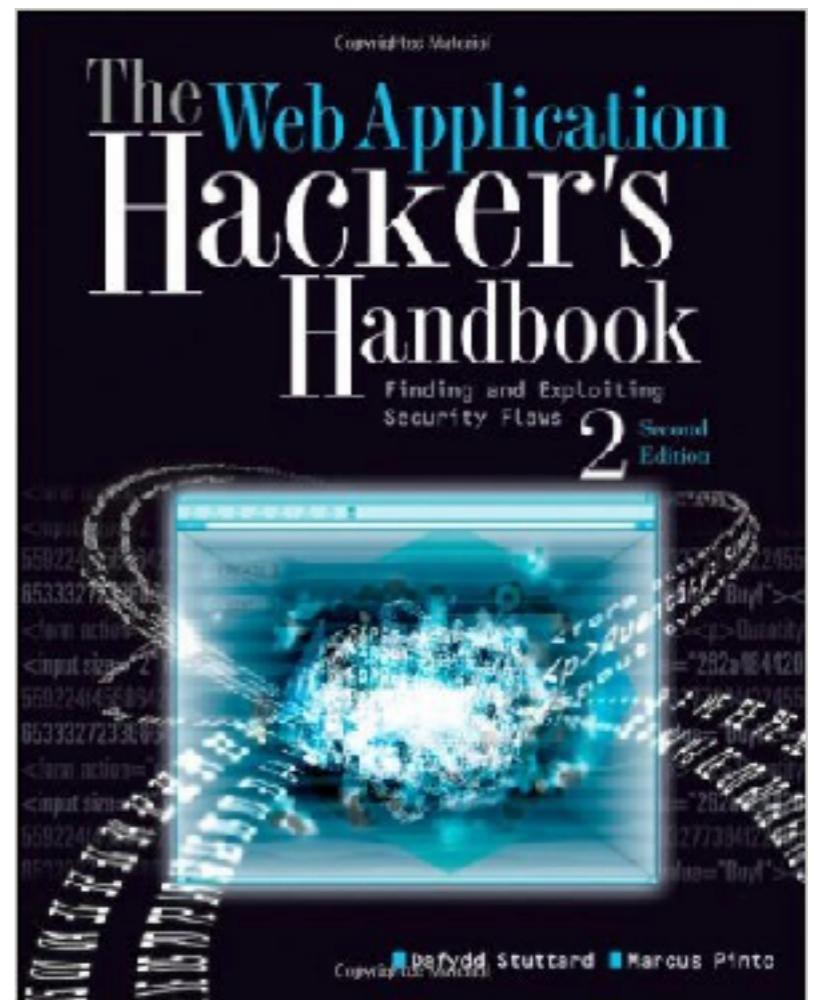


CNIT 129S: Securing Web Applications

Ch 7: Attacking Session Management

Updated 2-21-18



Session Management

- **Enables application to identify a given user over a number of different requests**
 - **Ex: login, then later requests**
- **Fundamental security component**
- **A prime target for attackers**

Session Management

- **Potential consequences of session management attacks**
 - A user can masquerade as another user
 - Privilege escalation to administrator, owning the entire application
 - Can be as simple as incrementing the value of a token

Wall of Sheep

- My Gmail was hacked at Defcon
- By stealing and replaying my session cookie
- Using Hamster and Ferret

| login | pass | domain_ip | application |
|---------------------|----------|-------------------------|-------------|
| bogususer | bmt***** | pop.west.cox.ie | POP |
| guyman | DUD***** | www.chroniclesofreal.co | HTTP |
| arm_619 | hi***** | myspace.com | HTTP |
| stealthdc5tuh0 | hon***** | yahoo.com | HTTP |
| RenLucas | hon***** | forum.geeksquadforums | IMAP |
| web61pl | dex***** | 62.153.108.10 | POP |
| adam | pen***** | 66.218.85.170 | POP |
| lmediss@prodigy.net | xy7***** | innoveted.com | POP |
| vlk | chi***** | 195.47.75.55 | HTTP |
| nash | eve***** | myspace.com | HTTP |
| | | cisco.com | HTTP |

The Need for State

Web 1.0

- **Stateless**
- **Static pages**
- **No custom content**
- **No logging in**

Logging In

- **Allow user to register and log in**
- **Require a session to maintain the "state" of being authenticated**
- **Otherwise user would have to log in to each page**

No Login

- **Application with no login function still use sessions**
- **Shopping basket**

Session Token

- **Server's first response contains a Set-Cookie: header**

Set-Cookie: ASP.NET_SessionId=mza2ji454s04cwbgb2ttj55

- **Each subsequent request from the client contains the Cookie: header**

Cookie: ASP.NET_SessionId=mza2ji454s04cwbgb2ttj55

Vulnerabilities

- **Two categories**
 - **Weakness in generation of session tokens**
 - **Weakness in handling session tokens throughout their life cycle**

Finding the Real Session Token

- Sometimes a platform like ASP.NET generates a token but the app doesn't really use it
- To find a token, establish a session and then replay a request
 - Systematically remove each item you suspect of being the real token
 - Wait till response is no longer customized for your session
 - Burp Repeater is good for this

Demonstration

Copy to Repeater

The screenshot shows the Charles Web Proxy interface. The top navigation bar includes tabs for Intercept, HTTP history, WebSockets history, and Options. The main area displays a list of network requests. A single entry is selected, showing a GET request to `/account` from the host `http://hackazon.samsclass.info`. Below this, there are tabs for Request, Response, Raw, Params, Headers, and Hex. The Request tab is active, showing the URL and method. The Headers tab shows the following cookies:

| Type | Name | Value |
|--------|-------------------------------|--|
| Cookie | <code>__cfduid</code> | <code>d702f29bef62072ec1b61d321a932a9861512448834</code> |
| Cookie | <code>visited_products</code> | <code>,81,72,64,1,16,</code> |
| Cookie | <code>PHPSESSID</code> | <code>snclqpgkaf4ltcq0vsdjki7i1</code> |

Modify & Go

The screenshot shows a user interface for a network proxy or testing tool. At the top, there is a horizontal navigation bar with tabs: Repeater, Sequencer, Decoder, Comparer, and Extender. Below this, a row of buttons allows selecting a specific item from a list, with '1' and '2' currently selected. A set of control buttons includes 'Go', 'Cancel', and navigation arrows. The main area is titled 'Request' in orange. Below it, there are four tabs: Raw, Params, Headers, and Hex, with 'Params' being the active tab. A message indicates a 'GET request to /account'. A table lists cookie parameters with their names and values. To the right of the table are four buttons: 'Add', 'Remove', 'Up', and 'Down'.

| Type | Name | Value |
|--------|-----------------|---------------------------|
| Cookie | _cfduid | d702f29bef62072ec1b6... |
| Cookie | visited_prod... | ,81,72,64,1,16, |
| Cookie | PHPSESSID | snclqpgkaf4ltcq0vsdjdk... |

Request

Raw Params Headers Hex

GET request to /account

| Type | Name | Value |
|--------|-----------------|---------------------------|
| Cookie | _cfduid | d702f29bef62072ec1b6... |
| Cookie | visited_prod... | ,81,72,64,1,16, |
| Cookie | PHPSESSID | snclqpgkaf4ltcq0vsdjdk... |

Add Remove Up Down

Demonstration

Amazon.com

The screenshot shows the Burp Suite interface. The top menu bar includes Burp, Intruder, Repeater, Window, and Help. The main toolbar has tabs for Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, and Comparer. Below the toolbar are buttons for Intercept, HTTP history, WebSockets history, and Options. A filter bar at the top says "Filter: Hiding CSS, image and general binary content". The main table displays network requests:

| # | Host | Method | URL |
|------|---------------------------|--------|------------------------------------|
| 1267 | https://fls-na.amazon.com | POST | /1/batch/1/OE/ |
| 1268 | https://www.amazon.com | GET | /gp/navigation/ajax/dynamic-menu.h |
| 1269 | https://www.amazon.com | GET | /gp/yourstore/home/ref=nav_cs_y |
| 1270 | https://dns.google.com | GET | /resolve?name=www.amazon.com |

Original Request

Burp Suite Free Edition v1.7.03 - Temporary Project

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

4 ...

Go Cancel < | > | ?

Request

Raw Params Headers Hex

GET request to /gp/yourstore/home/ref=nav_cs_y

| Type | Name | Value |
|--------|-----------------|---|
| Cookie | skin | noskin |
| Cookie | session-id | 135-4731125-9794301 |
| Cookie | session-id-time | 2082787201 |
| Cookie | csm-hit | {"tb": "YHXF87R2DGBY3AGQRTJ9+s-GHN2A8QNT8FARVXNF6H8 ...} |
| Cookie | x-wl-uid | 1yopolVDki6M1fjsP90iFmNWsZ9mrvo8B2EOHw s9KQLhjGTop... |
| Cookie | ubid-main | 130-4799368-8113216 |
| Cookie | session-token | "99IpMmxYKQ8k89zwdL JazpiVdmg67/sRP6Q2tYP4VzJJCctlpM... |
| Cookie | a-ogbcbff | 1 |
| Cookie | x-main | "ogEzTsmQe80akA4n9FSwypP2?td1OFA7XZaxpoormRz6PsLUu... |
| Cookie | at-main | Atza lwEBIBLB96EL80UPVcVuoUpNnYIRERzZz_G6TpVDDhxdnE9P... |
| Cookie | sess-at-main | "cAYVyKa7VR4cSkUBw8OkolQGQVsj/DW3ccQliYEJErE=" |
| Cookie | sst-main | Sst1 PQE2sYtPLx45ccwlXJ2Vrr6dCLEVIZG9_1MEs_o7lp-BFMkMI... |

Add Remove Up Down

Target: https://www.amazon.com

Response

Raw Headers Hex HTML Render

Interesting Finds Updated Daily

NEW & INTERESTING F

Amazon Try Prime

fire tv stick \$39.99

All Departments

Go

Search

Deliver to sanfrancisco 94102‌

EN Hello, samccsf Account & Lists Hi

samccsf Account & Lists Not samccsf? Sign Out

Orders Try Prime Cart 0

Departments

Your Pickup Location Browsing History samccsf's

Amazon.com Today's Deals Gift

Cards Registry Sell Help Disability Customer Support

Your Amazon.com Your Browsing History

Recommended For You Improve Your

Recommendations Your Profile Learn More

samccsf's Amazon Dashboard

Minimal Request

Target | Proxy | Spider | Scanner | Intruder | Repeater | Sequencer | Decoder | Comparer | Extender | Project options | User options | Alerts
4 × 5 × 6 × 7 × ...

Go Cancel < | > | Target: <https://www.amazon.com>  

Request
Raw Params Headers Hex

GET request to /gp/yourstore/home/ref=nav_cs_ys

| Type | Name | Value |
|--------|--------|----------------------------|
| Cookie | x-main | "ogEzTsmQe80akA4n9FSwy..." |

Add Remove Up Down Go

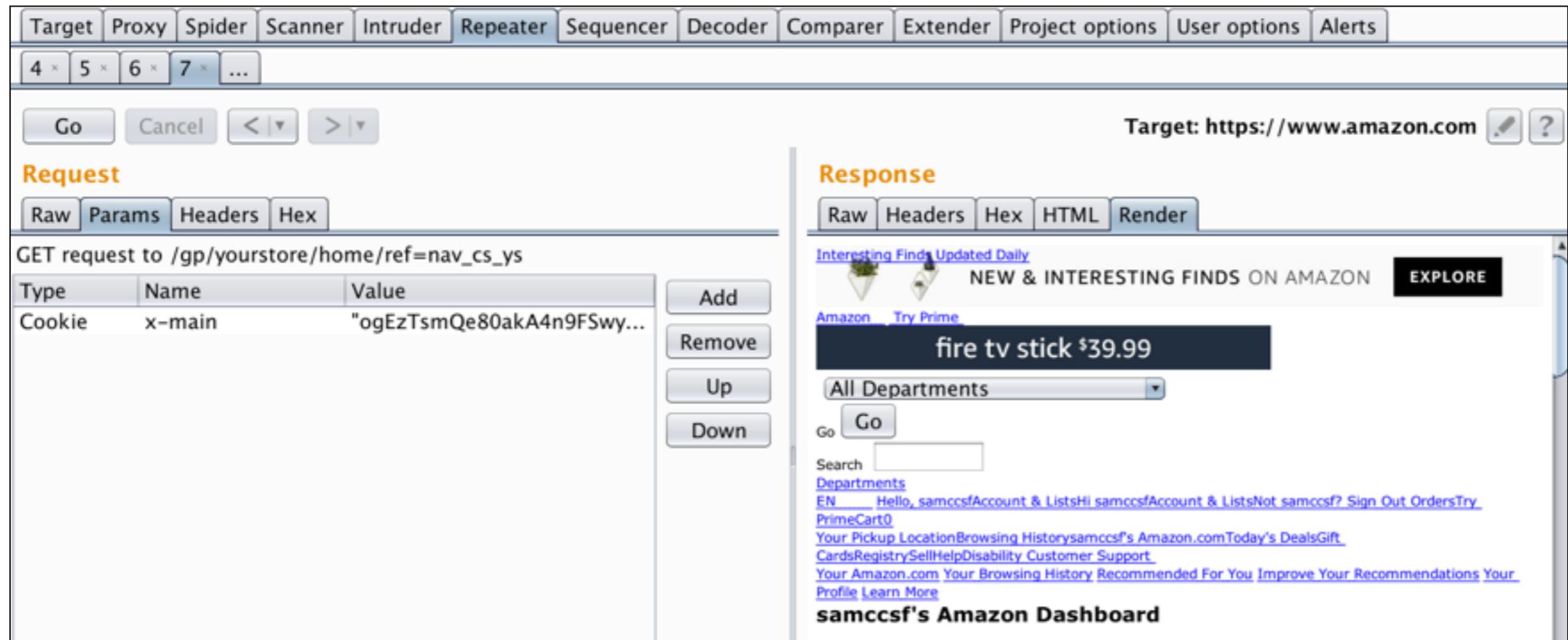
Response
Raw Headers Hex HTML Render

Interesting Finds Updated Daily NEW & INTERESTING FINDS ON AMAZON EXPLORE

Amazon Try Prime fire tv stick \$39.99

All Departments Go

Search
Departments
EN Hello, samccsf Account & Lists Hi samccsf Account & Lists Not samccsf? Sign Out Orders Try Prime Cart 0
Your Pickup Location Browsing History samccsf's Amazon.com Today's Deals Gift Cards Registry Sell Help Disability Customer Support
Your Amazon.com Your Browsing History Recommended For You Improve Your Recommendations Your Profile Learn More
samccsf's Amazon Dashboard



Alternatives to Sessions

HTTP Authentication

- **Basic, Digest, NTLM**
- **Pass credentials with every request in HTTP headers**
- **Not via application-specific code**
- **Rarely used on Internet-based applications**

Sessionless State Mechanisms

- Application doesn't issue session tokens or manage the state
- Transmit all data required to manage the state via the client in a cookie or hidden form field
- Ex: ASP.NET ViewState
 - Link Ch 7a

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
<script runat="server">
    // Sample ArrayList for the page.
    ArrayList PageArrayList;

    ArrayList CreateArray()
    {
        // Create a sample ArrayList.
        ArrayList result = new ArrayList(4);
        result.Add("item 1");
        result.Add("item 2");
        result.Add("item 3");
        result.Add("item 4");
        return result;
    }

    void Page_Load(object sender, EventArgs e)
    {
        if (ViewState["arrayListInViewState"] != null)
        {
            PageArrayList = (ArrayList)ViewState["arrayListInViewState"];
        }
    }
}
```

Data Types You Can Store in View State

You can store objects of the following types in view state:

- Strings
- Integers
- Boolean values
- Array objects
- ArrayList objects
- Hash tables
- Custom type converters (see the [TypeConverter](#) class for more information)

You can store other types of data also, but the class must be compiled with the [Serializable](#) attribute so that its values can be serialized for view state.

Securing View State

- **Data must be protected, usually as a binary blob that is encrypted or signed**
 - **To prevent re-use on another machine**
- **ASP.NET View State uses Base64 and a hash made from a Machine Authentication Code**
- **Includes the machine's MAC address**
- **Expiration time enforces session timeouts**

Indicators of Sessionless State Mechanisms

- **Token-like data items ≥ 100 bytes long**
- **New token-like item in response to every request**
- **Data is encrypted (structureless) or signed (structured accompanied by a few random bytes)**
- **Application rejects attempts to submit the same item with more than one request**

Weaknesses in Token Generation

Token Use Cases

- **Password recovery tokens sent to user's email address**
- **Tokens in hidden form fields to prevent cross-site forgery attacks**
- **Tokens used to grant one-time access to protected resources**
- **Persistent tokens used to "remember me"**
- **Tokens used to allow customers of shopping application to see the status of an order**

Unpredictability

- **The security of the application depends on all those tokens being unpredictable**

Meaningful Tokens

For example, the following token may initially appear to be a long random string:

```
757365723d6461663b6170703d61646d696e3b646174653d303  
12f31322f3131
```

- **It's just hexadecimal ASCII for**

```
user=daf;app=admin;date=10/09/11
```

- **Easy to guess other token values, like**
user=admin

ASCII

- 75 73 65 72
- U S e r

| Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|-----|----|-----|-------|-------|-----|----|-----|-------|-----|-----|----|-----|--------|-----|
| 32 | 20 | 040 | | Space | 64 | 40 | 100 | @ | Ø | 96 | 60 | 140 | ` | ` |
| 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 36 | 24 | 044 | $ | \$ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 40 | 28 | 050 | (| (| 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 41 | 29 | 051 |) |) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [| [| 123 | 7B | 173 | { | { |
| 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | |
| 61 | 3D | 075 | = | = | 93 | 5D | 135 |] |] | 125 | 7D | 175 | } | } |
| 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | | DEL |

Source: www.LookupTables.com

Components in Structured Tokens

- The account username
- The numeric identifier that the application uses to distinguish between accounts
- The user's first and last names
- The user's e-mail address
- The user's group or role within the application
- A date/time stamp
- An incrementing or predictable number
- The client IP address

Common Encoding Schemes

- **XOR**
- **Base64**
- **Hexadecimal ASCII codes**

Hack Steps

- Obtain a single token
- Modify it in systematic ways
- Change it one byte at a time, or one bit at a time
- Resubmit it to see if it's still accepted
- Some fields may be ignored
- Burp Intruder's "char frobber" function

Hack Steps

- **Log in as several users at different times**
- **Record the tokens**
- **If you can, register similar usernames like A, AA, AAA, AAAB, etc.**
- **Try similar series for other data, such as email addresses**

Hack Steps

- **Analyze the tokens for correlations**
- **Look for encoding or obfuscation**
- **A series of repeating letters like AAA will produce repeating encoded characters like zzz if XOR is used**
- **Base64 often ends in = or ==**

Hack Steps

- **Use the patterns you've found to try guessing tokens of other users**
- **Find a page that is session-dependent**
- **Use Burp Intruder to send many requests using guessed tokens**
- **Monitor results to see if any pages load correctly**

Predictable Tokens

Patterns

- From a sample of tokens, it may be possible to predict valid tokens
- Commercial implementations such as web servers or application platforms may be more vulnerable
 - Because it's easier to gather a large sample of tokens, from your own test system

Sequential Tokens

- Burp trying sequential payloads
- Winners shown at to the top

The screenshot shows the Burp Suite interface during an 'intruder attack'. The main window title is 'intruder attack 4'. At the top, there are buttons for 'attack', 'save', and 'columns'. A filter bar below says 'Filter: showing all items'. The main area is a table with several columns: 'request', 'payload', 'status', 'error', 'timeo...', 'length', 'Logged in as:', and 'comment'. The table contains 10 rows, with the first three rows highlighted in blue, indicating they are the 'winners' (sequential tokens found). The 'payload' column for these rows shows values 07, 1b, and 22 respectively. The 'Logged in as:' column shows user names: John Herman, Administrator, and Pabilna. In the bottom right corner of the table, the text 'baseline request' is visible. Below the table, there is a preview pane titled 'request' which shows an HTTP GET request to '/auth/340/Home.ashx' with various headers and a cookie 'Cookie: SessionId=516062E93E9FB22'.

| request | payload | status | error | timeo... | length | Logged in as: | comment |
|---------|---------|--------|-------|----------|--------|---------------|------------------|
| 8 | 07 | 200 | | | 1314 | John Herman | |
| 28 | 1b | 200 | | | 1348 | Administrator | |
| 35 | 22 | 200 | | | 1329 | Pabilna | |
| 73 | 48 | 200 | | | 1312 | GUnit | |
| 128 | 7f | 200 | | | 1357 | PSCDMGIIJr | |
| 147 | 92 | 200 | | | 1332 | Keycock | |
| 0 | | 302 | | | 546 | | baseline request |
| 1 | 00 | 302 | | | 546 | | |
| 2 | 01 | 302 | | | 546 | | |
| 3 | 02 | 302 | | | 546 | | |

request response

raw params headers hex

```
GET /auth/340/Home.ashx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.10)
Gecko/20100914 Firefox/3.6.10
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: close
Referer: https://mdsec.net/auth/340/Default.ashx
Cookie: SessionId=516062E93E9FB22
```

+ < > finished 0 matches

Three Sources of Predictable Session Tokens

- **Concealed sequences**
- **Time dependency**
- **Weak random number generation**

Concealed Sequences

```
lwjVJA  
Ls3Ajq  
xpKr+A  
xleXYg  
9hyCzA  
jeFuNg  
JaZZoA
```

- This sequence appears to be Base64-encoded
- Decodes to the gibberish on the right

```
--õ$  
.íÀž  
æ' «ø  
^w-b  
ö,ì  
?án6  
ö!y
```

Hexadecimal Form

```
9708D524  
2ECDC08E  
C692ABF8  
5E579762  
F61C82CC  
8DE16E36  
25A659A0
```

- **Calculate difference between sequential tokens**
- **For negative differences, add 0x1000000000 so it starts with FF**

```
FF97C4EB6A  
97C4EB6A  
FF97C4EB6A  
97C4EB6A  
FF97C4EB6A  
FF97C4EB6A
```

Script to Decode

```
#!/usr/bin/python

import base64

raw = ['lwjVJA', 'Ls3Ajq', 'xpKr+A', 'XleXYg', ' 9hyCzA', 'jeFuNg', 'JaZZoA']
dec = []

print "Base64 Decode"
for i in range(7):
    dec.append(base64.b64decode(raw[i] + '=='))
    print i, raw[i], dec[i]

print
print "Convert to hex"
h = []
for i in range(7):
    x = ''
    for j in range(4):
        x += str(hex(256+ord(dec[i][j])))[3:]
    h.append(x)
    print i, raw[i], dec[i], h[i]

print
print "Gather hex digits together"
n = []
for i in range(7):
    n.append(int('0x' + h[i], 16))
    print i, hex(n[i])

print
print 'Calculate differences'
diff = []
for i in range(6):
    diff.append(n[i] - n[i+1])
    if diff[i] < 0:
        diff[i] += 0x100000000000
    print i, hex(diff[i])
```

Generate Valid Tokens

```
#!/usr/bin/python

token = 0x9708d524
print hex(token)

for i in range(7):
    token += 0x97C4EB6A
    if token > 0xffffffff:
        token -= 0x100000000
    print hex(token)
```

```
0x9708d524
0x2ecdc08e
0xc692abf8
0x5e579762
0xf61c82cc
0x8de16e36
0x25a659a0
0xbd6b450a
```

Time Dependency

| | |
|-----------------------|-----|
| 3124538-1172764258718 | 344 |
| 3124539-1172764259062 | 219 |
| 3124540-1172764259281 | 453 |
| 3124541-1172764259734 | 312 |
| 3124542-1172764260046 | 110 |
| 3124543-1172764260156 | 140 |
| 3124544-1172764260296 | 125 |
| 3124545-1172764260421 | 391 |
| 3124546-1172764260812 | 78 |
| 3124547-1172764260890 | |

- **Left number simply increments by 1 each time**
- **Right number moves up by a varying value, as shown on the right**

Another Sample

```
3124553-1172764800468  
3124554-1172764800609  
3124555-1172764801109  
3124556-1172764801406  
3124557-1172764801703  
3124558-1172764802125  
3124559-1172764802500  
3124560-1172764802656  
3124561-1172764803125  
3124562-1172764803562
```

- **Ten minutes later**
- **First number has jumped by 6**
- **Second number has jumped by 539578**
- **Ten minutes = 600 sec = 600,000 milliseconds**

Attack Steps

- **Poll the server frequently to gather session tokens**
- **When first number increases by more than 1, there's another user in between**
- **We know the second number will be between the two numbers we have**
 - **Simply brute-force the value**

Weak Random Number Generation

- Jetty is a Java-based Web server
- Calculates pseudorandom session tokens with a "linear congruent generator"
- Multiplies previous number by a constant, adds another constant, truncates to 48 bits
- Given one value, all others can be predicted

```
synchronized protected int next(int bits) {  
    seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);  
    return (int)(seed >>> (48 - bits));  
}
```

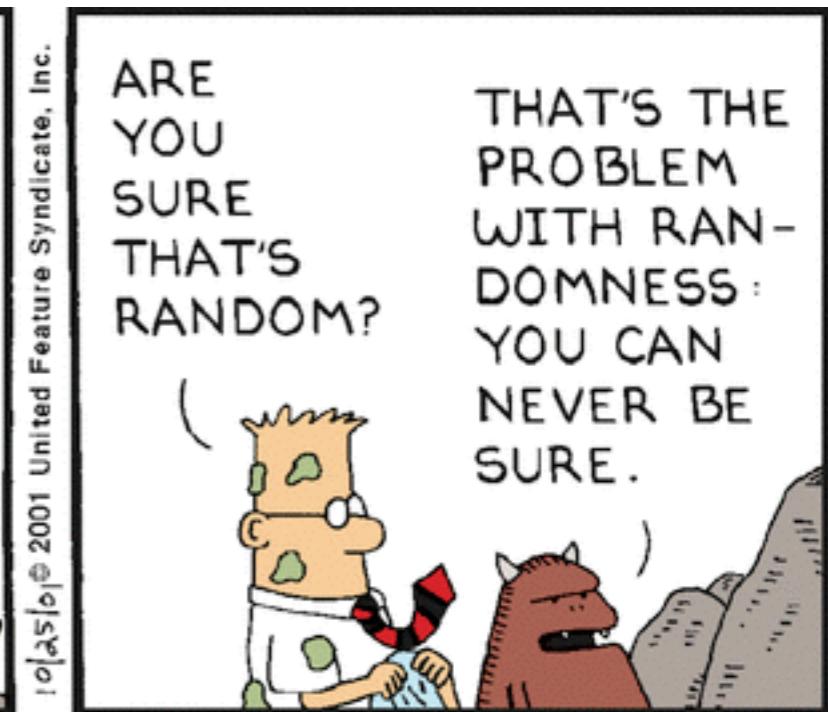
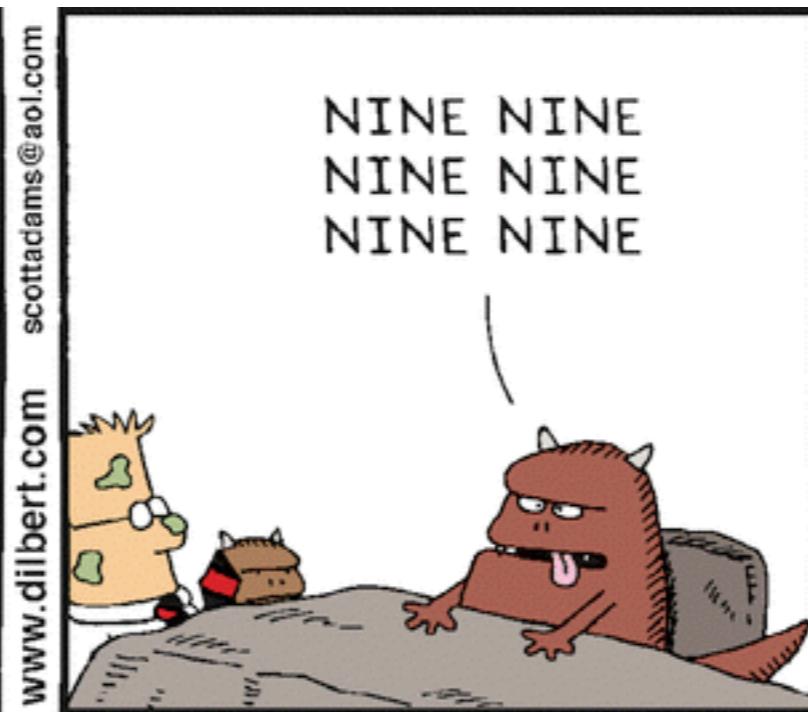
PHP 5.3.2 and Earlier

- **Session token generated from**
 - **Client's IP address**
 - **Epoch time at token creation**
 - **Microseconds at token creation**
 - **Linear congruential generator**

phpwn

- **The vulnerability was found in 2001**
- **But no one wrote a practical attack tool until Samy Kamkar in 2010**
 - **Link Ch 7b**

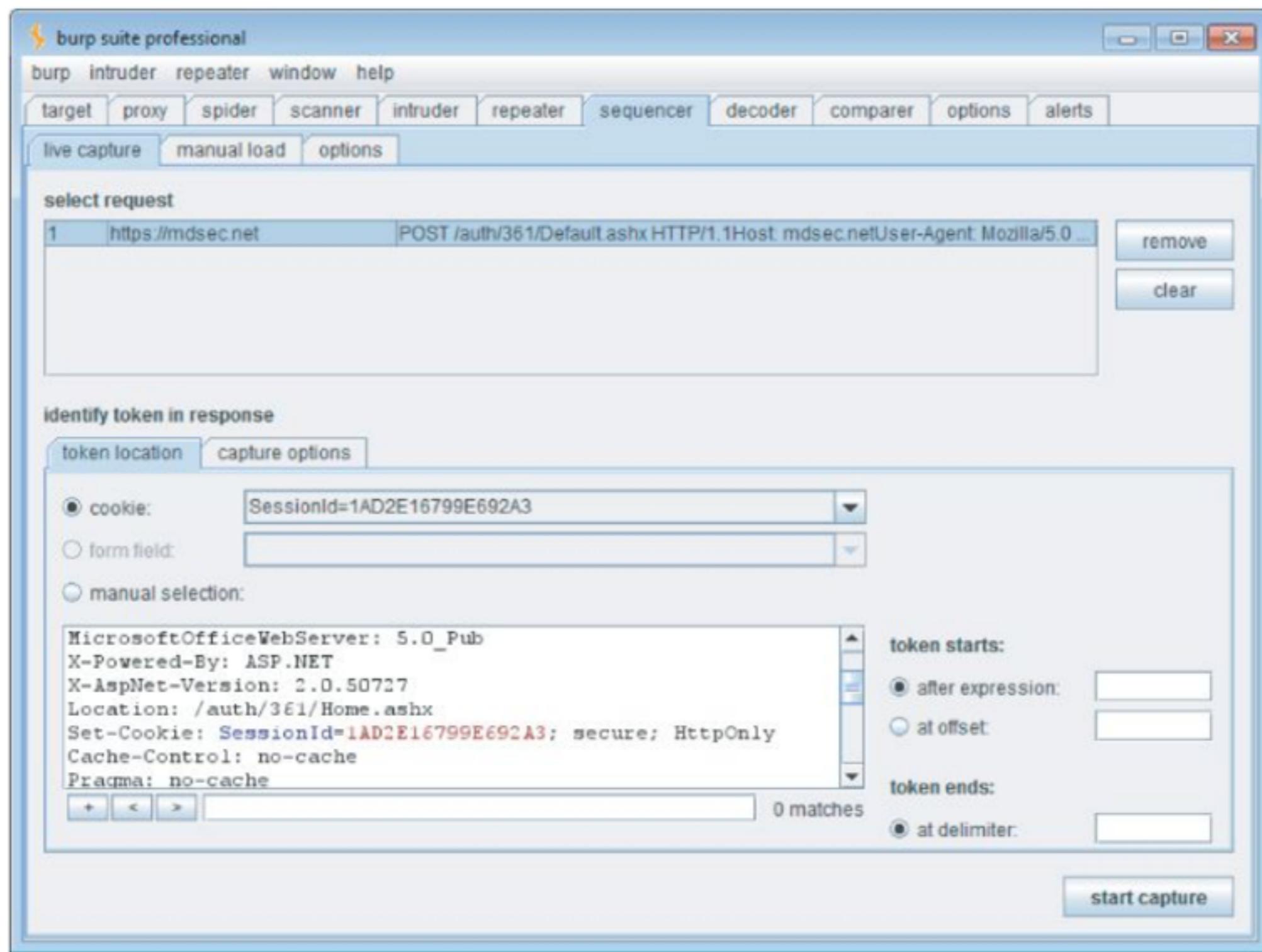
Testing the Quality of Randomness



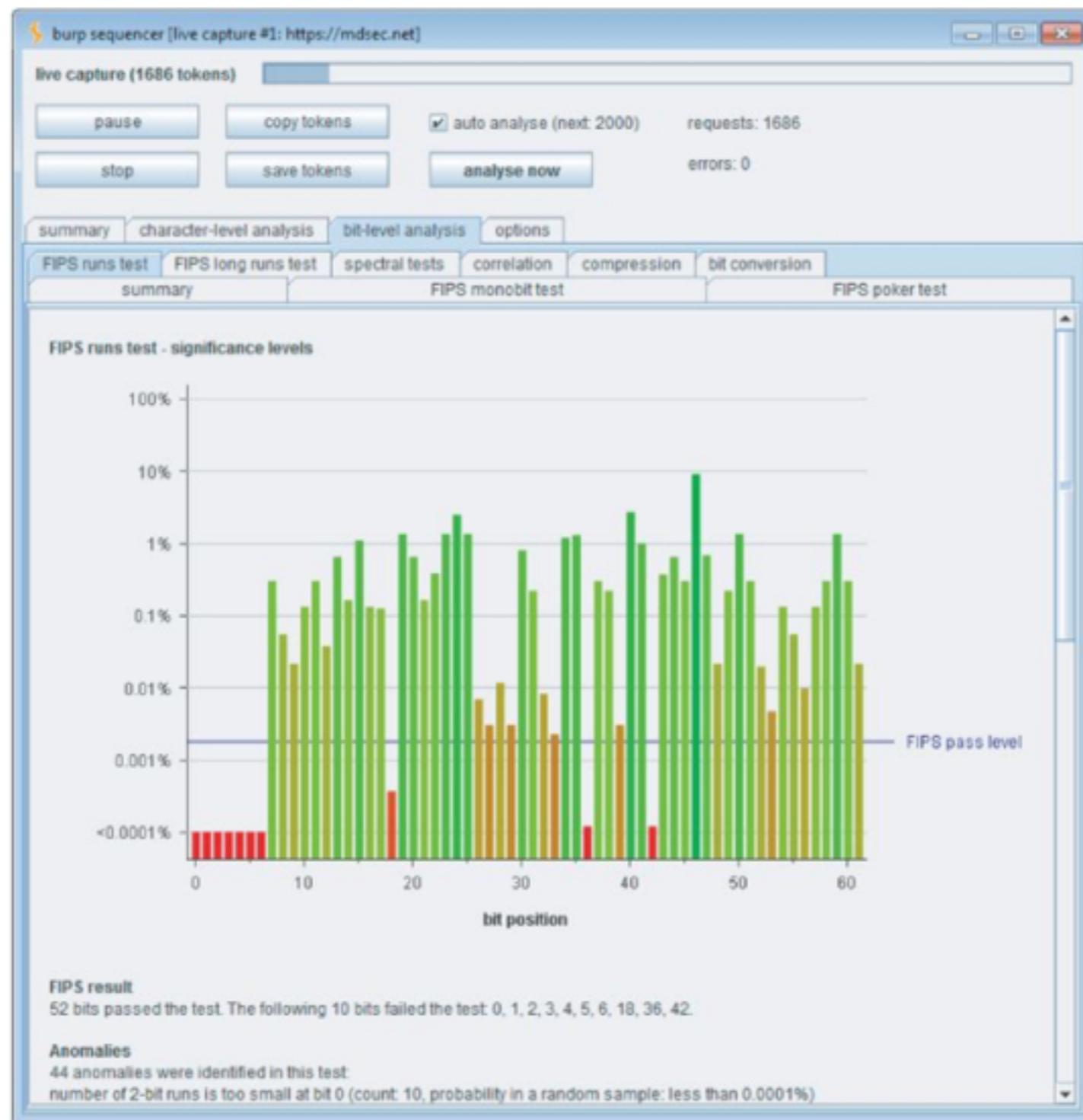
Algorithm

- 1.** Start with the hypothesis that the tokens are randomly generated.
- 2.** Apply a series of tests, each of which observes specific properties of the sample that are likely to have certain characteristics if the tokens are randomly generated.
- 3.** For each test, calculate the probability of the observed characteristics occurring, working on the assumption that the hypothesis is true.
- 4.** If this probability falls below a certain level (the “significance level”), reject the hypothesis and conclude that the tokens are not randomly generated.

Burp Sequencer



Results: Red Bits are Non-Random



Encrypted Tokens

Design Goal

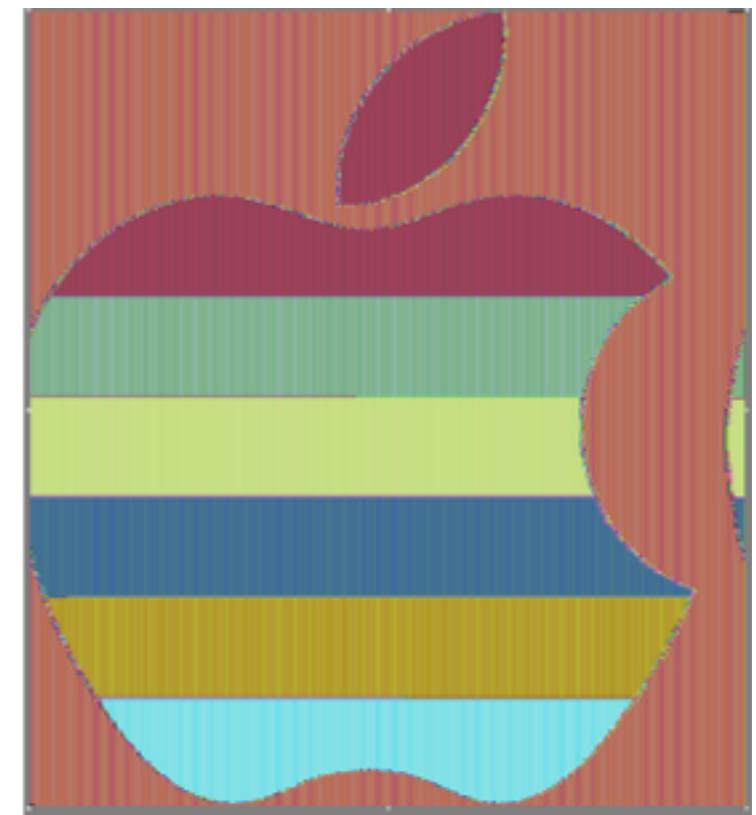
- **Token built from meaningful content, such as username**
- **Encrypted with a secret key not known to the attacker**
- **Sounds good, but sometimes the attacker can tamper with token's meaningful values without decrypting them**

ECB Ciphers

- **Electronic Code Book**
- **Input broken up into blocks, often 8 bytes long**
- **Symmetric encryption, no randomness**
- **Each input block encodes to a single output block**
- **This preserves patterns in input**

Image Encrypted with ECB

- Patterns preserved in output



Example of ECB

- **Token contains several meaningful fields, including numeric user id**

```
rnd=2458992;app=iTradeEUR_1;uid=218;username=dafydd;time=634430423694715  
000;
```

- **Encrypted form appears meaningless**

```
68BAC980742B9EF80A27CBBBC0618E3876FF3D6C6E6A7B9CB8FCA486F9E11922776F0307  
329140AABD223F003A8309DDB6B970C47BA2E249A0670592D74BCD07D51A3E150EFC2E69  
885A5C8131E4210F
```

8-Byte Blocks

| | |
|----------|------------------|
| rnd=2458 | 68BAC980742B9EF8 |
| 992;app= | 0A27CBBBC0618E38 |
| iTradeEU | 76FF3D6C6E6A7B9C |
| R_1;uid= | B8FCA486F9E11922 |
| 218;user | 776F0307329140AA |
| name=daf | BD223F003A8309DD |
| ydd;time | B6B970C47BA2E249 |
| =6344304 | A0670592D74BCD07 |
| 23694715 | D51A3E150EFC2E69 |
| 000; | 885A5C8131E4210F |

Copy a Whole Block

| | |
|-----------------|-------------------------|
| rnd=2458 | 68BAC980742B9EF8 |
| 992;app= | 0A27CBBBC0618E38 |
| iTradeEU | 76FF3D6C6E6A7B9C |
| R_1;uid= | B8FCA486F9E11922 |
| 992;app= | 0A27CBBBC0618E38 |
| 218;user | 776F0307329140AA |
| name=daf | BD223F003A8309DD |
| ydd;time | B6B970C47BA2E249 |
| =6344304 | A0670592D74BCD07 |
| 23694715 | D51A3E150EFC2E69 |
| 000; | 885A5C8131E4210F |

- Token is now for user 992

Register a New User "daf1"

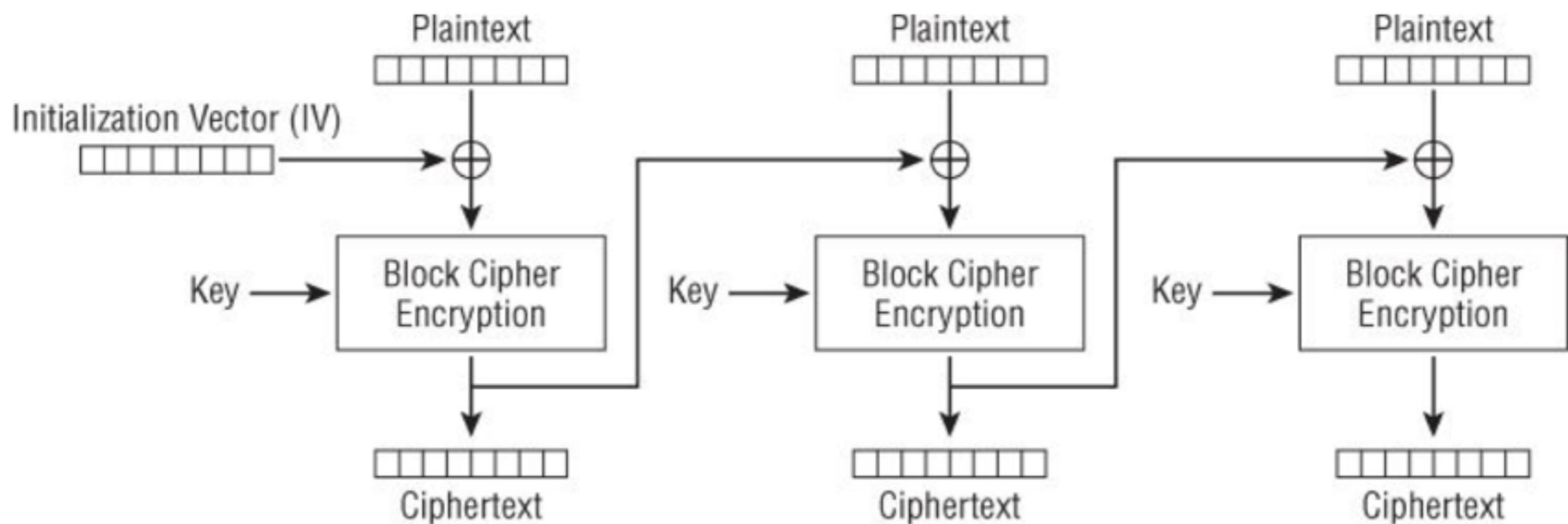
- Now attacker can target uid=1

| | |
|----------|------------------|
| rnd=9224 | 9A5A47BF9B3B6603 |
| 856;app= | 708F9DEAD67C7F4C |
| iTradeEU | 76FF3D6C6E6A7B9C |
| R_1;uid= | B8FCA486F9E11922 |
| 219;user | A5BC430A73B38C14 |
| name=daf | BD223F003A8309DD |
| 1;time=6 | F29A5A6F0DC06C53 |
| 34430503 | 905B5366F5F4684C |
| 61065250 | 0D2BBBB08BD834BB |
| 0; | ADEBC07FFE87819D |

| | |
|----------|-------------------------|
| rnd=9224 | 9A5A47BF9B3B6603 |
| 856;app= | 708F9DEAD67C7F4C |
| iTradeEU | 76FF3D6C6E6A7B9C |
| R_1;uid= | B8FCA486F9E11922 |
| 1;time=6 | F29A5A6F0DC06C53 |
| 219;user | A5BC430A73B38C14 |
| name=daf | BD223F003A8309DD |
| 1;time=6 | F29A5A6F0DC06C53 |
| 34430503 | 905B5366F5F4684C |
| 61065250 | 0D2BBBB08BD834BB |
| 0; | ADEBC07FFE87819D |

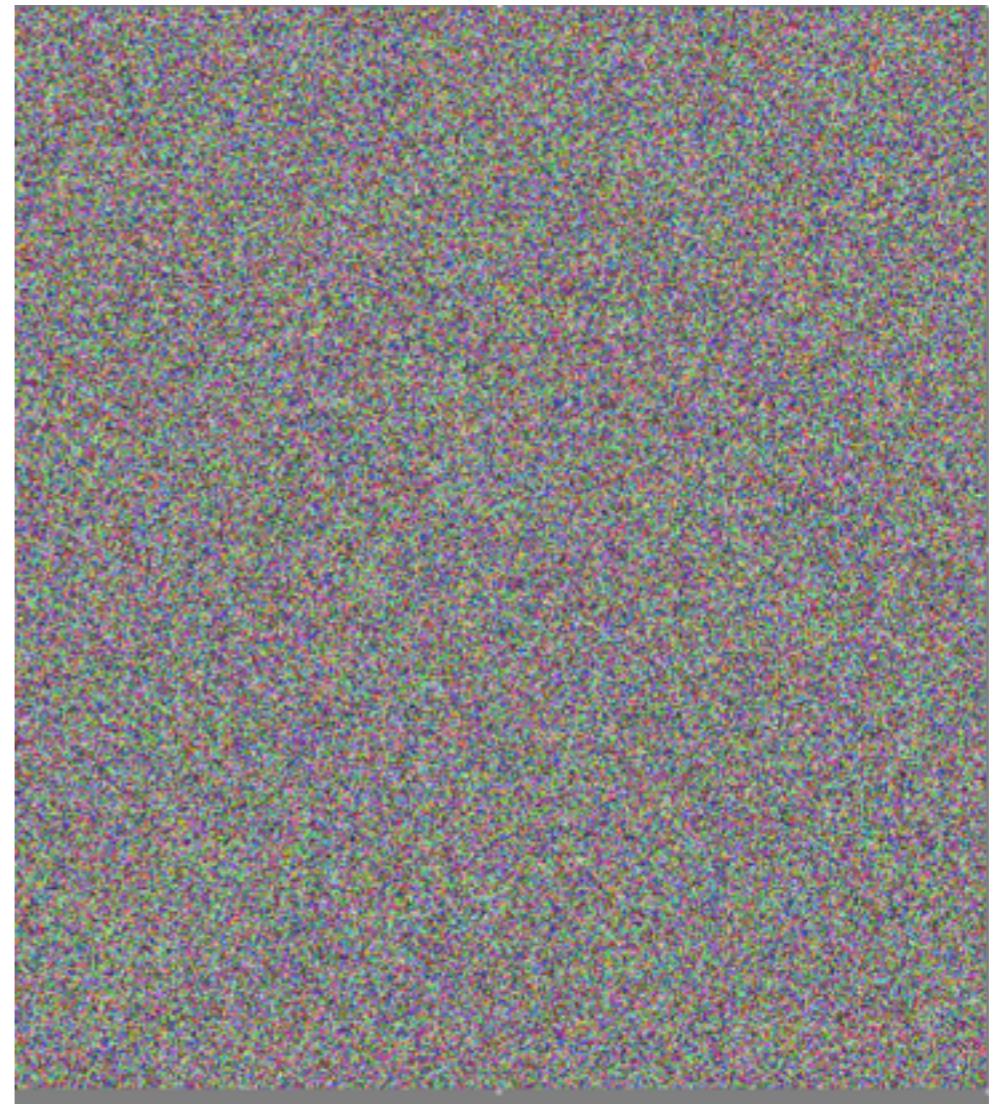
CBC Ciphers

- **Cipher Block Chain mode**
- **XORs each block of plaintext with the preceding block of ciphertext**



CBC Ciphers

- **Removes all visible patterns from the apple logo**



Example: CBC

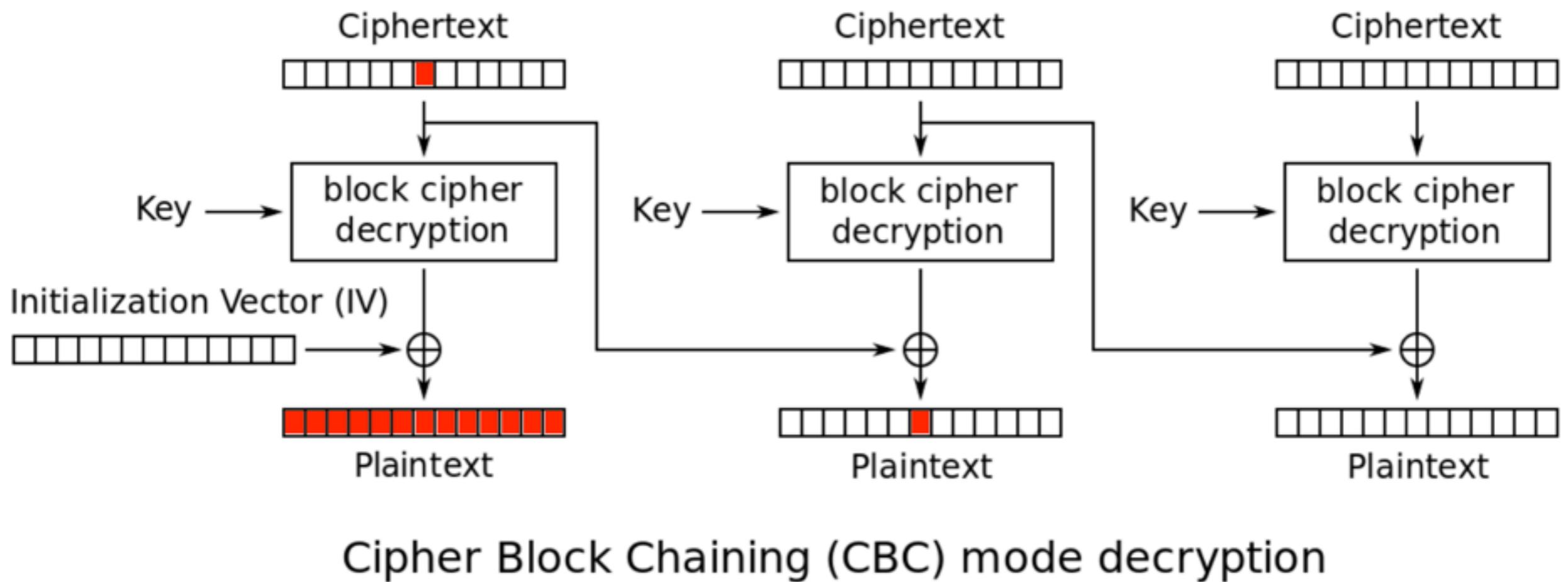
- **Token contains uid and other fields**

```
rnd=191432758301;app=eBankProdTC;uid=216;time=6343303;
```

- **Encrypted version appears random**

```
0FB1F1AFB4C874E695AAFC9AA4C2269D3E8E66BBA9B2829B173F255D447C51321586257C  
6E459A93635636F45D7B1A43163201477
```

Modify a Single Byte of Ciphertext



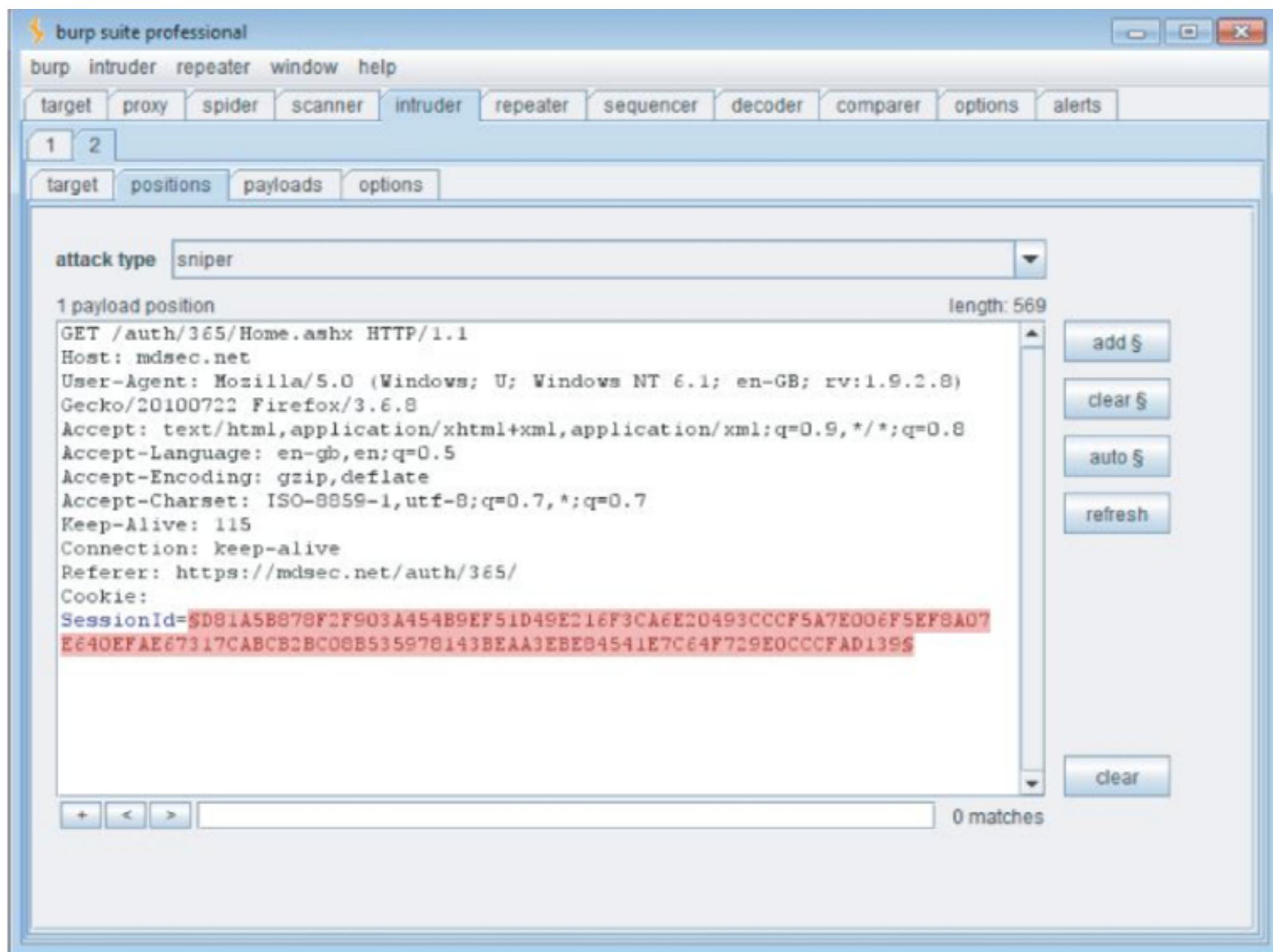
Modify a Single Byte of Ciphertext

- **That block will decrypt to junk**
- **But the next block will remain meaningful, only slightly altered by the XOR**
- **Some of the altered blocks will have valid uid values**

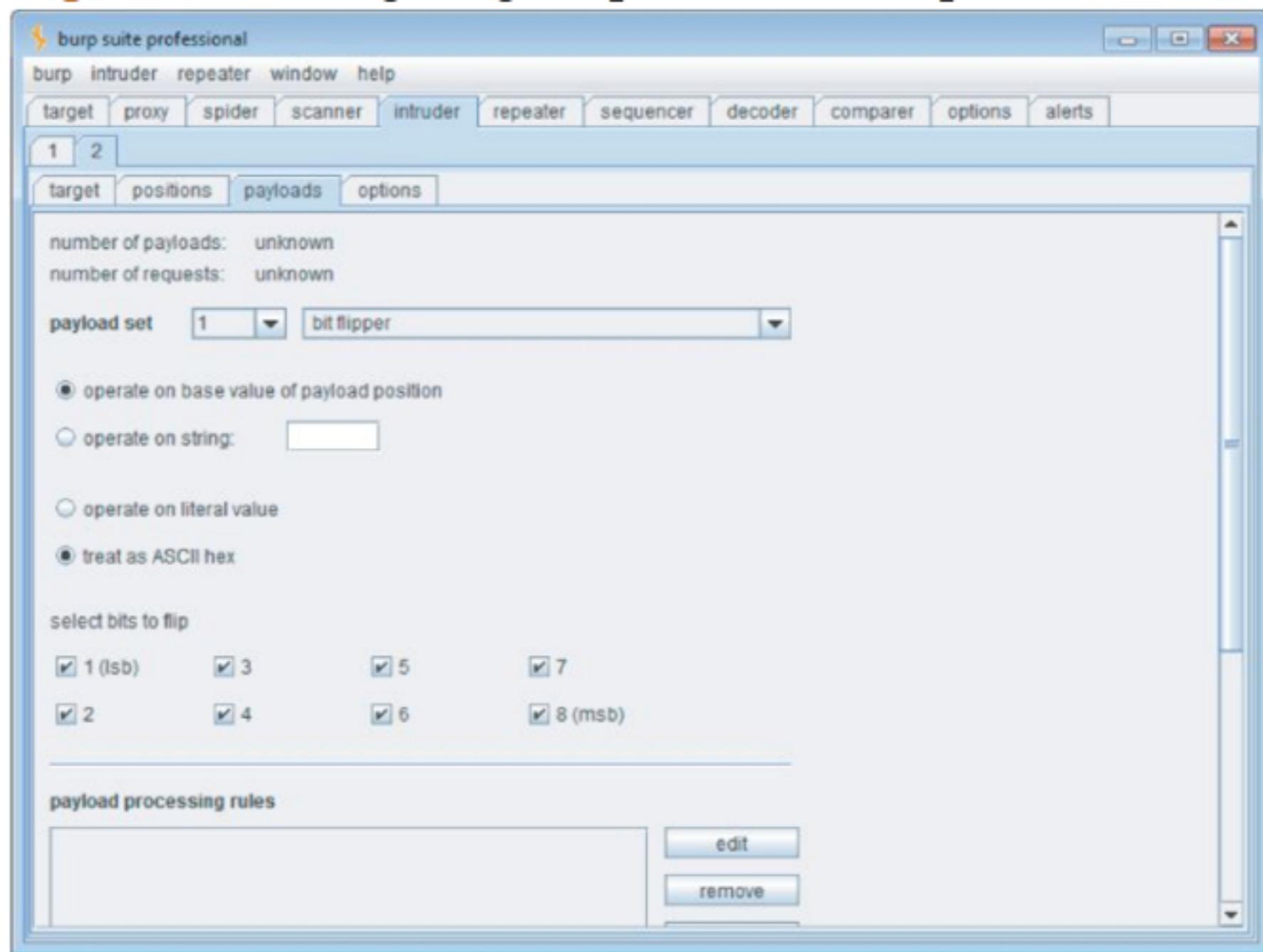
Example Altered Values

```
??????32858301;app=eBankProdTC;uid=216;time=6343303;  
??????32758321;app=eBankProdTC;uid=216;time=6343303;  
rnd=1914????????;aqp=eBankProdTC;uid=216;time=6343303;  
rnd=1914????????;app=eAankProdTC;uid=216;time=6343303;  
rnd=191432758301?????nkPqodTC;uid=216;time=6343303;  
rnd=191432758301?????nkProdUC;uid=216;time=6343303;  
rnd=191432758301;app=eBa????????;uie=216;time=6343303;  
rnd=191432758301;app=eBa????????;uid=226;time=6343303;  
rnd=191432758301;app=eBankProdTC????????;timd=6343303;  
rnd=191432758301;app=eBankProdTC????????;time=6343503;
```

Modify a Session Token



Flip Each Bit



Fast Method

- **8 requests per byte**
- **Won't take long to try all single bit flips**
- **Will confirm whether application is vulnerable**

Results

- **Some flips change user id to "invalid"**
- **Others reach real accounts for other users!**

The screenshot shows the 'intruder attack 2' interface. At the top, there's a menu bar with 'attack', 'save', and 'columns'. Below it is a filter bar set to 'Filter: hiding 3xx responses'. The main area has tabs for 'results', 'target', 'positions', 'payloads', and 'options', with 'results' currently selected. A table lists 221 rows of data, each containing a 'request' ID, a 'payload' (hex value), 'status', 'error', 'timeo.', 'length', 'Logged in as:', and a 'comment'. The 204th row is highlighted in blue and shows a comment 'Peter Weiner.'. Below the table, there's another tabbed interface with 'request' and 'response' tabs, and sub-tabs for 'raw', 'headers', 'hex', 'html', and 'render'. The 'html' tab displays an HTML page source with a red highlight on the text 'Logged in as: Peter Weiner.'.

| request | payload | status | error | timeo. | length | Logged in as: | comment |
|---------|--------------------|--------|-------|--------|--------|---------------|---------|
| 164 | D81A5B878F2F903... | 200 | | | 1303 | Daf. | |
| 165 | D81A5B878F2F903... | 200 | | | 1303 | Daf. | |
| 166 | D81A5B878F2F903... | 200 | | | 1303 | Daf. | |
| 167 | D81A5B878F2F903... | 200 | | | 1303 | Daf. | |
| 168 | D81A5B878F2F903... | 200 | | | 1303 | Daf. | |
| 197 | D81A5B878F2F903... | 200 | | | 1313 | unknown user. | |
| 198 | D81A5B878F2F903... | 200 | | | 1313 | unknown user. | |
| 199 | D81A5B878F2F903... | 200 | | | 1313 | unknown user. | |
| 204 | D81A5B878F2F903... | 200 | | | 1313 | Peter Weiner. | |
| 205 | D81A5B878F2F903... | 200 | | | 1312 | John Herman. | |
| 206 | D81A5B878F2F903... | 200 | | | 1313 | unknown user. | |
| 207 | D81A5B878F2F903... | 200 | | | 1313 | unknown user. | |
| 218 | D81A5B878F2F903... | 200 | | | 1303 | Daf. | |
| 219 | D81A5B878F2F903... | 200 | | | 1303 | Daf. | |
| 220 | D81A5B878F2F903... | 200 | | | 1303 | Daf. | |
| 221 | D81A5B878F2F903... | 200 | | | 1303 | Daf. | |

Information Leakage

- Application may re-use the encryption code elsewhere
- It may allow user to submit arbitrary input and see the ciphertext
- Such as to create a download link for an uploaded file
- Submit desired plaintext as a filename, such as
 - uid=1

Weaknesses in Session Token Handling

Common Myths

- **"SSL protects tokens in transmission"**
 - Some attacks still work, such as XSS
 - "Our platform uses mature, sound cryptography so the token is not vulnerable"
 - But cookie may be stolen in transit

Disclosure on the Network

- **Tokens transmitted without encryption**
- **Can be sniffed from many locations**
 - **User's local network**
 - **Within ISP**
 - **Within IT department of server hosting the application**

Credential Theft v. Token Theft

- Stealing a user's password may not work
 - Two-factor authentication, requiring a PIN in addition to the password
 - Login performed over HTTPS
- Stealing a session token and hijacking an authenticated session may still work
 - And the user won't be notified of a successful login

Not Always HTTPS

- An app may use HTTPS during login
- But use HTTP after login to see authorized content
- Gmail did this until recently
- Eavesdropper cannot steal password, but can steal session token

Upgradeable Token

- **App may use HTTP for preauthenticated pages**
 - Such as the site's front page
- **And use HTTPS for login and all subsequent pages**
 - But continue to use the same token; upgrade to an authenticated session
- **Attacker can steal the token before login**

Back Button

- **App uses HTTPS for login and all subsequent pages**
- **With a new token**
- **But user navigates back to an HTTP page with the Back button**
- **Exposing the token**

sslstrip

- **"Log In" link goes to an HTTPS page**
- **Attacker in the middle alters the page to use HTTP instead**
- **And forwards requests to the server via HTTPS**
- **User won't see any obvious difference in the page**

Mixed Content

Starting with [Firefox 23](#), Firefox blocks [active mixed content](#) by default. This follows a practice adopted by Internet Explorer ([since version 9](#)) and [Chrome](#).

- **HTTPS page with some HTTP content**
 - Such as images, style sheets, etc.
- **This can send the session token over HTTP**
- **Browsers now block some mixed-content by default**

Social Engineering

- App uses HTTPS for every page
- Send user an HTTP link via email or IM, or added to some page the user views
 - To `http://target.com` or `http://target.com:443`
- Clicking that link may send a session token over HTTP

Hack Steps

- **Walk through the app, from start page, through login, and all its functionality**
- **Record every URL and note every session token you receive**
- **Note transitions between HTTP and HTTPS**

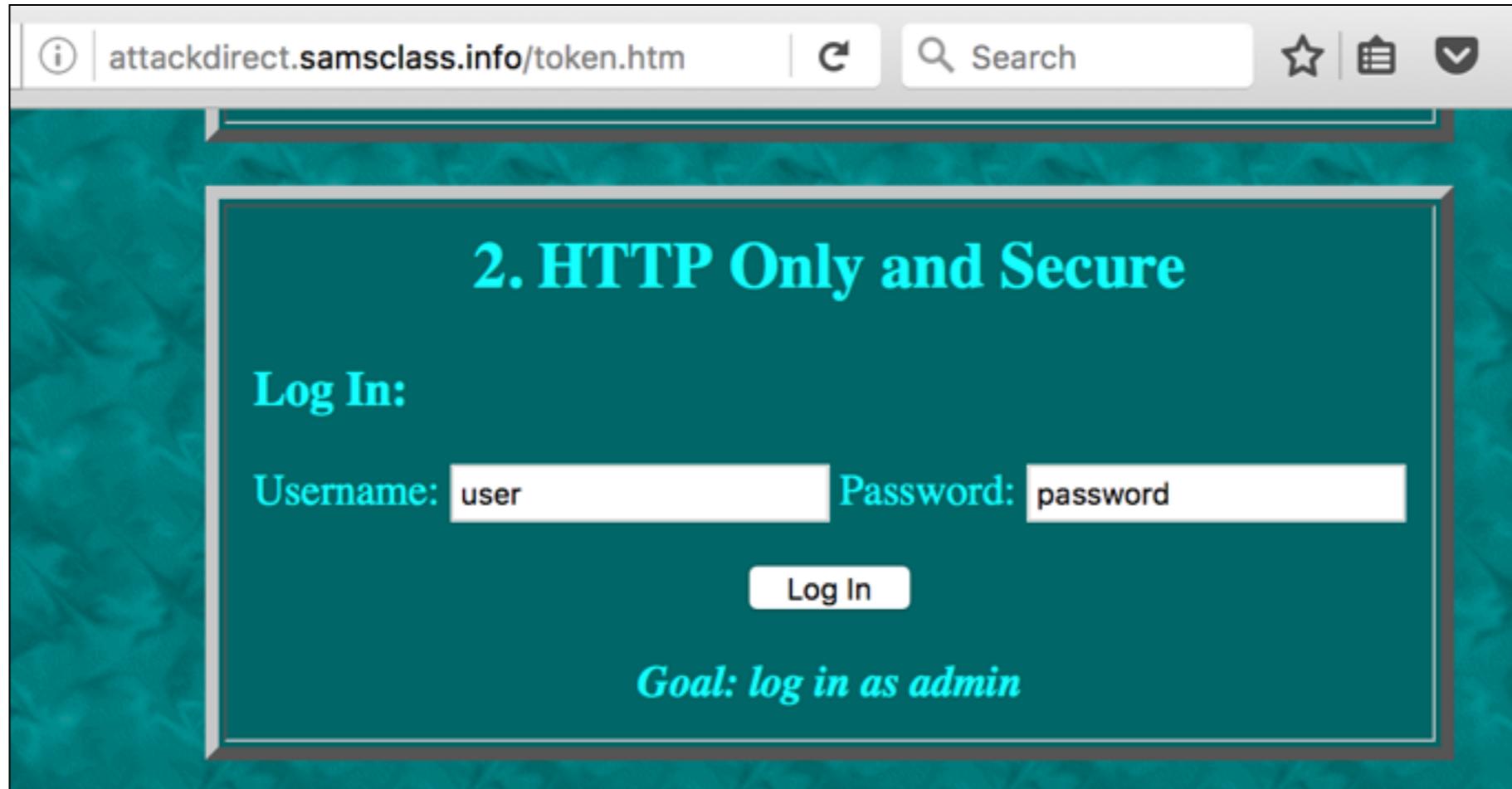
Burp's HTTP History

The screenshot shows the Burp Suite Free Edition interface. The title bar reads "Burp Suite Free Edition v1.7.03 - Temporar". The menu bar includes "Burp", "Intruder", "Repeater", "Window", and "Help". The toolbar below the menu has buttons for "Target", "Proxy", "Spider", "Scanner", "Intruder", "Repeater", "Sequencer", "Decoder", and "Comparer". The "HTTP history" button is highlighted. Below the toolbar is a sub-menu with "Intercept", "HTTP history", "WebSockets history", and "Options". A filter bar at the top of the main content area says "Filter: Hiding CSS, image and general binary content". The main table displays the following data:

| # | Host | Method | URL | Cookies |
|---|------------------------------|--------|---------------------|---------------|
| 1 | http://attack.samsclass.info | GET | /token.htm | |
| 3 | http://attack.samsclass.info | GET | /favicon.ico | |
| 4 | http://attack.samsclass.info | POST | /token1.php | username=user |
| 5 | http://attack.samsclass.info | GET | /token1-welcome.php | |
| 6 | http://attack.samsclass.info | POST | /token1.php | username=user |
| 7 | http://attack.samsclass.info | GET | /token1-welcome.php | |

Secure Cookies

- **If secure flag is set on a cookie, browser will only send it via HTTPS**
- **If the connection is only HTTP, that cookie won't be sent**



```
<?php
$p = strip_tags($_POST['password']);
$u = strip_tags($_POST['username']);
$date_of_expiry = time() + 3000 ;

if ( ($u == "user") && ($p == "password") ) {
    setcookie( "username", $u, $date_of_expiry, '', '', true, true );
    header ("Location: token2-welcome.php");
}
```

setcookie

Change language: English

[Edit](#) [Report a Bug](#)

(PHP 4, PHP 5, PHP 7)

setcookie — Send a cookie

Description

```
bool setcookie ( string $name [, string $value = "" [, int $expire = 0 [, string $path =
"" [, string $domain = "" [, bool $secure = false [, bool $httponly = false ]]]]]] )
```

secure

Indicates that the cookie should only be transmitted over a secure HTTPS connection from the client. When set to **TRUE**, the cookie will only be set if a secure connection exists. On the server-side, it's on the programmer to send this kind of cookie only on secure connection (e.g. with respect to `\$_SERVER\["HTTPS"\]`).

Setting Secure Cookie

The screenshot shows a web proxy interface with two requests listed:

- Request 40: POST /token2.php with parameter username=user (checkbox checked).
- Request 41: GET /token2-welcome.php (checkbox unchecked).

Below the requests, there are tabs for Request, Response, Raw, Headers, Hex, HTML, and Render. The Response tab is selected, displaying the following headers and content:

| Name | Value |
|----------------|--|
| HTTP/1.1 | 302 Found |
| Date | Mon, 10 Oct 2016 21:28:07 GMT |
| Server | Apache/2.4.7 (Ubuntu) |
| X-Powered-By | PHP/5.5.9-1ubuntu4.14 |
| Set-Cookie | username=user; expires=Mon, 10-Oct-2016 22:18:07 GMT; Max-Age=3000; secure; httponly |
| Location | token2>Welcome.php |
| Content-Length | 137 |
| Connection | close |
| Content-Type | text/html |

Transmitting Secure Cookie

The screenshot shows a browser developer tools Network tab. The request details are as follows:

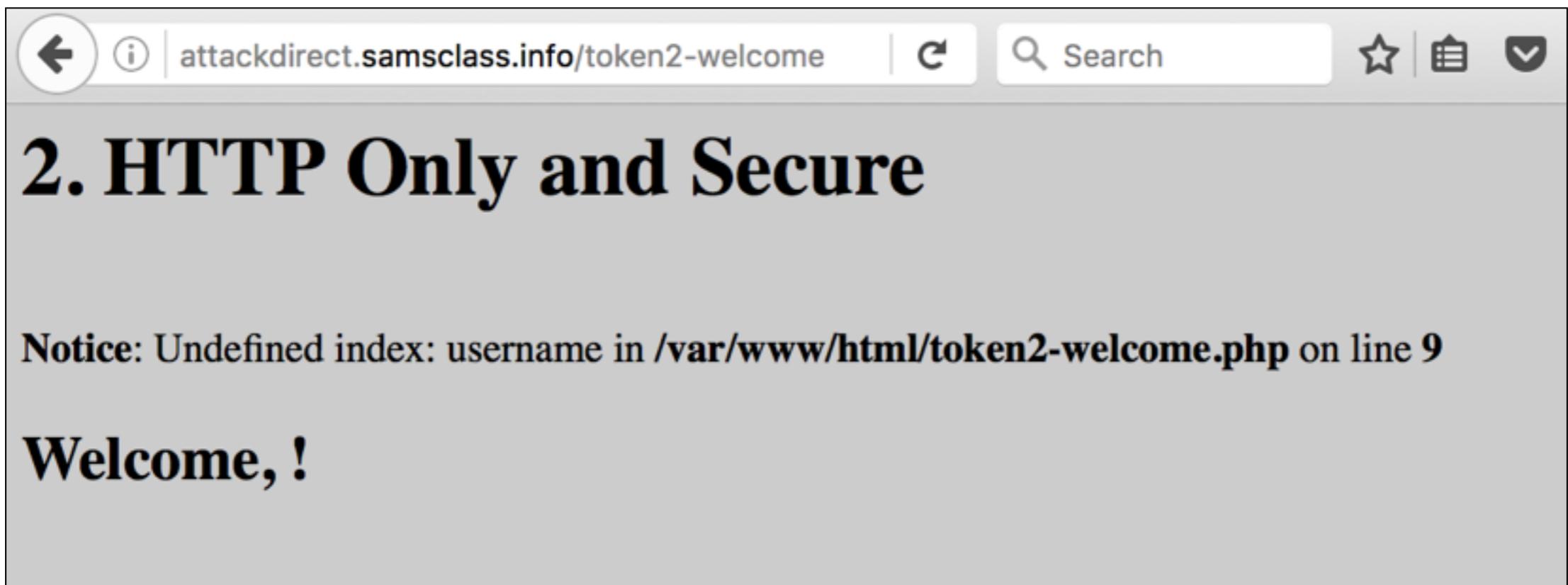
- Request ID: 41
- URL: <http://attackdirect.samsclass.info>
- Method: GET
- Path: /token2-welcome.php

The Request tab is selected. Below it, the Params tab shows a single parameter:

| Type | Name | Value |
|--------|----------|---|
| Cookie | __cfduid | d9c56d090ba7a8cde02df2adcce0fb34f1453389371 |

- **http session, so "username" is NOT sent**

Welcome Page Can't See Username



Try a Secure Connection

- Go to **<https://attack.samsclass.info/token.htm>**
- Add an exception to allow Burp to proxy traffic

Burp Suite Free Edition v1.7.03 - Temporary Pro

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extend

Intercept HTTP history WebSockets history Options

Filter: Hiding CSS, image and general binary content

| # | Host | Method | URL | Cookies |
|----|-------------------------------|--------|---------------------|---------------|
| 49 | https://attack.samsclass.info | POST | /token2.php | username=user |
| 50 | https://attack.samsclass.info | GET | /token2>Welcome.php | |

Request Response

Raw Params Headers Hex

GET request to /token2>Welcome.php

| Type | Name | Value |
|--------|----------|---|
| Cookie | __cfduid | d9c56d090ba7a8cde02df2adcce0fb34f1453389371 |
| Cookie | username | user |

- **Cookie sent**
- **Welcome page knows my name**



httponly

- **httponly cookie cannot be used by JavaScript**

The screenshot shows a web browser window with two stacked login forms, each with a teal header and a white input field.

1. Not Encoded

Log In:

Username: Password:

Goal: log in as admin

`alert(document.cookie)`

2. HTTP Only and Secure

Log In:

Username: Password:

Goal: log in as admin

`alert(document.cookie)`

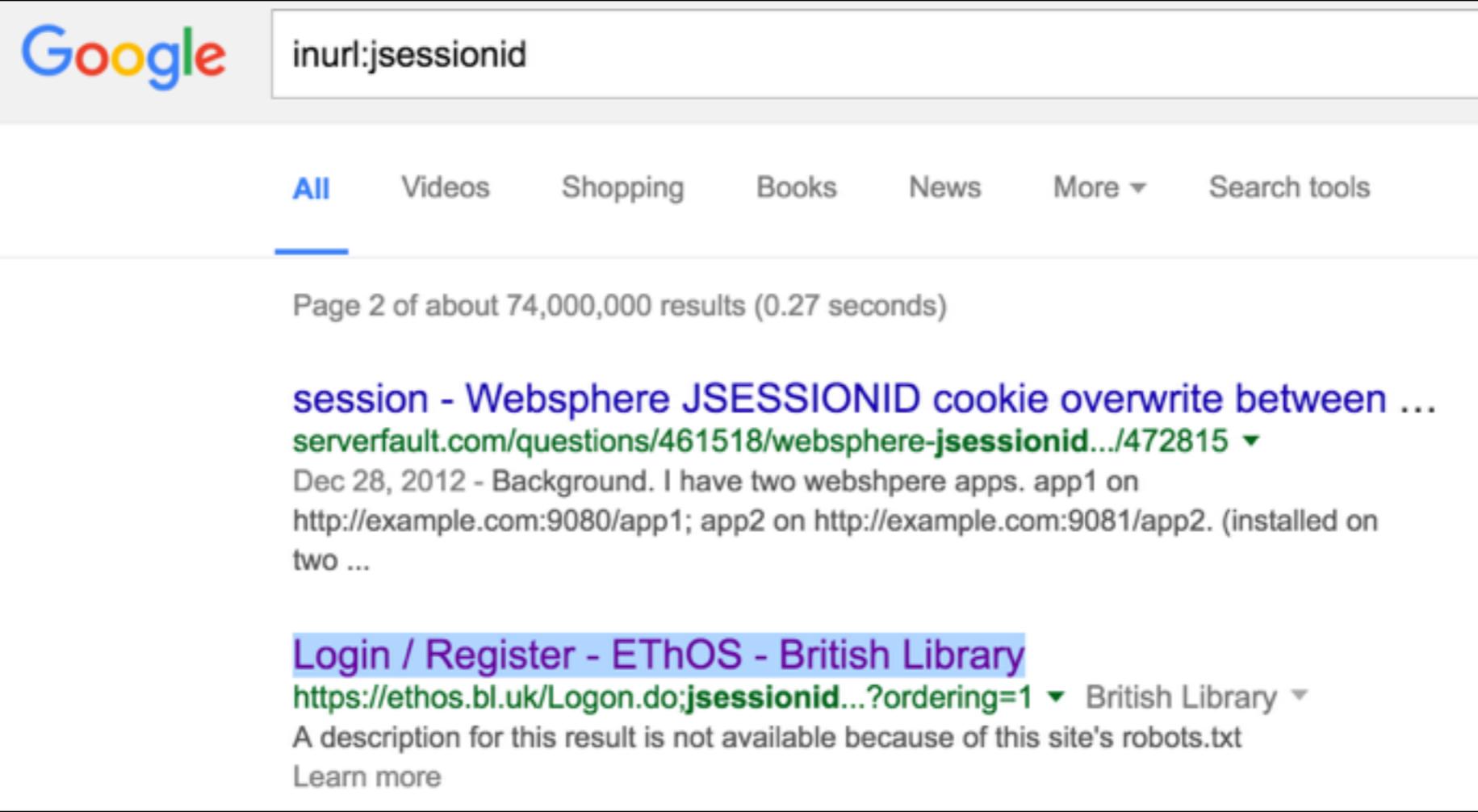
`alert(1)`

The browser's address bar shows the URL `attackdirect.samsclass.info/token.htm`. The top right corner of the browser window includes standard icons for search, refresh, and navigation.

Disclosure of Tokens in Logs

- **Less common as unencrypted network traffic but more serious**
- **Logs often visible to more attackers**

inurl:jsessionid



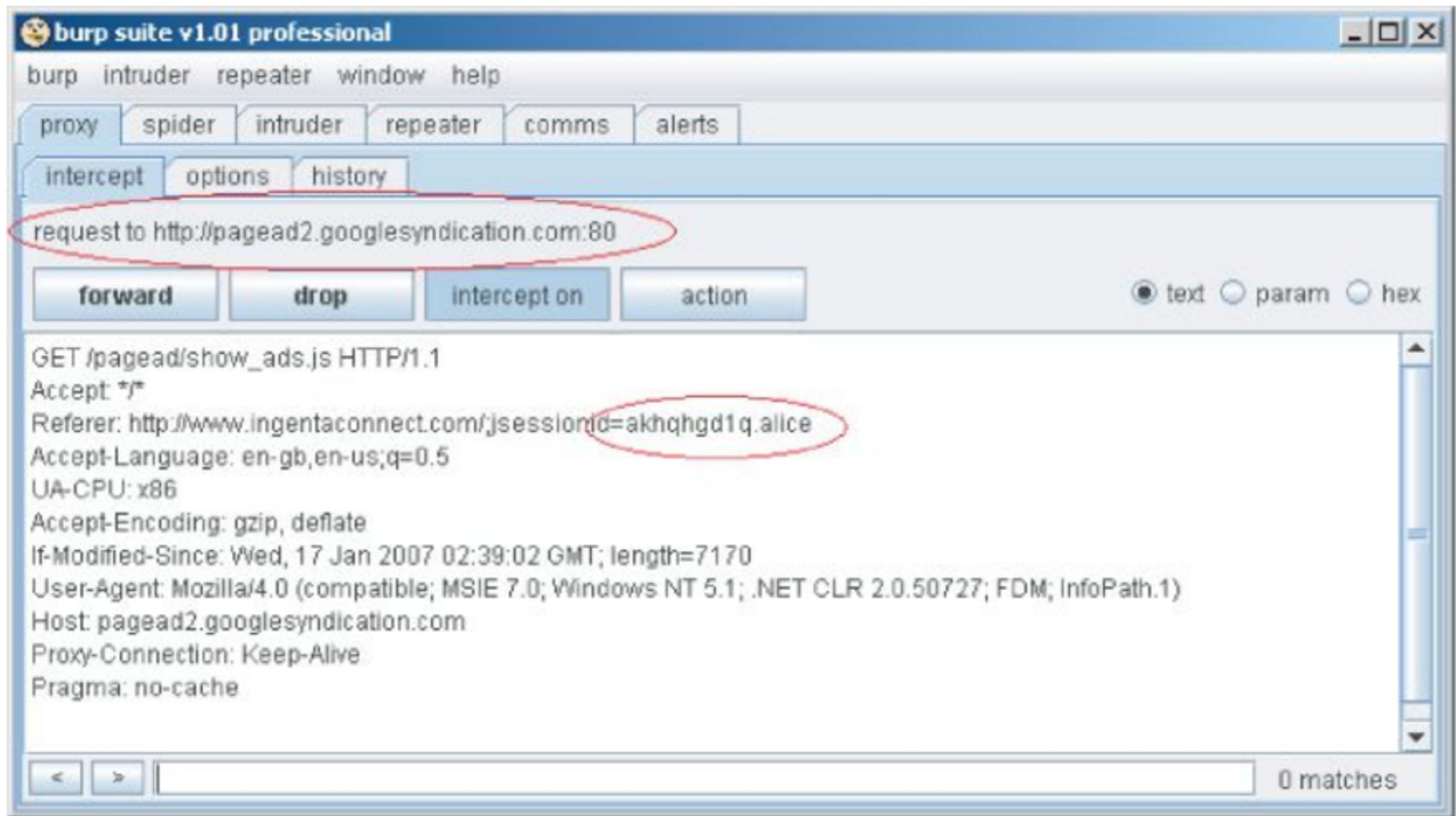
Google search results for "inurl:jsessionid". The search bar shows the query. The results page has a navigation bar with "All" selected, followed by "Videos", "Shopping", "Books", "News", "More", and "Search tools". It displays 74,000,000 results in 0.27 seconds. Two results are shown:

- session - Websphere JSESSIONID cookie overwrite between ...**
serverfault.com/questions/461518/websphere-jsessionId.../472815 ▾
Dec 28, 2012 - Background. I have two webshpere apps. app1 on http://example.com:9080/app1; app2 on http://example.com:9081/app2. (installed on two ...)
- Login / Register - EThOS - British Library**
<https://ethos.bl.uk/Logon.do;jsessionId...?ordering=1> ▾ British Library ▾
A description for this result is not available because of this site's robots.txt
Learn more

- **Example**

Where SessionIDs in URL Will Appear

- Users' browser logs
- Web server logs
- Logs of corporate or ISP proxy servers
- Logs of any reverse proxies employed within the application's hosting environment
- The Referer logs of any servers that application users visit by following off-site links, as shown in [Figure 7.11](#)



Referer Attack

- A web mail application transmits session tokens in the URL
- Send email to targets containing a URL on the attacker's Web server
- The Referer headers from people who click will appear in the server logs

Vulnerable Mapping of Tokens to Sessions

- **Allowing users to have two sessions open at the same time**
- **Using static tokens (same token is sent to the user each time they log in)**
 - **Misunderstanding of what a session is**

Flawed Logic

- **Token value**

`dXNlcj1kYWY7cjE9MTMwOTQxODEyMTM0NTkwMTI=`

which Base64-decodes to:

`user=daf;r1=13094181213459012`

- **But app accepts the same "r1" with a different "user"**

Vulnerable Session Termination

- A session may remain valid for days after the last request is received
- Ineffective logout functionality
- Logout merely deletes cookie but does not invalidate it
 - Logout merely runs a client-side script, server doesn't know a logout has occurred

← → ⌂ ⌂ <https://samsclass.info/123/proj10/cookie-reuse.htm>

Cookie Re-Use in Office 365 and Other Web Services

Topics

- [American Express and Chase](#)
- [Background](#)
- [List of Vulnerable Sites](#)
- [ASP.NET and Cookie-Re-Use](#)
- [Instructions for Testing Sites](#)
- [Media Coverage](#)
- [Changelog](#)

Hacking into my American Express Account Without a Password

Token Hijacking

- **Cookie theft**
- **Session fixation:** attacker feeds a token to the user, then user logs in, then attacker hijacks the session
- **Cross-Site Request Forgery (CSRF)**
 - Tricks user into submitting a request containing a cookie that goes to an attacker's server

Liberal Cookie Scope

- When a cookie is set, the server can set the *domain* and *url* (path) the cookie is used for
- By default, all subdomains are included
 - Cookie set by `games.samsclass.info`
 - Will be sent to `foo.games.samsclass.info`
 - But NOT to `samsclass.info`

Specifying the Domain

- **App at foo.wahh-app.com sets this cookie:**

```
Set-cookie: sessionId=19284710; domain=wahh-app.com;
```

The browser then resubmits this cookie to all subdomains of wahh-app.com, including bar.wahh-app.com.

- **A domain can only set a cookie for the same domain or a parent domain**
 - **And not a top-level domain like .com**

Example

- **blogs.com sets a cookie for each user**
- **Each user can create blogs**
 - **joe.blogs.com**
 - **sally.blogs.com**
- **A blog with JavaScript can steal tokens of other users who read the attacker's blog**

Fix

- **There is no way to prevent cookies for an application from being sent to subdomains**
- **Solution: use a different domain name for main app, and scope the domain to this fully qualified name**
 - **www.blogs.com**
 - **Cookie won't be sent to joe.blogs.com**

Path

- **Application returns this HTTP header:**

```
Set-cookie: sessionId=187ab023e09c00a881a; path=/apps/;
```

the browser resubmits this cookie to all subdirectories of the /apps/ path.

- **Easily defeated by getting a handle to a JavaScript window (Link Ch 7f)**

Securing Session Management

Securing Session Management

- **Generate Strong Tokens**
- **Protect them throughout life cycle, from creation to disposal**

Strong Tokens

The most effective token generation mechanisms are those that:

- Use an extremely large set of possible values
- Contain a strong source of pseudorandomness, ensuring an even and unpredictable spread of tokens across the range of possible values

Strong Tokens

- **Tokens should contain no meaning or structure**
- **All data about the session's owner and status should be stored on the server in a session object**
- **Some random functions, like `java.util.Random`, are predictable from a single value**

Sources of Entropy (Randomness)

- The source IP address and port number from which the request was received
 - The User-Agent header in the request
 - The time of the request in milliseconds
- **Good method:**
 - **Add a secret known only to the server, then hashing it all**
 - **Change the secret on each reboot**

Protecting Tokens Throughout Their Life Cycle

- **Only transmit over HTTPS**
 - **secure, httponly**
 - **Use HTTPS for every page in application**
- **Don't put session tokens in the URL**
- **Implement a logout function that invalidates session token on the server**

Protecting Tokens Throughout Their Life Cycle

- Session should expire after a brief period of inactivity, such as 10 minutes
- Don't allow concurrent logins
 - If a user starts a new session, a new token should be generated, and the old one invalidated
- Protect diagnostic or administrative functions that save tokens
 - Or don't save tokens in them

Protecting Tokens Throughout Their Life Cycle

- **Restrict domain and path for session cookies**
- **Audit codebase and remove XSS vulnerabilities**
- **Don't accept tokens submitted by unrecognized users**
 - **Cancel tokens after such a request**

Protecting Tokens Throughout Their Life Cycle

- **Use two-factor authentication**
 - Makes cross-site request forgery and other session-hijacking methods more difficult
- **Use hidden fields rather than session cookies**
 - More difficult to steal because they aren't sent in every request

Protecting Tokens Throughout Their Life Cycle

- **Create a fresh session after every authentication**
 - **To prevent session fixation attacks**

Per-Page Tokens

- **A new page token is created every time the user requests an application page**
- **Page token verified on every request**
 - **If it doesn't match, the session is terminated**
- **Prevents pages being used out of order**
- **And blocks an attacker from using a page at the same time as a real user**

Log, Monitor, and Alert

- **Requests with invalid tokens should raise IDS alerts**
- **Alert users of incidents relating to their sessions**

Reactive Session Termination

- **Terminate session if a request has**
 - **Modified hidden form field or URL query string parameter**
 - **Strings associated with SQL injection or XSS**
 - **Input that violates validation checks, such as length restrictions**