Global Information Assurance Certification Paper

COVERT CHANNELS OVER SOCIAL NETWORKS

GIAC (GCIH) Gold Certification

Author: Jose Selvi, jselvi@pentester.es
Advisor: Rob VandenBrink

Accepted: March 20th 2012

Abstract

There are many ways to embed data into unused fields of some network protocols like IP, TCP, etc in order to send and receive data in a hidden way. Nowadays, malware coders are hiding their communications using https, but techniques such as DNS sinkhole can help network administrators to stop some of them. The following step in Covert Channels is to embed data into known applications such as, for instance, social networks. Since it uses known domain names, it is difficult to detect the difference between real communications and evil ones. In this paper, we review some ways to embed data into these social networks, and how this can affect to corporate and personal security. As a proof of concept, we have released a tool called "facecat" (FaceBook Cat). With this tool we can relay ports using a FaceBook Wall as a Pipe, so it can be used through proxies and other network protections.

Introduction

Today we live in a malware age, with the malware industry growing exponentially (AV-Test, 2012). Anti-malware software companies are working hard in order to stop this growing trend, but malware detection is a really complex problem, since it can be a very high number of different codes that result in the same evil actions.

While anti-malware software companies often concentrate on host based detection, network administrators work trying to detect and block unwanted or suspicious network communications. These network communications are needed by many malware applications in order to communicate with a coder or botmaster, since most of the malware needs to connect to a command and control console to report back stolen information. There are only a few known fully independent malwares, for instance Stuxnet (Falliere, O'Murchu & Chien, 2011), which is designed to work without Internet connection and without human control. However, this is not a common architecture in the malware industry today.

One of the most extreme countermeasures against malware is blocking outgoing traffic. However, since computers need to share information and communicate between each other, most of the time it is not possible to block all outgoing traffic.

When any connection is allowed, attackers and malware can take advantage of it to hide an evil communication inside a permitted one. This is the root idea of Covert Channels (Lampson, 1973).

Covert Channels Techniques have been evolving from the 1970's until today (Thyer, 2008), trying to avoid protections used by network administrators. As happens in most security fields, the battle between attackers and security engineers is on.

Author Name, email@address

The Covert Channels' Past

Steganography & Covert Channels

Steganography is the art of hiding information inside any kind of message, so Covert Channels can be thought of as Network Steganography.

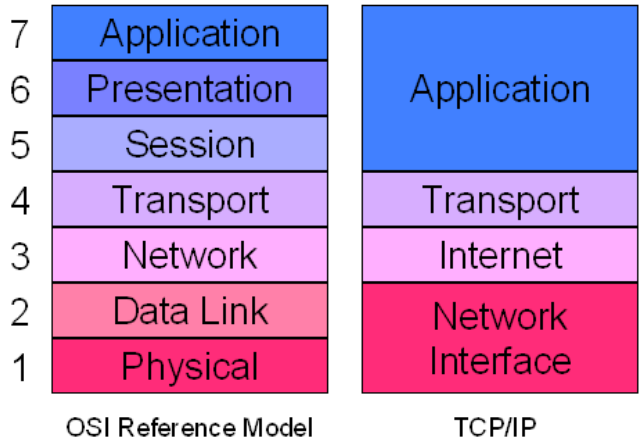
Steganography uses unused or less significant fields in file formats, network header or any other field in order to store a hidden message, with the main goal of having the same appearance as with no hidden message.

Once information has been hidden, it can be difficult to detect, since sometimes it is impossible to set a baseline of allowed values for a field, and a deep knowledge of each protocol and some highly complex statistical techniques are needed (Geetha, Sivatha, Siva & Kamaraj, 2009) (Geetha, Ishwarya & Kamaraj, 2010) in order to detect an abnormal field use. This field of study isb called *steganalysis*, and it applies to file formats, and also to network connections.

TCP/IP Covert Channels

Almost all modern networks are TCP/IP-based. As we know, TCP/IP model is a suite of protocols, specified by RFCs (Request For Comments).

RFCs specifies headers, fields, type of data, sizes and much more, but it is easy to find reserved or unused fields, or any other way of hiding data.



Each TCP/IP Layer, and each protocol field, can be potentially exploited in order

to establish a Covert Channel. Some used fields are (Thyer, 2008):

- Network Layer: Not commonly used, since LAN access to the target is needed.
- Internet Layer: IP and ICMP Header fields like "IP Identification Field" (IPID) in non-fragmented datagrams, "IP Source Address", "ICMP Identification Number" (ICMP ID) and different ICMP Control Messages.
- Transport Layer: TCP and UDP Header fields like "TCP Initial Sequence Number" (ISN), "TCP Acknowledge Number" (ACK), "TCP Options", etc.
- Application Layer: DNS, HTTP, etc, like "DNS Identification Number" (DNS ID).

In addition to exploiting specific fields, relationship between layers in the TCP/IP stack can also be abused. For instance, specially crafted packets can be utilized to create gaps between IP and TCP headers which can be leveraged as a Covert Channel (Caudle, 2007).

Some of these techniques can be used in almost all kinds of environment and some of them only in specific ones, since environments can be very different from each other.

Traditional Network Protections

Covert Channels are not a new threat. Network Administrators have been aware of Covert Channels for near 40 years (Lampson, 1972) and they have been working hard with the purpose of detecting, stopping, or at least mitigating their impact.

"Traditional" protections include:

- Block connections: Any kind of communication can be used to hide information as a Covert Channel. The best security protection against Covert Channels is just not to allow a connection. Of course, this is not feasible because computers need to communicate with each other, but blocking unneeded traffic can help to avoid some risks. Currently, common protections are to block all incoming connections except the only ones really needed, and

block all outgoing connections except established ones.

- Proxies: Normal or Reverse Proxies are used as a second layer of protection. Since incoming and outgoing connections finish at the proxy, all the hidden information stored in the lower layers is going to be lost. Most of them also work in application layer (mostly HTTP), and check for an abnormal use of the protocol.

These kinds of protection are widely known and implemented in almost any big company, but as always happen, it's not a silver bullet.

New trends in Covert Channels

Protocol Encapsulation

As we have just seen, common network protections limit the possibility of hiding data in all TCP/IP stack layers.

If we focus on the most common application protocol (HTTP), while a proxy can block abnormal uses, it is completely unable to handle the protocol content on account that it can be very different from one website to any other.

Because of this, it is perfectly possible to hide information as an HTML body, or any other application content. The Hackers Choice (THC) published a tool called RWWWShell (Hauser, 1998), as a proof of concept of a reverse HTTPS shell written in Perl, hiding all the shell traffic as HTTP Requests and Responses.

Some years later, Sensepost researchers published Setiri (SensePost, 2002), as a proof of concept of a Trojan developed using HTTPS Covert Channels.

At the moment, most trojans and botnets use HTTP Covert Channel communications. One of the most notorious ones is Zeus (Falliere & Chien, 2009) which uses HTTP Connections in order to communicate with its command and control.

Encrypted Protocols

Other widely used countermeasures against Covert Channel communications are network intrusion detection systems (NIDS) or content filtering proxies. These work by looking for known patterns inside communication content, like a CMD banner, a /etc/passwd line, or other keywords.

The atackers response to this countermeasure is encryption. Content information can be encoded and encrypted in such a way that only the attacker can decode the information. However, despite content encoding and encryption, it is possible for intrusion detection systems to look for non-content patterns, for example the URI of a known Trojan (like Zeus), tag where encoded content is, or any other means.

The common way to encrypt full HTTP communications is just to use HTTPS. As we know, SSL communications are a big problem for proxies and intrusion detection

systems, since it's not possible for them to check the communication content. When we want to protect our own SSL services, it is possible to configure an external SSL-Decrypter Appliance (as SSL Accelerators do) since we know both public and private encryption keys, but when checking outgoing communications it becomes difficult, because of course we don't know every site's private keys. Some companies set an internal CA and configure their proxies as an SSL-MitM to be able to decrypt SSL communications. This countermeasure can really help in Covert Channel detection, but user's privacy is totally exposed.

Application Encapsulation

The next step in HTTP Covert Channels is Application Encapsulation. Since it uses a known application in order to hide data, it can be difficult to split from real communications.

Some Trojans are using this new scenario, like "Naz" (Kartaltepe, Morales, Xu & Sandhu, 2010), which used a Twitter account (@upd4t3) as a command channel. The tweets were base64 short-url encoded links, which pointed to payloads to execute.

This new trend in Covert Channels is not widely used, since nowadays protocol encapsulation and encryption are more than enough most of the time, but it has some advantages because of using known applications as a communication channel.

One of these advantages is that, when encryption is used, the only researchable data is domain name, since it has to be known by proxies or hosts in order to connect, and domain name using known applications encapsulation can be as usual as Twitter.com, FaceBook.Com, Google.Com or any other similar.

New Countermeasures

We have just seen that DNS name is the only information we can get when an evil application uses encapsulation and encryption. In order to limit this issue security analysts are working on different kinds of DNS-blacklist.

Some of them work at browser level, like the "Google SafeBrowsing API" (http://code.google.com/apis/safebrowsing/), used by Firefox while many others work at network level, like NIDS reputation lists or other similar techniques. Security in the

browser can be less effective than security in the network, since malware can use direct HTTPS connections. Security in the browser's main goal is to prevent user infection through browsers vulnerabilities, and not to block users which are already infected.

DNS Sinkhole techniques (Bruneau, 2010) are used in corporate environments as a way to block connections to well known names used by botnets. These techniques work by hijacking DNS Queries to hostnames or domains presented in the blacklist, and then making a fake response to a non existing or local IP address. The advantage of this technique is that it prevents any kind of connection to domain names used by malware and botnets, but is not effective when common domain names are used, for instance if malware is hosted on Google.com.

Author Name, email@address

Known Applications Protections

PasteBin

PasteBin (http://pastebin.com) is a website that allows users to upload any kind of simple text. It can be used within a user account, or in a public way. As a consecuence, it's often used by hackers to publish information leakages.

Despite its anonymity, PasteBin has a Captcha protection in order to avoid automatic publications, so it would be difficult to use it as a pipe for hidden communications.

DropBox

DropBox (http://www.dropbox.com) is a website that allows users to upload files and share them in an easy way. Hiding a TCP communication inside files could be possible, but a user account is needed in order to upload information.

Regarding a few tests we did, DropBox is not blocking massive file uploads. As a consecuence, it would be easy to hide a TCP communication over file uploads, using each file as a container for packet payloads.

Google Sites

One of the Google Services is Google Sites (http://sites.google.com). This web application provides users with the capability of creating new websites with some standard functions like texts, images, attachments or comments. Google Sites is very useful in order to create a new and simple website in just a couple of clicks.

In the same way, it can also be used in order to hide communications within its content (comment, boddy, etc). Only authorized users can add information to a site but, as almost everyone has a Google account, it would be easy to find a Google cookie in almost any computer. Since a session variable is usually stored in a cookie, It can be used to access into a user's session, without any knowledge of his login credentials. We can consider a cookie as a container for a temporal password (session variable).

Regarding a few tests we did, Google is not blocking massive comment postings when a user is authenticated, so it would be easy to hide a TCP communication within

Google Sites comments, using them as containter for packet payloads.

Twitter

Twitter (http://twitter.com) has already been used by "Naz" backdoor in order to hide its commands, so it is obvious that TCP communications can be hidden within Twitter tweets.

With Twitter, you can't use a user account to post evil tweets, since every tweet is automatically seen by all the user followers, so it would be too noisy. This is the reason why "Naz" uses its own and independent Twitter account.

FaceBook

FaceBook (http://www.facebook.com) is probably the best known social network. Almost everyone has a FaceBook account and many seldom logout since they're using it constantly. This is a great advantage for hiding communications, because we can use a different FaceBook account for each communication, and it would be more difficult for FaceBook to detect.

FaceBook has some protections against massive posting, but it is a big and complex application with lots of features, so it is possible to find the non-protected ones.

Summary

After reviewing some well-known web applications, we can make a summary in order to choose one of them for a proof of concept:

	Non-Auth	Captcha	Widely-Used	Covert Ranking
PasteBin	Yes	Yes	No	Low
DropBox	No	No	No	Medium
Google Sites	**No**	**Sometimes**	**Yes (Google)**	**High**
Twitter	No	No	Sometimes	Medium
FaceBook	**No**	**Sometimes**	**Yes**	**High**

Author Name, email@address

Regarding all the analysis, we chose FaceBook because it is probably the most widely-used in user environments and its massive posting protections can be easily bypassed. FaceBook is also a threat in corporate environments, since lot of people use it at work.

Author Name, email@address

Proof of Concept: FaceCat

Introduction

FaceBook is one of the most widely used social networks. Almost everybody has a FaceBook profile used daily in order to communicate and to share information with friends and acquaintances. This is why FaceBook becomes a perfect sample of widely used social networks where we can hide communication channels. FaceBook offers some communication channels: private messages, profile information, the wall, pictures, comments or just chat. Security countermeasures are used in some of these channels in order to stop automatic access like crawling for profile information or some other threats.

Despite the efford of FaceBook's Security Team, there are still some ways to create Covert Channels through the FaceBook walls and maybe some other alternative ways.

The following is an example of Covert Channles within FaceBook, called FaceCat.

Stealing user cookies

A browser can handle cookies in two different ways. A cookie that is used for a long duration is stored on disk, and a cookie that is valid only for a short period of time (for instance, a session) is stored in the memory, and it is removed later when the browser closes.

Authentication cookies are an example of short time storage, but social network users prefer not to login every time they want to use FaceBook. For this reason, an option "Keep me logged in" exists on the FaceBook login page. When a user checks this checkbox prior to login, the browser stores the authentication cookie on disk, so every time the user goes onto FaceBook, they are already authenticated.

With regard to this fact, most of the time we can find a FaceBook cookie stored on disk that we can use in order to authenticate its account. Then we can access it as different and real user accounts, not only with accounts specificly created for the purpose, such as Naz does.

Author Name, email@address

As a first action, FaceCat fingerprints the host operating system, and looks for FaceBook cookies in the most frequently used browsers like Internet Explorer, Mozilla Firefox, Google Chrome or Apple Safari. If found, FaceCat steals this cookie and starts working as the last user logged in FaceBook.

A FaceBook Wall as a Pipe

The Wall is a FaceBook object where users publish whatever they want mainly their thoughts. One of the communication advantages of the Wall is that people can write on a friend's wall or comment on any wall's publication, if privacy settings allow it.

As a result of this capability of receiving information from different users, it can be a perfect Covert Channel. Both sides of FaceCat binaries can read from and write messages to the wall, in order to establish a TCP connection through a FaceBook Wall, just as NetCat does with sockets. This is the main idea behind FaceCat, using a FaceBook Wall as a Pipe.

FaceBook HTML content can be very difficult to parse so we use the mobile version (http://m.facebook.com) since it has a simpler code, simpler interfaces and fewer changes than the main site.

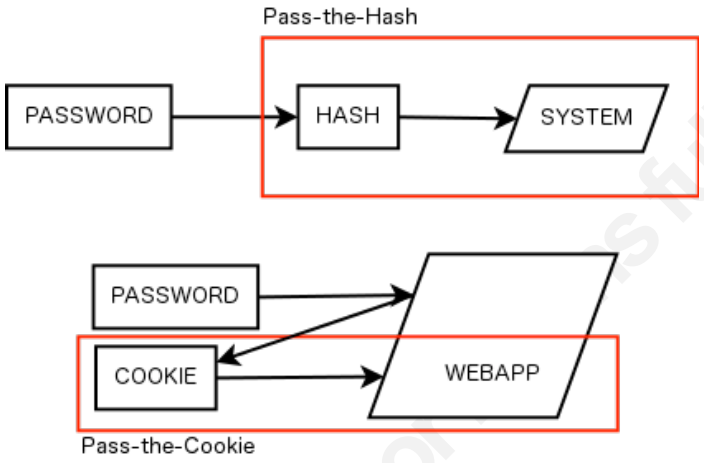
On a FaceBook's earlier version, a user could allow anyone to write on his wall, despite not being friends with him. On a recent FaceBook update, the wall's privacy options have been updated in such a way that you can't allow everyone to write on your wall, only your friends. This forced a change in FaceCat design, since the stolen cookie can't be used for writing on walls anymore.

Pass-the-Cookie

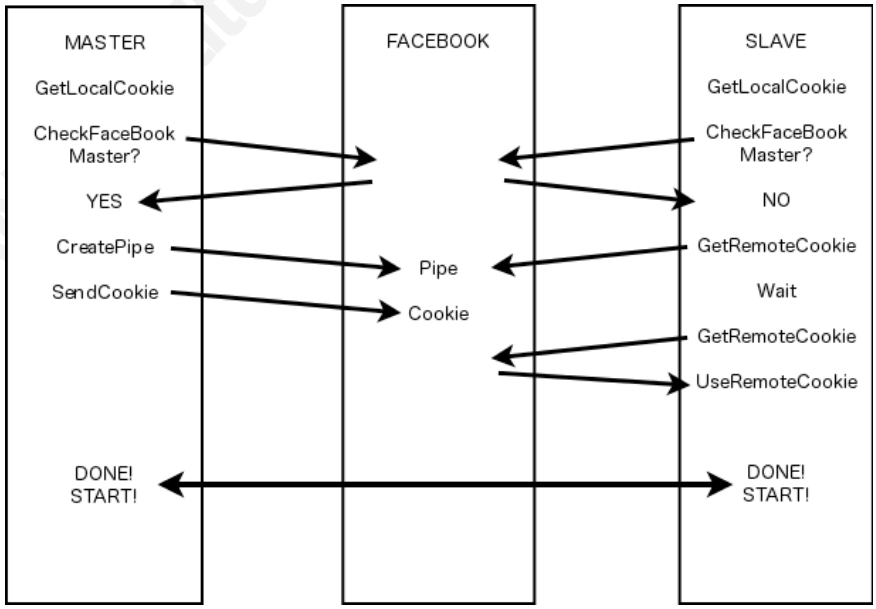
One of the possible solutions for the last FaceBook updates would be to force the hijacked user to be a "pipe's friend", but that would be easy to detect by users or their other friends, since they would have a new unusual friend, without any other friends, photos, etc.

Author Name, email@address

In the end, the chosen option was to do a "Pass-the-Cookie". We call it "Pass-the-Cookie" because of its similarities with "Pass-the-Hash" technique (Ewaida, 2010), since both use a "second step" authentication object (hash or cookie) instead of "first step" one (user and password).



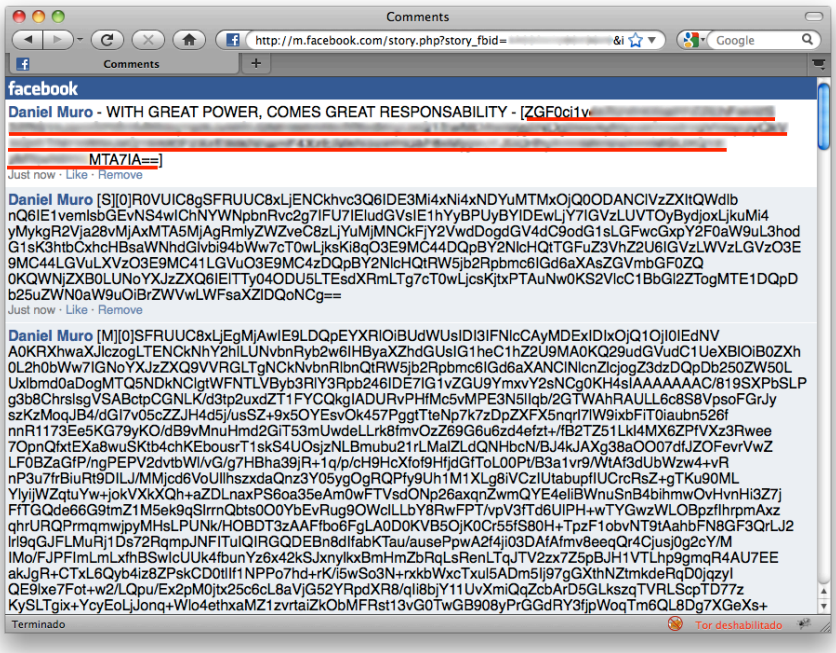
In FaceCat's architecture, we have two different roles. On the one hand, we have The Master (M), who is the wall's owner or a friend capable of writing on it (the attacker). On the other hand we have The Slave (S), who is a non-related FaceBook user (the victim).



In the first step of FaceCat's operation, Slave FaceCat looks at the wall, waiting for a cookie. When Master FaceCat starts running, it writes its own cookie on the wall,

Author Name, email@address

between brackets and base64 encoded. Then Slave FaceCat reads it and uses this cookie as an authentication factor in the following steps, so from this point it has the ability of writing on the wall.



Message Encoding

Not each hexadecimal number has a printable ASCII equivalent, so some kind of encoding is needed in order to be able to write it in a channel thought for human language.

Base64 was designed to represent binary data as an ASCII string, so it fits perfectly in this situation. Base64 is used in FaceCat's encoding schema, but some other encoding or cryptography could also be used.

Certainly, a very long base64 string doesn't look like human language so it could be detected as a binary channel through a wall. However, it could be split into randomized sized words (2 to 7 characters, for instance) and used ";", "," and "." instead of "/", "+" and "=". Since FaceBook supports lots of languages, it should be more difficult to detect that this message isn't a human language.

In practice, FaceBook doesn't check the comments content, so it's not necessary

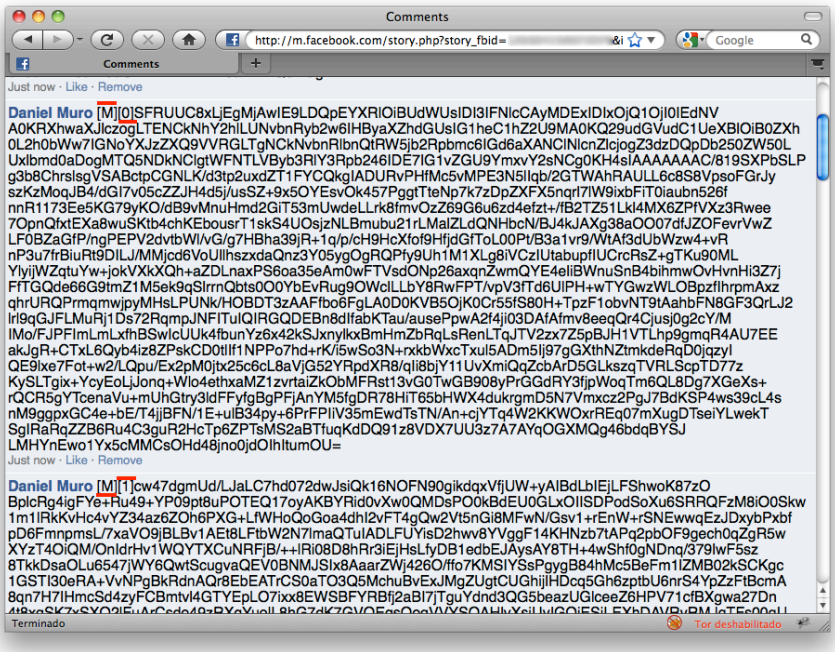
Author Name, email@address

to use a more complex technique than the base64 encoding.

Sequence numbers and acknowledgment

When requesting a comments page, we have no simple way of detecting what comments have already been read or not. This is the reason why we need some kind of labeling in order to detect which comment was read last.

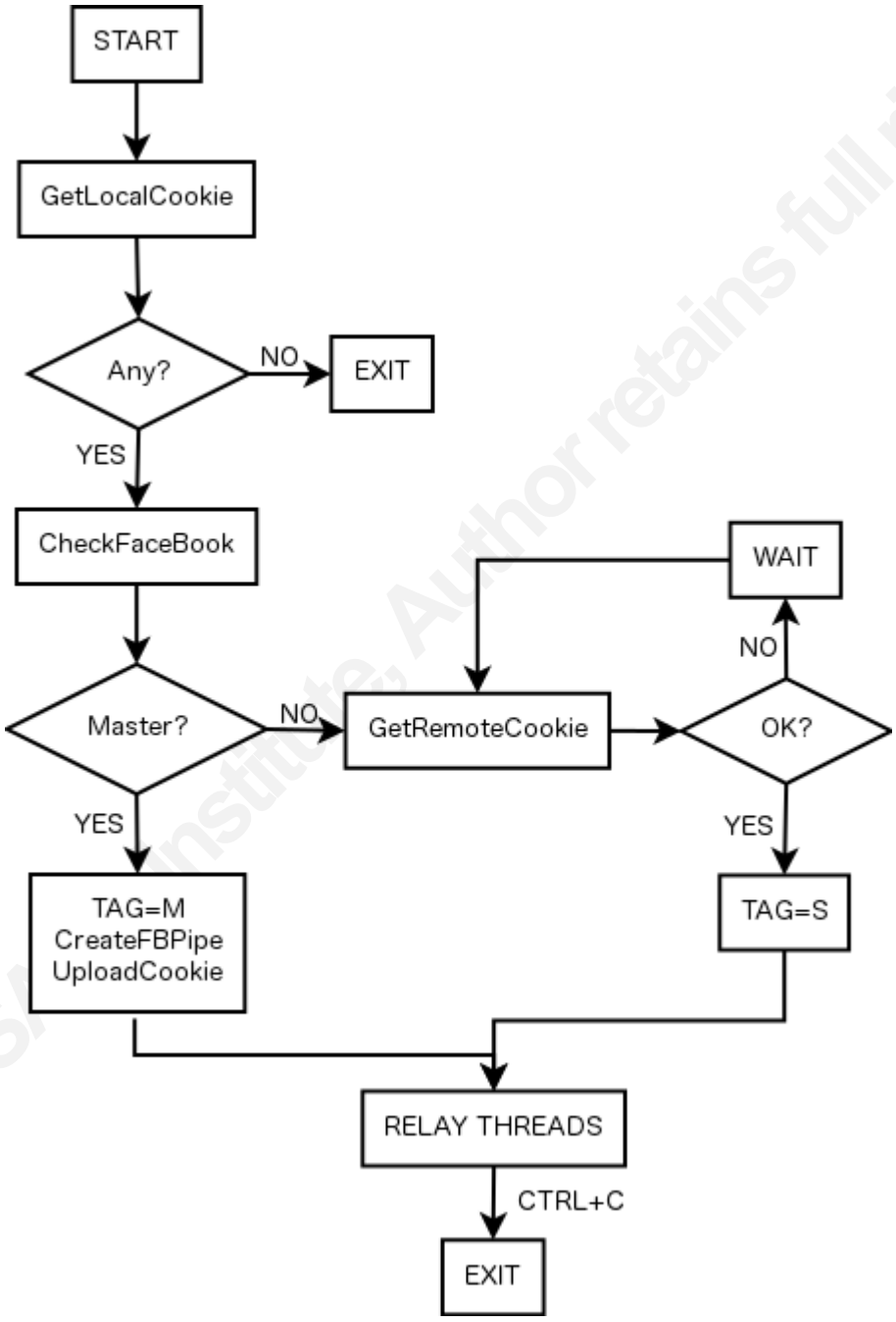
The simplest way we found was to use a sequence number for comments, in the same way that TCP sequence numbers work. FaceCat stores a sequence number for writing, and an acknowledge number for reading, so at any time it knows which sequence number it has to use and which new comments there are. This sequence number is written into brackets, at the comment's beginning, after the master-slave tag.



Author Name, email@address

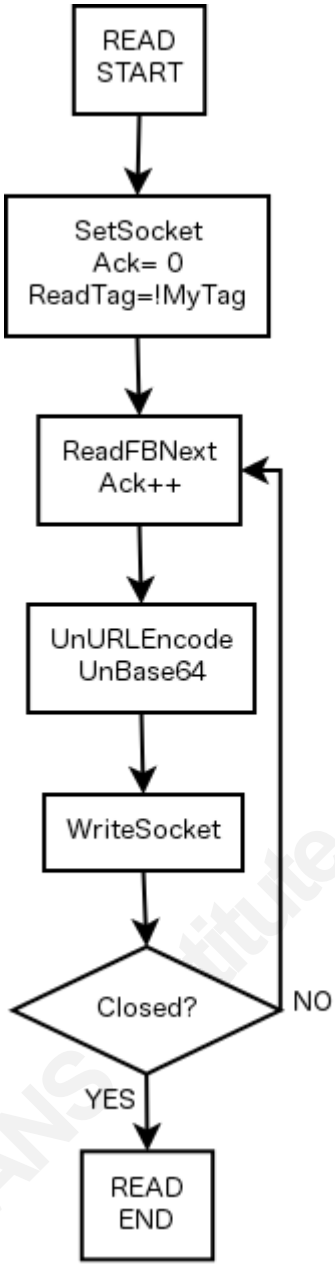
FaceCat Algorithm Diagram

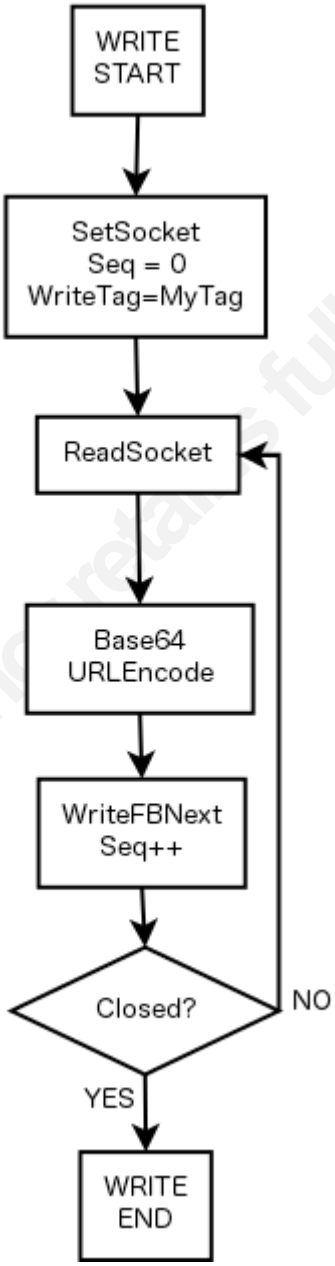
As a result, the FaceCat's full algorithm is following:



Read and Write Relays are following:

Author Name, email@address





FaceCat Options

In a more practical view, let's see FaceCat's command line options:

Author Name, email@address

```
$ ./facecat.py

Usage: facecat.py [options]

Options:

 -h, --help                    show this help message and exit

 -w WALL,    --wall=WALL       wall pipe account

 -c HOST,    --host=HOST       connection host

 -p PORT,    --port=PORT       listening or connection port

 -v,         --verbose         verbose output
```

In a closer view of each option:

- Help: Just show the help
- Wall: Email of the master's wall. It has to be previously configured in order to allow writing on it.
- Host: FaceCat can work by listening or connecting, as NetCat does. If you chose a host, connection mode is used. If not, listening mode is.
- Port: Port where FaceCat is listening for new connections, or port to connect to (Host:Port connection).
- Verbose: Shows each step of the process. Useful for educational purposes.

FaceCat in Action

As a Proof of Concept, we used FaceCat in order to hide a known Backdoor communication, for instance **Poison Ivy** (http://www.poisonivy-rat.com/).

Poison Ivy, as some other **Remote Administration Tools** (RAT), is a piece of software that allows an operator to control a system as if he had physical access to it. This kind of software is widely used in corporate environments, in order to give remote technical support. It is also used as a backdoor, when the system is controlled in a hidden way, without the knowledge of the user or administrator.

Author Name, email@address

In this Proof of Concept, we have chosen Poison Ivy as a RAT tool, but any other Tool or Protocol could also be used. The next steps are followed:

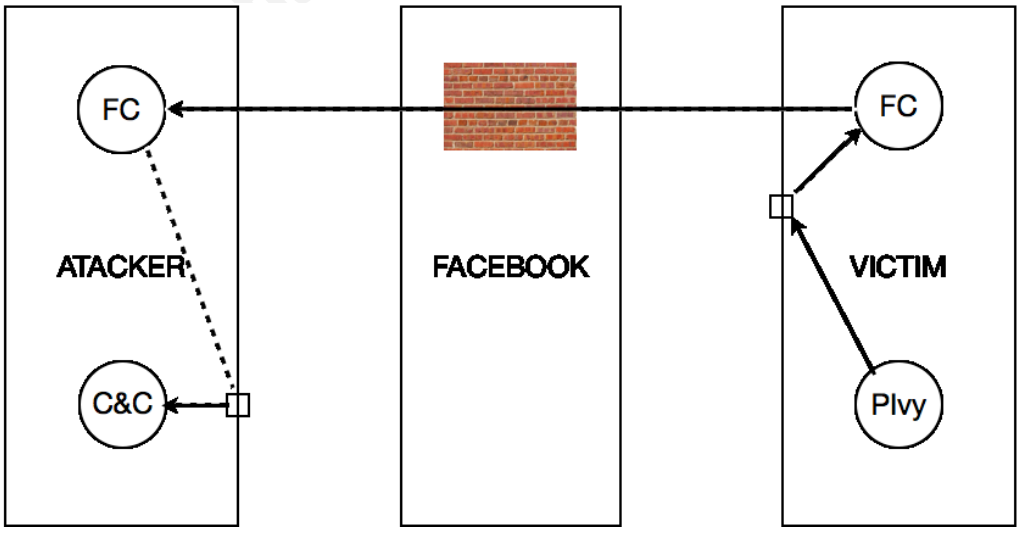
1. Create a Poison Ivy server that will try to connect to 127.0.0.1 at port 3460. We also start a Poison Ivy client listening at the same port
2. Create and configure a FaceBook account in order to write on its wall, for instance wall1@gmail.com.
3. Run Internet Explorer and login in our newly created account.
4. Run FaceCat in order to read wall1@gmail.com's wall and to relay to our local poison ivy's client:

   facecat.py –v –m wall1@gmail.com –c 127.0.0.1 –p 3460

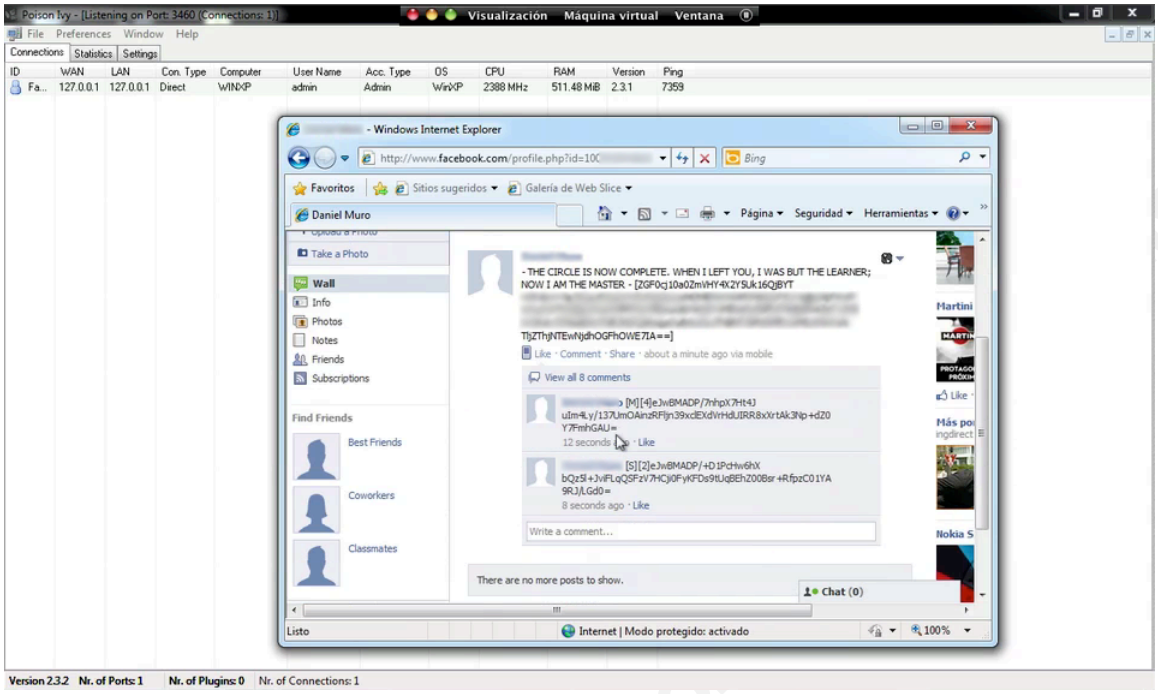
5. Copy (or infect) FaceCat and Poison Ivy's server to the victim's machine.
6. Run FaceCat in order to listen to port 3460 in the victim's machine and to relay to wall1@gmail.com's wall:

   facecat.py –v –m wall1@gmail.com –p 3460

7. Run Poison Ivy's client in the victim's machine.



8. Use Poison Ivy normally, but through a FaceBook's Covert Channel.



Author Name, email@address

Covert Channels Detection and Prevention

Network Layer Protections

In the corporate environment, we can take two different approaches: detection or prevention, or perhaps both of them.

Covert Channel detection can be a very difficult task, since they are specifically designed in order to avoid detection. Despite this, they can sometimes be detected using an anomaly detection approach. If you have previously deployed some sensor and you have made a baseline profile of your network communications, perhaps you could be able to detect abnormal circumstances such as too much traffic in non-office time and other similar situations.

Covert Channel prevention can be a bit more simple in some circumstances. You can block outgoing connections to social networks if it is not needed, with techniques such as DNS Sinkhole or just blocking by IP address. In a more sophisticated approach, we could use our own captcha protection in our proxy in order to avoid massive automatic connections to external web sites.

Application Layer Protections

Of course, the best approach is that all applications should have protections against massive automatic messages. If social networks such as FaceBook or Twitter had a captcha or similar protection like PasteBin does, it would be much harder to use them as a Covert Channel.

The problem is that a captcha (or similar) protection makes users feel uncomfortable with the application handling, since it becomes more difficult and slower. It can make it really hard to be deployed in intensively used applications like Twitter or Facebook.

Author Name, email@address

Conclusions

Protections against application layer Covert Channels are very difficult to design and deploy, since it can be totally different from one application to another. The best protection should always be deployed in each application, but it isn't always possible.

In these circumstances, the least-privilege rule is a must. We can try to avoid Covert Channels by blocking all unnecessary websites or, in a more complex approach, deploying network protections against massive automatic connections.

In user environments, where all network protections are entirely within a home-grade firewall router, we must focus on prevention, since usually all outgoing connections are allowed.

Author Name, email@address

References

Av-Test. (2012, march 05). *Malware statistics* . Retrieved from http://www.av-test.org/en/statistics/malware/

Bruneau, G. (2010). *DNS sinkhole*. Retrieved from http://www.sans.org/reading_room/whitepapers/dns/dns-sinkhole_33523

Caudl, R. (2007). *Assumptions in intrusion analyst – Gap analysis*. Retrieved from http://www.sans.org/reading_room/whitepapers/honors/assumptions-intrusion-detection-blind-spots-analysis_1751

Ewaida, B. (2010). *Pass-the-hash attacks: Tools and mitigations*. Retrieved from http://www.sans.org/reading_room/whitepapers/testing/pass-the-hash-attacks-tools-mitigation_33283

Falliere, N., & Chien, E. (2009). *Zeus: King of the bots*. In Symantec Corporation. Retrieved from http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/zeus_king_of_bots.pdf

Falliere, N., O'Murchu, L., & Chien, E. (2011, February). *W32 Stuxnet Dossier*. Retrieved from http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Geetha, S., Ishwarya, N., & Kamaraj, N. (2010). *Evolving decision tree rule based system for audio stego anomalies detection based on hausdorff distance statistics*. Information Sciences, 180(13), 2540-2559.

Geetha, S., Sivatha, S., Siva, S., & Kamaraj, N. (2009). *Blind image steganalysis based on content independent statistic al measures maximizing the specificity and sensitivity of the system*. Computers and Security, 28(27), 683-697.

Hauser, V. (1998). *Placing backdoors through firewalls*. Retrieved from http://www.thc.org/papers/fw-backd.htm

Author Name, email@address

Kartaltepe, E. J., Morales, J. A., Xu, S., & Sandhu, R. (2010). *Social network-based botnet command-and-control: Emerging threat and countermeasures*. Springer-Verlag Berlin Heidelberg, 511-528.

Lampson, B. W. (1973). *A note on the confinement problem*. Communications of the ACM, 16(10), 613-615.

SensePost. (2002). *Seiri: Advances in trojan technologies*. Paper presented at Black hat asia. Retrieved from http://www.blackhat.com/presentations/bh-asia-02/Sensepost/bh-asia-02-sensepost.pdf

Thyer, J. S. (2008, January 30). *Covert data storage channel using ip packet headers*. Retrieved from http://www.sans.org/reading_room/whitepapers/covert/covert-data-storage-channel-ip-packet-headers_2093

Author Name, email@address

Appendix

FaceCat YouTube Demos

FaceCat TCP chat: http://www.youtube.com/watch?v=flZUuRK2R-k

FaceCat and Poison Ivy: http://www.youtube.com/watch?v=C_c8KNvVSVg

FaceCat SourceCode

Download from: http://tools.pentester.es/facecat/

Author Name, email@address