Biting into the forbidden fruit

Lessons from trusting Javascript crypto

Krzysztof Kotowicz, Hack in Paris, June 2014

About me

- Web security researcher
  - HTML5
  - UI redressing
    - browser extensions
    - crypto
- I was a Penetration Tester @ Cure53
- Information Security Engineer @ Google

Disclaimer: “My opinions are mine. Not Google’s”.
Disclaimer: All the vulns are fixed or have been publicly disclosed in the past.

Introduction

JS crypto history

- **Javascript Cryptography Considered Harmful**
  http://matasano.com/articles/javascript-cryptography/

- **Final post on Javascript crypto**
  http://rdist.root.org/2010/11/29/final-post-on-javascript-crypto/

JS crypto history

- **It's not needed**
  - Implicit trust in the server
  - SSL / TLS required
- **It's dangerous**
  - Any XSS can circumvent the code
- **It's hard**
  - Poor crypto support in the language
  - Mediocre library quality
- **JS crypto is doomed to fail!**

Doomed to fail?

Multiple crypto primitives libraries, symmetric & asymmetric encryption, TLS implementation, a few OpenPGP implementations, and a lot of user applications built upon them. Plus custom crypto protocols.

https://crypto.cat/

Mailvelope

https://www.mailvelope.com/

http://openpgpjs.org/

Action plan

- Look at the code
- Find the vulnerabilities
- Understand the root cause
- Compare to native crypto

JS crypto vulns in the wild

- Language issues
  - Caused by a flaw of the language
- Web platform issues
  - “The web is broken”

Language issues

Language issues matter


```
if (you_think_they_dont)
    goto fail;
    goto fail;
```

Javascript in a glance

- a dynamic language
- a weakly typed language
- with prototypical inheritance
- with a global object
- and a forgiving parser

It’s a flexible language

- Code in 6 characters only!

```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!!
[]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]
+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+
[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+
(![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+([![]]+[][[]])[+!
+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+
[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]
+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]((![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]
+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]+(![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]
+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]]+[+!+[]]+
(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+
[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]])()
```

- alert(1), obviously

Weak typing

- A lot of gotchas & silent type conversions

```
// From wtfjs.com

true == 'true'
false != 'false'

Math.min() > Math.max()

typeof null == 'object'
!(null instanceof Object)
```

- Devs don’t use types. This matters to crypto!

Weak typing

- Cryptocat adventures with entropy
  http://tobtu.com/decryptocat.php

```
// Generate private key (64 random bytes)
var rand = Cryptocat.randomString(64, 0, 0, 1, 0);
```

```
// Generates a random string of length `size` characters.
// If `alpha = 1`, random string will contain alpha characters,
// and so on.
// If 'hex = 1', all other settings are overridden.
Cryptocat.randomString = function(
    size, alpha, uppercase, numeric, hex)
```

- `"70654517326151964 58..."` != 64 random bytes.
- Entropy loss - 512 bits => 212 bits

Magic properties

- Cryptocat - a multiparty chat application

- Check if we don't yet have the user's key (=new user). Generate shared secrets (hmac key + encryption key)

```
if (!publicKeys[sender]) {
    publicKeys[sender] = receivedPublicKey;
    multiParty.genSharedSecret(sender);
}
```

- Decrypt incoming message (if you have a secret already)

```
if (sharedSecrets[sender]) {
    if (message[myName]['hmac'] === HMAC(ciphertext,
            sharedSecrets[sender]['hmac'])) {
        message = decryptAES(ct, sharedSecrets[sender]['msg']);
        return message;
     }
```

Magic properties

- Meet __proto__. Always there

```
publicKeys = {one: "1", two: "2"}
publicKeys['__proto__'] // {}
Boolean(publicKeys['__proto__']) // true
```

- publicKeys['__proto__'] == true, so shared secret is never generated

- But sharedSecrets['__proto__'] == true, so decryption throws exception

- [CVE 2013-4100] Joining chat as __proto__ breaks chat for everyone. http://www.2ality.com/2012/01/objects-as-maps.html

Magic properties

- Python has them too!
- Kill an application by submitting a hash algorithm __delattr__
- http://blog.kotowicz.net/2013/12/breaking-google-appengine-webapp2.html

Silent errors

```
a = [1];
a[0] // 1
a[1000] // undefined. No error!
```

- Out-of-bounds array access does not throw error

- At least it returns harmless *undefined*
  (I‘m looking at you, C)

Unicode 

- JS strings are unicode, not byte arrays

- *String.charCodeAt(index)* returns the numeric **Unicode** value of the character

- Not a byte value!

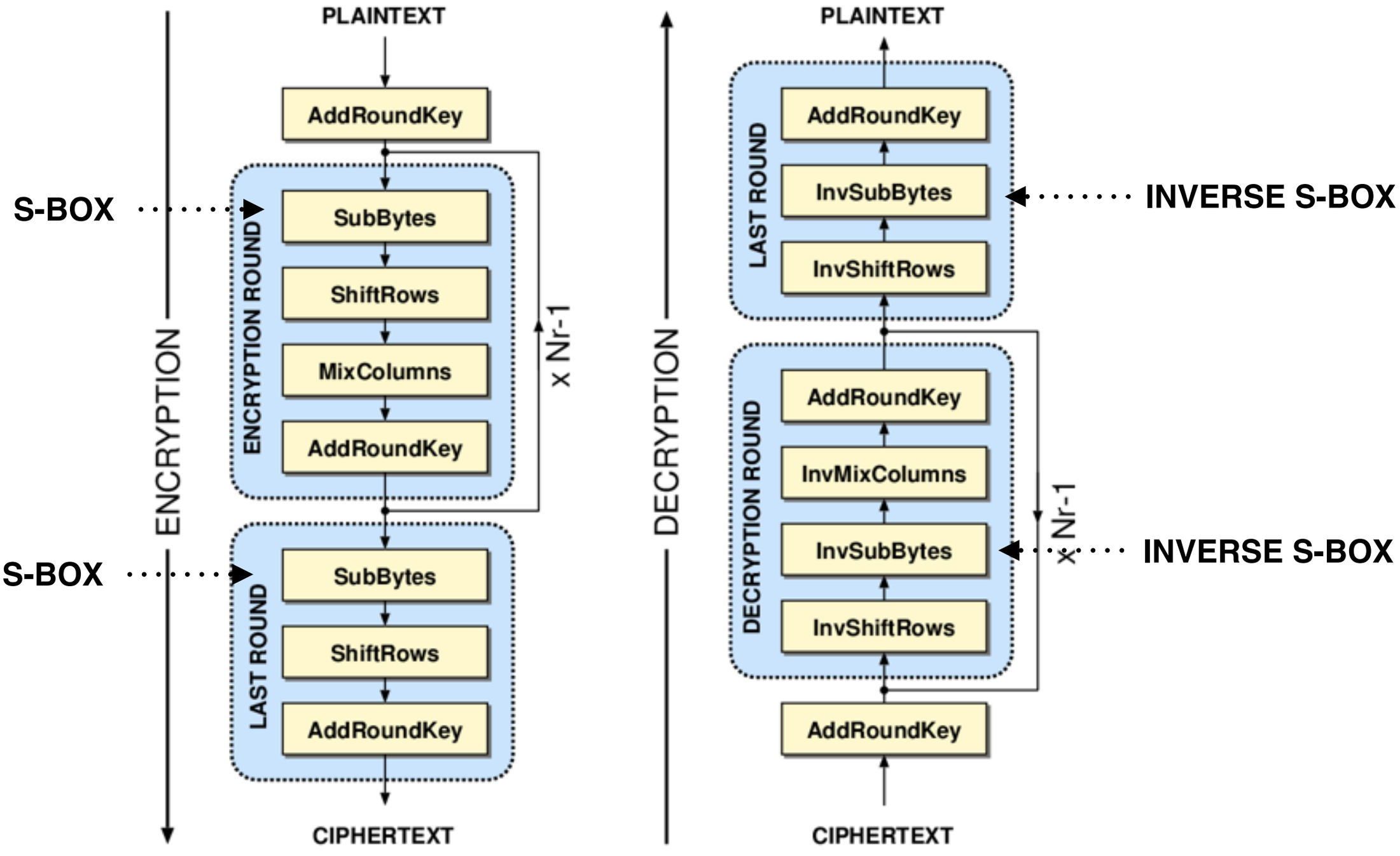
- https://speakerdeck.com/mathiasbynens/hacking-with-unicode

16 snowmen attack!

☃☃☃☃☃☃☃☃☃☃☃☃☃☃☃☃

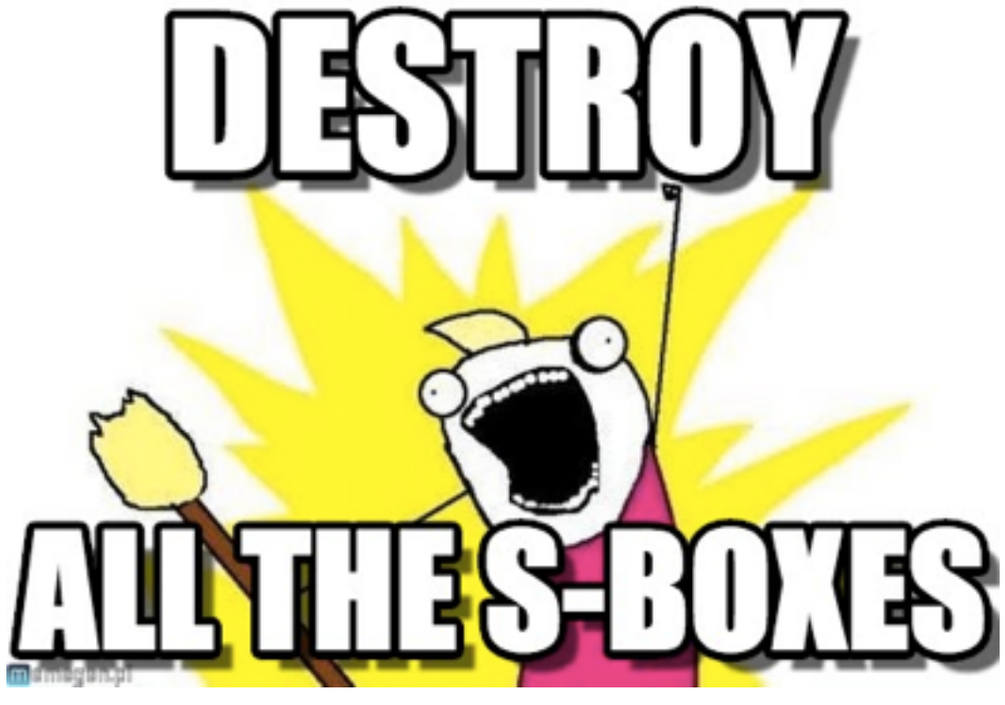
- Reveals AES key by encrypting Unicode and decrypting the result
http://vnhacker.blogspot.com/2014/06/why-javascript-crypto-is-useful.html

AES


PLAINTEXT
AddRoundKey
SubBytes
ShiftRows
MixColumns
AddRoundKey
ENCRYPTION ROUND
x Nr-1
S-BOX
SubBytes
ShiftRows
AddRoundKey
LAST ROUND
CIPHERTEXT
ENCRYPTION
PLAINTEXT
AddRoundKey
InvSubBytes
InvShiftRows
LAST ROUND
INVERSE S-BOX
AddRoundKey
InvMixColumns
InvSubBytes
InvShiftRows
DECRYPTION ROUND
x Nr-1
INVERSE S-BOX
AddRoundKey
CIPHERTEXT
DECRYPTION

Encrypting…

```
function SubBytes(state, Sbox) // state = [9740, 9796, 9743, ...]
{
    var i;
    for( i=0; i<16; i++ )
        state[i] = Sbox[ state[i] ];
    return state; // [undefined, undefined, ...]
}
```



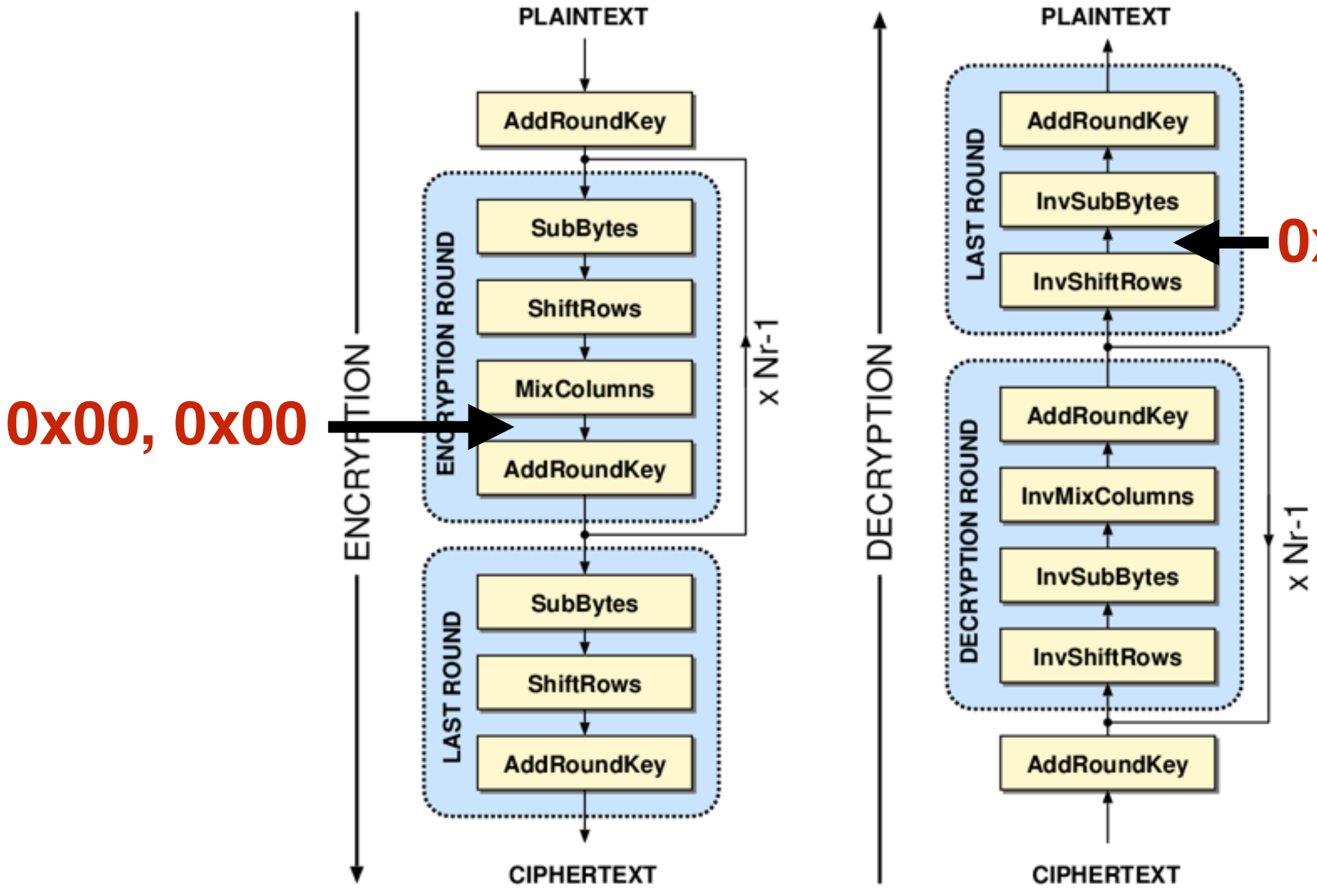
Implicit type coercion

```
function MixColumns(state) { // [undefined, undefined, ...]
    c0 = state[I(0,col)]; // c0 = undefined,...
    state[I(0,col)] = aes_mul(2,c0) ^ aes_mul(3,c1) ^ c2 ^ c3;
    return state
}

function aes_mul(a, b) { // 2, undefined
    var res = 0;
    res = res ^ b; // 0 ^ undefined = 0 :)
}
```

```
aes_mul(2,c0) ^ aes_mul(3,c1) ^ c2 ^ c3;
undefined     ^ undefined     ^ 0  ^ 0  // 0
```

AES


0x00, 0x00
ENCRYPTION
PLAINTEXT
AddRoundKey
ENCRYPTION ROUND
SubBytes
ShiftRows
MixColumns
AddRoundKey
x Nr-1
LAST ROUND
SubBytes
ShiftRows
AddRoundKey
CIPHERTEXT
DECRYPTION
PLAINTEXT
LAST ROUND
AddRoundKey
InvSubBytes
InvShiftRows
0x00, 0x00
DECRYPTION ROUND
AddRoundKey
InvMixColumns
InvSubBytes
InvShiftRows
x Nr-1
AddRoundKey
CIPHERTEXT

Decrypting…

- Decrypt the ciphertext with the same key
- In last round:

```
function SubBytes(state, Sbox) // state = [0, 0, …]
{
    var i;
    for( i=0; i<16; i++ )
        state[i] = Sbox[ state[i] ];
    return state; // [0x52, 0x52, …]
}
```

- plaintext = key ⊕ [0x52, 0x52, …]
- key = plaintext ⊕ [0x52, 0x52, …]

Type coercion

CVE-2014-0092 GnuTLS certificate validation bypass

http://blog.existentialize.com/the-story-of-the-gnutls-bug.html

```
/* Checks if the issuer of a certificate is a
 * Certificate Authority
 * Returns true or false, if the issuer is a CA,
 * or not.
 */
static int
check_if_ca (gnutls_x509_crt_t cert, gnutls_x509_crt_t issuer,
             unsigned int flags)
```

- C has no exceptions. Errors were reported as negative numbers. But callers treated return value as a boolean:

```
if (ret == 0) { /*cert invalid, abort */}
```

Language issues

- They are not unique to Javascript
- You can overcome them!
  - ES 5 strict mode
    https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions_and_function_scope/Strict_mode
  - Type enforcing - e.g. Closure Compiler
    https://developers.google.com/closure/compiler/
  - Development practices: tests, continuous integration, code reviews

Web platform issues

Web platform

- Javascript code runs in a JS engine…
  **Monkey, v8, Nitro, Chakra, SunSpider*

- In an execution environment…
  browser renderer process, server process

- With different APIs available…
  DOM, WebCrypto, browser extension API

- With different restriction/isolation policies…
  Same Origin Policy, CSP, iframe sandbox, extension security policies

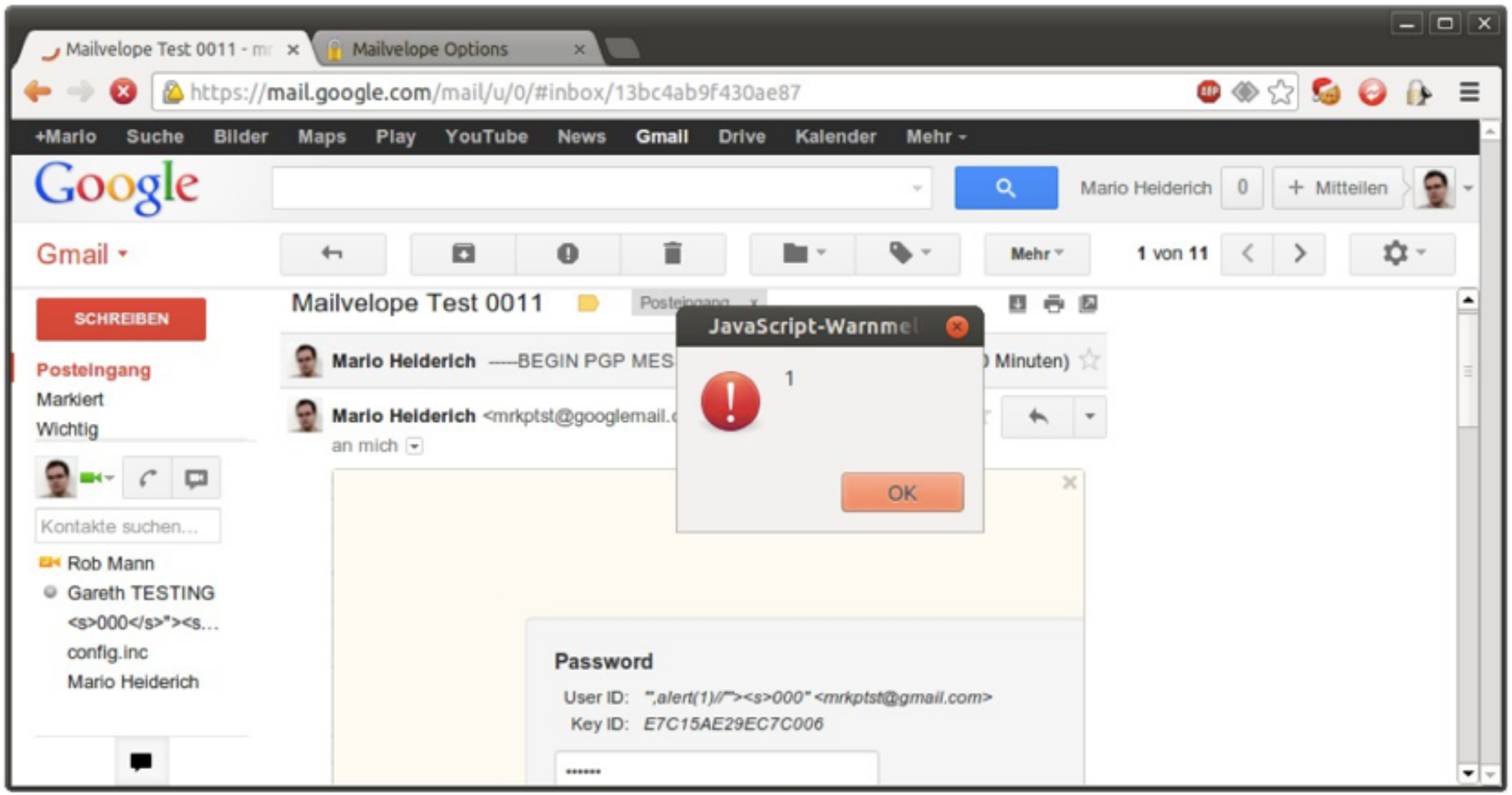
- These conditions are much more important to crypto!

XSS

- Web is full of it
- Any XSS is RCE equivalent for web
- XSS can bypass any crypto code in the same origin
    - replace a PRNG
    - exfiltrate the key or plaintext
    - replace the public key
- There are XSSes in crypto code

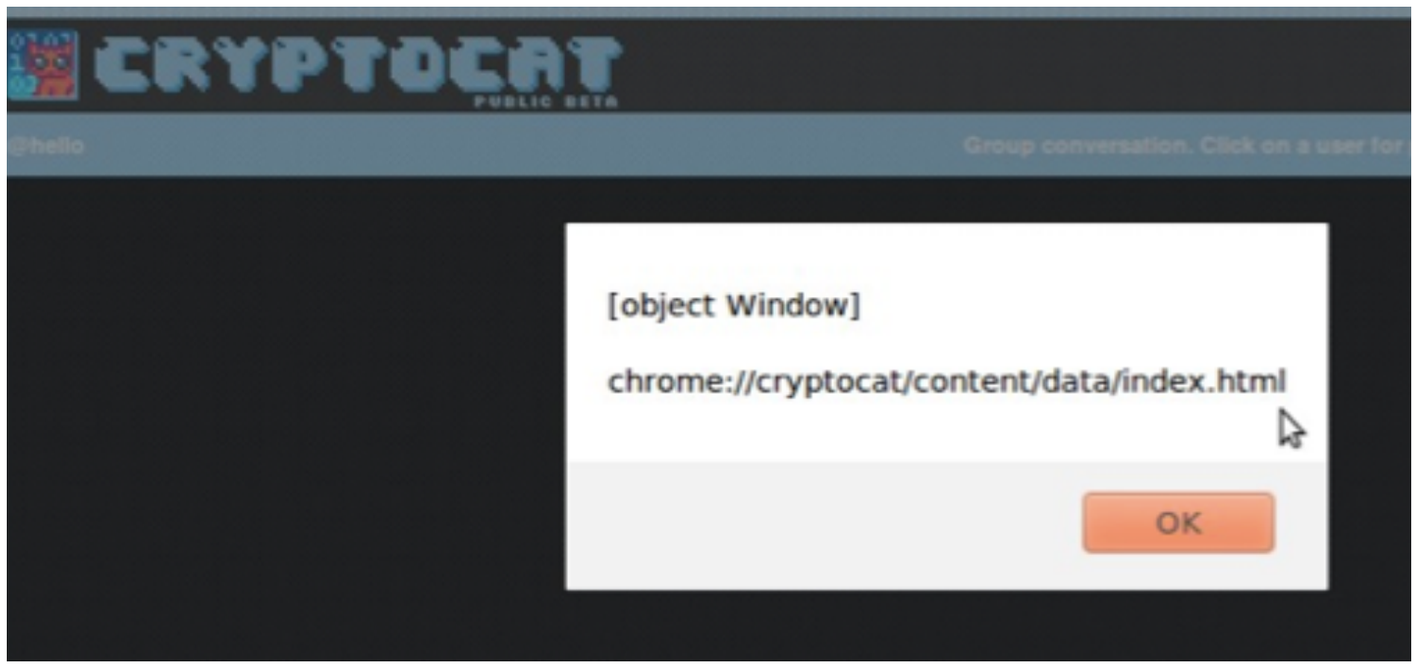
XSS

- Mailvelope - DOM XSS in Gmail by sending encrypted **<img onerror=alert(1)>** to the victim



XSS

- [CVE 2013-2259] Cryptocat used client side filtering of nickname / conversation name.



- Chrome extension: CSP, only UI Spoofing
- Firefox extension: XSS = RCE in the OS

RCE in non-JS crypto

- [CVE-2014-3466] A flaw was found in the way **GnuTLS** parsed session IDs from ServerHello messages of the TLS/SSL handshake. A malicious server could use this flaw to send an **excessively long session ID** value, which would trigger a **buffer overflow** in a connecting TLS/SSL client application using GnuTLS, causing the client application to crash or, **possibly, execute arbitrary code.**

Timing side-channels

- OpenPGP.js RSA decryption unpadding

```
/**
 * Decodes a EME-PKCS1-v1_5 padding
 */
decode: function(message, len) {
    if (message.length < len)
        message = String.fromCharCode(0) + message; // branching
    if (message.length < 12 || message.charCodeAt(0) !== 0 ||
            message.charCodeAt(1) != 2) // branching
        return -1; // early exit
    var i = 2;
    return message.substring(i + 1, message.length);
}
```

- This needs to be constant time to avoid Bleichenbacher's attack http://archiv.infsec.ethz.ch/education/fs08/secsem/Bleichenbacher98.pdf

Timing side-channels

- Similar problem in Java - JSSE (RSA used in TLS) http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/MeyerChristopher/diss.pdf

- [CVE-2012-5081] Different error messages

- [CVE-2014-0411] Timing side-channel - random numbers were generated only on invalid padding

Direct memory access

- Remember Heartbleed?

- Not a crypto vulnerability, but it allowed to bypass the encryption by just reading memory

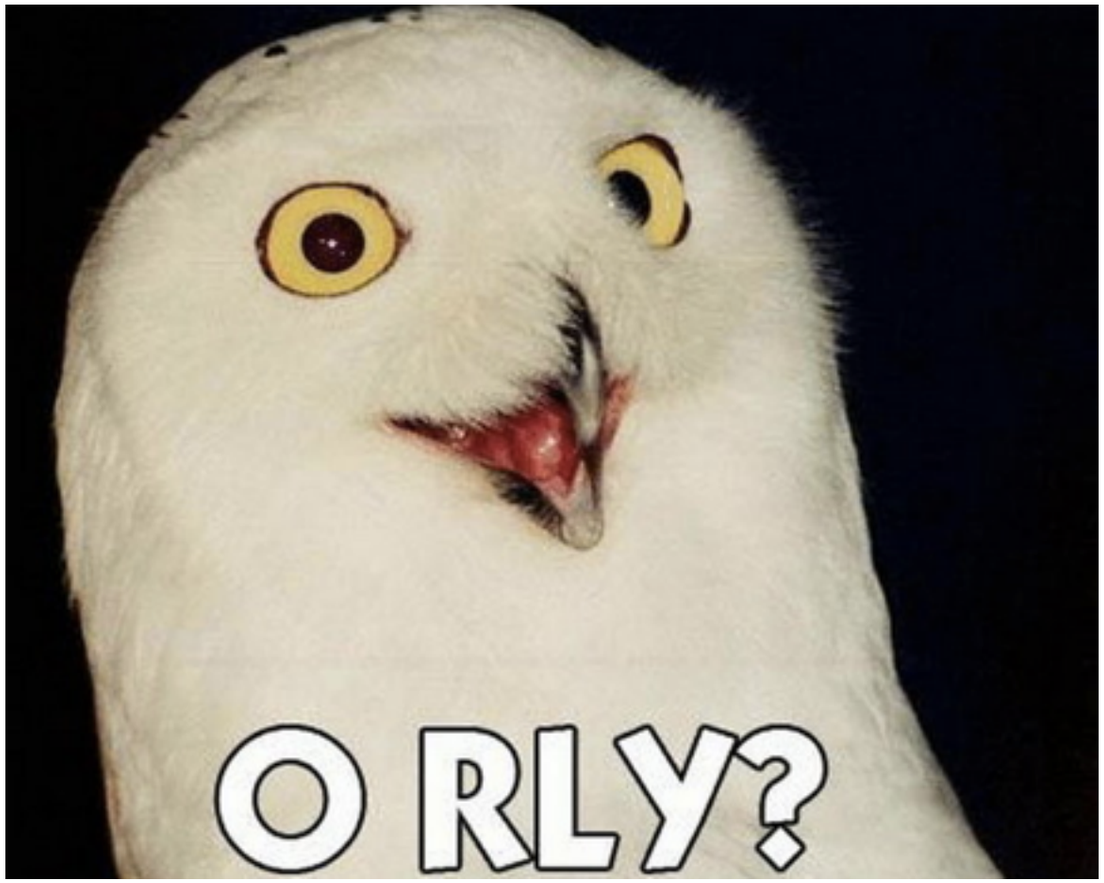
  - client sends a large payload length + a tiny payload

  - no bounds check in the server

  - server replies with leaked memory contents

Direct memory access

- Thankfully, JS is a memory-safe language. We have no buffers to overflow…



Direct memory access

- Pwn2Own 2014, Firefox 28, Jüri Aedla
  “TypedArrayObject does not handle the case where ArrayBuffer objects are neutered, setting their length to zero while still in use. This leads to **out-of-bounds reads and writes into the Javascript heap**, allowing for **arbitrary code execution**.”
  https://www.mozilla.org/security/announce/2014/mfsa2014-31.html

- Pwnium 4, Chrome 33, geohot (George Hotz)
  https://code.google.com/p/chromium/issues/detail?id=351787

```
var ab = new ArrayBuffer(SMALL_BUCKET);
ab.__defineGetter__("byteLength",function(){return 0xFFFFFFFC;});
var aaa = new Uint32Array(ab);
// all your base are belong to us
```

Direct memory access

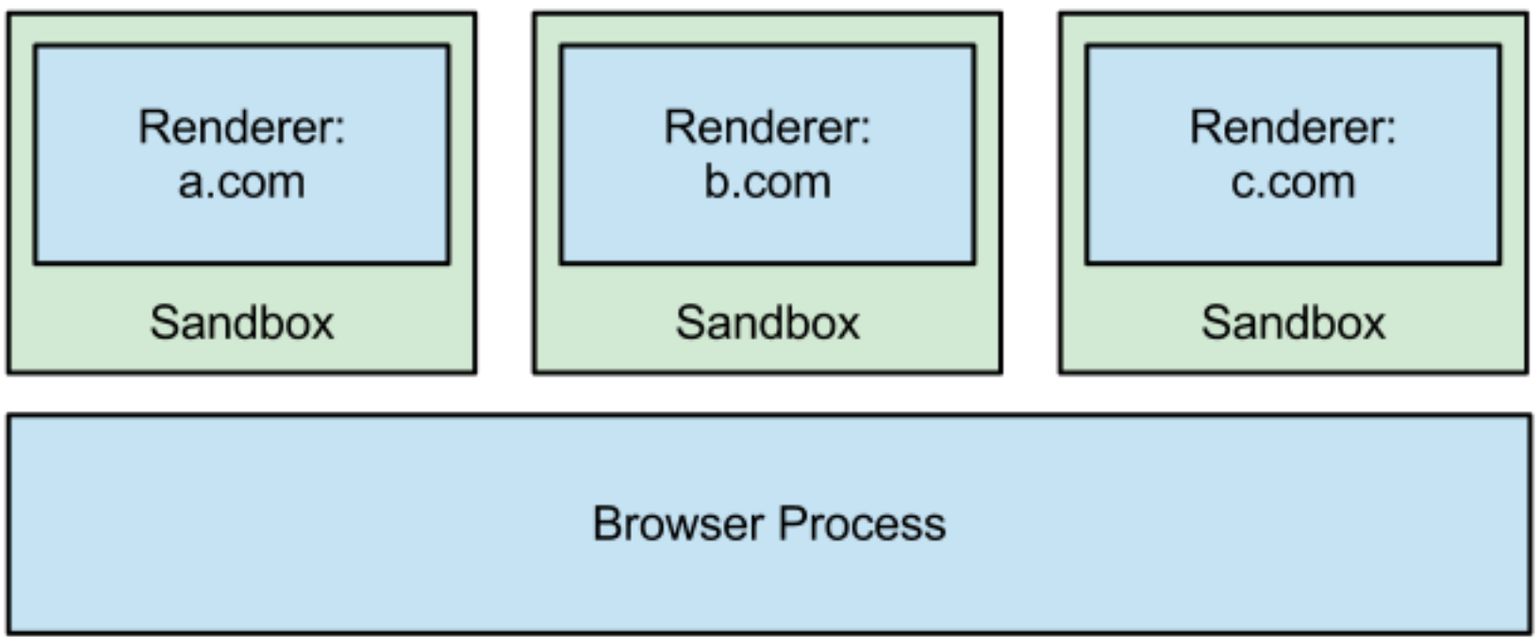
- Browsers are an attack surface as well
  - network stack
  - HTML parser
  - JS engine
- Any URL in any tab can trigger an exploit

Browser architecture

- Firefox - single process
  http://lwn.net/Articles/576564/

- IE - multiprocess, sandboxed from OS
  http://blogs.msdn.com/b/ie/archive/2012/03/14/enhanced-protected-mode.aspx

- Chrome - multiprocess, sandboxed from other tabs
  http://www.chromium.org/developers/design-documents/sandbox



Malware problem

- Any malware can circumvent native crypto software as well. Kernels have vulnerabilities too.

- GnuPG was bypassed by the authorities by simply installing a keylogger.
  https://www.gnupg.org/faq/gnupg-faq.html#successful_attacks

- For JS crypto - your browser is the OS. Browser security = host security

- There is one difference though…

Application delivery

- You don’t install websites
- Code delivery and execution is transparent (drive-by download)
- Huge code execution playground, running code separated by Same Origin Policy only
- Roughly half of the users use the browser with *any* kind of sandbox

Is JS crypto doomed?

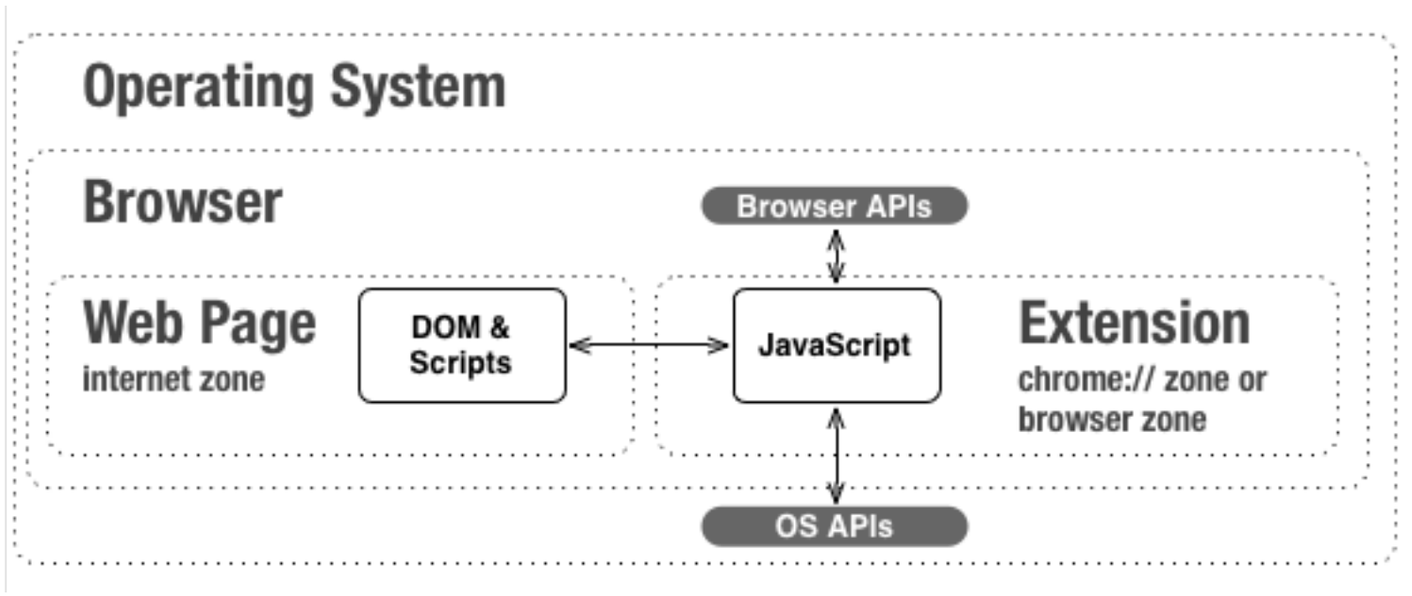
- Create perfect, XSS-free, constant time JS code
- Ensure server will never be compromised
- Put it in a website, serve over HTTPS

- You’re safe until someone uses:
  - a browser exploit
  - a Same Origin Policy bypass
- How can we fix this?

Extensions to the rescue

Browser extension

- Not a plugin (Java, Flash, PDF reader)
- A Javascript application running in privileged execution environment
- You need to install it



Browser extension

- Secure, signed code delivery
- Better separation from websites than just Same Origin Policy
- Much smaller attack surface
- Process isolation in Chrome http://www.chromium.org/developers/design-documents/site-isolation

Open problems

- Timing sidechannels are exploitable and hard to fix http://sirdarckcat.blogspot.com/2014/05/matryoshka-web-application-timing.html
- No *mlock()* equivalent - secrets can be swapped to disk
- No secure store yet (wait for WebCrypto)
- Extensions silently auto-update
- Lack of full process isolation yet

Summary

- JS crypto is way better than it used to be
- A lot of perceived “JS crypto flaws” are present in other languages as well
- The platform issues are much more difficult to mitigate
  - in-website crypto has too large attack surface
  - use extensions only

The end

Me:
http://blog.kotowicz.net, @kkotowicz, krzysztof@kotowicz.net

More vulns:
https://cure53.de/pentest-report_mailvelope.pdf
https://cure53.de/pentest-report_openpgpjs.pdf
https://blog.crypto.cat/wp-content/uploads/2012/11/Cryptocat-2-Pentest-Report.pdf

Thanks to people who helped and inspired
(*in Math.random() order*):
Mario Heiderich, Franz Antesberger, Juraj Somorovsky, Ian Beer, Ivan Fratric, Eduardo Vela Nava, Thai Duong, Frederic Braun, Ben Hawkes, Stephan Somogyi, Daniel Bleichenbacher, Adam Langley, Mathias Biennia