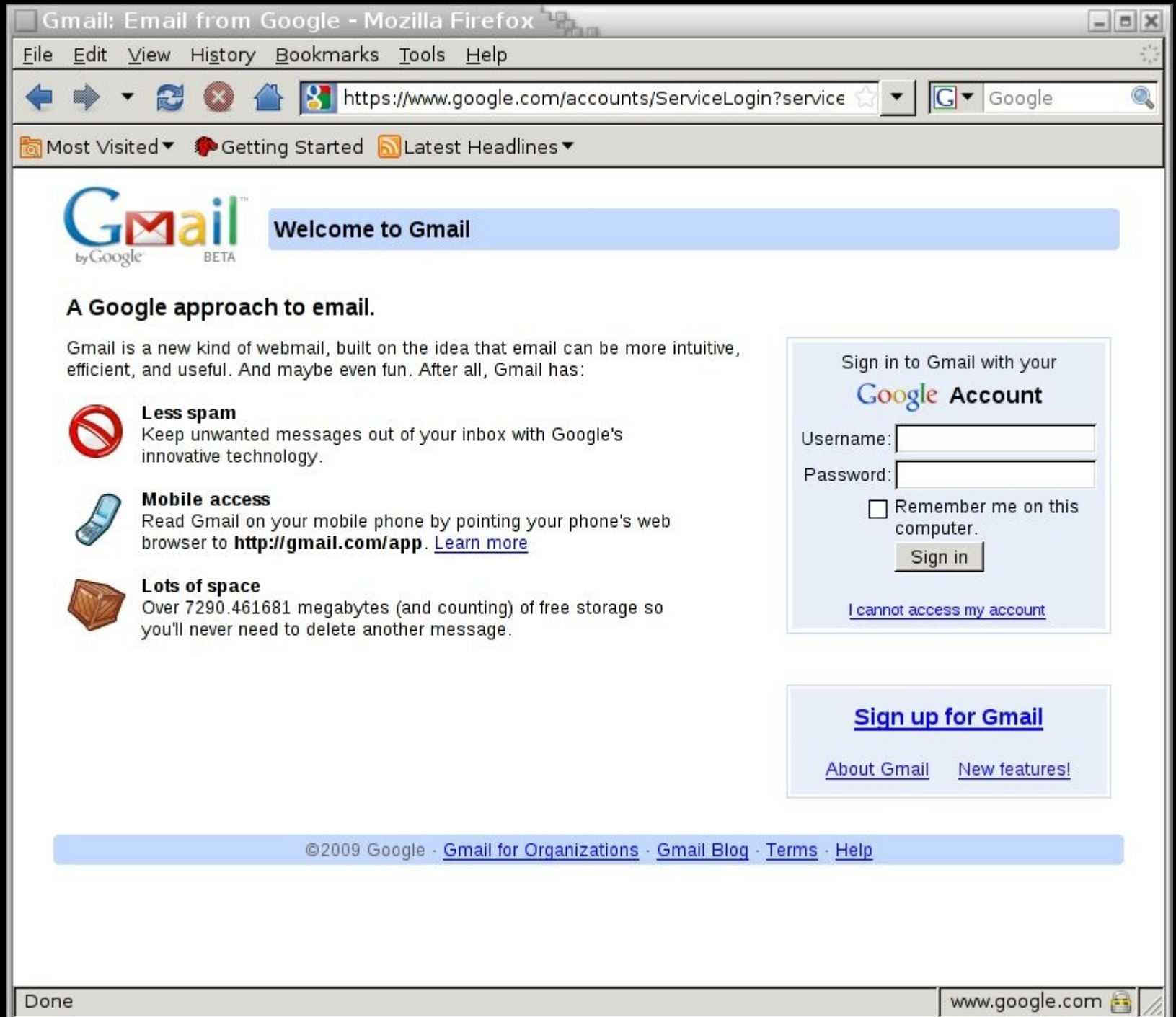


SSLSTRIP

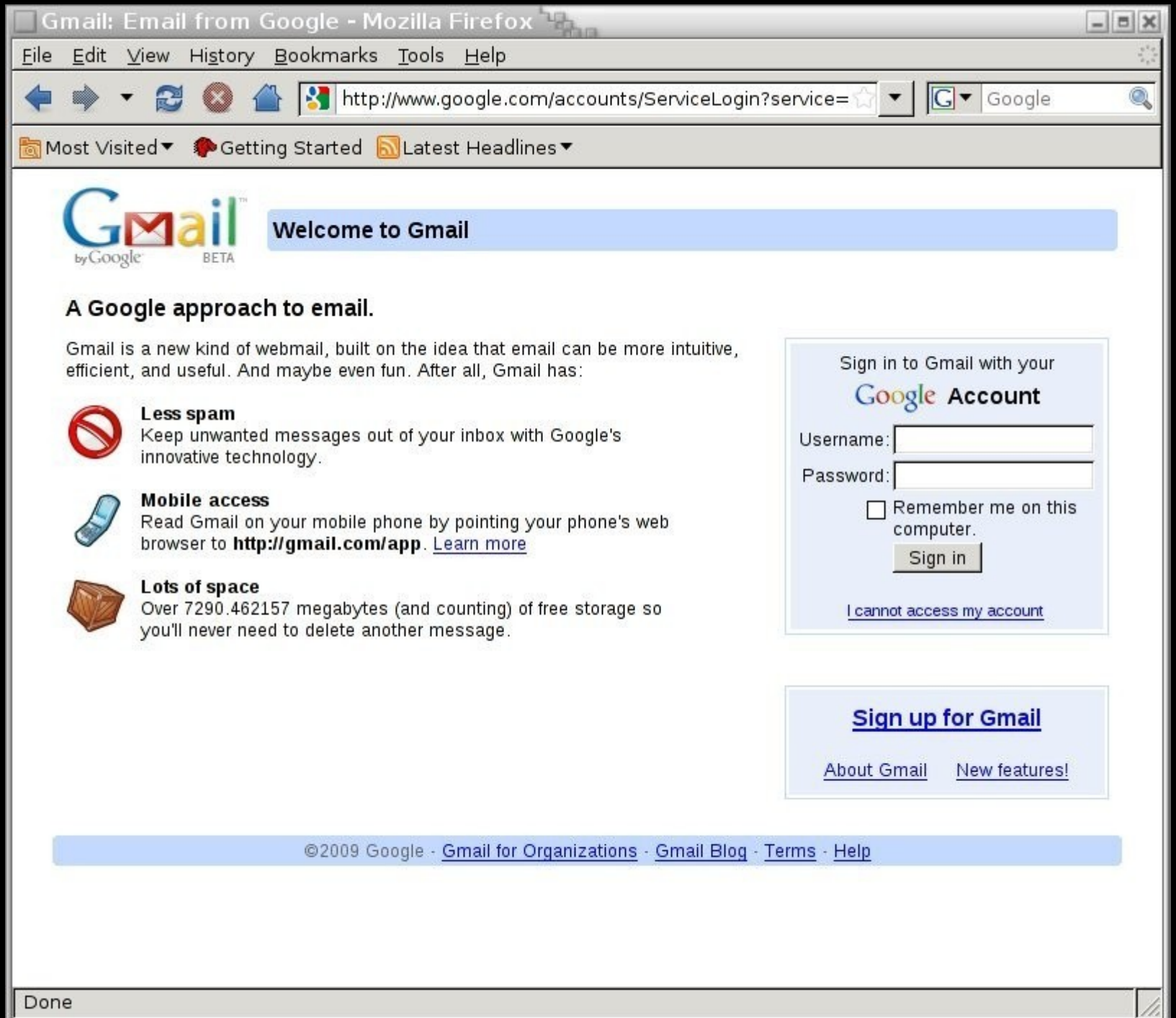
- Watch HTTP traffic go by.
- When we see an HTTP request for a URL that we've stripped, proxy that out as HTTPS to the server.
- Watch the HTTPS traffic go by, log everything that we want, and keep a map of all relative, CSS, and JS links that go by.



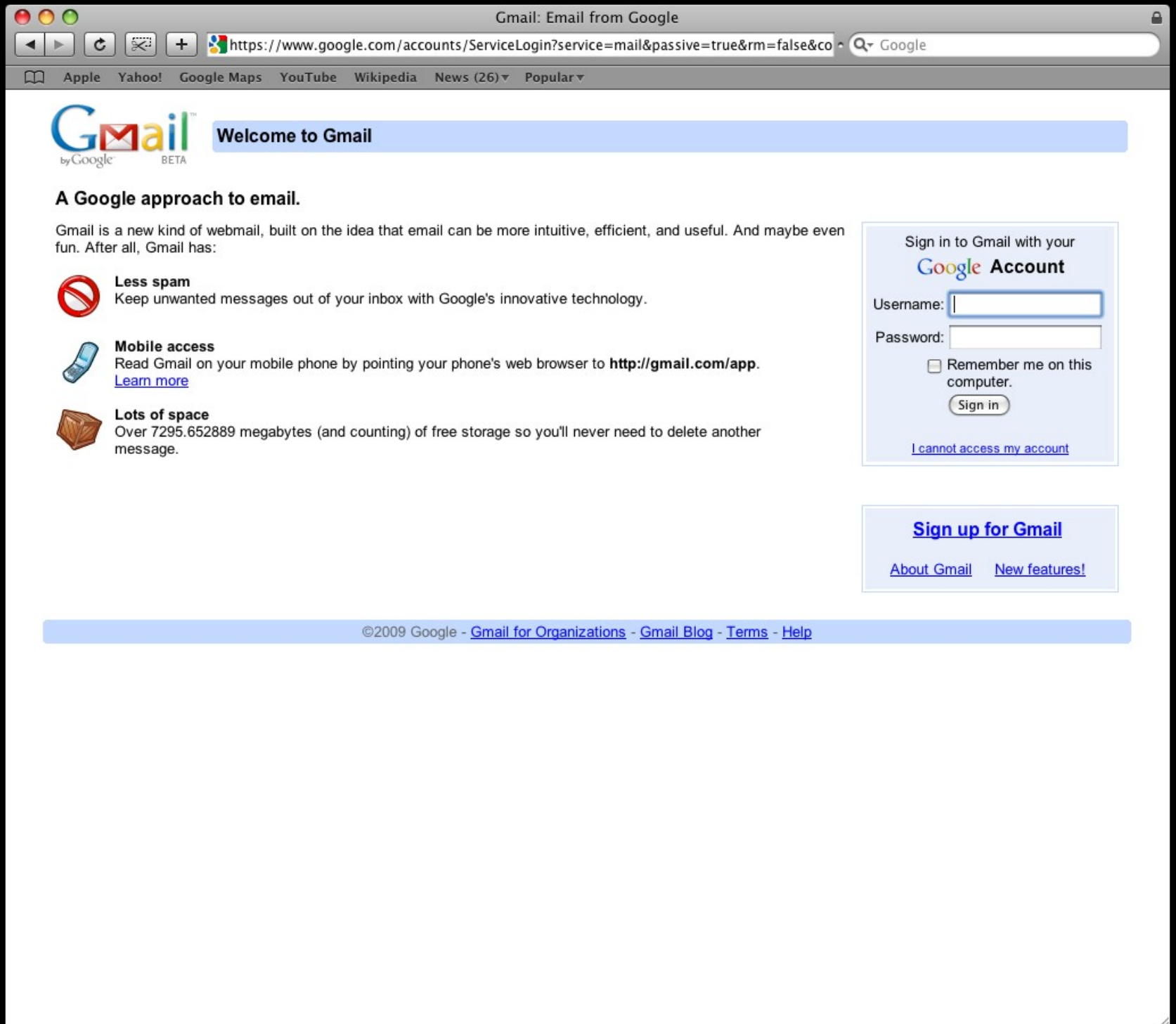
How Does It Look?



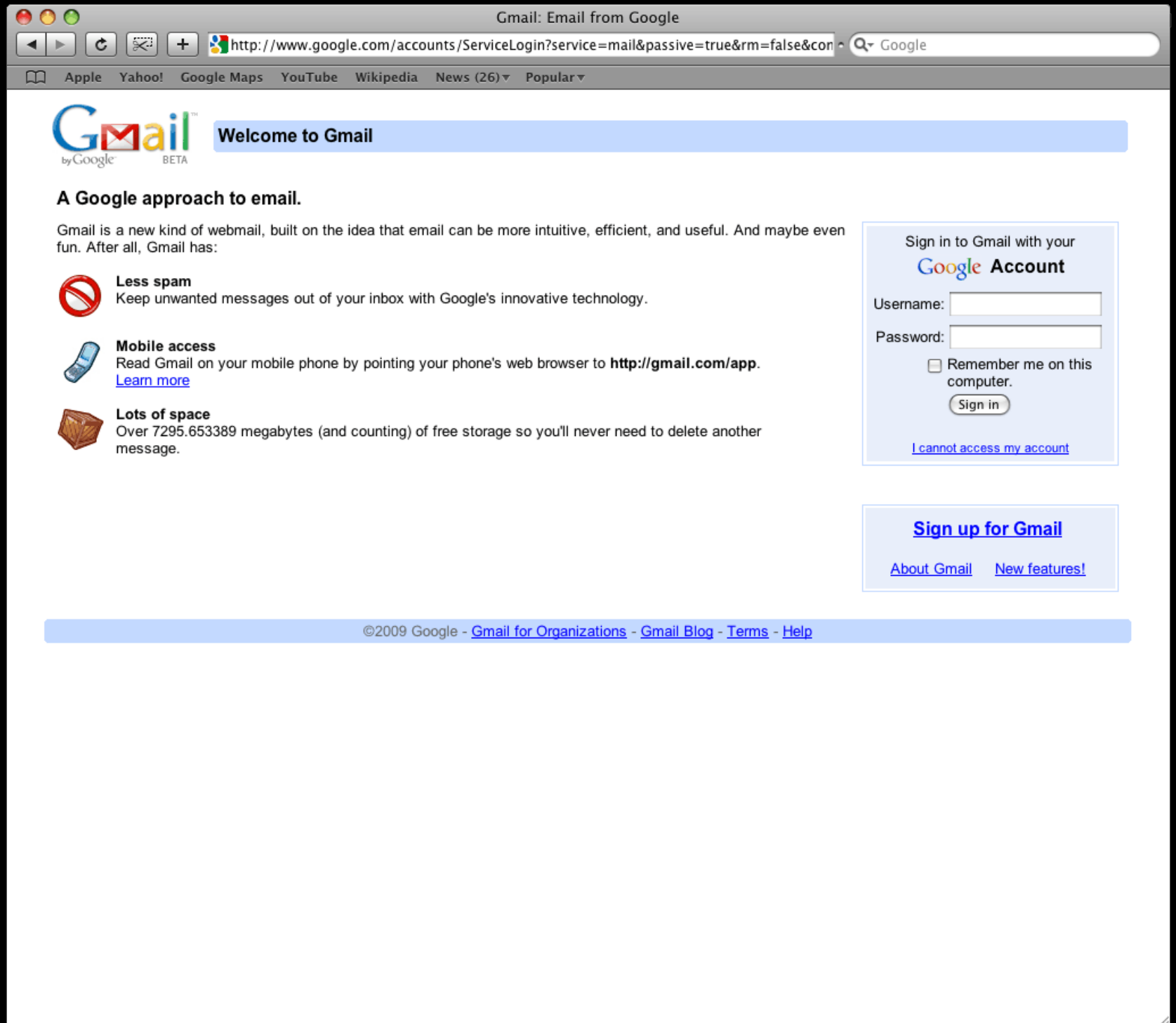
How Does It Look?



How Does It Look?



How Does It Look?



Where can we go from here?

Where do we *need* to go from here?

What's with certificates, anyways?

- X509Certificate
  - Version
  - Serial Number
  - Issuer
  - Validity (not before X or after Y)
  - Subject
  - PublicKey
- SignatureAlgorithm
- Signature

What's with certificates, anyways?

- X509Certificate
  - Version
  - Serial Number
  - Issuer
  - Validity (not before X or after Y)
  - Subject
  - PublicKey
- SignatureAlgorithm
- Signature

What's with certificates, anyways?

X509Certificate

- Version
- Serial Number
- Issuer
- Validity (not before X or after Y)
- Subject
- PublicKey

SignatureAlgorithm

Signature

What's with certificates, anyways?

X509Certificate

- Version
- Serial Number
- Issuer
- Validity (not before X or after Y)
- Subject
- PublicKey

SignatureAlgorithm

Signature

What's with certificates, anyways?

X509Certificate

- Version
- Serial Number
- Issuer
- Validity (not before X or after Y)
- Subject
- PublicKey

SignatureAlgorithm

Signature

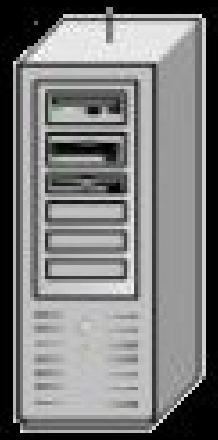
```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            01:2a:39:76:0d:3f:4f:c9:0b:e7:bd:2b:cf:95:2e:7a
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd., CN=Thawte SGC CA
        Validity
            Not Before: Mar 27 22:20:07 2009 GMT
            Not After : Mar 27 22:20:07 2010 GMT
        Subject: C=US, ST=California, L=Mountain View, O=Google Inc, CN=www.google.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:d6:b9:e1:ad:b8:61:0b:1f:4e:b6:3c:09:3d:ab:
                    e8:e3:2b:b6:e8:a4:3a:78:2f:d3:51:20:22:45:95:
                    d8:00:91:33:9a:a7:a2:48:ea:30:57:26:97:66:c7:
                    5a:ef:f1:9b:0c:3f:e1:b9:7f:7b:c3:c7:cc:af:9c:
                    d0:1f:3c:81:15:10:58:fc:06:b3:bf:bc:9c:02:b9:
                    51:dc:fb:a6:b9:17:42:e6:46:e7:22:cf:6c:27:10:
                    fe:54:e6:92:6c:0c:60:76:9a:ce:f8:7f:ac:b8:5a:
                    08:4a:dc:b1:64:bd:a0:74:41:b2:ac:8f:86:9d:1a:
                    de:58:09:fd:6c:0a:25:e0:79
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication, Netscape Server Gated Crypto
            X509v3 CRL Distribution Points:
                URI:http://crl.thawte.com/ThawteSGCCA.crl

            Authority Information Access:
                OCSP - URI:http://ocsp.thawte.com
                CA Issuers - URI:http://www.thawte.com/repository/Thawte_SGC_CA.crt

            X509v3 Basic Constraints: critical
                CA:FALSE
    Signature Algorithm: sha1WithRSAEncryption
        39:b6:fb:11:bc:33:2c:c3:90:48:e3:6e:c3:9b:38:b1:42:d1:
        00:09:58:63:a0:e1:98:1c:85:f2:ef:10:1d:60:4e:51:09:62:
        f5:05:bd:9d:4f:87:6c:98:72:07:80:c3:59:48:14:e2:d6:ef:
        d0:8f:33:6a:68:31:fa:b7:bb:85:cc:f7:c7:47:7b:67:93:3c:
        c3:16:51:9b:6f:87:20:fd:67:4c:2b:ea:6a:49:db:11:d1:bd:
        d7:95:22:43:7a:06:7b:4e:f6:37:8e:a2:b9:cf:1f:a5:d2:bd:
        3b:04:97:39:b3:0f:fa:38:b5:af:55:20:88:60:93:f2:de:db:
```

The Big Three

- Secrecy
- Authenticity
- Integrity

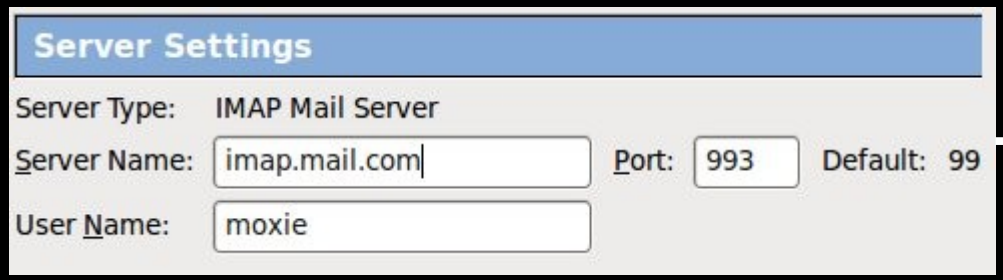
SSL/TLS Handshake Beginnings

ClientHello



ServerHello, ServerCertificate

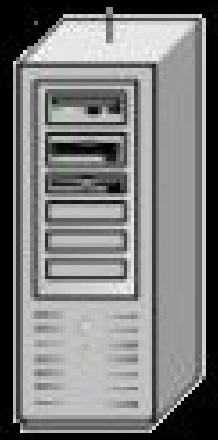
SSL Handshake Beginnings



- X509Certificate
  - Version
  - Serial Number
  - Issuer
  - Validity
  - **Subject**
  - PublicKey
- SignatureAlgorithm
- Signature

The Problems For Us Begin

Attacker



ClientHello

ServerHello,
ServerCertificate?

Let's start by looking back once more.

In 2000, things were different.

Notaries!

Identification!

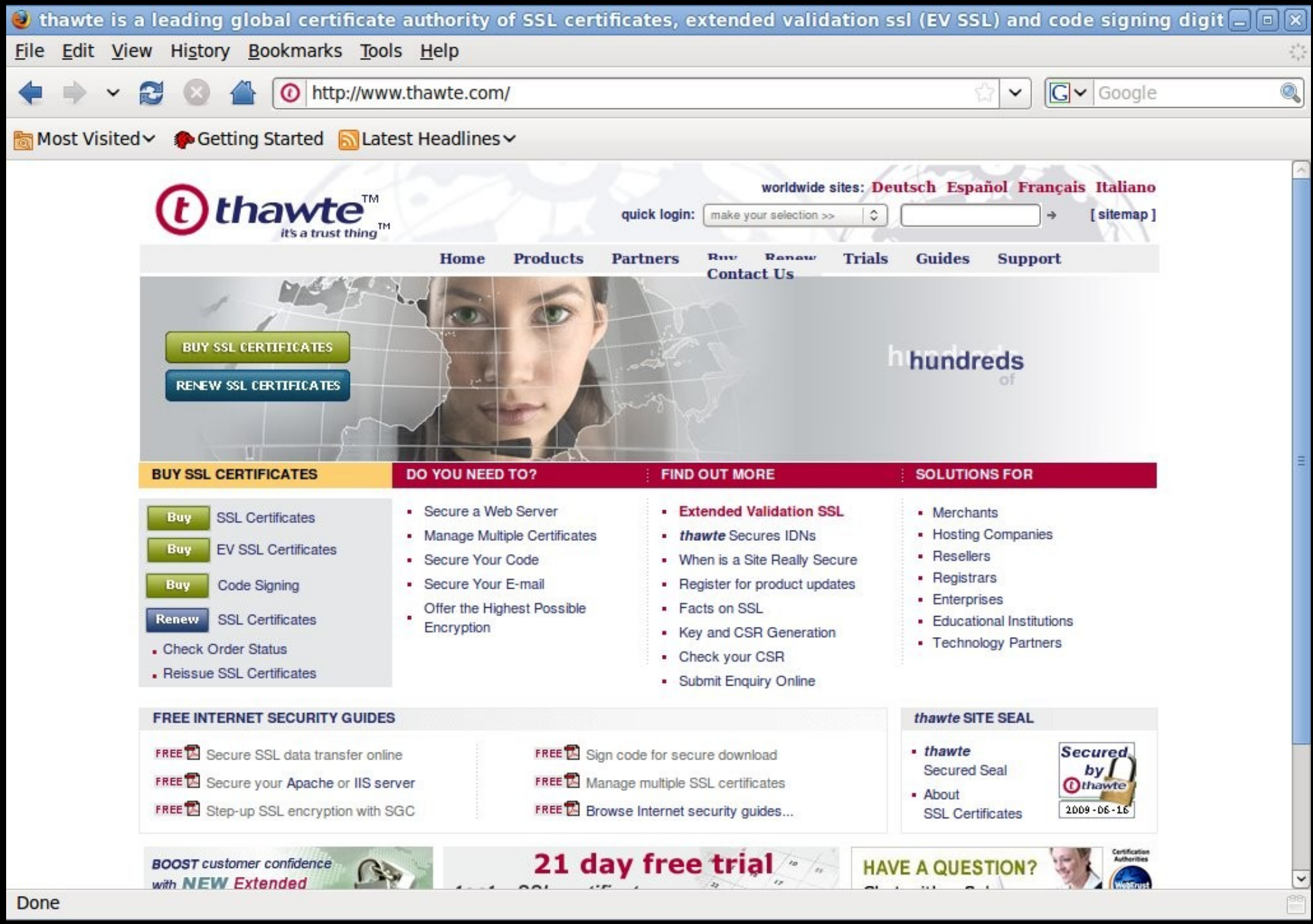
Phone Calls!

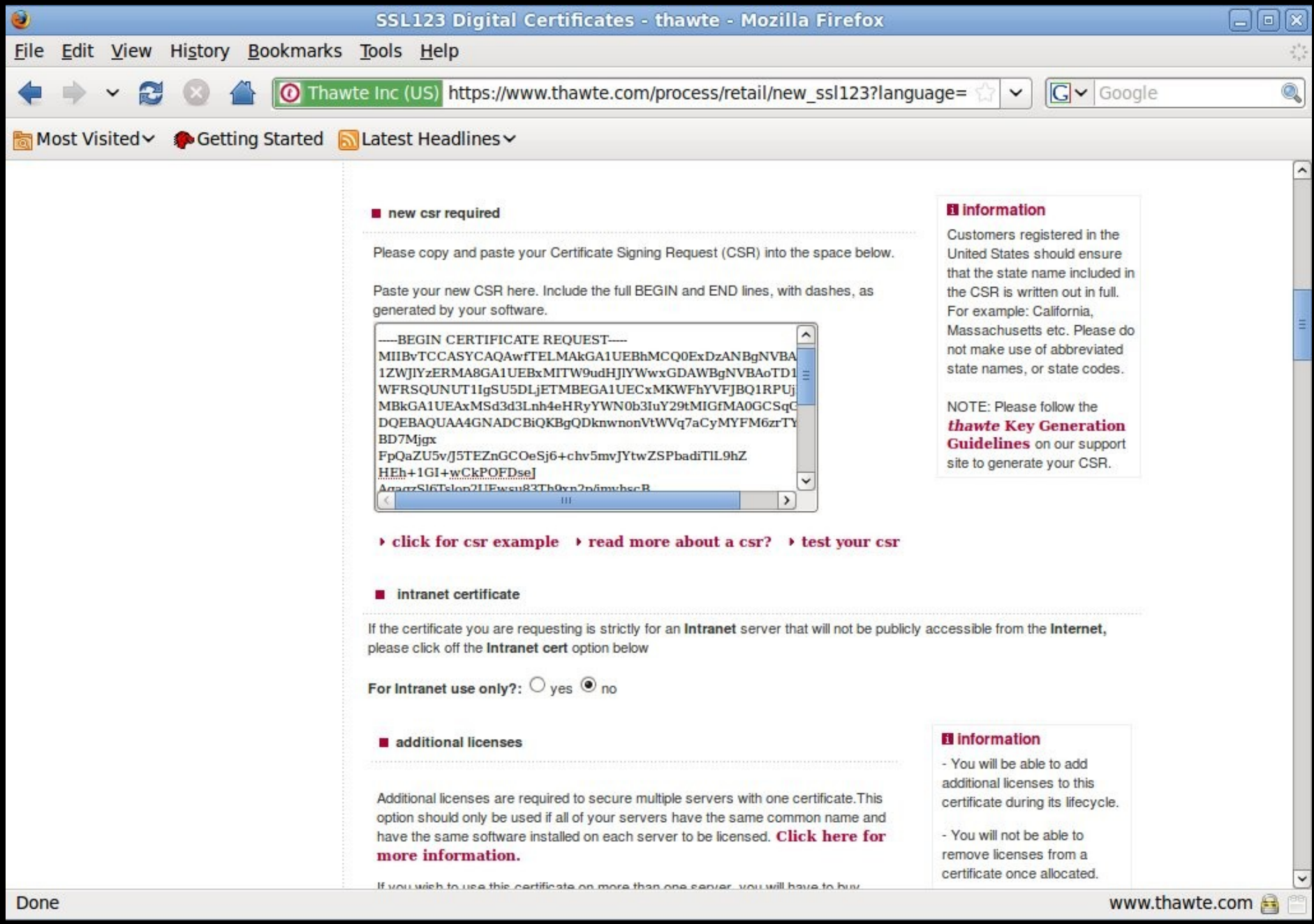
Actual people involved...

That is a bygone era

These days it's all about: online domain validation

Moxie Marlinspike
Institute For Disruptive Studies


thawte is a leading global certificate authority of SSL certificates, extended validation ssl (EV SSL) and code signing digit
File Edit View History Bookmarks Tools Help
http://www.thawte.com/
Google
Most Visited
Getting Started
Latest Headlines
thawte™
it's a trust thing™
worldwide sites: Deutsch Español Français Italiano
quick login: make your selection >>
[sitemap]
Home Products Partners Buy Renew Trials Guides Support
Contact Us
BUY SSL CERTIFICATES
RENEW SSL CERTIFICATES
hundreds
BUY SSL CERTIFICATES
Buy SSL Certificates
Buy EV SSL Certificates
Buy Code Signing
Renew SSL Certificates
Check Order Status
Reissue SSL Certificates
DO YOU NEED TO?
Secure a Web Server
Manage Multiple Certificates
Secure Your Code
Secure Your E-mail
Offer the Highest Possible Encryption
FIND OUT MORE
Extended Validation SSL
thawte Secures IDNs
When is a Site Really Secure
Register for product updates
Facts on SSL
Key and CSR Generation
Check your CSR
Submit Enquiry Online
SOLUTIONS FOR
Merchants
Hosting Companies
Resellers
Registrars
Enterprises
Educational Institutions
Technology Partners
FREE INTERNET SECURITY GUIDES
FREE Secure SSL data transfer online
FREE Secure your Apache or IIS server
FREE Step-up SSL encryption with SGC
FREE Sign code for secure download
FREE Manage multiple SSL certificates
FREE Browse Internet security guides...
thawte SITE SEAL
thawte Secured Seal
About SSL Certificates
Secured by thawte
2009-08-18
BOOST customer confidence with NEW Extended
21 day free trial
HAVE A QUESTION?
Done

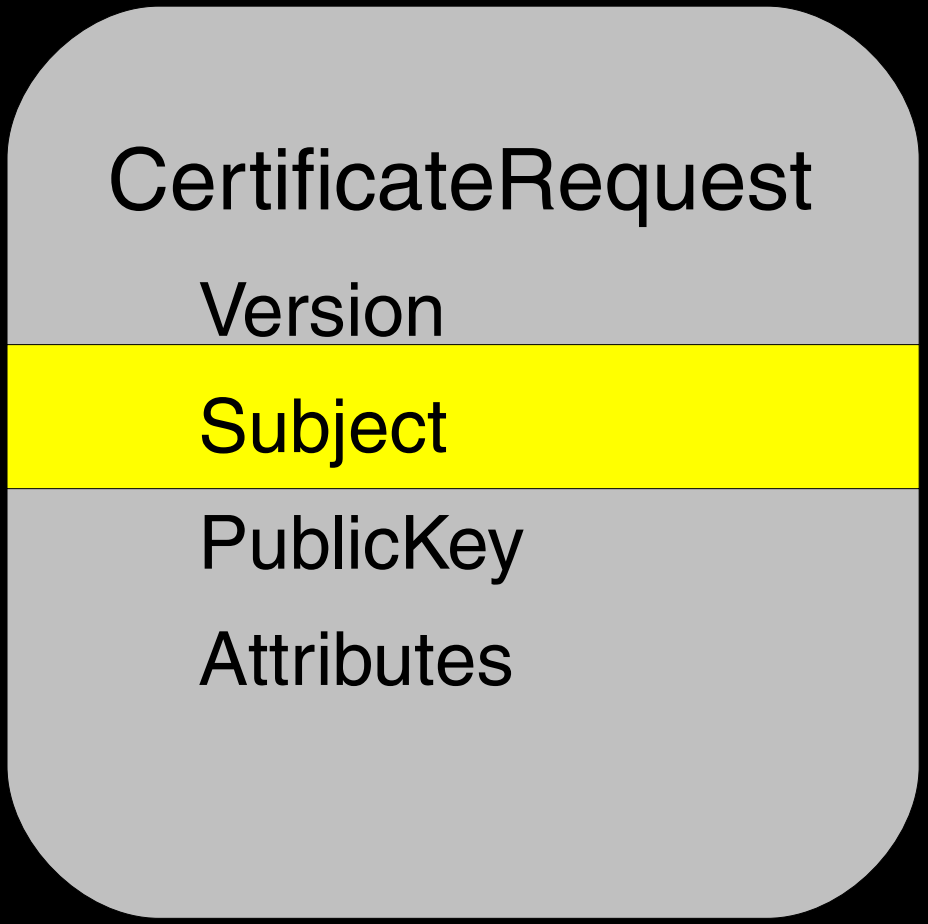

SSL123 Digital Certificates - thawte - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Thawte Inc (US) https://www.thawte.com/process/retail/new_ssl123?language=
Google
Most Visited
Getting Started
Latest Headlines
new csr required
Please copy and paste your Certificate Signing Request (CSR) into the space below.
Paste your new CSR here. Include the full BEGIN and END lines, with dashes, as generated by your software.
-----BEGIN CERTIFICATE REQUEST-----
MIIBvTCCASYCAQAwfTELMAkGA1UEBhMCQ0ExDzANBgNVBA
1ZWJlYzERMA8GA1UEBxMITW9udHJlYWwxGDAWBgNVBAoTD1
WFRSQUNUT1IgSU5DLjETMBEGA1UECxMKWFhYVFJBQ1RPUj
MBkGA1UEAxMSd3d3Lnh4eHRyYWN0b3IuY29tMIGfMA0GCSqG
DQEBAQUAA4GNADCBiQKBgQDknwnonVtWVq7aCyMYFM6zrTY
BD7Mjgx
FpQaZU5v/J5TEZnGCOeSj6+chv5mvJYtwZSPbadiTlL9hZ
HEh+1GI+wCkPOFDseJ
click for csr example
read more about a csr?
test your csr
information
Customers registered in the United States should ensure that the state name included in the CSR is written out in full. For example: California, Massachusetts etc. Please do not make use of abbreviated state names, or state codes.
NOTE: Please follow the thawte Key Generation Guidelines on our support site to generate your CSR.
intranet certificate
If the certificate you are requesting is strictly for an Intranet server that will not be publicly accessible from the Internet, please click off the Intranet cert option below
For Intranet use only?: yes no
additional licenses
Additional licenses are required to secure multiple servers with one certificate.This option should only be used if all of your servers have the same common name and have the same software installed on each server to be licensed. Click here for more information.
information
- You will be able to add additional licenses to this certificate during its lifecycle.
- You will not be able to remove licenses from a certificate once allocated.
Done
www.thawte.com

PKCS #10

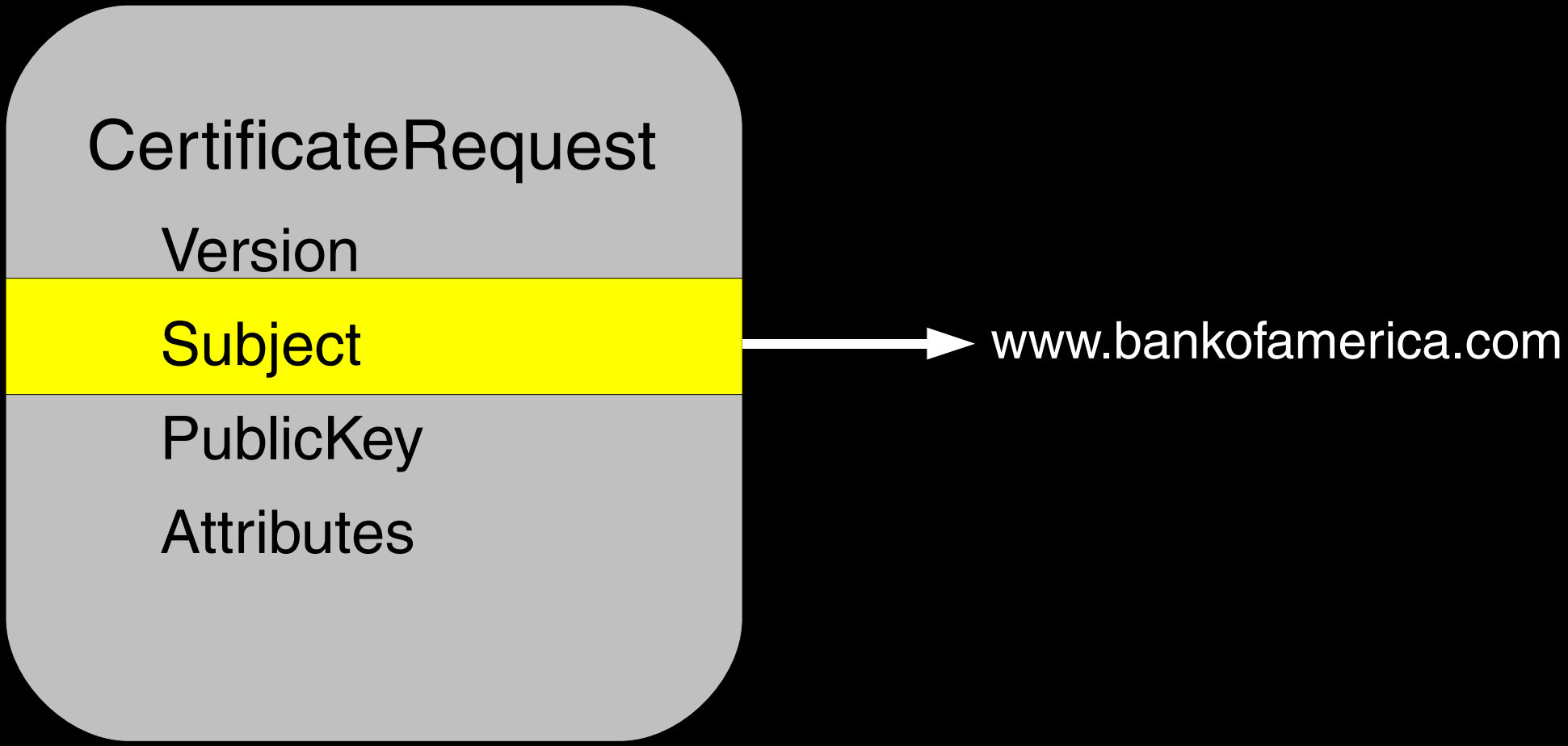
CertificateRequest

- Version
- Subject
- PublicKey
- Attributes

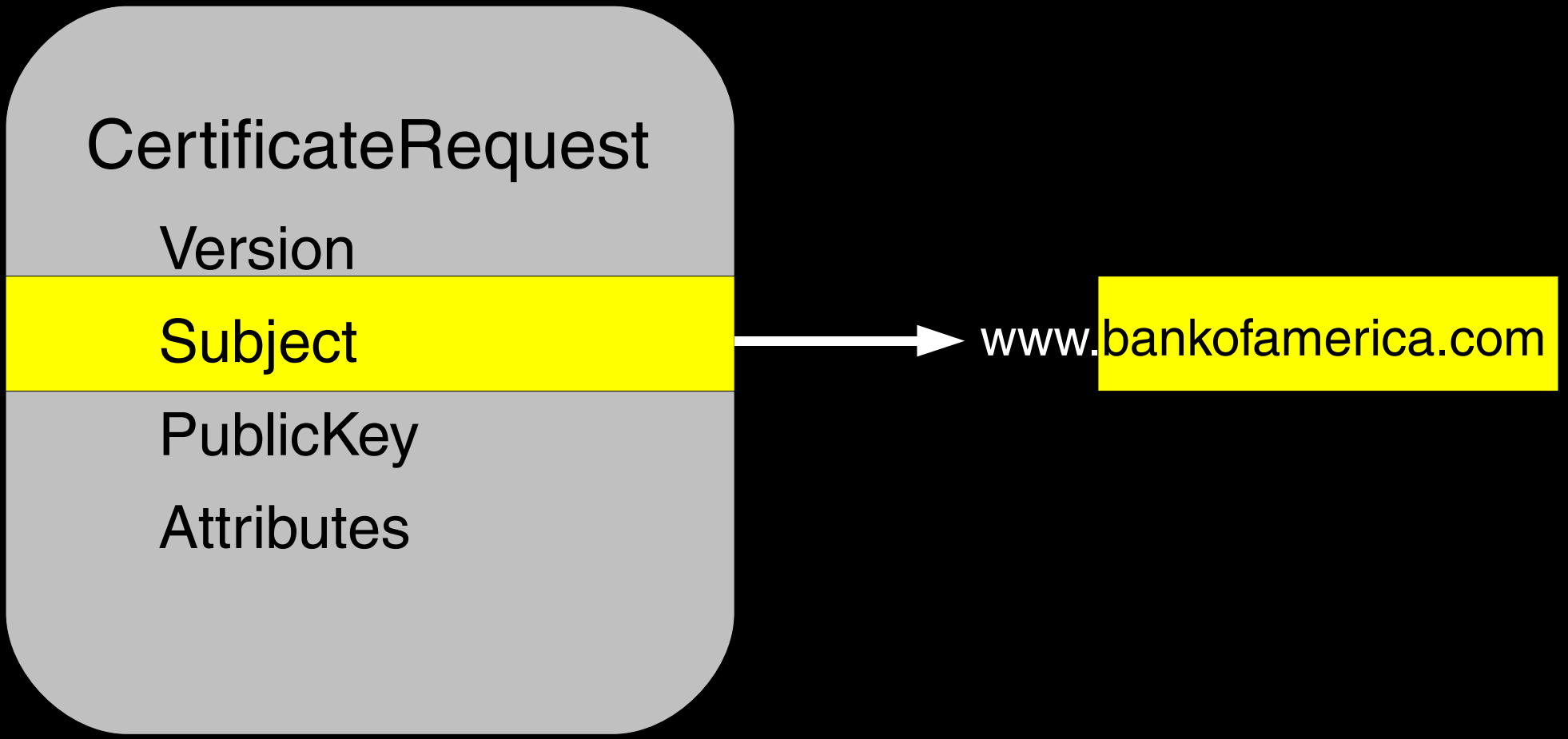
PKCS #10


CertificateRequest
Version
Subject
PublicKey
Attributes

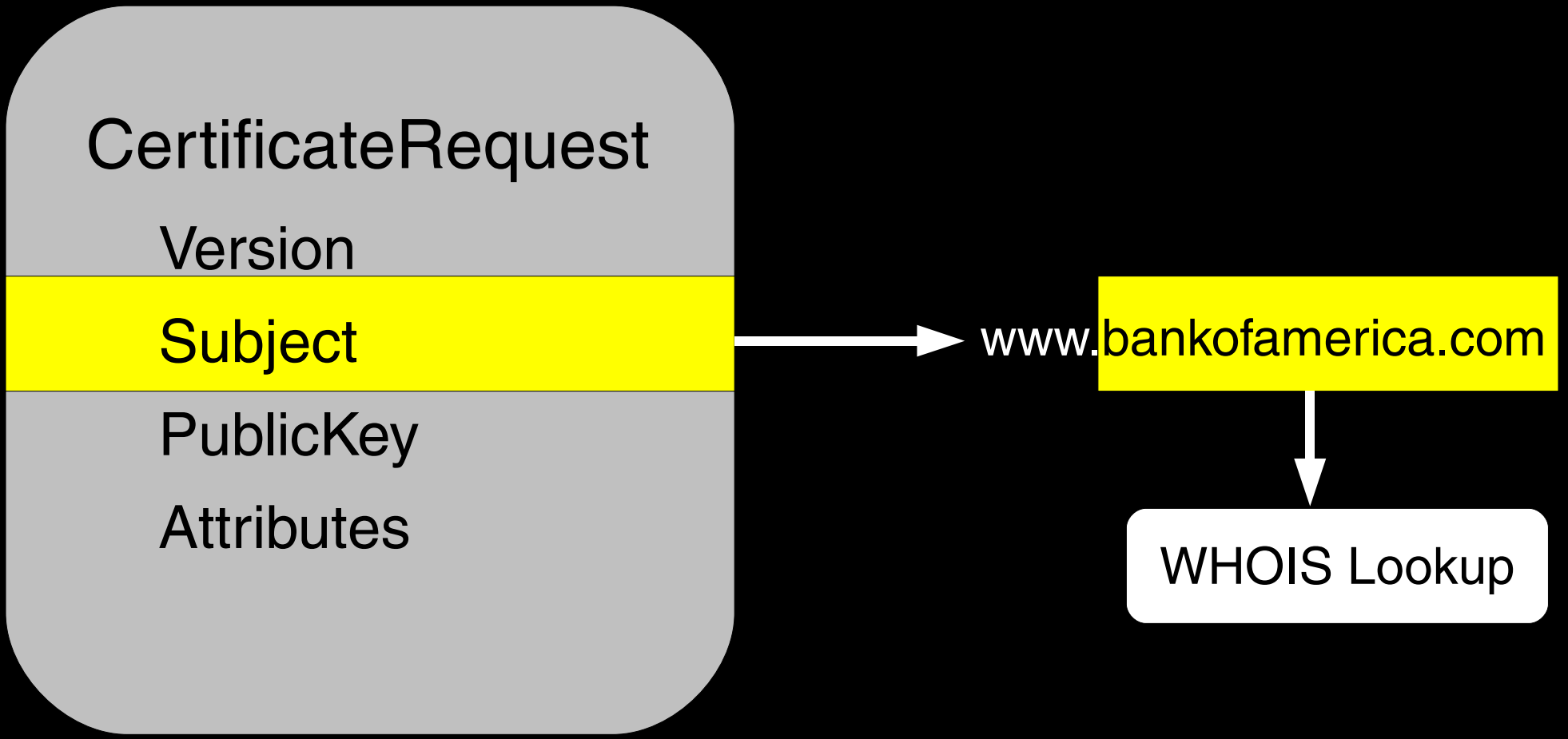
PKCS #10


CertificateRequest
Version
Subject
PublicKey
Attributes
www.bankofamerica.com

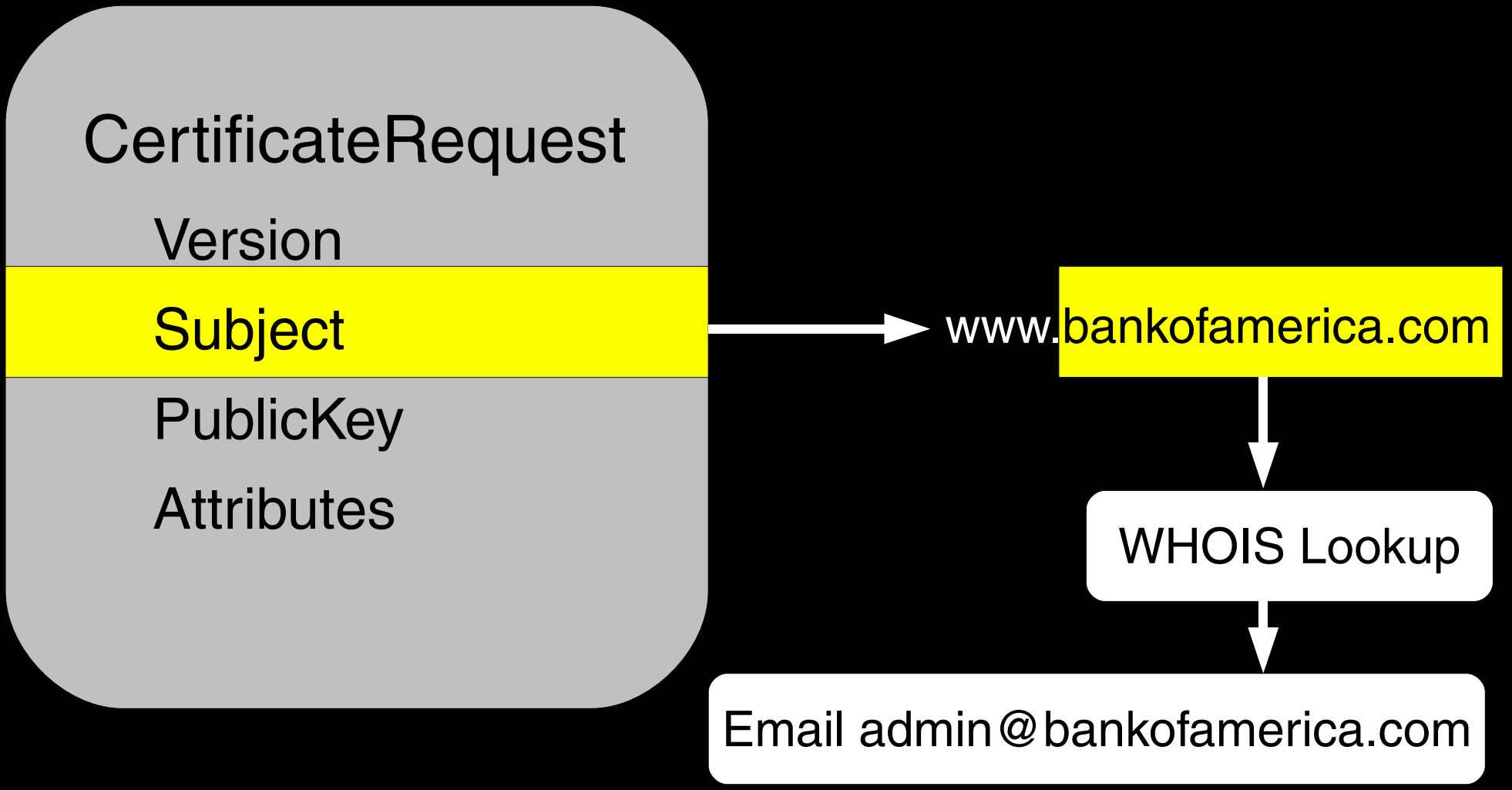
PKCS #10


CertificateRequest
Version
Subject
PublicKey
Attributes
www.bankofamerica.com

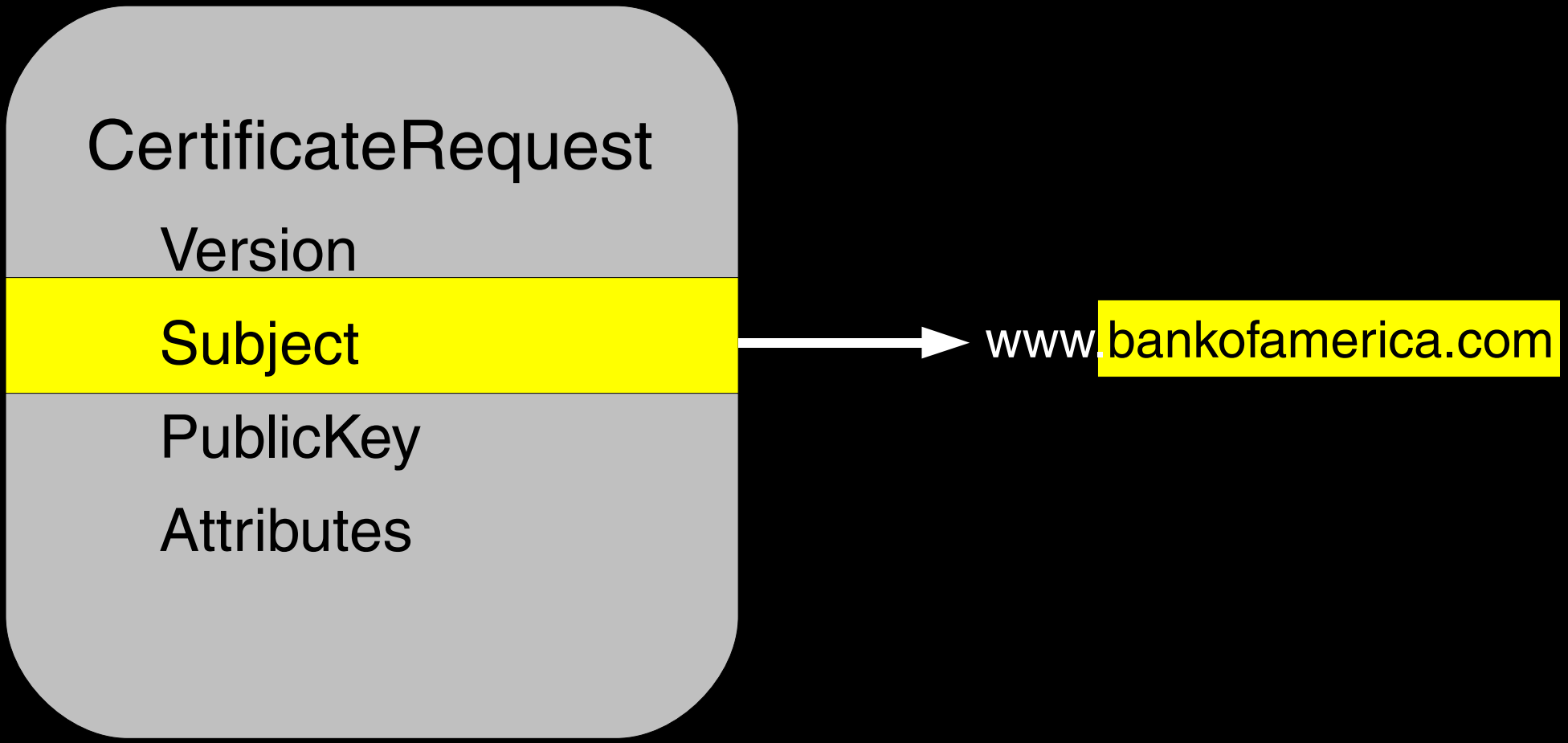
PKCS #10


CertificateRequest
Version
Subject
PublicKey
Attributes
www.bankofamerica.com
WHOIS Lookup

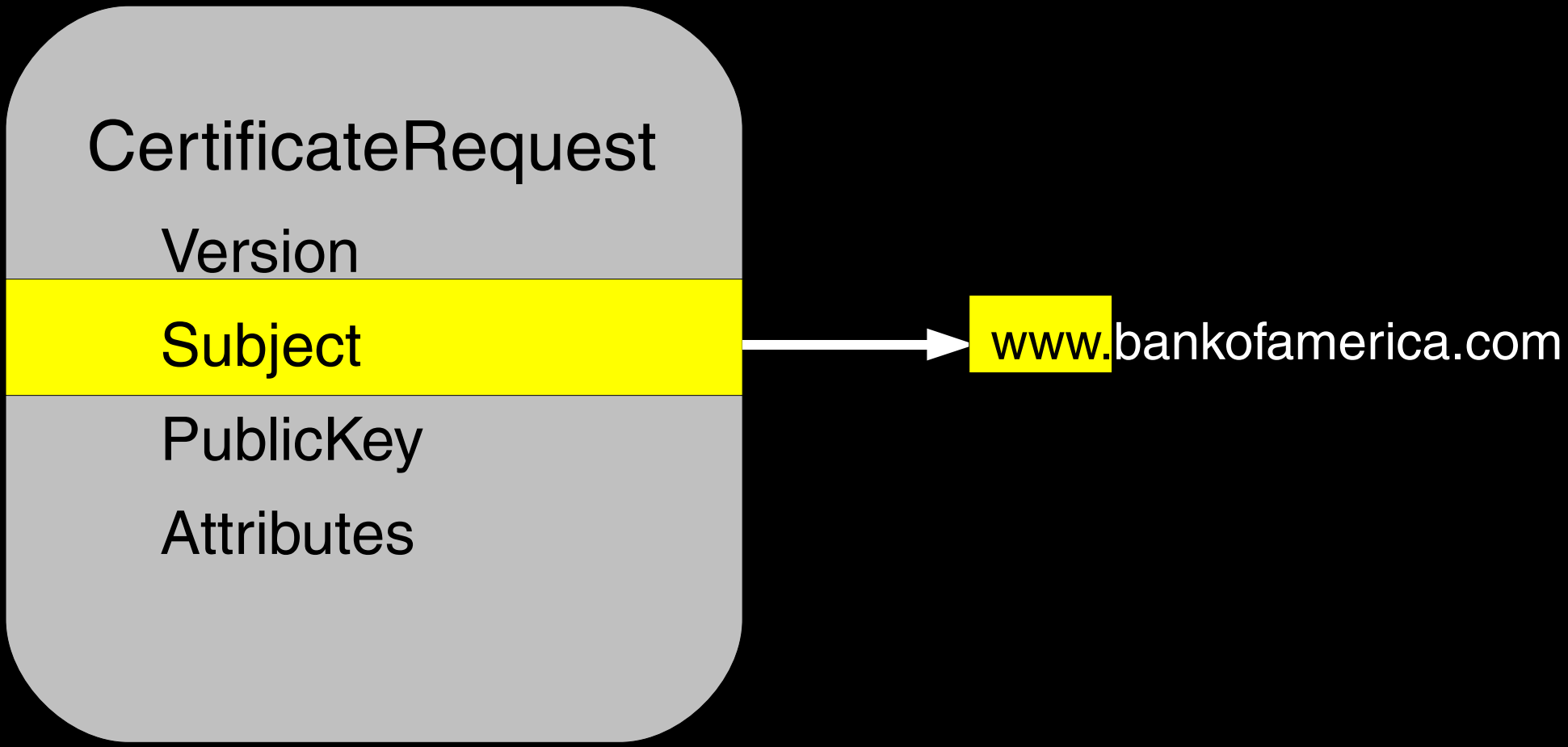
PKCS #10


CertificateRequest
Version
Subject
PublicKey
Attributes
www.bankofamerica.com
WHOIS Lookup
Email admin@bankofamerica.com

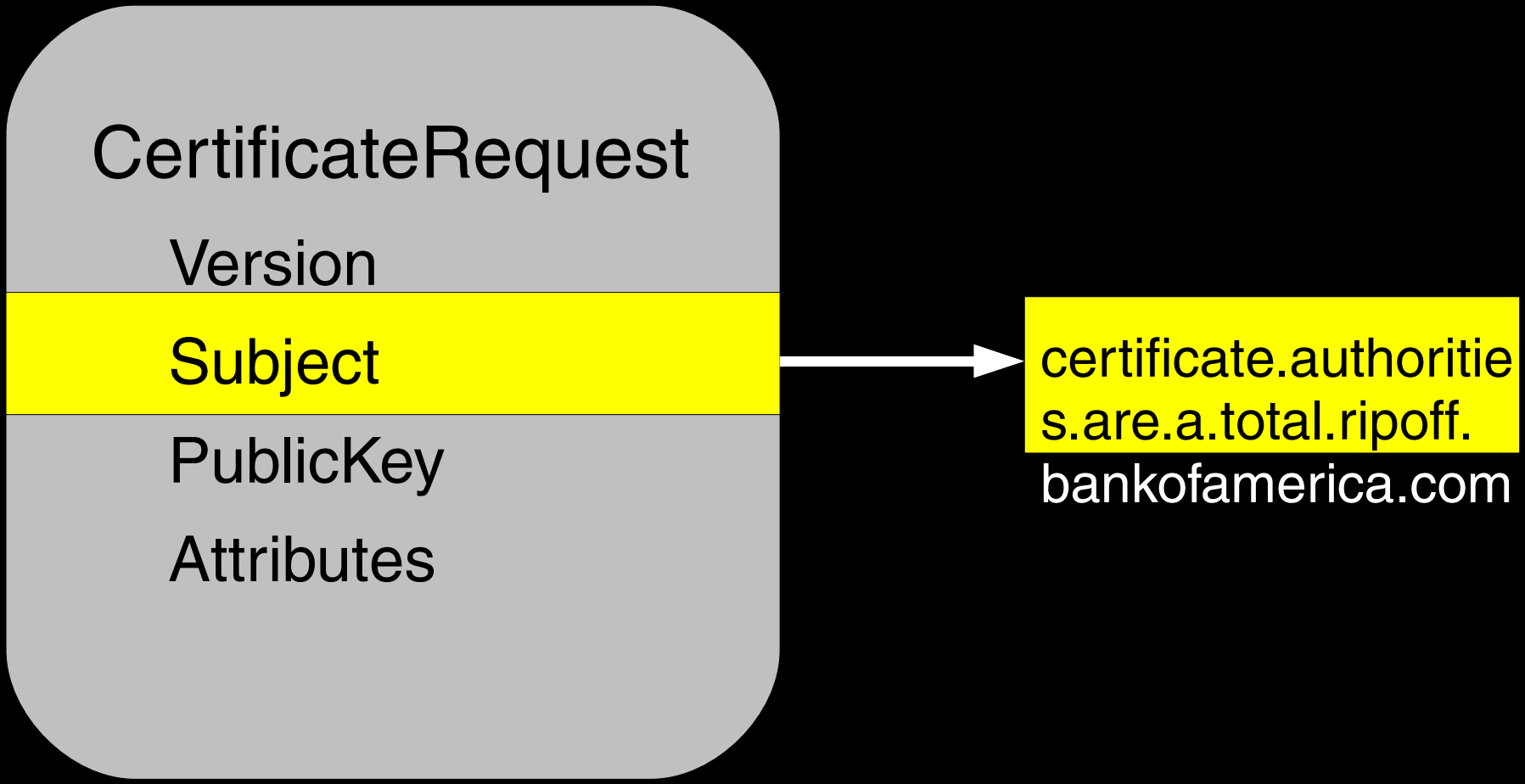
PKCS #10


CertificateRequest
Version
Subject
PublicKey
Attributes
www.bankofamerica.com

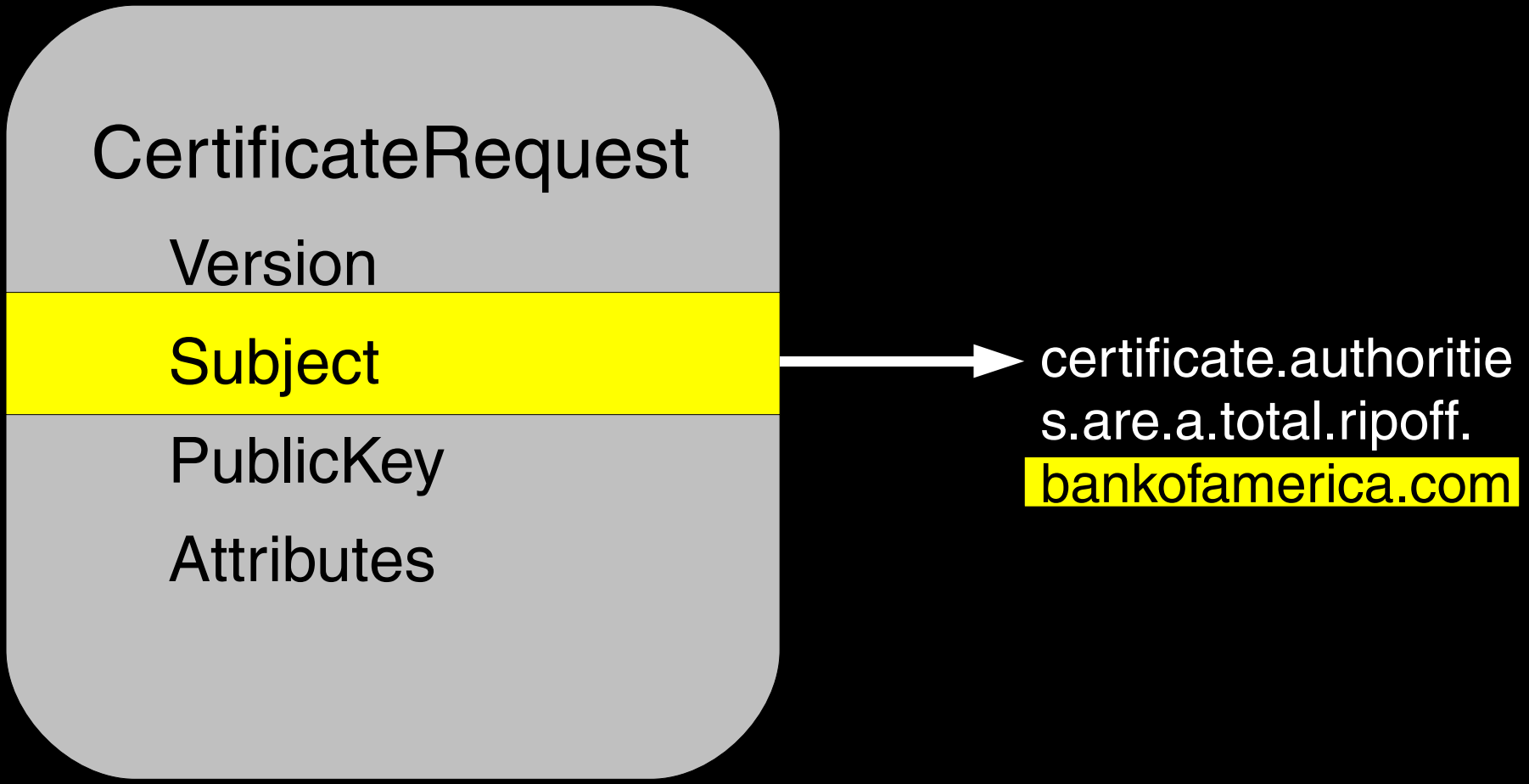
PKCS #10


CertificateRequest
Version
Subject
PublicKey
Attributes
www.bankofamerica.com

PKCS #10


CertificateRequest
Version
Subject
PublicKey
Attributes
certificate.authorities.are.a.total.ripoff.
bankofamerica.com

PKCS #10


CertificateRequest
Version
Subject
PublicKey
Attributes
certificate.authorities.are.a.total.ripoff.
bankofamerica.com

Subjects

DistinguishedName

- Country
- State
- Locale
- Organization
- Organizational Unit
- Common Name

Subjects

DistinguishedName

- Country
- State
- Locale
- Organization
- Organizational Unit
- Common Name

- The X.509 standard is a total nightmare.
- Three revisions, twenty years.
- Parts of the standard have literally been “lost” and then later “found” again.

Subjects

DistinguishedName

- Country
- State
- Locale
- Organization
- Organizational Unit
- Common Name

- The original vision for the DN was that each DN would fit into some global Directory Information Tree.
- In practice, the standard is weak, everyone does everything differently, and the global DIT never materialized.

Subjects

DistinguishedName

- Country
- State
- Locale
- Organization
- Organizational Unit
- Common Name

- “There is nothing in any of these standards that would prevent me from including a 1 gigabit MPEG movie of me playing with my cat as one of the RDN components of the DN in my certificate.”

  -- Bob Jueneman on IETF-PKIX

Subjects

DistinguishedName

- Country
- State
- Locale
- Organization
- Organizational Unit
- Common Name

www.bankofamerica.com

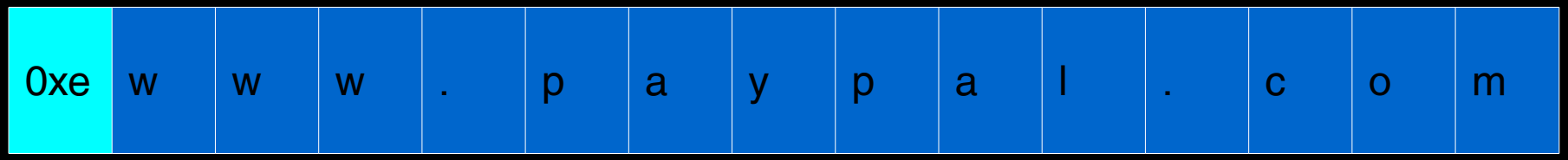
CN

```
commonName      ::=
        SEQUENCE { { 2 5 4 3 }, StringType( SIZE( 1...64 ) ) }
```

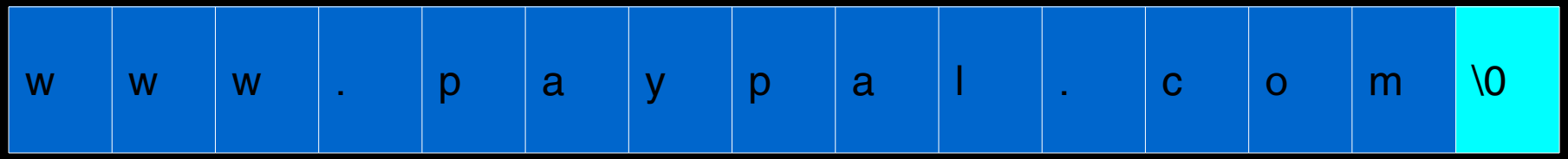
- IA5String:
  - 0x16 – ID
  - 0x05 – Length (5 Chars)
  - 0x76, 0x61, 0x6c, 0x75, 0x65 – v, a, l, u, e

CN Encoding

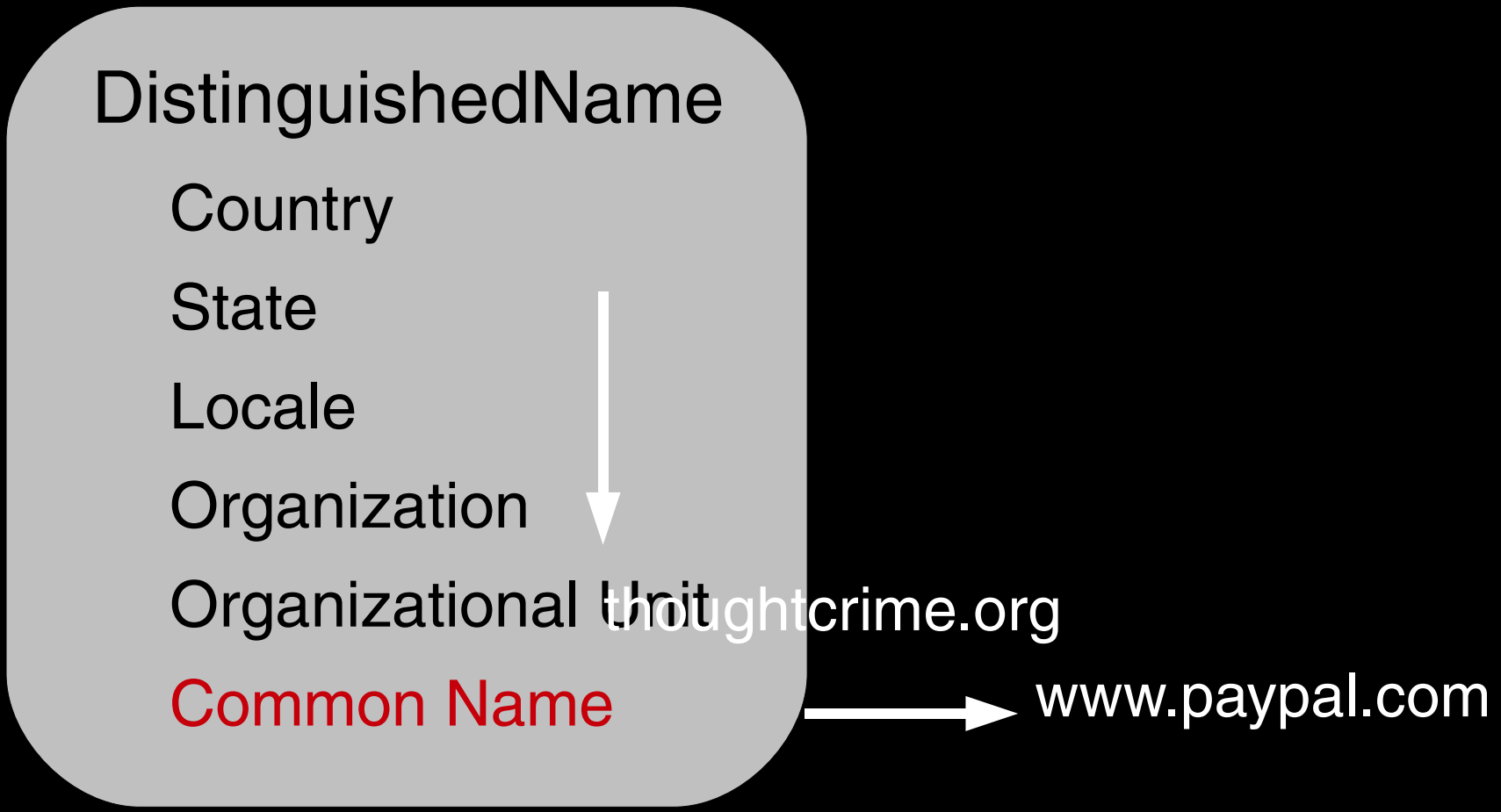
- Essentially, the CN field is represented as a PASCAL String.



- This is different from how C strings are represented.



PKCS #10 Subject



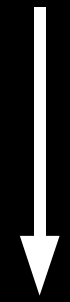
PKCS #10 Subject

Common Name

www.thoughtcrime.org

PKCS #10 Subject

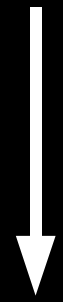
Common Name



verisign.eats.children.thoughtcrime.org

PKCS #10 Subject

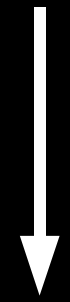
Common Name



iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii.thoughtcrime.org

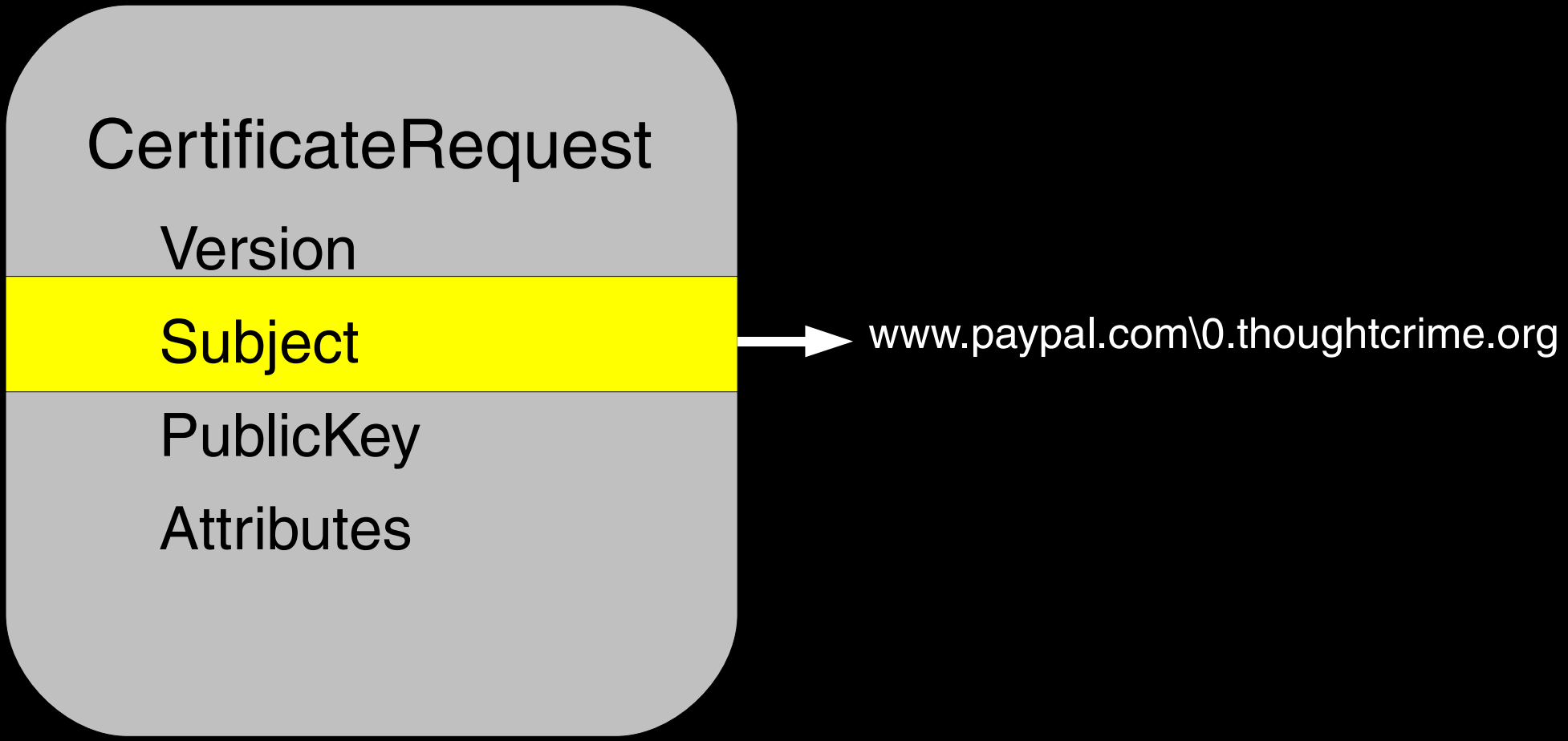
PKCS #10 Subject

Common Name

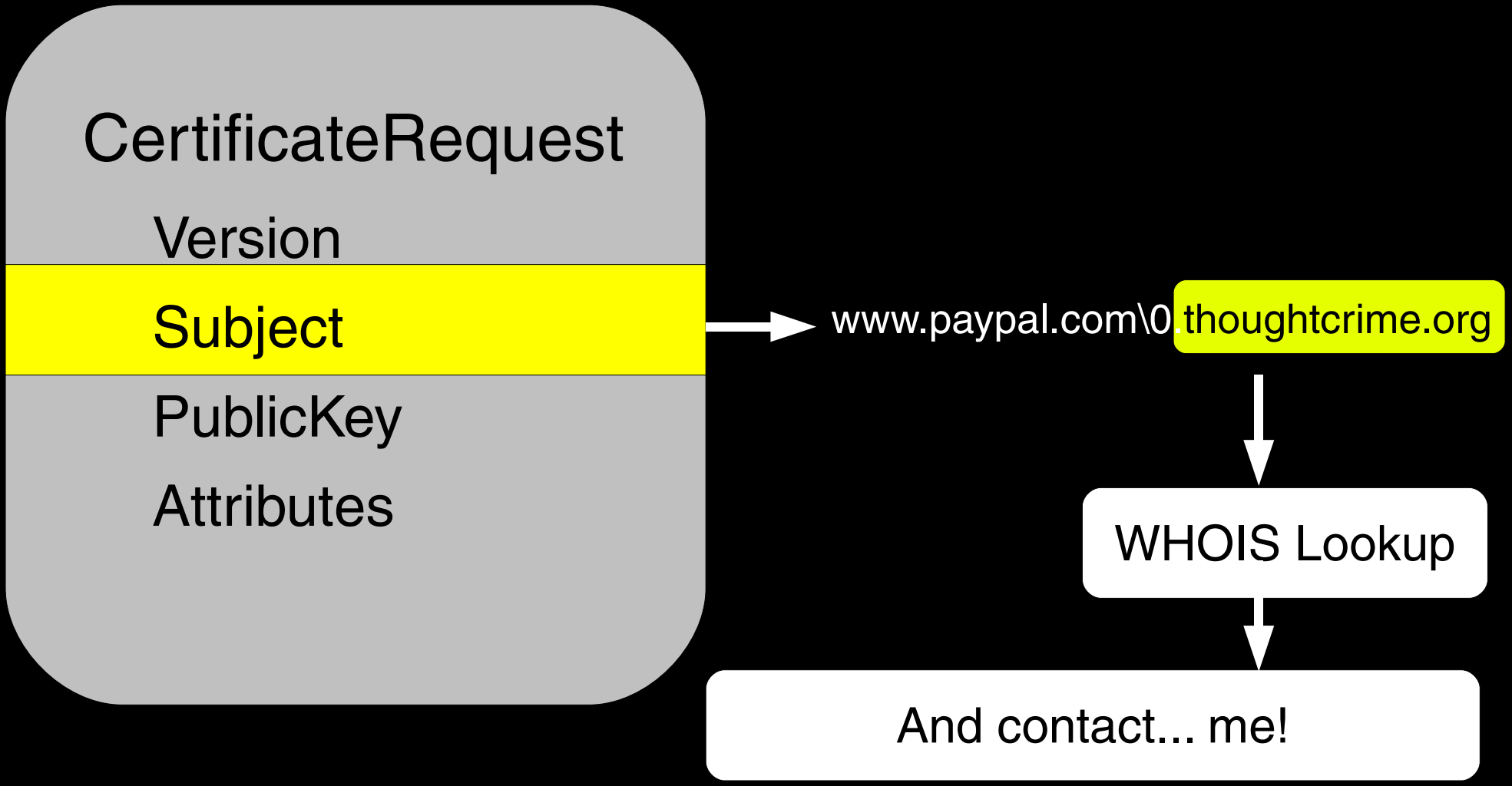


www.paypal.com\0.thoughtcrime.org

PKCS #10 Certificate Signing Request

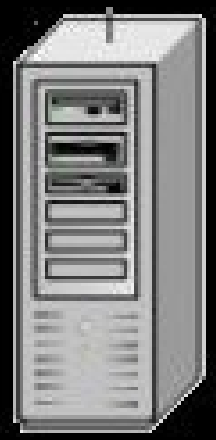


PKCS #10 Certificate Signing Request



Our Original Scenario

Attacker



ClientHello

ServerHello,
ServerCertificate
[www.paypal.com\0.thoughtcrime.org]

Our Original Scenario

X509Certificate

- Version
- Serial Number
- Issuer
- Validity
- Subject
- PublicKey

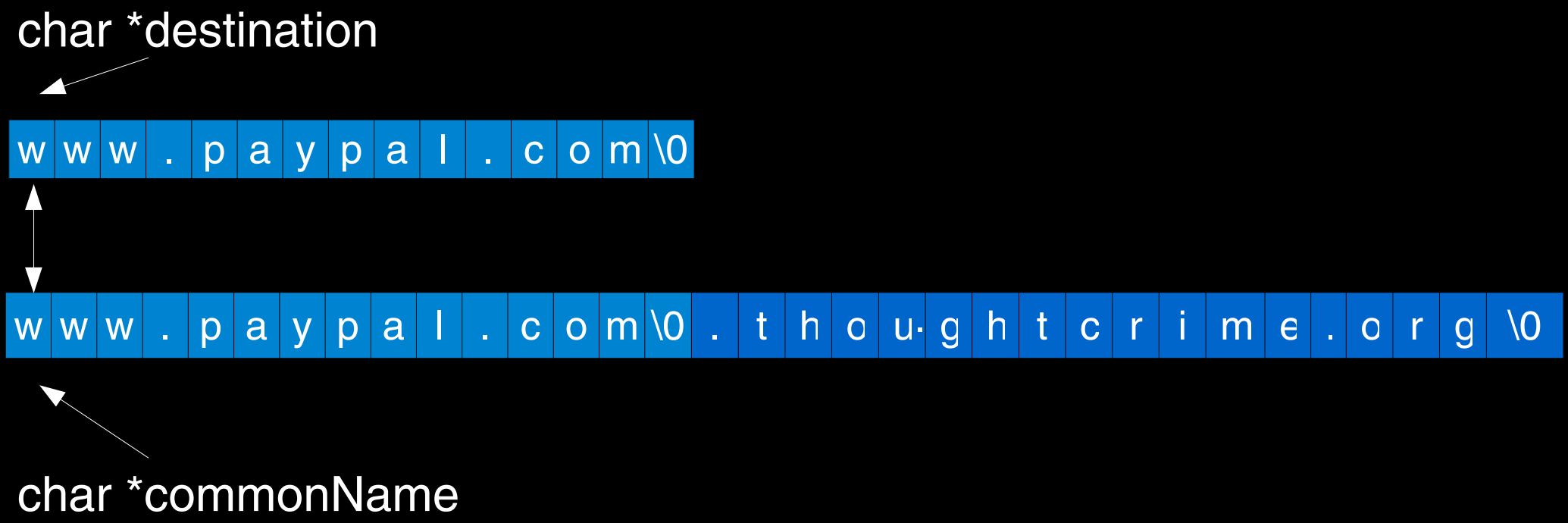
SignatureAlgorithm

Signature

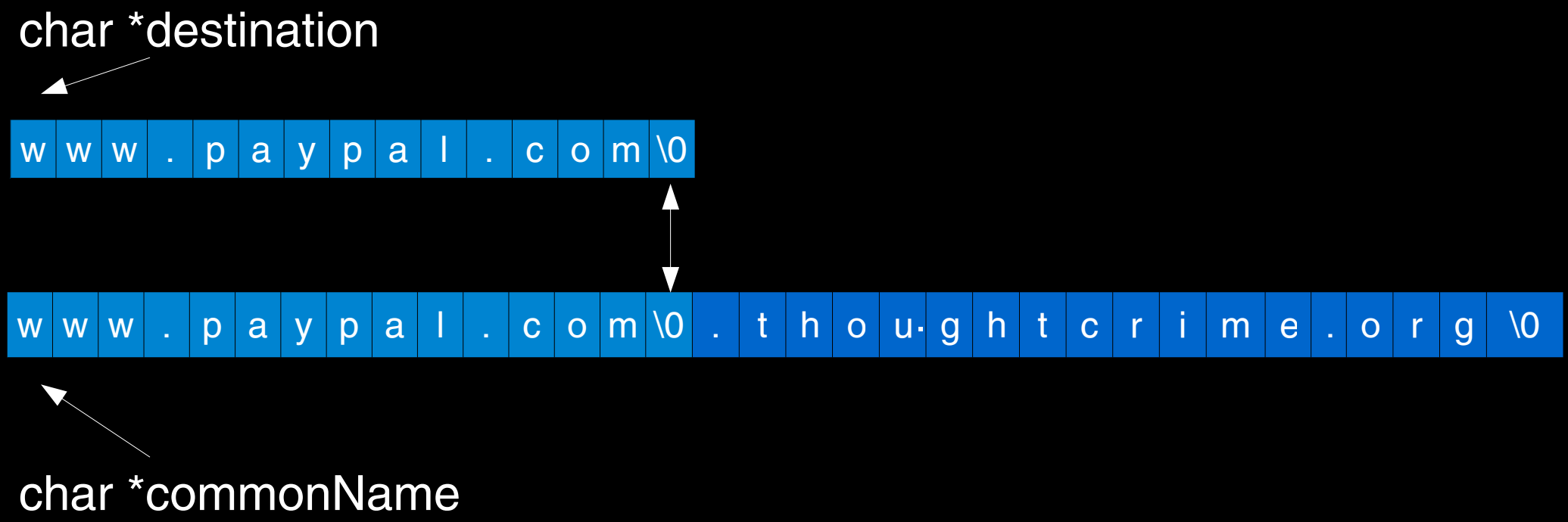
Our Original Scenario

```
char *destination = getDomainWeAreConnectingTo();
char *commonName = getCommonNameFromCertificate();
bool everythingIsOk = (strcmp(destination, commonName) == 0);
```

In memory, though...



In memory, though...

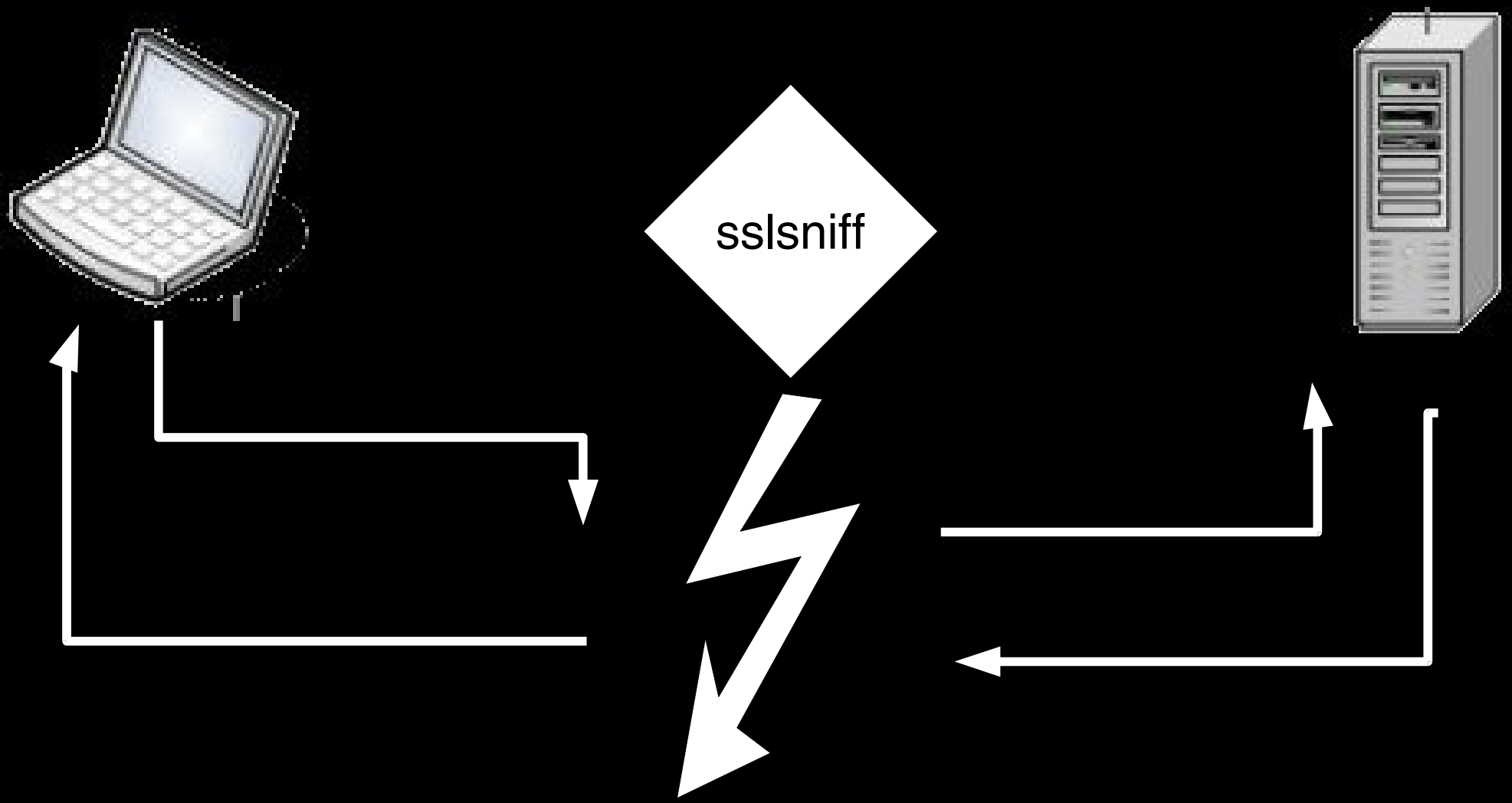


In the eyes of most SSL implementations, this certificate is *completely valid* for www.paypal.com

What are "most" SSL implementations?

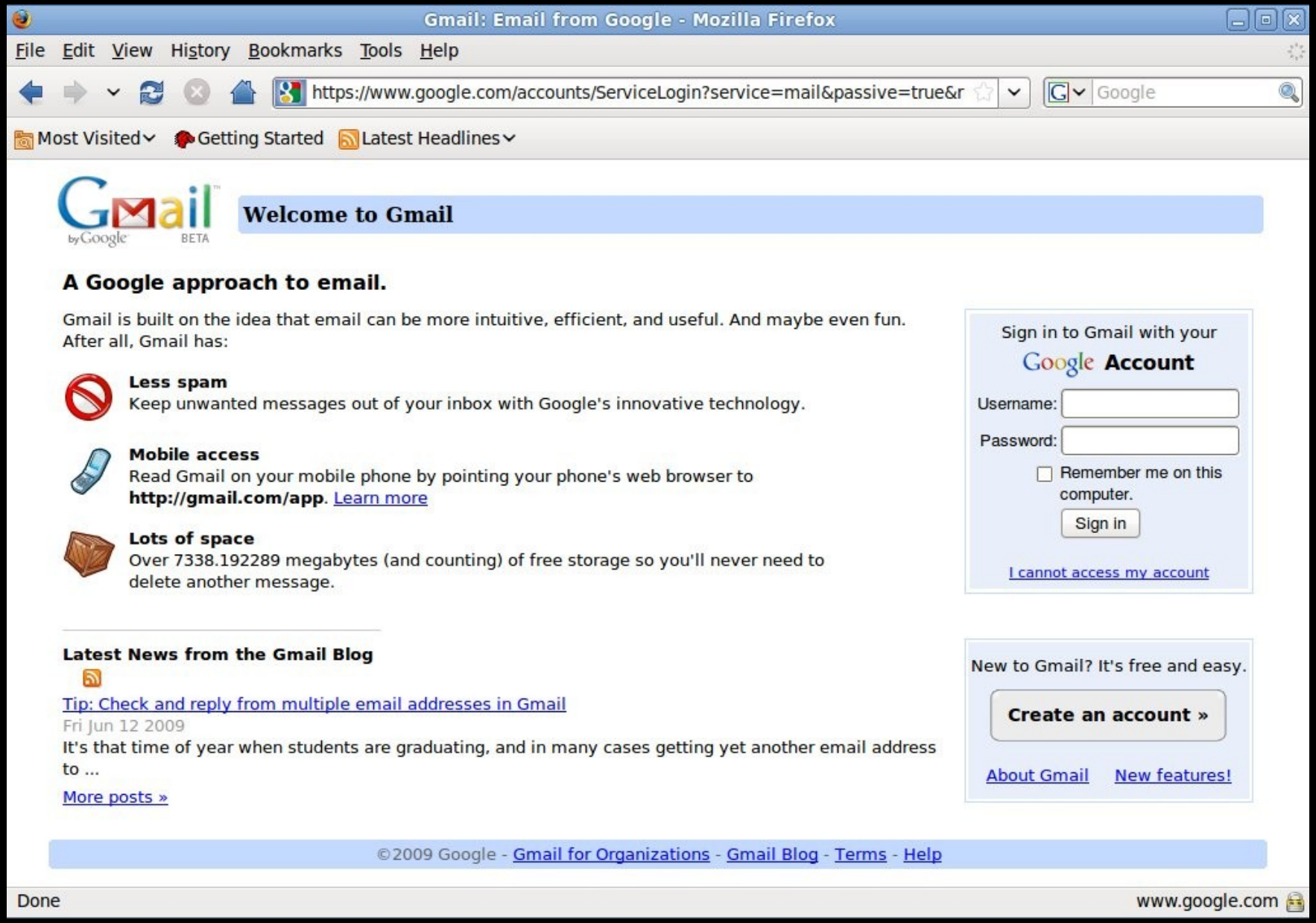
- Web Browsers
  - Firefox (all versions), IE (all versions), Lynx, Curl,
- Mail Clients
  - Thunderbird, Outlook, Evolution
- Chat Clients
  - Pidgin, AIM, irssi, centericq
- SSL VPNs
  - AEP, Citrix, etc...

A First Cut: updated sslsniff

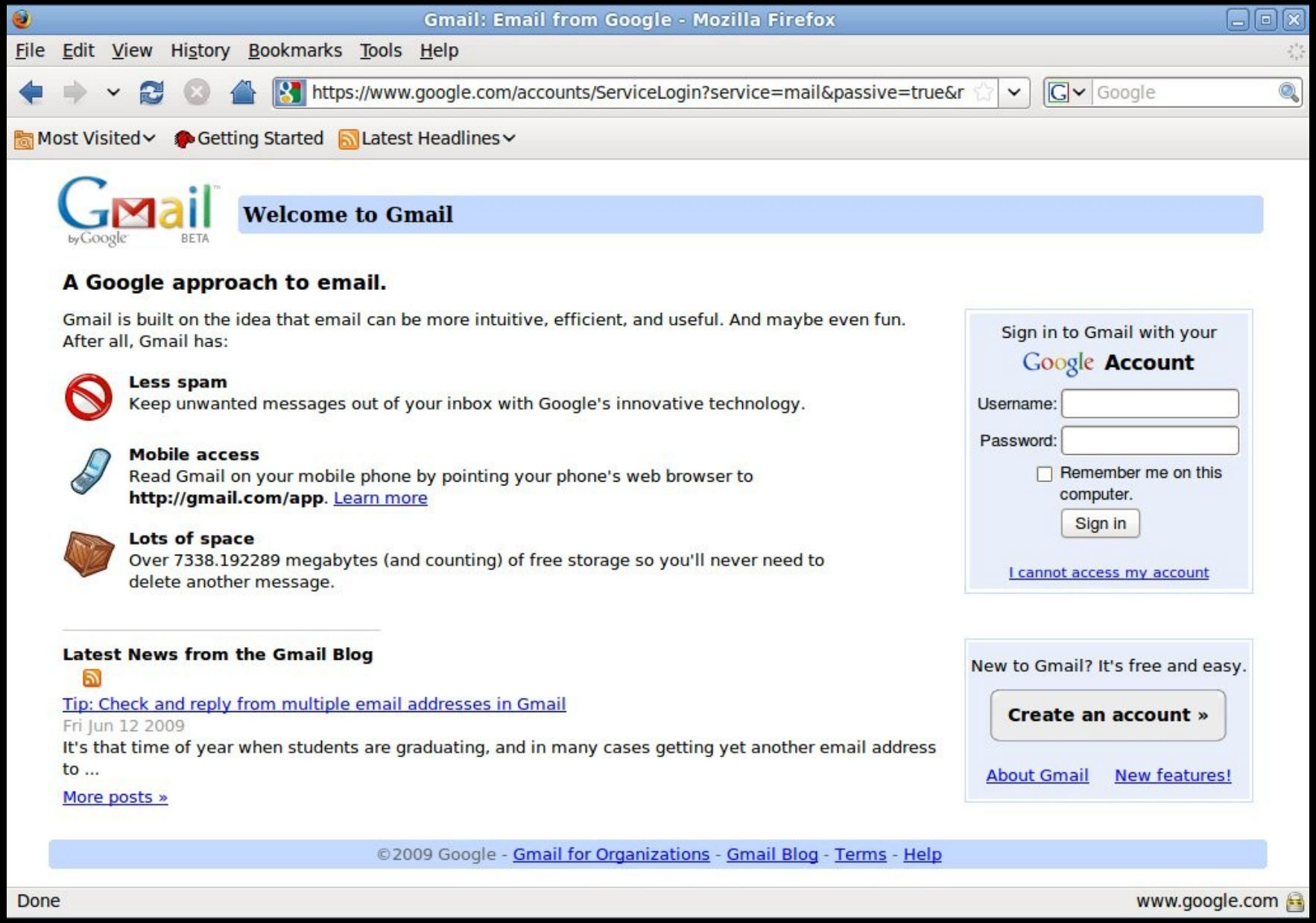


Iff “null prefix attack” certificate
is available

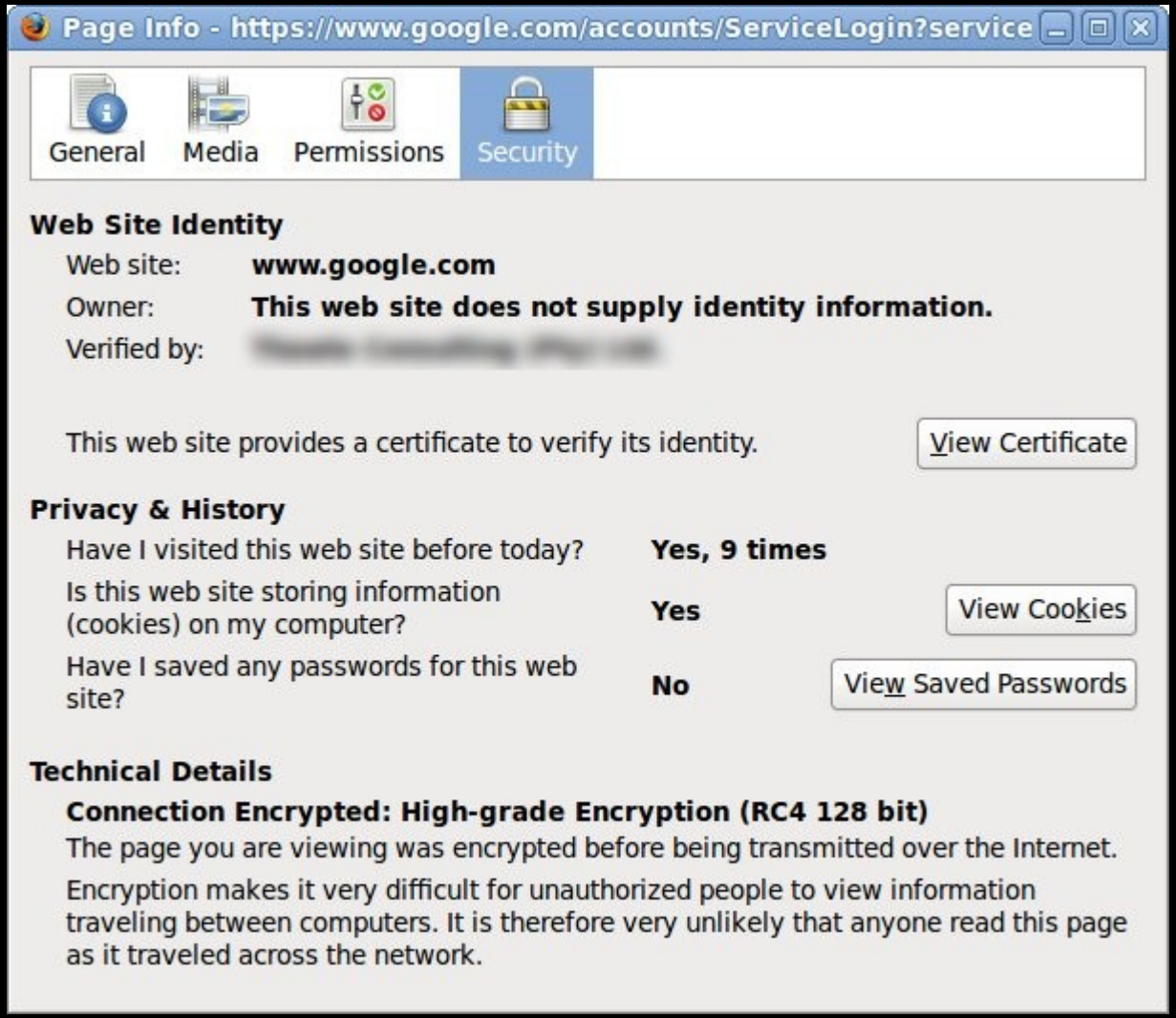
How does it look?



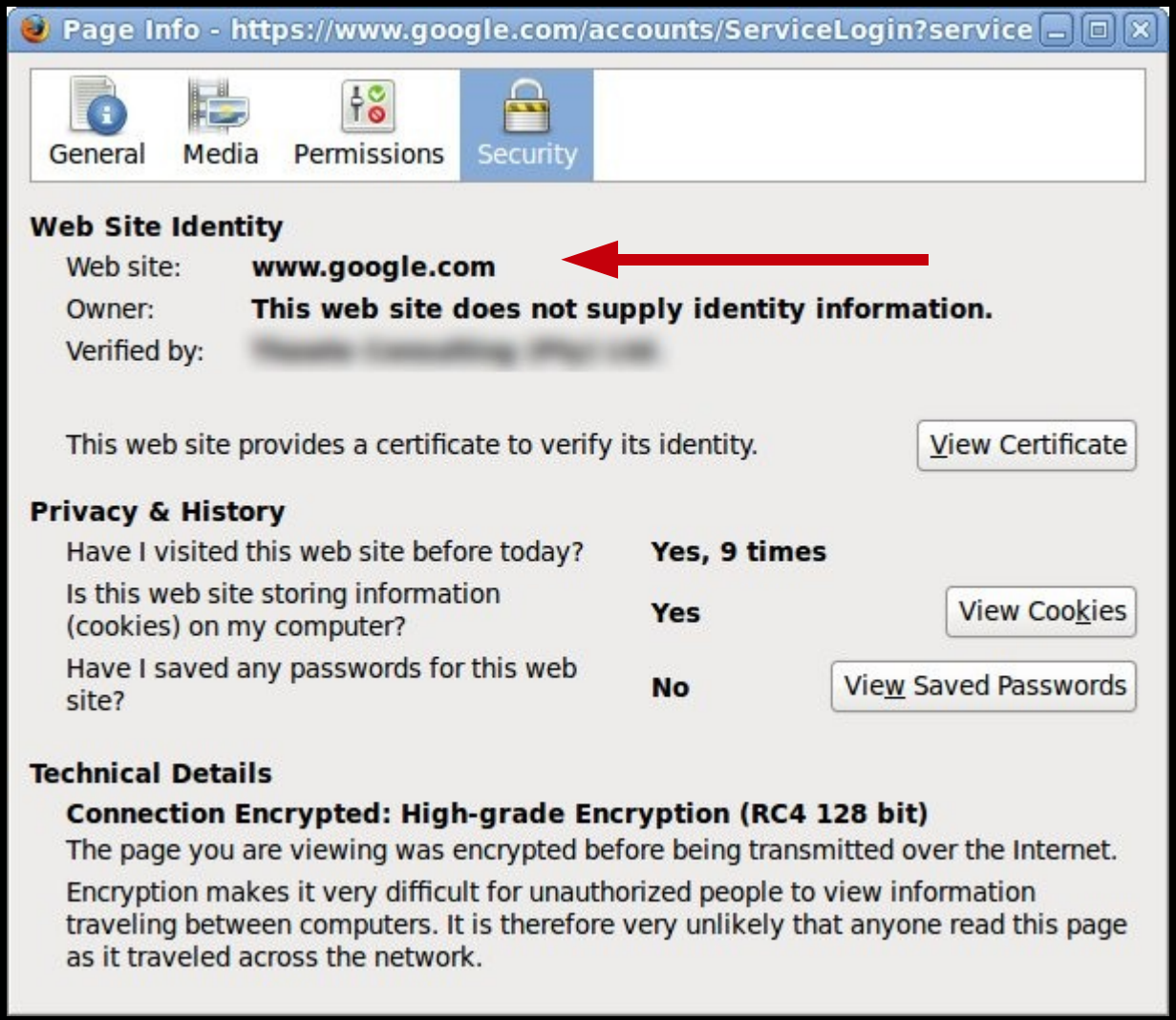
How does it look?



How does it look?



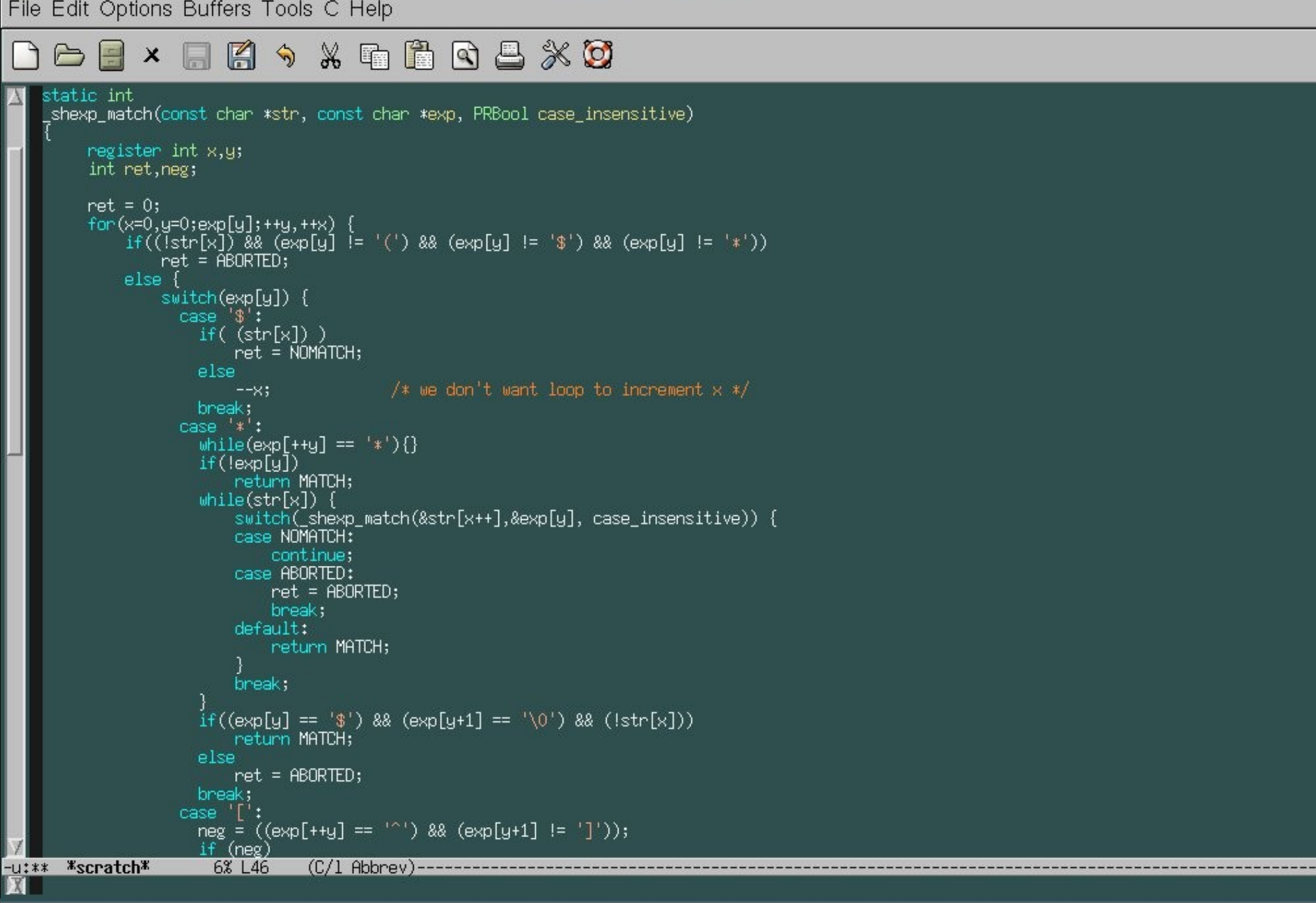
How does it look?



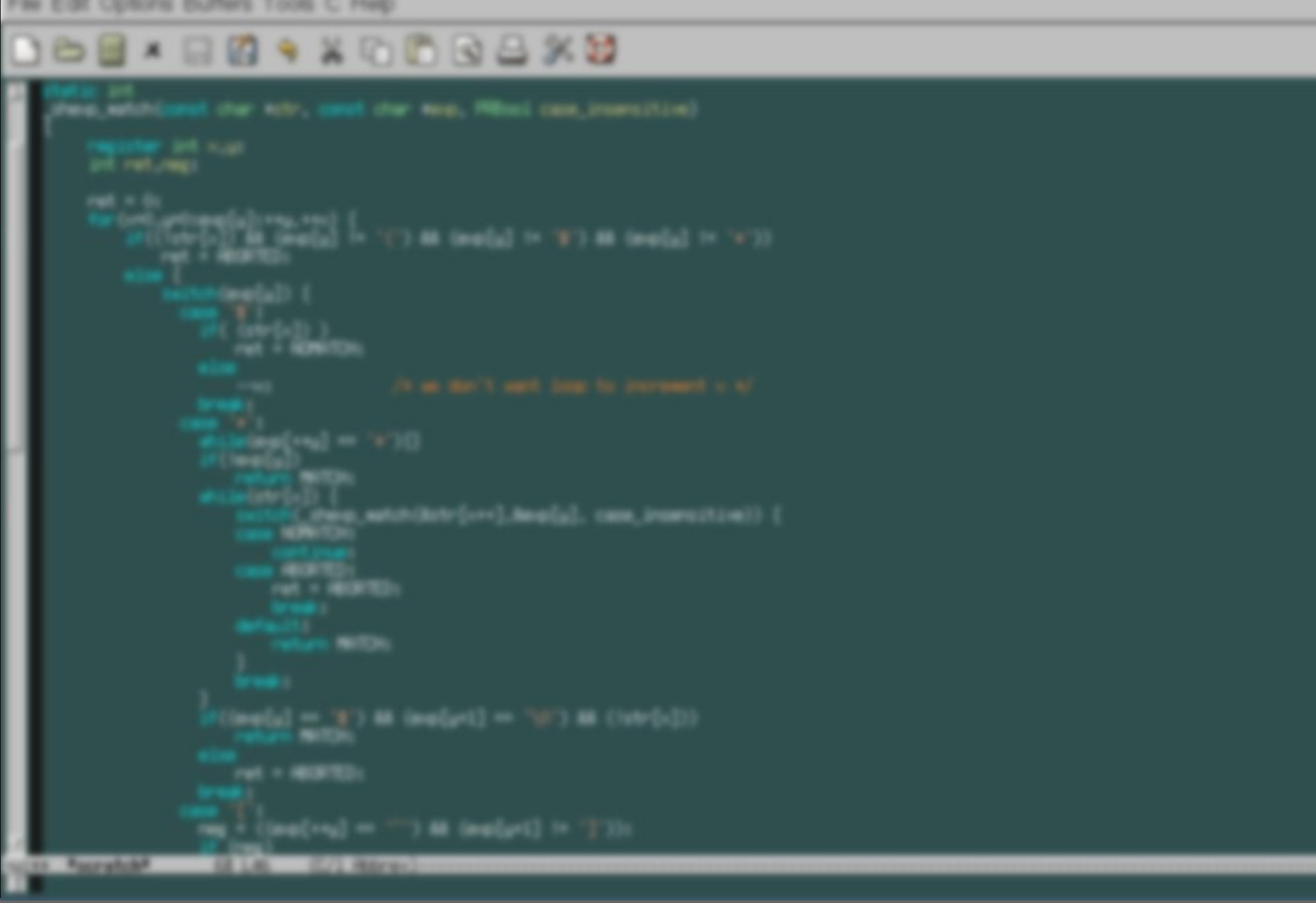
Disadvantages

1) Targeted attacks are kind of lame.

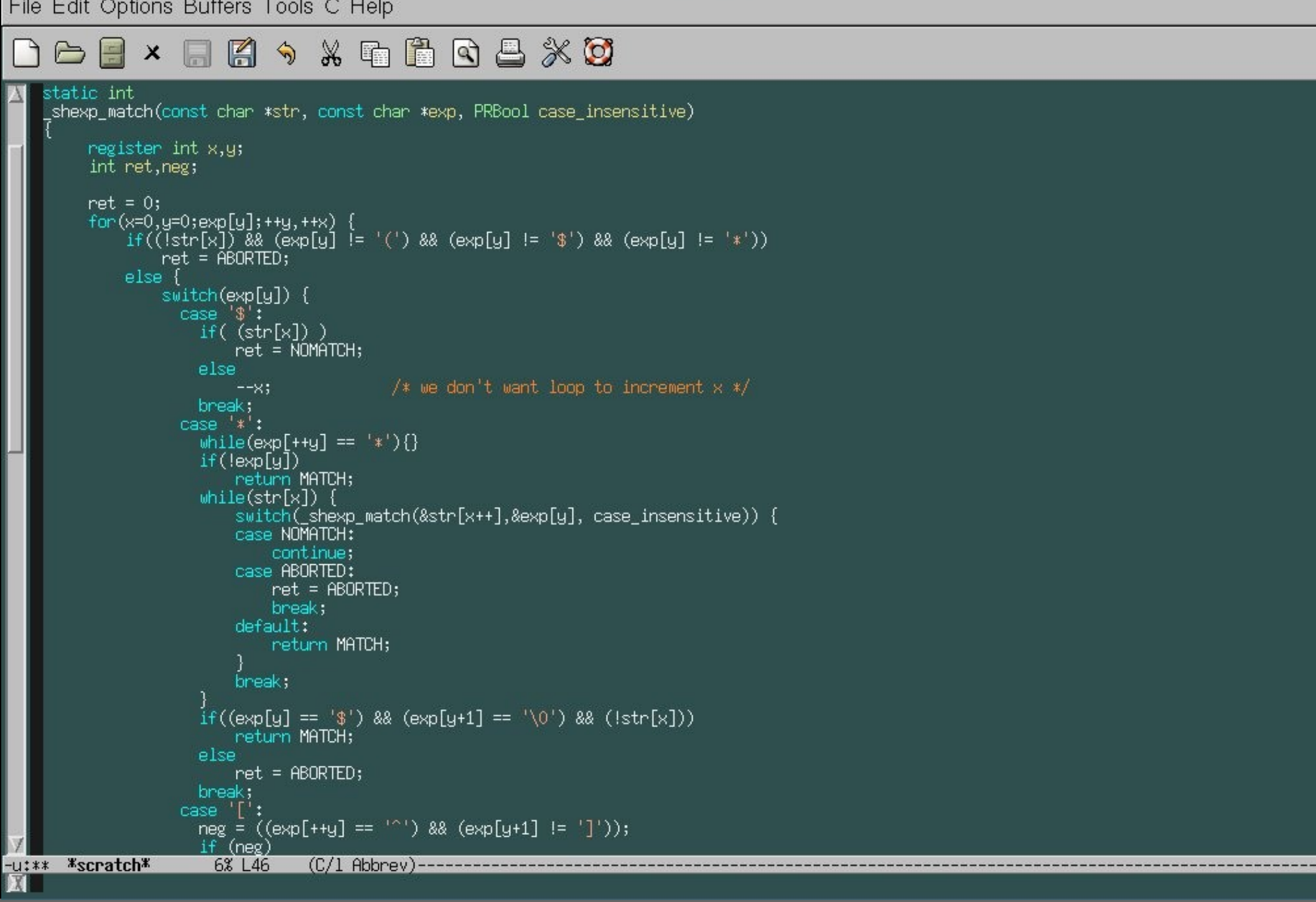
Maybe there's another trick in here somewhere...



Moxie Marlinspike
Institute For Disruptive Studies



Moxie Marlinspike
Institute For Disruptive Studies



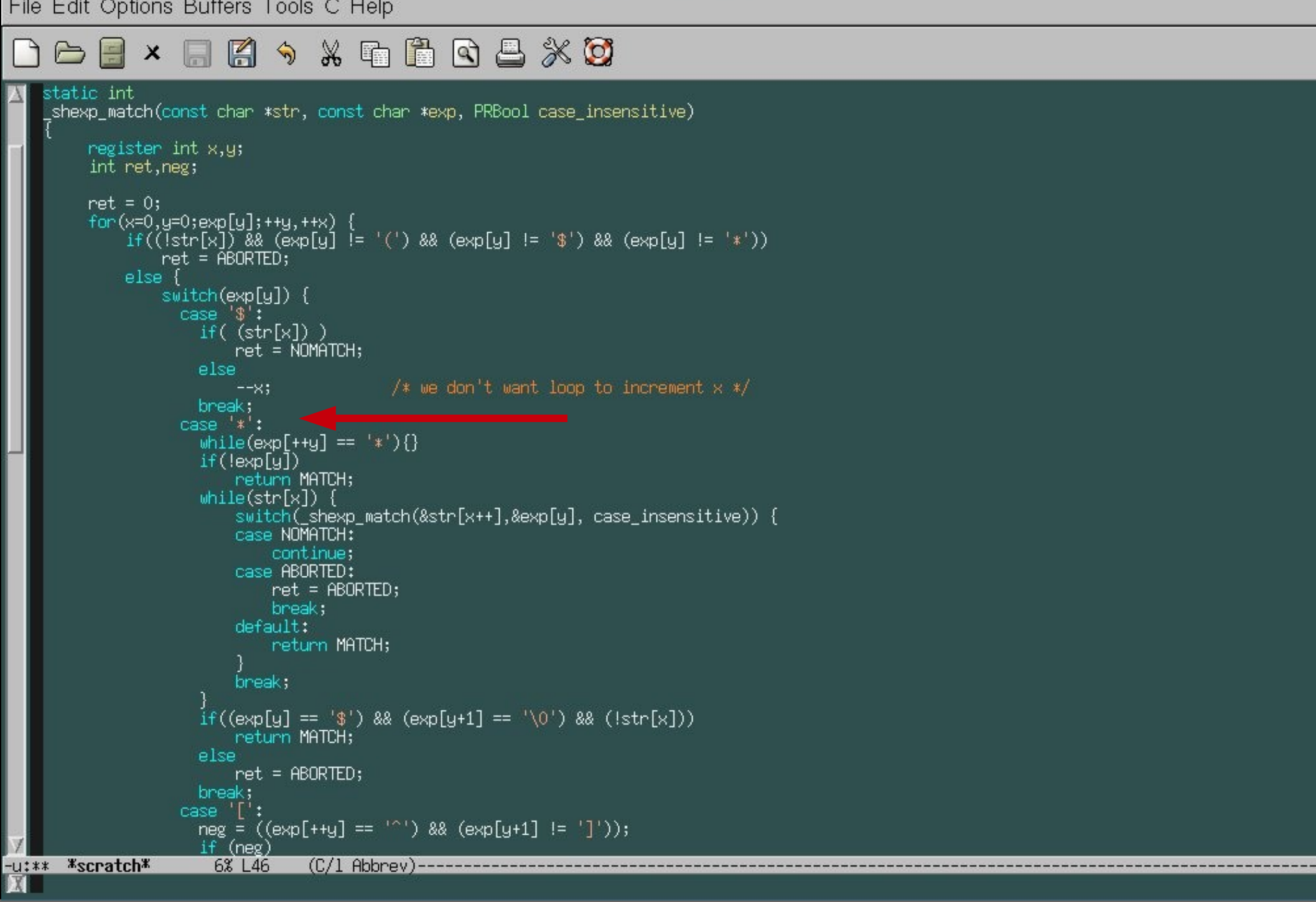
Moxie Marlinspike
Institute For Disruptive Studies

File Edit Options Buffers Tools C Help

```
static int
_shexp_match(const char *str, const char *exp, PRBool case_insensitive)
{
    register int x,y;
    int ret,neg;

    ret = 0;
    for(x=0,y=0;exp[y];++y,++x) {
        if((!str[x]) && (exp[y] != '(') && (exp[y] != '$') && (exp[y] != '*'))
            ret = ABORTED;
        else {
            switch(exp[y]) {
              case '$':
                if( (str[x]) )
                    ret = NOMATCH;
                else
                    --x;            /* we don't want loop to increment x */
                break;
              case '*':
                while(exp[++y] == '*'){}
                if(!exp[y])
                    return MATCH;
                while(str[x]) {
                    switch(_shexp_match(&str[x++],&exp[y], case_insensitive)) {
                    case NOMATCH:
                        continue;
                    case ABORTED:
                        ret = ABORTED;
                        break;
                    default:
                        return MATCH;
                    }
                    break;
                }
                if((exp[y] == '$') && (exp[y+1] == '\0') && (!str[x]))
                    return MATCH;
                else
                    ret = ABORTED;
                break;
              case '[':
                neg = ((exp[++y] == '^') && (exp[y+1] != ']'));
                if (neg)
```

-u:** *scratch* 6% L46 (C/l Abbrev)----------

Moxie Marlinspike
Institute For Disruptive Studies



Moxie Marlinspike
Institute For Disruptive Studies

Universal Wildcard

*\0.thoughtcrime.org

Universal Wildcard

*~.thoughtcrime.org

Other Weird Stuff

- (www.paypal.com|mail.google.com|
  www.etrade.com|www.bankofamerica.com|
  www.wachovia.com|www.pnc.com|
  www.wellsfargo.com)\0.thoughtcrime.org

And... your remote exploit.

```
144    char *e2 = (char *) PORT_Alloc(sizeof(char)*strlen(exp));
145    register int t,p2,p1 = 1;
146    int cp;
147
148    while(1) {
149        for(cp=1;exp[cp] != ')';cp++)
150            if(exp[cp] == '\\')
151                ++cp;
152        for(p2 = 0;(exp[p1] != '|') && (p1 != cp);p1++,p2++) {
153            if(exp[p1] == '\\')
154                e2[p2++] = exp[p1++];
155            e2[p2] = exp[p1];
156        }
157        for (t=cp+1; ((e2[p2] = exp[t]) != 0); ++t,++p2) {}
158        if(_shexp_match(str,e2, case_insensitive) == MATCH) {
159            PORT_Free(e2);
160            return MATCH;
161        }
162       ...
```

And... your remote exploit.

```
144  char *e2 = (char *) PORT_Alloc(sizeof(char)*strlen(exp));
145  register int t,p2,p1 = 1;
146  int cp;
147
148  while(1) {
149      for(cp=1;exp[cp] != ')';cp++)
150          if(exp[cp] == '\\')
151              ++cp;
152      for(p2 = 0;(exp[p1] != '|') && (p1 != cp);p1++,p2++) {
153          if(exp[p1] == '\\')
154              e2[p2++] = exp[p1++];
155          e2[p2] = exp[p1];
156      }
157      for (t=cp+1; ((e2[p2] = exp[t]) != 0); ++t,++p2) {}
158      if(_shexp_match(str,e2, case_insensitive) == MATCH) {
159          PORT_Free(e2);
160          return MATCH;
161      }
162      ...
```

And... your remote exploit.

```
144  char *e2 = (char *) PORT_Alloc(sizeof(char)*strlen(exp));
145  register int t,p2,p1 = 1;
146  int cp;
147
148  while(1) {
149      for(cp=1;exp[cp] != ')';cp++)
150          if(exp[cp] == '\\')
151              ++cp;
152      for(p2 = 0;(exp[p1] != '|') && (p1 != cp);p1++,p2++) {
153          if(exp[p1] == '\\')
154              e2[p2++] = exp[p1++];
155          e2[p2] = exp[p1];
156      }
157      for (t=cp+1; ((e2[p2] = exp[t]) != 0); ++t,++p2) {}
158      if(_shexp_match(str,e2, case_insensitive) == MATCH) {
159          PORT_Free(e2);
160          return MATCH;
161      }
162     ...
```

And... your remote exploit.

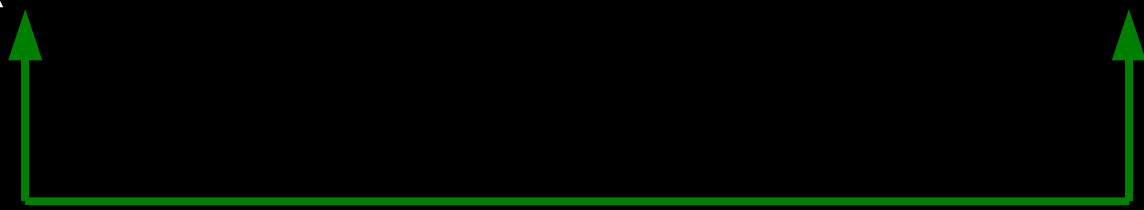
```
144    char *e2 = (char *) PORT_Alloc(sizeof(char)*strlen(exp));
145    register int t,p2,p1 = 1;
146    int cp;
147
148    while(1) {
149        for(cp=1;exp[cp] != ')';cp++)
150            if(exp[cp] == '\\')
151                ++cp;
152        for(p2 = 0;(exp[p1] != '|') && (p1 != cp);p1++,p2++) {
153            if(exp[p1] == '\\')
154                e2[p2++] = exp[p1++];
155            e2[p2] = exp[p1];
156        }
157        for (t=cp+1; ((e2[p2] = exp[t]) != 0); ++t,++p2) {}
158        if(_shexp_match(str,e2, case_insensitive) == MATCH) {
159            PORT_Free(e2);
160            return MATCH;
161        }
162       ...
```

And... your remote exploit.

(AAAAAAAAAAAAAAAAAAAAAAAAA\0OVERWRITE).foo.com

And... your remote exploit.

(AAAAAAAAAAAAAAAAAAAAAAAAA\0OVERWRITE).foo.com



And... your remote exploit.

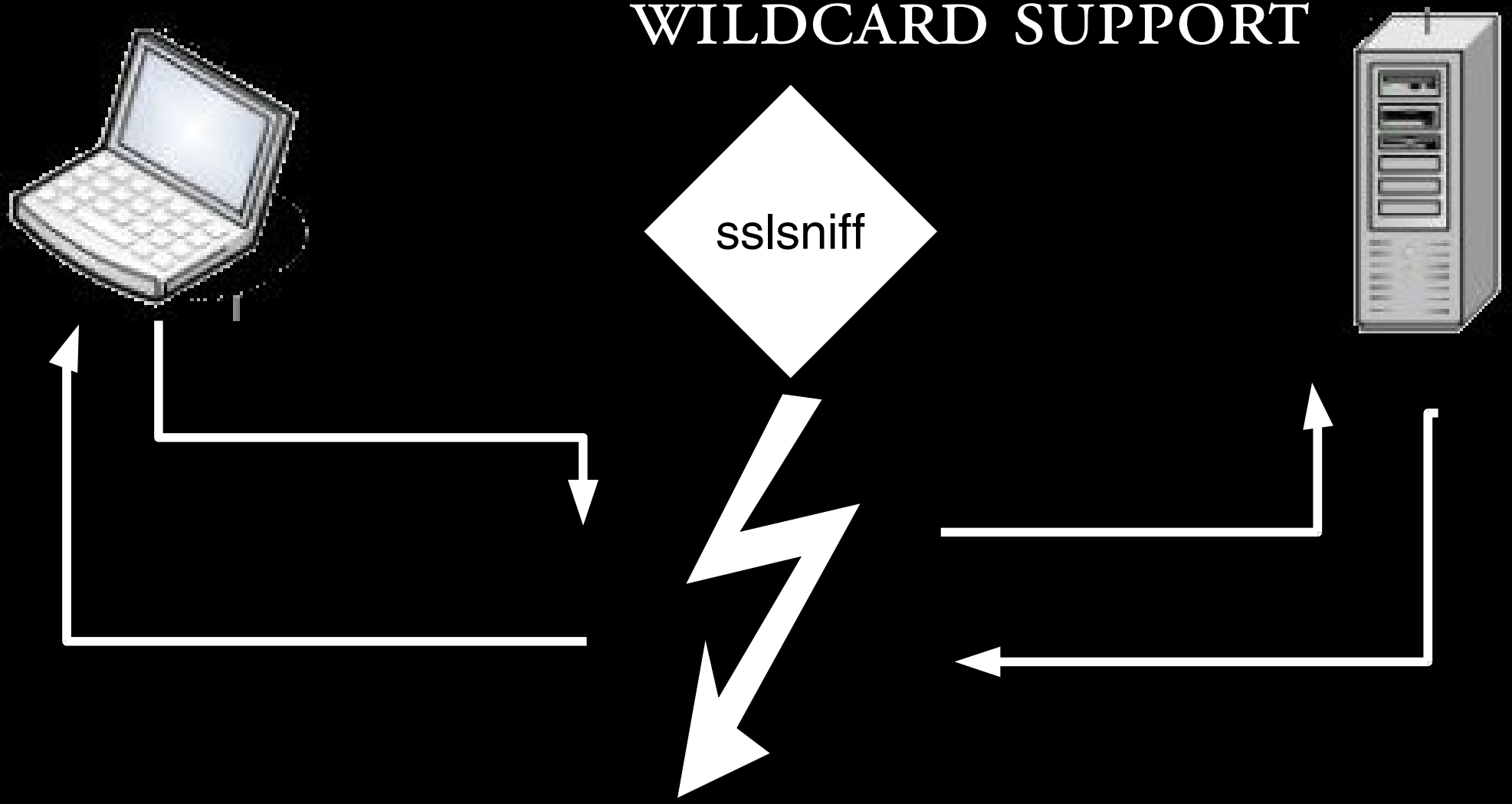
(AAAAAAAAAAAAAAAAAAAAAAAAA\0OVERWRITE).foo.com

And... your remote exploit.

(AAAAAAAAAAAAAAAAAAAAAAAAA\0OVERWRITE).foo.com

- No signed signature required!
- Possible to sneak non-ASCII characters past the NSS filters.
- This yields something exploitable in Firefox, Thunderbird, Evolution, Pidgin, and AIM.

A Second Cut: sslsniff with wildcard support



- Perform MITM if “null termination attack” cert is available.
- Or perform MITM with “universal wildcard” cert if client is NSS.

A Second Cut: updated sslsniff

- Watches network and fingerprints clients for level of vulnerability.
- Every NSS client's communication is intercepted – either with a specific “null termination” certificate, or with the “universal wildcard” certificate.
- Every non-NSS client that is vulnerable is intercepted with a “null termination” certificate if available for the destination host.
- Non-vulnerable clients are left alone to avoid detection.

What do we have to worry about?

What do we have to worry about?

1) Certificate Revocation

2) Updates

What do we have to worry about?

1) Certificate Revocation

- It would be unfortunate if some bitter Certificate Authority decided to revoke our universal wildcard certificates or any of our null-termination certificates.

2) Updates

- It would be unfortunate if some bitter SSL implementation decided to start paying attention to how ASN.1 is formatted.

What do we have to worry about?

1) Certificate Revocation

- These days, it's all about Online Certificate Status Protocol (OCSP).
- Whenever a SSL stack sees a new certificate, it makes a quick request to the OCSP URL that the signing CA embedded in it.
- The SSL stack receives a signed response from the OCSP provider indicating whether the certificate has been revoked or not.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            01:2a:39:76:0d:3f:4f:c9:0b:e7:bd:2b:cf:95:2e:7a
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd., CN=Thawte SGC CA
        Validity
            Not Before: Mar 27 22:20:07 2009 GMT
            Not After : Mar 27 22:20:07 2010 GMT
        Subject: C=US, ST=California, L=Mountain View, O=Google Inc, CN=www.google.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:d6:b9:e1:ad:b8:61:0b:1f:4e:b6:3c:09:3d:ab:
                    e8:e3:2b:b6:e8:a4:3a:78:2f:d3:51:20:22:45:95:
                    d8:00:91:33:9a:a7:a2:48:ea:30:57:26:97:66:c7:
                    5a:ef:f1:9b:0c:3f:e1:b9:7f:7b:c3:c7:cc:af:9c:
                    d0:1f:3c:81:15:10:58:fc:06:b3:bf:bc:9c:02:b9:
                    51:dc:fb:a6:b9:17:42:e6:46:e7:22:cf:6c:27:10:
                    fe:54:e6:92:6c:0c:60:76:9a:ce:f8:7f:ac:b8:5a:
                    08:4a:dc:b1:64:bd:a0:74:41:b2:ac:8f:86:9d:1a:
                    de:58:09:fd:6c:0a:25:e0:79
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication, Netscape Server Gated Crypto
            X509v3 CRL Distribution Points:
                URI:http://crl.thawte.com/ThawteSGCCA.crl

            Authority Information Access:
                OCSP - URI:http://ocsp.thawte.com
                CA Issuers - URI:http://www.thawte.com/repository/Thawte_SGC_CA.crt

            X509v3 Basic Constraints: critical
                CA:FALSE
    Signature Algorithm: sha1WithRSAEncryption
        39:b6:fb:11:bc:33:2c:c3:90:48:e3:6e:c3:9b:38:b1:42:d1:
        00:09:58:63:a0:e1:98:1c:85:f2:ef:10:1d:60:4e:51:09:62:
        f5:05:bd:9d:4f:87:6c:98:72:07:80:c3:59:48:14:e2:d6:ef:
        d0:8f:33:6a:68:31:fa:b7:bb:85:cc:f7:c7:47:7b:67:93:3c:
        c3:16:51:9b:6f:87:20:fd:67:4c:2b:ea:6a:49:db:11:d1:bd:
        d7:95:22:43:7a:06:7b:4e:f6:37:8e:a2:b9:cf:1f:a5:d2:bd:
        3b:04:97:39:b3:0f:fa:38:b5:af:55:20:88:60:93:f2:de:db:
```

Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}
```

Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}
ResponseBytes ::=       SEQUENCE {
      responseType   OBJECT IDENTIFIER,
      response       OCTET STRING
}
 BasicOCSPResponse       ::= SEQUENCE {
      tbsResponseData      ResponseData,
      signatureAlgorithm   AlgorithmIdentifier,
      signature            BIT STRING,
      certs               [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }
```

Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}
ResponseBytes ::=       SEQUENCE {
      responseType   OBJECT IDENTIFIER,
      response       OCTET STRING
}
 BasicOCSPResponse       ::= SEQUENCE {
      tbsResponseData      ResponseData,
      signatureAlgorithm   AlgorithmIdentifier,
      signature            BIT STRING,
      certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }
```

Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}
ResponseBytes ::=       SEQUENCE {
      responseType   OBJECT IDENTIFIER,
      response       OCTET STRING
}
 BasicOCSPResponse       ::= SEQUENCE {
      tbsResponseData      ResponseData,
      signatureAlgorithm   AlgorithmIdentifier,
      signature            BIT STRING,
      certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }
```

Moxie Marlinspike
Institute For Disruptive Studies

Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}
ResponseBytes ::=       SEQUENCE {
      responseType   OBJECT IDENTIFIER,
      response       OCTET STRING
}
 BasicOCSPResponse       ::= SEQUENCE {
      tbsResponseData      ResponseData,
      signatureAlgorithm   AlgorithmIdentifier,
      signature            BIT STRING,
      certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }
```

Moxie Marlinspike
Institute For Disruptive Studies

Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}
ResponseBytes ::=       SEQUENCE {
      responseType   OBJECT IDENTIFIER,
      response       OCTET STRING
}
 BasicOCSPResponse       ::= SEQUENCE {
      tbsResponseData      ResponseData,
      signatureAlgorithm   AlgorithmIdentifier,
      signature            BIT STRING,
      certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }
```

Moxie Marlinspike
Institute For Disruptive Studies

Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}
 OCSPResponseStatus ::= ENUMERATED {
      successful             (0),  --Response has valid confirmations
      malformedRequest       (1),  --Illegal confirmation request
      internalError          (2),  --Internal error in issuer
      tryLater               (3),  --Try again later
                                   --(4) is not used
      sigRequired            (5),  --Must sign the request
      unauthorized           (6)   --Request unauthorized
}
```

Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}
 OCSPResponseStatus ::= ENUMERATED {
      successful             (0),  --Response has valid confirmations
      malformedRequest       (1),  --Illegal confirmation request
      internalError          (2),  --Internal error in issuer
      tryLater               (3),  --Try again later
                                   --(4) is not used
      sigRequired            (5),  --Must sign the request
      unauthorized           (6)   --Request unauthorized
}
```

Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus = 3,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}
```

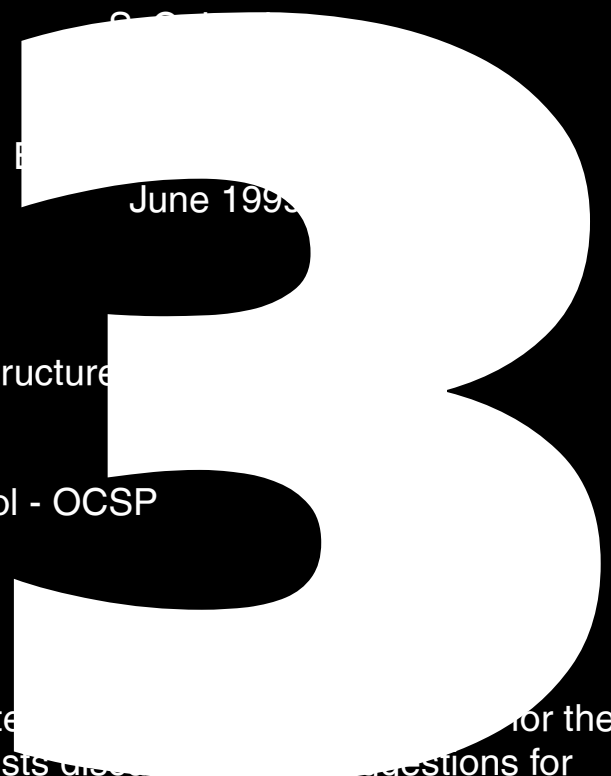
Defeating OCSP

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus = 3,

}
```

PROPOSED STANDARD

Network Working Group
Request for Comments: 2560
Category: Standards Track

M. Myers
VeriSign
R. Ankney
CertCo
A. Malpani
ValiCert



June 199

X.509 Internet Public Key Infrastructure

Online Certificate Status Protocol - OCSP

Status of this Memo

This document specifies an Inte for the Internet community, and requests d ggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

A Third Cut: ocsp-aware sslsniff

- Watch network and fingerprints clients for level of vulnerability.
- Every NSS client's communication is intercepted – either with a specific “null termination” certificate, or with the “universal wildcard” certificate.
- Every non-NSS client that is vulnerable is intercepted with a “null termination” certificate if available for the destination host.
- Non-vulnerable clients are left alone to avoid detection.
- Optionally watch for OCSP requests corresponding to certificates we're using, and “tryLater” them to defeat OCSP.

What do we have to worry about?

2) Updates

- It used to be that people, you know, downloaded and installed updates.
- As software gets more complicated, it is inevitably shipped with more bugs, and attackers are situated to exploit them on a larger scale.
- So some have felt the need to deploy self-updating software in order to fix problems rapidly.

What do we have to worry about?

2) Updates

- This is bad news for us, because by standing here and talking to you about this stuff, it probably means that SSL implementations are going to fix these problems.
- But their update mechanisms in themselves seem like kind of a dangerous idea, right?
- Maybe there's something we can do about our problem.

Firefox/Thunderbird: A Case Study

- When you install Firefox, it comes with a feature called “automatic update service,” which happens to be enabled by default.
- Here be dragons.

Firefox/Thunderbird: A Case Study

Update Server In The Sky

TLS Connection to: aus2.mozilla.org

Hello, do you have any updates for me? Here's my product, version, build ID, OS, locale, and channel.

TLS Connection to: aus2.mozilla.org

As a matter of fact, I do. Here's an unsigned blob of data – you'd do well to install it.

Firefox/Thunderbird: A Case Study

- Firefox and Thunderbird depend on their TLS connection to the update server to defend them against all possible attacks.
- Code is returned from the update server either as a binary diff against the distribution binary the client is running, or as a complete image of the binary.
- By default, “minor” updates are downloaded and installed silently – only prompting the user to restart their browser once everything is done.
  - The update server is the one who reports the version number of the update, so it is effectively up to the server whether the image it provides is installed silently or not.

Firefox/Thunderbird: A Case Study

- As vendors start to release patches for this vulnerability, the update mechanisms themselves will be vulnerable.
- All we need is a universal wildcard cert, or alternately a null-termination prefix cert for aus2.mozilla.org, and we can take control of the update mechanism to deliver payloads of our choice.
  - This could be anything:
    - A rootkit that logs keystrokes.
    - Something that sends all traffic/email through a server of our choosing.
    - A completely legitimate image that just happens to include our own CA certs.
    - Or, just to be confusing, a totally different web browser (“Thank you for updating to Galeon 0.0.3!”) or even a completely different type of application – notepad.exe comes to mind.

Firefox/Thunderbird: A Case Study

- In order to patch your system effectively, you will not be able to trust anything that comes through automatic updates.

A Fourth Cut: update-aware sslsniff

- Watch network and fingerprints clients for level of vulnerability.
- Every NSS client's communication is intercepted – either with a specific “null prefix” certificate, or with the “universal wildcard” certificate.
- Every non-NSS client that is vulnerable is intercepted with a “null prefix” certificate if available for the destination host.
- Non-vulnerable clients are left alone to avoid detection.
- Optionally watch for OCSP requests corresponding to certificates we're using, and “tryLater” them to defeat OCSP.
- Optionally watch for Firefox/Thunderbird update polls, and respond with a “custom” build.

Postscript: Stripping NULL is no solution

- Some SSL/TLS implementations (Safari, Opera) appear to strip '\0' from commonName strings before comparing.
- Thus:

  www.paypal.com\0.thoughtcrime.org

- Becomes:

  www.paypal.com.thoughtcrime.org

Postscript: Stripping NULL is no solution

- These implementations are vulnerable to a variation of our attack.
- The key is that some Certificate Authorities are vulnerable to this attack internally.
  - When presented with www.paypal.com\0.thoughtcrime.org, some CAs internally validate it as www.paypal.com
  - But the whole string (www.paypal.com\0.thoughtcrime.org) is what ends up in the subject of the cert they later issue.

Postscript: Stripping NULL is no solution

- So if we register a domain like sitekey.ba
- We can get a certificate for sitekey.ba\0nkofamerica.com
- The CAs that are internally vulnerable to this attack will validate that certificate against sitekey.ba, which we own.
- When the cert is later presented to a SSL implementation that strips \0, the certificate's common name becomes: sitekey.bankofamerica.com

Conclusion

- We have a MITM attack that will intercept communication for almost all SSL/TLS implementations.
- In the case of NSS (Firefox, Thunderbird, Evolution, AIM, Pidgin) we only need a single certificate.
- We've defeated the OCSP protocol as implemented.
- We've hijacked the Mozilla auto-updates for both applications and extensions.
- We've got an exploitable overflow.
- In short, we've got your passwords, your communication, and control over your computer.