

Techniken

Wie man einen versteckten Kanal kocht

Simon Castro und Gray World Team



Schwierigkeitsgrad



Bevor Sie mit dem Kochen Ihres versteckten Kanals anfangen, müssen Sie erst über das Rezept nachdenken: Entscheiden Sie, wie Ihr versteckter Kanal aussehen wird, wofür er benutzt wird (Antipasti oder Dessert?) und schließlich wann Sie Ihr Mahl nehmen werden. Das heutige Menü konzentriert sich auf HTTP Cookies, also lassen Sie uns das Rezept erneut anschauen und anfangen zu kochen.

Wir alle kennen HTTP und Cookies. Wenn Sie jemals etwas über das Internet gekauft haben (oder jemand das für Sie erledigt hat), haben Sie möglicherweise Cookies verwendet, um eine logische Sitzung mit dem Remoteserver aufrecht zu erhalten. Wie also könnte es möglich sein, Cookies als versteckten Kommunikationskanal zu verwenden?

Die Cookie Theorie

Lassen Sie uns das [RFC_2109]-Dokument durchsehen, welches verschiedene interessante Punkte beschreibt, die die Erstellung von logischen Sitzungen betreffen.

Es sollte verstanden werden, dass die Sitzung durch Server (wir werden danach *Server* verwenden, um vom HTTP-Server zu sprechen und *Client*, um vom HTTP-Client zu sprechen) oder durch den Client beendet werden kann und das Sitzungen nicht zu lange dauern sollten (1).

Beachten Sie, dass ein Client Cookie(s) mit jeder Anfrage an den Server senden *sollte* und dass der Server ein Cookie an einen Client senden kann, sogar, wenn der Client nicht danach gefragt hat (2 und 3). Beachten Sie auch, dass der Cookiewert für den Client *undurchsichtig* ist und somit auch *undurchsichtig* für einen Host ist, der die Sitzung überwachen will (4).

Schließlich nehmen wir an, dass es nichts verdächtiges ist, die Caching-Dienste aufzufordern, die Cookies, die der Client und der Server gesendet haben, nicht im Cache zu speichern, wenn der Cookie für die Benutzung eines Einzelbenutzers gedacht ist. (5)

Beachten Sie, dass das Lesen von (5) auf andere Art bedeutet, dass wir eine Gelegenheit haben, diese Caching-Dienste als Relais der zweiten Ebene zu verwenden, um Daten zu

In diesem Artikel erfahren Sie...

- Wie man einen versteckten Kontroll-Kommunikationskanal vorbereitet.

Was Sie vorher wissen/können sollten...

- Das HTTP Protokoll;
- Sie sollten Grundkenntnisse über die Programmiersprache Python haben.

speichern und an mehrere Clients weiterzuleiten, mit oder ohne Server. Wir wissen bereits, dass dies für jede HTTP-Einheit möglich ist, aber vielleicht wird es für Cookies auch möglich sein.

Listing 1. *Beobachtung gewöhnlicher Cookies*

```
Unser Cookie ist: 582c76b3d761f5741774f9786603e2438853b8b0
und ohne Auffüllung: 582c76b3d761f5741774f97866

Andere sind (eins pro Zeile):
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=
```

RFC 2109

- (1) [...] Das Paradigma des Designers für Sitzungen, die durch den Austausch von Cookies erstellt werden, hat diese Schlüsselattribute: Jede Sitzung hat einen Beginn und ein Ende; Jede Sitzung ist relativ kurzlebig; entweder der Benutzervermittler oder der Ursprungs-Server können eine Sitzung beenden.
- (2) Um eine Sitzung zu initiieren, gibt der Ursprungsserver einen besonderen Antwort-Header an den Client zurück. [...] Ein Benutzervermittler gibt einen Cookie-Anfrage-Header [...] an den Ursprungsserver zurück, wenn er enstcheidet, eine Sitzung fortzuführen. Der Ursprungsserver kann es ignorieren oder es verwenden, um den aktuellen Status der Sitzung zu ermitteln. Er kann an den Client einen Set-Cookie Antwortheader mit der selben oder unterschiedlicher Information zurücksenden, oder er kann überhaupt keinen Set-Cookie-Header senden.
- (3) Server können einen Set-Cookie Antwortheader mit jeder Antwort senden. Benutzervermittler sollten Cookie-Anfrageheader, Thema anderer Regeln, die unten ausführlich erläutert werden, mit jeder Anfrage senden. Ein Ursprungsserver kann mehrere Set-Cookie-Header in einer Antwort einfügen.
- (4) Set-Cookie Syntax: [...] `cookie = NAME "=" VALUE *(";" cookie-av)` [...] `NAME=VALUE Required`. Der Name der Statusinformation (*cookie*) ist `NAME`, und sein Wert ist `VALUE`. [...] Der `VALUE` ist für den Benutzervermittler undurchsichtig und kann irgendetwas sein, was der Ursprungsserver zu senden entscheidet, möglicherweise ein von Server ausgewähltes, druckbares ASCII Encoding. *Undurchsichtig* impliziert, dass der Inhalt nur für den Ursprungsserver interessant und relevant ist. Der Inhalt kann vielmehr für jeden lesbar sein, der den Set-Cookie-Header betrachtet.
- (5) Ein Ursprungsserver muss für den Effekt des Caching von sowohl der zurückgegebenen Quelle als auch des Set-Cookie-Headers zuständig sein. [...] Wenn der Cookie für die Benutzung durch einen einzelnen Benutzer gedacht ist, sollte der Set-Cookie-Header nicht in den Cache. Ein Set-Cookie-Header, der dazu gedacht ist, von mehreren Benutzern geteilt zu werden, kann in den Cache.

Über das Rezept nachdenken

Ein versteckter Kanal ist ein Kommunikationskanal, der nicht entworfen wurde und/auch nicht als existierend gedacht ist und der verwendet werden kann, Informationen auf eine Art zu übermitteln, die die bestehende Sicherheitsrichtlinie verletzt. [...] Verschiedene Parameter existieren, um versteckte Kanäle zu charakterisieren: Rauschen, Bandbreite/Kapazität, Synchronisation und Aggregation [...], Latenz und Verstohlenheit [CC].

Das Rezept des Tages wird sich darauf konzentrieren, Schritt für Schritt einen *neuen* Kommunikationskanal vorzubereiten (wenden Sie sich an [CC] wegen des Unterschieds zwischen Kontroll- und Datenkommunikationkanälen), der so verdeckt wie möglich sein wird.

Da wir einen versteckten Kommunikationskanal kochen, betrachten wir die Bandbreiten-/Kapazität- und Latenzparameter nicht als Schlüsselfaktoren.

Wir kochen einen Kommunikationskanal über das HTTP-Protokoll. Das bedeutet, dass der HTTP-Server einen Kontakt zum HTTP-Client braucht, bevor er in der Lage ist, irgendwelche Daten zu senden. Da wir uns auf einen Kontrollkommunikationskanal konzentrieren, müssen wir auch die Menge an Daten und die Frequenzausstrahlungsparameter, die der HTTP-Client benutzt, um Daten zu senden und vom HTTP-Server zu empfangen, begrenzen.

Wir werden das Problem des aktiven Wächters nicht besprechen, da es ihn dazu bringen würde, jedes Cookie, das er entdeckt, zu verändern und im Auge zu behalten (keine so gute Idee, nur Teile des Cookies zu verändern...) und schließlich werden wir annehmen, dass alles bis auf unser Cookie als Standard von einem leistungsfähigen Erkennungssystem gesehen wird (Netzwerkschichten-Faktoren und HTTP-Protokoll-Verhalten).

Listing 2. *Standardsitzung 1*

```
HTTP GET auf A.XXX
=> Antworte mit einer Dokumentspeicherstelle zu www.A.XXX mit :
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
=> Antworte mit:
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

Nun schließen wir den Browser, warten ein paar Sekunden und tun es erneut:
HTTP GET on A.XXX having :
Cookie: PREF=ID=ef6ed1bdb2a7b217:TM=11821401:LM=1221401:S=-MwFEtY3L1_Xe
=> Antworte mit einer Dokumentspeicherstelle zu www.A.XXX ohne Set-Cookie

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

Listing 3. *Standardsitzung 2*

```
HTTP GET auf B.XXX
=> => Antworte mit einer Dokumentspeicherstelle zu www.B.XXX mit
    Set-Cookie: ASPSESSIONIDATRSCS=HAEBGHTVCSXZFJLLLDIAJJMN; path=/

HTTP GET auf www.B.XXX ohne Cookie
```

Listing 4. *Den Client-Teil ablaufen lassen*

```
$ ./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')
```

Unser Beta-Rezept

Das Informationsbehälter-Modell, die der HTTP-Client und der HTTP-Server benutzen werden, ist so einfach wie:

```
Checksum: default size 2 bytes
Command : default size 1 byte
=> ist eine Anfrage oder eine Antwort
Info    : request or response
Padding : default size to 20 bytes
```

Checksum ist eine gewöhnlich berechnete Prüfsumme über die Command- und Info-Parameter. Command zeigt an, ob das Cookie eine Anfrage oder eine Antwort enthält. Auffüllen (Padding) ist etwas optionales, das ermöglicht, die Cookie-Größe zu verändern.

Lassen Sie uns anschauen, welche Art von Cookie wir bei einem wesentlichen Client-Befehl haben können, der dem Server mitteilt: *Ich bin da, hier ist meine lokale IP-Adresse, meine Startzeit und meine Kontaktverzögerung*:

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

wird ein Cookie ergeben: `'7e5801010202030 4075bcd15000a42424242424242'` .

Nun, da wir ein Cookie haben, wäre es eine gute Idee, es nicht im Klartext zu senden. Wenn wir genügend *zufällige* Bytes haben können, die wir verwenden können, um eine XOR-Verknüpfung des Cookies zu machen, können wir etwas, das ein bisschen weniger verdächtig ist, erhalten. Lassen Sie uns also annehmen, dass wir einen statischen Schlüssel haben und x *zufällige* Bytes, die Server und Client bekannt sind, können wir dann eine Extraktionsfunktion verwenden, um genug *pseudo-zufällige* Bytes zu erhalten, um unser Cookie vor dem Senden zu Server XOR zu verknüpfen. So werden wir, anstelle von `'7e58010102030 4075bcd15000a42424242424242'` so etwas wie `'582c76b3d761f5741774f9786603e2438853b8b0'` haben.

Wir können nun Cookies verwenden, um Daten zu senden und zu empfangen und wir haben einen Weg, um sie zu verändern, so dass sie *verworren* und *zufällig* aussehen. Konzentrieren wir uns auf einige Befehlsarten, die interessant zu implementieren wären:
Clientbefehle:

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

Serverbefehle:

```
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'
```

Mit diesen Befehlen können wir im Wesentlichen unseren Kontrollkommunikationskanal steuern, so dass er so lange, wie wir es brauchen, online bleibt, aber wir könnten einem weiteren Problem gegenüberstehen: Wie wissen wir, ob ein Client oder ein Server den Befehl bekommen hat, den wir gesendet haben? Lassen Sie uns einen Befehl/Bestätigung-Mechanismus, so wie den im folgenden beschriebenen, verwenden:
Clientbefehle:

```
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
```

Serverbefehle:

```
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
```

Der Hauptvorteil, eine Bestätigung für die Client-UP-Nachricht nicht zu verwenden, ist, dass der Client in der Lage sein wird, dasselbe Cookie zu senden und wieder zu senden, ohne *zufällige* Bytes zu verlieren. Wie jeder Standard-Webclient es macht.

Freunden von dem Rezept erzählen

Wir entscheiden beliebig, unser Cookie zu *hex-ifizieren*, aber Sie können einen anderen Algorithmus auswählen, um Ihr Cookie zu verschlüsseln. Lassen Sie uns unseren bevorzugten MS13 Browser starten und auf unsere Cookies achten (Listing 1):

- der Name ist gewöhnlich '`_1-9a-zA-Z`' und 1 < x < 24 Bytes lang;

Listing 5. *Verbindung zum Server*

```
$ ./cook_cgi
Wie man einen versteckten Kanal kocht - cook_cgi.py - v0.1

Bryan sagt: Größenaktualisierung auf 24 aufgestockt mit Auffüllung auf 0 für
                    Client 2. (1)
Bryan sagt: Willkommen in der Küche, wir haben 2 Clients (Mi Apr [..]
  o Entferne Clients, die seit über 3600 Sekunden leise sind
  o Verdopple nicht den Bestand desselben Befehls: 1
  o gefälschte Cookies für Standardclients: Keine
  o Brenn die Küche ab

Clientliste:

  #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'\
         / Sync verloren: 0
    Was hast du?
      Neue Kontaktperiode, neue rbytes, Ändere Cookie-Größe,\
      Deaktiviert/ Aktiviert Auffüllen, Entfern Befehle
      Gespeicherte Befehle:
        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'\
         / Sync verloren: 0
    Was hast du?
      Neue Kontaktperiode, neue rbytes, Ändere Cookie-Größe,\
      Deaktiviert/ Aktiviert Auffüllen, Entfern Befehle
      Gespeicherte Befehle:
$ _
```

Listing 6. *Senden der gekochten Befehle an den Client*

```
(1) 19:54:27 - Sende Cookie an 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 - Befehl Aktualisiere Kontaktzeit
(4) 19:54:27 - Aktualisiere Kontaktperiode auf 5 Sek
(5) 19:54:27 - 24 bytes cookie (6/16) erhalten:\
               '\xe8\xab\x03\x00\x18\x00BBBBBBBBBBBBBBBBBB'
(6) 19:54:27 - Befehl Aktualisiere Größe
(7) 19:54:27 - Aktualisiere Cookie-Größe auf 24 (Auffüllungsaktivierung: 0)
(8) 19:54:27 - Sende Cookie an ip:80/cgi-bin/cook_cgi (7/7):\
               22984fcc75fc01b0af217350eb
(9) 19:54:27 - Sende Cookie an ip:80/cgi-bin/cook_cgi (8/7):\
               14a4087e1e5cf3d5724b522fe6
(10) 19:54:32 - Sende Cookie an ip:80/cgi-bin/cook_cgi (9/7):\
               943d58cb1fd5864a98a1a47067
(11) 19:54:38 - Sende Cookie an ip:80/cgi-bin/cook_cgi (9/7):\
               943d58cb1fd5864a98a1a47067
```

- die Domain ist zu 50% *fqdn* und zu 50% *.fqdn*;
- der Pfad ist zu 90% '/' (ist er?)
- das Ablaufdatum ist gewöhnlich zwischen diesem Jahr+1 und 2016 oder 2038 (?);
- der Inhalt ist manchmal raw ASCII, aber oft gilt Schlüssel = Wert (Wert = Raw ASCII).

Nun ist unser nächster Schritt zu ermitteln, wie das Verhalten unserer Freunde ist, wenn sie einem Cookie begegnen, so dass wir wissen, wann und wie wir Daten senden und empfangen können. Unten stehend sind Sitzungen von bekannten, verdeckten Webseiten beschrieben.

Wir schließen daraus, dass unser gekochter Client Cookies an den Server senden kann, sogar wenn der Server kein Set-Cookie (bis 2038?) gesendet hat, da der Server dieses Cookie vor 32 Jahren gesenden hat?

Wir folgern, dass wir ein paar (nur) praktische (nicht nur theorethisch geschrieben im *rfc*) Lösungen für den Server haben, um ein Cookie zu senden, ohne dass der Client mit dem Cookie antworten muss:

- wir führen Set-Cookie mit einer Domain aus, die sich von der in der HTTP URI unterscheidet => [Standardsitzung 1];
- wir führen Set-Cookie aus, ohne die Domain anzugeben => [Standardsession 2].

Es scheint, dass unser Beta-Rezept recht interessant aussieht, lassen Sie uns mit den Kochen anfangen.

Rezept

Nun da wir in etwa wissen, was wir kochen werden, müssen wir entscheiden, welche Art von Bryan (der immer in der Küche ist, wie wir alle wissen) uns helfen wird, ein schnelles Essen für unsere (möglicherweise) neuen Freunde zu kochen.

Wir haben uns entschieden, einen PYTHON Bryan zu verwenden, so dass Sie und Ihre *Freunde* die Mahlzeit probieren können, egal ob Sie eine Win32- oder eine **Nix*-Küche haben. Allerdings kann es sein,

Lisitng 7. *Client akzeptierte Befehle*

```
$ ./cook_cgi
Wie man einen versteckten Kanal kocht - cook_cgi.py - v0.1

Bryan sagt: Größenaktualisierung auf 24 aufgestockt mit Auffüllung auf 0 für
                    Client 2. (1)

Bryan sagt: Willkommen in der Küche, wir haben 2 Clients (Mi Apr [..]
  o Entferne Clients, die seit über 3600 Sekunden leise sind
  o Verdopple nicht den Bestand desselben Befehls: 1
  o gefälschte Cookies für Standardclients: Keine
  o Brenn die Küche ab

Clientliste:

  #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' / Sync verloren: 0
    Was hast du?
      Neue Kontaktperiode, neue rbytes, Ändere Cookie-Größe,\
      Deaktiviert/ Aktiviert Aufüllen, Entfern Befehle
      Gespeicherte Befehle:

  #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'\
         / Sync verloren: 0
    Was hast du?
      Neue Kontaktperiode, neue rbytes, Ändere Cookie-Größe,\
      Deaktiviert/ Aktiviert Aufüllen, Entfern Befehle
      Gespeicherte Befehle:

$ _
```

Listing 8. *Glücksspiel für den Client #1*

```
# ./cook_cl.py -d ip -s 10 -v
(1) : 20:02:59 - Sende Cookie an ip:80/cgi-bin/cook_cgi (2/16):\
               96a152e6aeeb52b5726263b02d16bb5095b27c9a28586498

      20:03:09 - Sende Cookie an ip:80/cgi-bin/cook_cgi (2/16):\
               96a152e6aeeb52b5726263b02d16bb5095b27c9a28586498

      20:03:09 - 24 Bytes erhalten (4/16): '5\x80\x02\x00\
                     x10priatnovoapetitaBBB'

      20:03:09 - Befehl Aktualisiere Rbytes

(2) : 20:03:09 - Aktualisiere rbytes mit 'priatnovoapetita'

      20:03:09 - Sende Cookie an ip:80/cgi-bin/cook_cgi (6/16):\
               4df16f06ec172a8b8a1bbca7ed2d154944584b2a5b2a31f0

      20:03:19 - Sende Cookie an ip:80/cgi-bin/cook_cgi (8/16):\
               7f8db0cc75fc0eb0b1cd3f50e4e3fce0ec63cbaaf742b636
```

Lisitng 9. *Glücksspiel für den Client #2*

```
# ./cook_cl.py -d ip -s 10 -v
(1) : 20:07:33 - Sende Cookie an ip:80/cgi-bin/cook_cgi (2/16):\
               d55d52e6aeeb5db5726577b02d16bb5095b27c9a28586498
      20:07:43 - Sende Cookie an ip:80/cgi-bin/cook_cgi (2/16):\
               d55d52e6aeeb5db5726577b02d16bb5095b27c9a28586498
      20:07:43 - 24 Bytes erhalten (4/16): 'Q\x08\x02\x00\
                     ndozvidaniaBBBBBBBBB'
      20:07:43 - Befehl Aktualisiere Rbytes
(2) : 20:07:43 - Aktualisiere rbytes mit 'dozvidania'
      20:07:43 - Sende Cookie an ip:80/cgi-bin/cook_cgi (6/16):\
               0e0d6f06ec17258b8a1ca8a7ed2d154944584b2a5b2a31f0
      20:07:53 - Sende Cookie an ip:80/cgi-bin/cook_cgi (8/16):\
               3c71b0cc75fc01b0b1ca2b50e4e3fce0ec63cbaaf742b636
```

dass Sie, wenn Sie dieses Rezept lesen, möglichweise ein anderes Gericht probieren möchten, dass in einer Win32 C/C++-Küche gekocht wurde und von dem niemand vorher gehört hat, da es immer besser ist, niemandem etwas zu verraten, wenn Sie eine Überraschung vorbereiten.

So ist unser Mahl aus zwei Zutaten aufgebaut: Der Client-Teil, der eine unabhängige Python-Anwendung ist und der Server-Teil, der ein CGI-Script ist, das Sie auf einen Webserver heraufladen müssen.

Der Client

Der Cleint verbindet sich mit dem Webserver und sendet eine GET-Anfrage zusammen mit einem Cookie, in dem der *I am up* (*Ich bin hochgefahren*)-Befehl eingebettet ist. Wenn die Serverantwort einen Cookie enthält, entschlüsselt der Client den Cookie und sendet die zugehörige Bestätigung zurück. Wenn der Server nicht auf ein Client-Cookie antwortet, schläft der Client für x Sekunden.

Da der Server mit mehreren Cookies in einer einzigen Antwort antworten kann, analysiert der Client alle Cookie-Befehle, bevor er die zugehörige Bestätigung senden (so dass der Server und der Client die Synchronisation für die Zufallsbytes einhalten).

Der Client sendet seine HTTP-Anfrage mit einem MS13 oder Firefox-Verhalten: Beide Browser verhalten sich auf die gleiche Weise auf der TCP-Ebene für unser CGI (TCP HandShake, HTTP GET, HTTP REPLY, TCP FIN HandShake), aber sie senden nicht die selben HTTP-Header, wenn sie den remote HTTP-Server anfragen.

Der Server

Der CGI-Server bietet uns zwei Dienste an:

- er steuert Client-Anfragen: Cookie-Entschlüsselung, Informationen über Clients und zu sendende Admin-Befehle aufbewahren;
- er implementiert eine Basis-Webschnittstelle, die dem Admin ermöglicht, Client-Informationen und erteilte Befehle anzuzeigen.

Wenn ein Client eine GET-Anfrage sendet, das CGI das Cookie überprüft und versucht, es zu entschlüsseln, aktualisiert es die Client-Informationen (speichert sie in einer Datei) und sendet schließlich die Antwort an den Client, gemeinsam mit den Befehlen, die der Administrator vorbereitet hat.

Wenn ein Administrator auf die Webschnittstelle zugreift, kann er Client-Informationen anzeigen und Befehle vorbereiten, die in der nächsten Kontaktperiode an den Client gesendet werden.

Wenn der Administrator mehr als einen Befehl vorrätig hat, der an den Client gesendet wird, wird jeder Befehl ein Cookie werden und alle Cookies werden in einer einzelnen HTTP-Antwort an den Client gesendet.

Wie sieht es aus?

Wir greifen auf die Adminschnittstelle *http://ip:port/cgi-bin/cook_cgi?pass=grayworld* zu, die uns mitteilt, dass aktuell kein Client registriert ist. Wir starten einen client auf 10.1.1.7 und wir machen Schluss:

```
./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
```

Wir starten einen zweiten Client auf 10.1.1.8 und lassen ihn laufen:

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

Listing 10. *Glücksspiel für den Server*

```
$ ./cook_cgi
Wie man einen versteckten Kanal kocht - cook_cgi.py - v0.1
[...]
Clientliste:
  #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'
[...]
```

Lassen Sie uns unsere Adminschnittstelle ansehen und zwei Befehle für den 10.1.1.8 Client speichern. Wir werden ein *Neue Kontaktperiode* auf 5 Sekunden (2) und *Deaktiviere das Auffüllen* (1) uns (3) (Verbindung zum Server) speichern.

Unser Client verbindet 180 Sekunden später zurück (Zeile 1) und sendet dasselbe Cookie wie vorher. Das CGI sendet die zwei gespeicherten Befehe (Zeilen 2-7): Der Client aktualisiert seine Kontaktperiode auf 5 Sekunden und deaktiviert dann das Auffüllen. Dann sendet er die zwei Bestätigungen mit zwei Verbindungen an den Server zurück (Zeilen 8 und 9). Er schläft für 5 Sekunden und kontaktiert dann den Server mit einer neuen *I am up*-Nachricht (Zeile 10). Dann schläft er wieder und wiederholt das *I am up* alle 5 Sekunden (Zeile 11) (Senden der gekochten Befehle an den Client.

Wenn wir die Adminschnittstelle überprüfen, bemerken wir, dass der Client 10.1.1.8 aktualisiert ist und dass gespeicherte Befehle nicht mehr aufgelistet sind (Client akzeptierte Befehle).

Glücksspiel

Jeder Client, der sich zum ersten Mal mit dem Server verbindet, verwendet diesselben Zufallsbytes (Zeile 1, [Glücksspiel für den Client #1] und [Glücksspiel für den Client #2]). Allerdings, jedes Mal, wenn Sie neue Zufallsbytes an einen Client senden (Zeile 2, [Glücksspiel für den Client #1] und [Glücksspiel für den Client #2] und dann Zeilen ½ [Glücksspiel für den Server]), sind sie nur für diesen Server bestimmt.

Wie Ihnen an [Glücksspiel für den Client #1] und [Glücksspiel für den Client #2] aufgefallen sein dürfte, wenn ein Client dieselben Zufallsbytes mit Auffüllen aktiviert verwendet, ist der Auffüll-Teil des Cookies exakt gleich. Dieser Teil wird natürlich anders sein, sobald der Client mit neuen *rbytes* aktualisiert wird, aber dieses Verhalten könnte verdächtig sein. Aus diesem Grund ist Auffüllen standardmäßig deaktiviert. Um die Auffüll-Option zu verwenden wäre der beste Vorgang, das Auffüllen zu deaktivieren, einige Initialisierungszufallsbytes für jeden Client zu setzen und sobald ein Client sich zum ersten Mal verbindet, die folgenden Befehle zu speichern oder einen nach dem anderen zu senden (mehrere HTTP Anfragen/Antworten):

- aktualisiere die Kontaktperiode auf kurze Verzögerung;
- aktualisiere die Cookie-Größe auf einen *hohen* Wert;
- Füge *hohe* neue Zufallsbytes hinzu;
- aktualisiere die Cookie-Größe auf Standardgröße und aktiviere Auffüllen;
- aktualisiere die Kontaktperiode auf Standardwartezeit.

Sie haben dadurch den Client mit bestimmten Zufallsbytes und die Initialisierungscookies werden verschieden sein, solange zwei Clients nicht mit derselben lokalen IP-Adresse zur selben Zeit beginnen.

Genießen Sie Ihr Mahl

Guten Appetit: *http://gray-world.net/projects/cooking_channels/*. Sicher, Fast Food zum Mittagessen ist nicht so gut für die Gesundheit, oder? Unser Mahl bietet verschiedene Probleme: Zum Beispiel setzt sein Entwurf voraus, dass jeder Client mit den selben Zufallsbytes starten muss (und somit, dass sie Auffüllen nicht während der Initialisierung verwenden *können*). Es heisst ebenso, dass, wenn ein Client *kompromittiert* ist, die gesamte Kommunikation für diesen Client Klartext sein wird. Eine Lösung wäre, `RBYTES` sicher von Zeit zu Zeit für jeden Client zu löschen.

Ein anderes Problem liegt in der Synchronisation. Wenn sie aus irgendeinem Grund verloren ist, ist der Client verloren. Eine Lösung wäre, zum Beispiel, einen anderen Cookie (oder irgendein HHTP-Anfragefeld) zu verwenden, um den Client zu resynchronisieren: Der Server sendet `RBYTES_POS+x` an den Client und der Client muss es für seine nächste *I am up*-Nachricht verwenden. Wenn die y nächsten Nachrichten falsch sind, heisst das, der Client ist *kompromittiert* und bald wird der Server auch untersucht werden.

Außerdem liegt ein anderes Problem an dem Schema, das wir verwenden, um die Clients zu registrieren. Da sie mit ihrer öffentlichen IP-Adresse registriert sind, ist ein einziger Client pro öffentlicher IP-Adresse möglich. Einige Lösungen zu diesem Problem können implementiert werden, Sie müssen sie nur finden.

Und was ist mit dem Server? Angenommen, Ihr Server ist *down*, wäre es nicht lustig, dass der Client automatisch einen zweiten registriert? Der Client kann demnach `RBYTES_POS[x]` für x Server verwenden. Natürlich könnten wir ebenso einen neuen Befehl implementieren, der dafür verwendet würde, den Client aufzufordern, auf einen anderen Server zu wechseln. Wenn Sie nicht wollen, dass jeder Server k*ompromittiert* wird, wenn es ein Client ist, speichern Sie nur 4 *ge-XOR-te* Bytes auf der Clientseite und senden Sie den *Schlüssel*, wenn Sie wechseln wollen.

Eine andere komische Idee ist, dass, sobald Sie überprüft haben, ob der Client mit der Außenwelt kommunizieren kann, sind Sie erledigt, oder? So wäre ein weiterer Befehl: *bitte, mein lieber Client, lösch dich selbst, aber <ironisch> achte auf dein Umfeld </ironisch>*.

Daher wäre der *Vorschlag des Küchenchefs* für morgen, ein sichereres `RBYTES`-Verhalten zu implementieren und einige Online-Verhaltensänderungen zu implementieren (so, dass unser Client immer nützlicher wird, sobald wir wissen, dass er online ist). Natürlich würde der *Chef* Ihnen gerne vorschlagen, mit ungewöhnlichen Gewürzen zu kochen, so dass wir etwas heißes zum Probieren bekommen: Browser-Prozessinjektion, da die Leute oft kein Python essen mögen und da Huckepack auf rechtmäßigen HTTP-Vorgängen *funky* wäre – zumindest, wenn Sie wollen, dass fremde Leute Ihr Rezept probieren.

Platz von Cookies

Wir entschieden uns, unsere Set-Cookie-Anweisung in die HTTP-Headerantwort einzubetten. Beachten Sie, dass wir auch eine META-Anweisung verwenden können, so wie:

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

Das heißt nicht viel für das aktuelle Projekt, aber Sie werden den Trick im folgenden Kapitel verstehen.

Caching der zweiten Ebene

Wie in *Die Cookie Theorie* beschrieben, ist es möglich, Caching-Dienste als eine Zwischenebene zu verwenden, um Daten zu speichern und an mehrere Clients weiterzuleiten und dann die Verwendung eines *remote*-Servers zu beenden.

Die einfachste Art, um diese Theorie zu implementieren (auch wenn kompliziertere Schemata existieren – folgen Sie dem weißen Kaninchen), liegt bei:

- Client C1 fragt eine URI von Server S durch Proxy P an;
- Server S antwortet und die Antwort ist in P gecachet;
- Client C2 fragt dieselbe URI von Server S durch Proxy P an;
- Proxy P antwortet mit der 2. Antwort.

Grundsätzlich heißt das, dass die Clients C1 und C2 kommunizieren können, ohne den *remote*-Server für jede Nachricht erreichen zu müssen. Es heißt etwas im Katz-und-Maus-Spiel, das wir eventuell gegen das Erkennungsteam spielen: Es heißt, dass der Erkennungsantrieb Verkehr zwischen den Clienten und ihrem ersten *Hop-zum-Ziel* fangen muss, wenn es ein Caching-Dienst ist.

Ist es so möglich. Diesen Punkt mit unseren Cookies zu implementieren? Lassen Sie uns die Squid FAQ ansehen. Die FAQ (*http://www.squid-cache.org/Doc/FAQ/*) erklärt: *Demnach macht Squid-2 Cache-Antworten mit Set-Cookie-Headern, aber es filtert die Set-Cookie-Header selbst für Cache-Treffer aus.* Ok. Das heißt (hier auch wieder), dass, wenn wir entscheiden, Set-Cookie-Headeranweisungen zu benutzen, werden wir nicht in der Lage sein, unsere Cookies zu cachen. Aber filtert Squid das Meta-Äquivalent aus (siehe *Platz von Cookies*)? Überprüfen Sie es selbst.

Wie in PRIATNOVO APETITA beschrieben, kann dieses Verhalten interessant sein, wenn Sie entscheiden, den Client aufzufordern, auf einen anderen Server zu wechseln. Sie müssen den Befehl nur einmal für den ersten Client senden und dann wird jeder Client, der durch den selben Cache geht, die Antwort erhalten, auf den zweiten Server zu wechseln. ●

Im Internet

- *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.

Über den Autor

Simon Castro ist ein Mitglied des Gray World Teams (*http://gray-world.net*). Diese internationale Forschungseinheit widmet sich der Computer- und Netzwerksicherheit mit einem speziellen Interesse an NACS-Umgehung (Tunneling, versteckte Kanäle, netzwerkbezogene stenografische Methoden).
Kontakt zu den Autoren: *simon@gray-world.net oder team@gray-world.net*

