Biting into the forbidden fruit

Lessons from trusting Javascript crypto

Krzysztof Kotowicz, OWASP Appsec EU, 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

- Implicit trust in the server to deliver the code
- SSL/TLS is needed anyway
- Any XSS can circumvent the code
- Poor library quality
- Poor crypto support
- No secure keystore
- **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/

JS crypto is a fact

- Understand it
- Look at the code
- Find the vulnerabilities
- Analyze them
- Understand the limitations and workarounds
- Answer the question: can it be safe?

JS crypto vulns in the wild

- Language issues
  - Caused by a flaw of the language
- Web platform issues
  - Cased by the web
- Other standard bugs
  - out of scope for this presentation

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

Bit quirks

- All numbers are floats, actually
  http://www.2ality.com/2012/04/number-encoding.html

- Bit shifts are tricky

```
1 << 31 // -2147483648
1 << 32 // 1
1 << 31 << 1 // 0
1 << 31 >> 31 // -1
1 << 31 >>> 31 // 1. Sigh!
```

- "The right operand should be less than 32, but if not **only the low five bits will be used**."
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators

Weak typing

- A lot of gotchas & silent type conversions

```
// From wtfjs.com

true == 'true'      // true
false == 'false';   // false

Math.min() < Math.max(); // false

typeof null // object
null instanceof Object // false
```

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

Weak typing

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

```
-// Generate private key (32 byte random number)
-// Represented in decimal
+// Generate private key (64 random bytes)
 var rand = Cryptocat.randomString(64, 0, 0, 1, 0);
 myPrivateKey = BigInt.str2bigInt(rand, 16);
```

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

Magic properties

- Cryptocat - a multiparty chat
- You must store public key of all your chat members
- Can’t overwrite existing key
- Add public key of a new chat member:

```
//multiParty.receiveMessage
if (!publicKeys[sender]) {
    if (validate(publicKey)) {
        publicKeys[sender] = publicKey;
    }
}
```

Magic properties

```
> dict = {'foo': 'bar', 'baz': 'bang'}
  Object {foo: "bar", baz: "bang"}
> dict['foo']
  "bar"
> dict['bazinga']
  undefined
> dict['__proto__']
  Object {}
> dict['__proto__'] == true
  false
> dict['__proto__'] = 'bzium'
  "bzium"
> dict
  Object {foo: "bar", baz: "bang"}
> dict['__proto__']
  Object {}
```

- [CVE 2013-4100] User __proto__ breaks chat for all participants
- 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[1] // undefined. No error!
```

- Does not throw errors

- At least it’s only 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

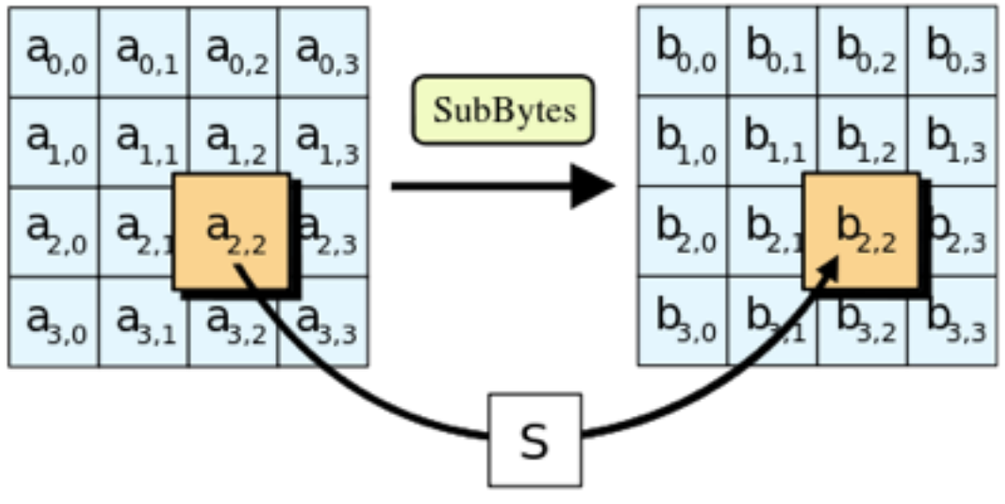
- Not a byte value!

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

16 snowmen attack!

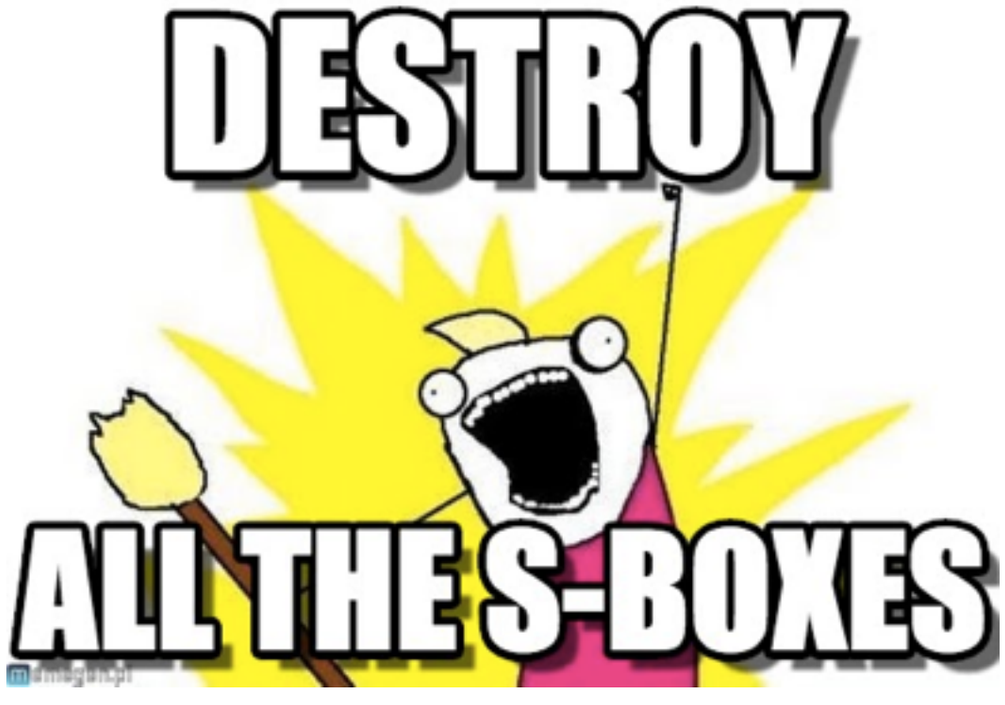
☃☃☃☃☃☃☃☃☃☃☃☃☃☃☃☃

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



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
```

After first round:

state = [0, 0, …] ⊕ Round key = Round key

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, …]

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 like 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 issues

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

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

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

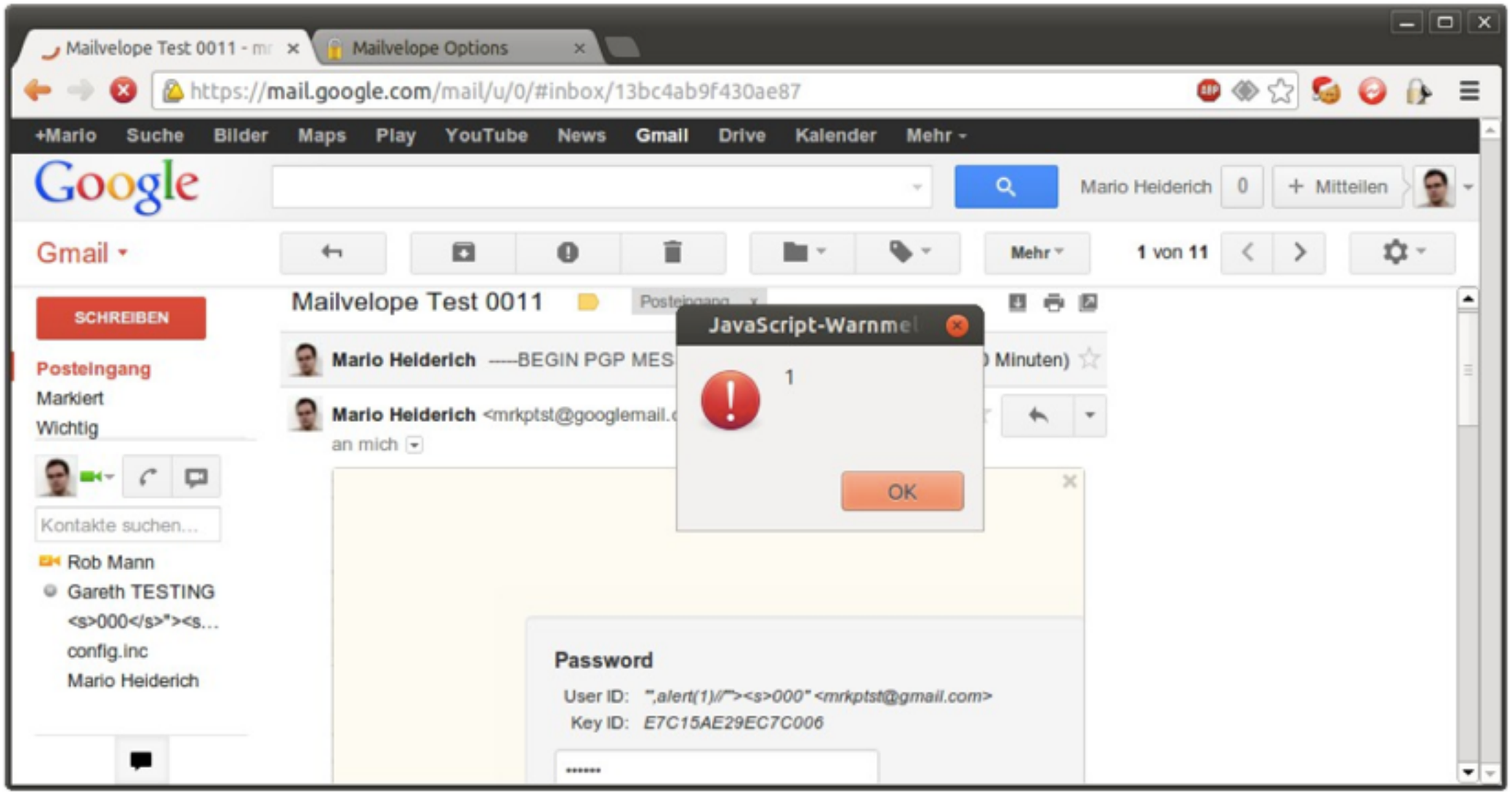
- With different restriction/isolation policies…
  Same Origin Policy, CSP, iframe sandbox, extension security policies

- These issues are much more important to crypto!

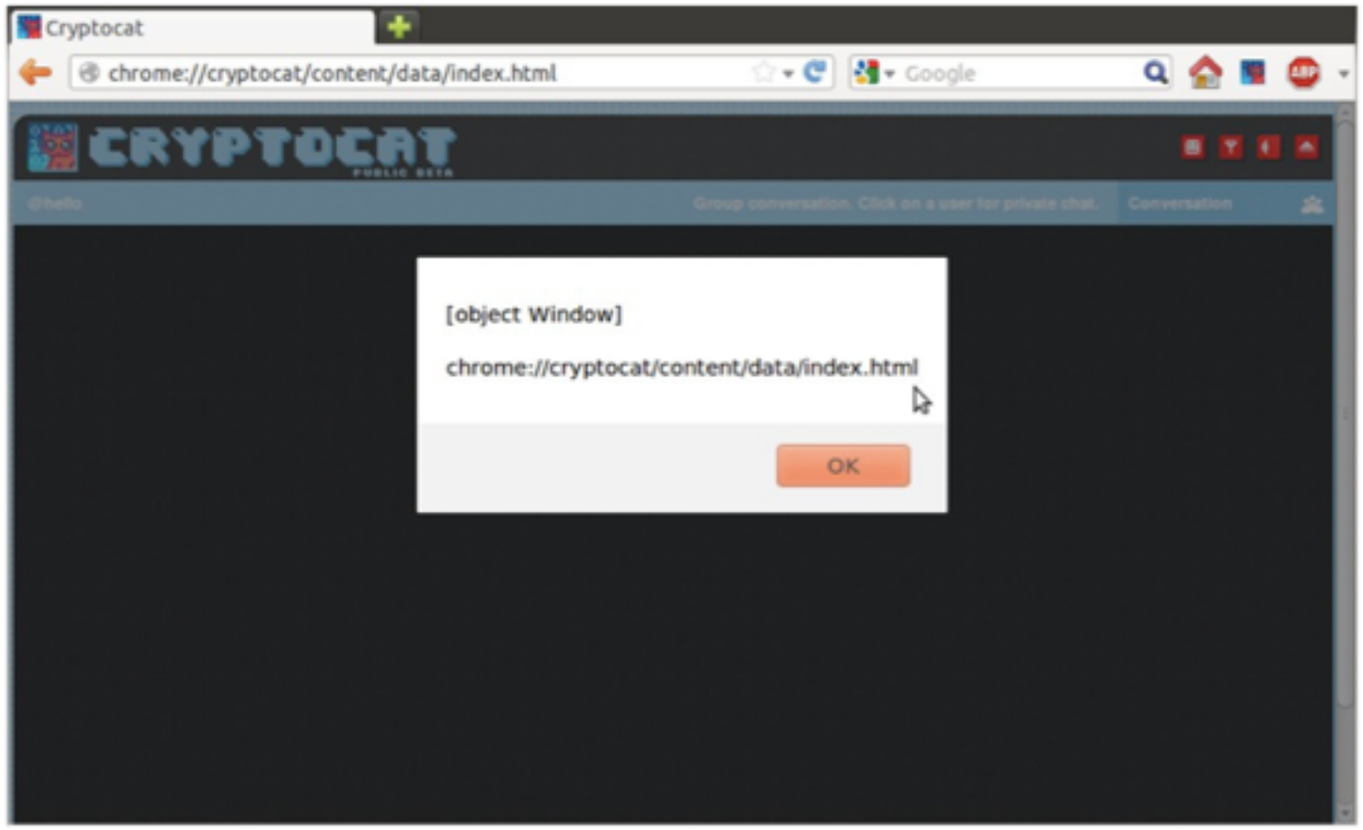
XSS

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

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



- [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.**

Poor randomness

- Math.random() is not good for crypto
- You can recover the state cross-origin in some browsers

  http://ifsec.blogspot.com/2012/05/cross-domain-mathrandom-prediction.html
- Use crypto.getRandomValues() in browsers* and crypto.randomBytes() in node.js.
- Still, Math.random() is common

* IE from 11, poor mobile browsers support

Poor randomness

- OpenPGP.js RSA encryption padding

```
/**
* Create a EME-PKCS1-v1_5 padding
*/
encode: function(message, length) {
    //...
    for (var i = 0; i < length - message.length - 3; i++) {
        result += String.fromCharCode(random.getPseudoRandom(1, 255));
    }
    return result;
}

random.getPseudoRandom: function(from, to) {
    return Math.round(Math.random() * (to - from)) + from;
}
```

Poor randomness

- [CVE-2013-4102] Cryptocat uses BOSH for XMPP transport.
  “The session identifier (SID) and initial request identifier (RID) are security-critical and therefore MUST be both unpredictable and non-repeating.”

```
this.rid = Math.floor(Math.random() * 4294967295);
```

Non-JS randomness fai

Not Javascript

Debian OpenSSL fiasco (2006-2008)

- OpenSSL used uninitialized memory buffers as entropy sources

- Debian maintainer analyzed OpenSSL with Valgrind, asked *openssl-dev* about the warnings. Group said - go ahead, just remove the calls.

- Only process ID remained in the entropy pool

- ssh-keygen - only **32K** possible keys
  http://research.swtch.com/openssl

Timing side-channels

- Timing differences are measurable, even cross-origin
- Exploits are not remote - all code runs on the same CPU, <iframe>s let you jump in a same thread even!
- Demonstrated by Eduardo Vela Nava
  http://sirdarckcat.blogspot.com/2014/05/matryoshka-web-application-timing.html
  “It is possible to bruteforce an 18 digit number in about 3 minutes on most machines.” (cross-domain!)

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

Not Javascript

Compiler optimisation

- JS engine is a blackbox. Even correct constant-time code can be optimised. http://stackoverflow.com/questions/18476402/how-to-disable-v8s-optimizing-compiler

- Problems are not unique to the web:

```
// golang.org/src/pkg/crypto/subtle/constant_time.go
func ConstantTimeByteEq(x, y uint8) int {
    z := ^(x ^ y)
    z &= z >> 4
    z &= z >> 2
    z &= z >> 1

    return int(z)
}
```

- Constant-time algorithm meets timing differences in Intel DIV instruction https://www.imperialviolet.org/2013/02/04/luckythirteen.html

Direct memory access

- Remember Heartbleed?

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

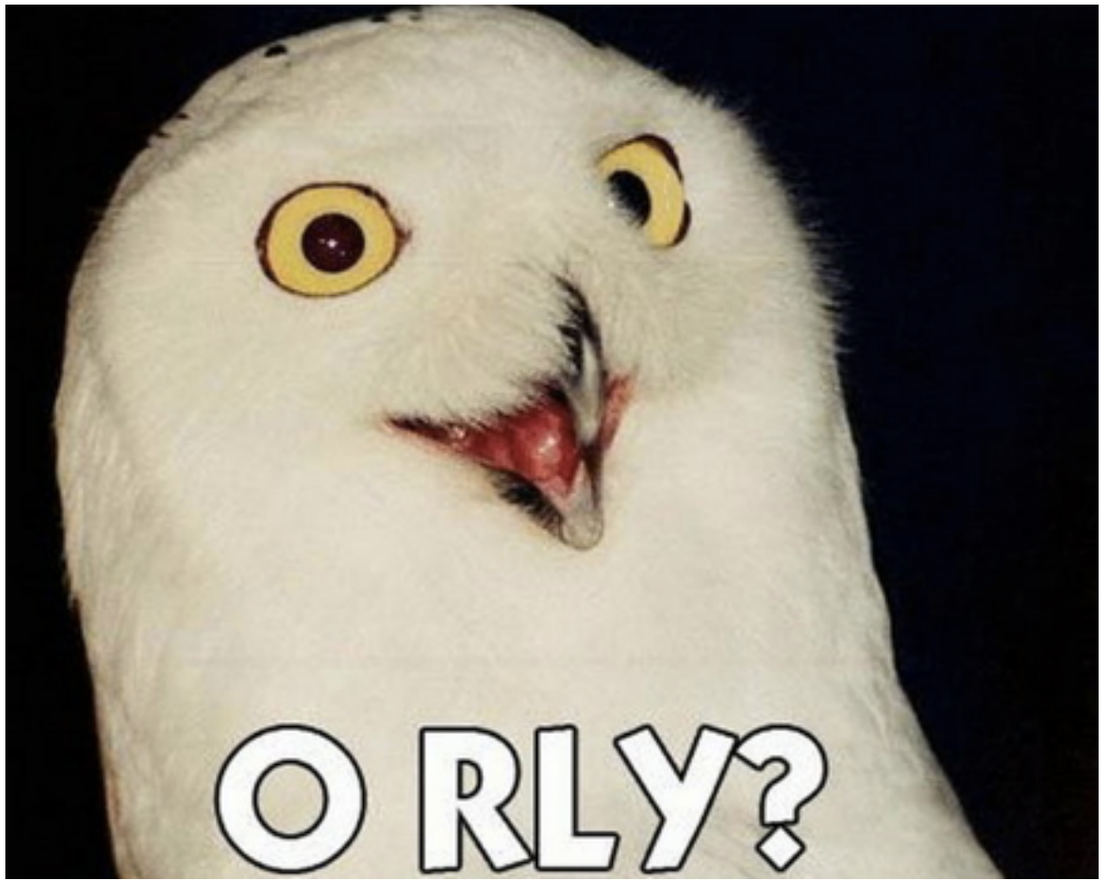
  - client sends a large payload length + a tiny payload

  - 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

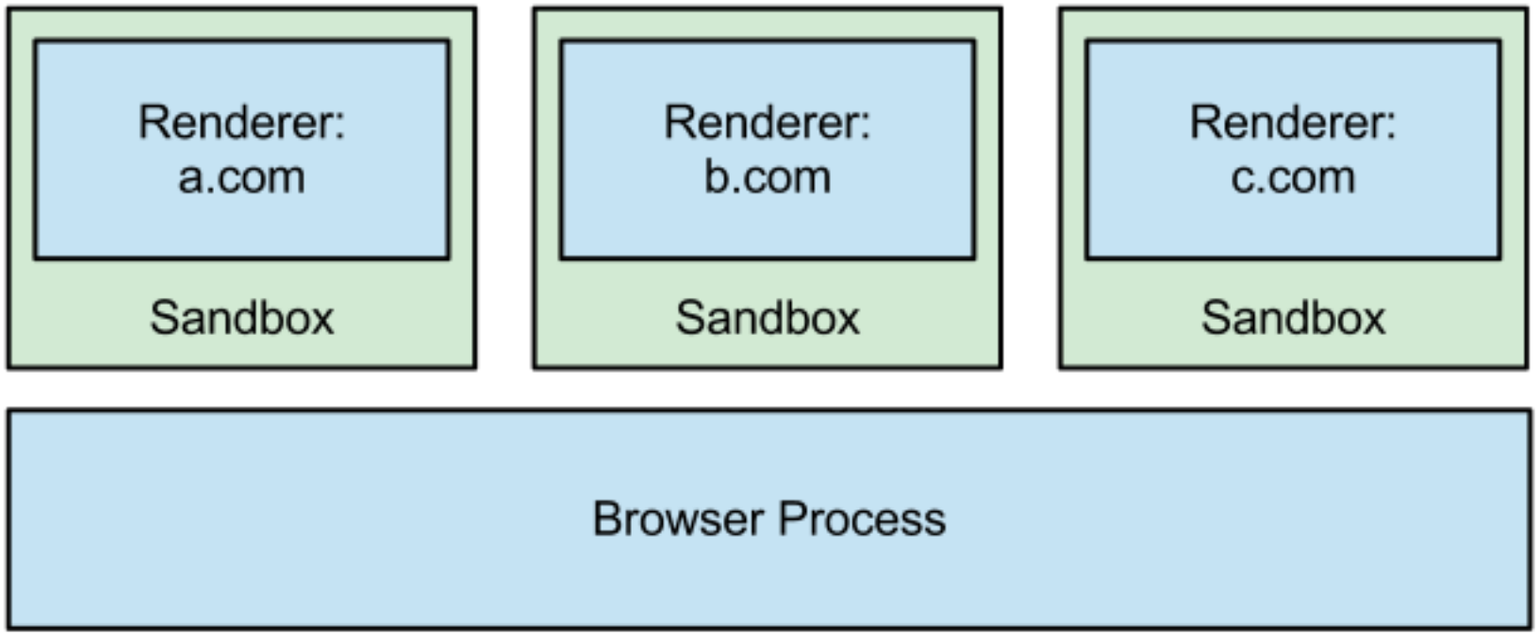
- JS crypto code executes in an environment
- Browsers are an attack surface as well
  - network stack
  - HTML parser
  - JS engine
- Exploited browser ~ malware

Browsers architecture

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

- IE - multiprocess

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



Malware problem

- Any malware can circumvent standard crypto software as well. Kernels have exploits 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
- It’s a huge code execution playground, running code separated by (hopefully) Same Origin Policy
- Few browsers have sandboxes to enforce further restrictions

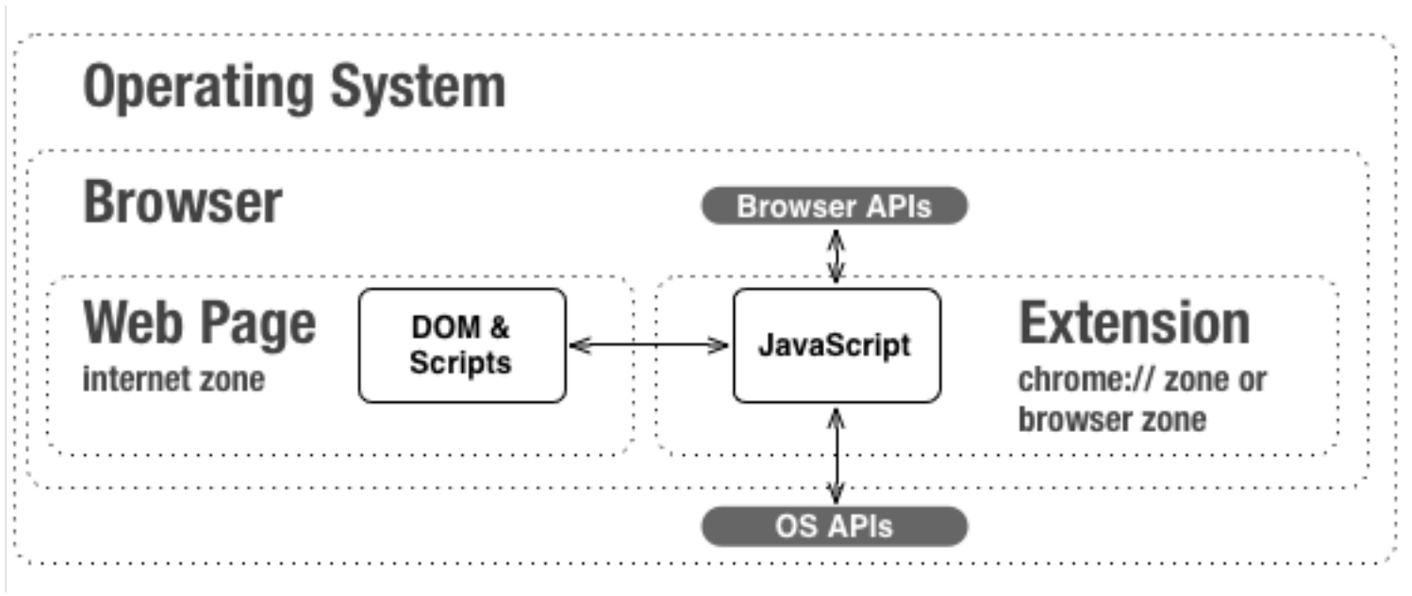
Is JS crypto doomed?

- Create perfect, XSS-free, constant time JS code
- Put in in a website, serve over HTTPS
- You're safe until someone uses:
  - a browser exploit
  - a SOP bypass

Extensions for 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
- Separate storage area
- Better separation from websites than just Same Origin Policy
- Process isolation in Chrome http://www.chromium.org/developers/design-documents/site-isolation

Browser extension

- Not a perfect solution!
  - Chrome Extensions can share processes when limits are hit (use Chrome App to be sure)
  - XSS in extension is possible. XSS = native code execution in Firefox
    http://www.slideshare.net/kkotowicz/im-in-ur-browser-pwning-your-stuff-attacking-with-google-chrome-extensions
  - Timings are readable

Open problems

- Optimisations in JS engines make timing side channels probable
- No mlock() equivalent - secrets can be swapped to disk
- No secure store yet (wait for WebCrypto)
- Extensions silent auto-update
- Lack of full process isolation yet

Summary

- A lot of perceived “JS crypto flaws” are present in other languages as well
- The platform issues are much more difficult to mitigate
- Only extension-based crypto can be secure
- Malware, as always, wins

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, Daniel Bleichenbacher, Adam Langley, Mathias Biennia