Instituto Superior Técnico, Universidade Técnica de Lisboa
Network and Computer Security
MERC: 1st year, 1st Semester, 2012/13

68208 – Artur Balanuta 68210 – Dário Nascimento 65963 – David Dias

Peer-to-Peer Botnet - Security &Authentication

1. Overview

A bot, or zombie, is a computer infected with a program, which allows an attacker to execute arbitrary commands remotely on it. *Botnets*, i.e., large network of bots, are one of the main concerns in today's Internet since they are often used to execute mass attacks like *Distributed Denial-Of-Service* (*DDoS*), send a large amount of spam, phishing emails, click-fraud and stealing personal data, such as email accounts or credit-card data.

At the center of many of these attacks is a large pool of compromised computers located in homes, schools, businesses and government around the world. Attackers use these *zombies* as anonymous proxies to hide their real identities and amplify their attacks.

Most part of botnets are based on centralized architectures like *Command & Control (C^2)*: in these scenarios, the bots receive info from a single central server. Locating the central server and then shutting it down can achieve mitigation of such botnets. However, in recent years, the first botnets that use peer-to-peer (P2P) based communications appear, like Walowdac or Storm Worm.

Almost no current botnet uses strong cryptography and they can't distinguish between valid bots and traitors. While C^2 architectures allow *botmaster* commands authentication, distributed architectures also needs a thrust mechanism to avoid sabotage, traitors, whitewashes, front men, collusions and other P2P attacks.

2. Communication & Organization

Most attackers would like the ability to rapidly send instructions to bots but also do not want that communication to be detected or the source of that commands to be revealed. There are 3 main topologies Centralized (C2), Unstructured Peer-to-Peer and Peer-to-peer structured overlay network.

2.1. Propagation

One change that happened few years ago is the shift from a propagation that might have required a manual installation process by attacker to multiple automated propagations. For example, the Slammer worm used a single vulnerability to infect hosts while more modern bots exploit distinct vulnerabilities. Another important shift in propagation is the move from random scanning to robust "hitlists" (e.g.: list of hosts, email lists, social networking lists, ARP cache entries, etc.) and from vulnerable service and applications to vulnerable users. Nowadays the targets are higher-level application like web browsers and social engineering attacks against users. For example, drive-by downloads and web-based infections are now commonplace.

The details of how bots are infected will not be addressed on our solution. We will assume the existence of a population of computers infected with a malicious binary containing: the code to perform malicious activities and the public key of the *botmaster,* used for command validation. Bots can receive commands from the *botmaster* through a common TCP port open to the Internet. UPnP or malware with firewall killers can achieve this.

2.2 Organization

There are many ways to create a botnet structure. The most common are the Centralized, Unstructured, P2P Structured.

2.2.1. Centralized (C^2): One master send commands to all botnet. Messages tend to have low latency, as they only need to transit few well-known hops. The major weakness: the attacker can be easily detected since many clients receive data from the same point and the discovery of the central location can compromise the whole system.

2.2.2. Unstructured: Each bot knows few other bots and flood the command to them. The major weakness is: the message latency could be extremely high and no guarantee of delivery.

2.2.3. Peer-to-peer structured overlay network:

Each *peer/bot* joins a *Distributed Hash Table* where he knows few other nodes. All peers route the messages along the *DHT*. The *botmaster* can join the ring like a common peer without being detected. This means that the compromise of single bot doesn't necessarily compromise the entire botnet. The major weaknesses are the message latency, crawlers and traitors.

Our choice is the *P2P Strutted Overlay Network.* It allow a botmaster to control a lot of slaves by efficient way and keep a communication channel between all of them, sending messages to specific bot ID. These messages will hop from bot to bot until destination, hiding the source. However recently researchers start to use honeypots to discover this kind of botnet. This honeypots are desirable to an attacker regarding it's location and low level of security. If one of these honeypots gets infected to enter the DHT, the botnet attacker gets a way to start tracking the requests that pass by that machine and try to make patters of message dissemination, being able to estimate the size of the botnet (crawling) or in the worst case scenario compromise and dismantle all the botnet.

3. The Godfather P2P Botnet

The Godfather is a structured Peer-to-Peer Botnet based on DHT algorithm Pastry, totally developed by the authors of this document. The objective is full distributed botnet with secure command dissemination and able to avoid attacks like traitors and crawlers.

We've set the following goals:

- Achieve a secure and untraceable way for peers to enter and exit the network,
- Hide the source of messages such as orders to execute DDOS attacks which are disseminated on DHT
- Be able to rent the services available by the botnet such as CPU Cycles, geographic distribution and network to enable third parties to do their attacks
- Avoid crawlers and Sybil Attacks

The global implementation architecture is the figure 3 of attachment.

3.1. Peer entry

The details of how bots are infected will not be addressed on our solution. We assume that each node is infected has the **binary with *botmaster* certificate.** The infected nodes join the Pastry *DHT* Overlay Network and stays in passive-mode waiting for commands. During infection each peer gets a bot list where he can make the bootstrap to join the network. To avoid network compromise with attacks, such as crawling or Denial Of Service, against the bootstrap nodes, the new node is only accepted on Botnet if it solves a proof-of-work, explained in section 3.4.

The bots keep an update bootstrap list to guarantee that it has an active peer at next bootstrap time (if the machine restarts for example). This bootstrap peers are called Relays. **Relays** are public IP peers, which allow other peers to perform a bootstrap. To bootstrap, the peer requesting must solve a *criptopuzzle*. This creates a huge delay between when the peer knows a new bot and when bootstraps the network. Since *criptopuzzle* are time exhausting task and CPU intensive, it is hard to get to know all the network instantly. The attacker only knows the *IP* of a new peer between long intervals of time, creating a great delay window until crawl the entire network. Even if an attacker creates hundreds of instances on same machine, since this task is CPU Intensive, it would take him a lot of time to get information about all the peers of the network at a given time and taking advantage of the network not being static, the botnet attacker can never have a snapshot of the entire botnet. This doesn't avoid in total the Sybil attack because nowadays computer hardware price is lowering everyday, which gives the attacker the opportunity to have a computing power a lot larger than what is predicted as the common CPU of a normal bot..

The difficult of this proof-of-work has a trade off between the number of false peers (on attacker machine) and the time to entry of true peers.

3.1.1 Twitter and DNS

Facebook and Twitter became a medium to share lot of information. Bot can get information about bootstrap nodes from Twitter if the bootstrap list is empty. The time until detection of this issue and close the Twitter account makes this system reliable.

We could use a DNS domain name to get a bootstrap peer list.

3.2 Secure dissemination of *Botmaster* Commands

All the commands sent by the *botmaster* are signed. Each bot peer has the *botmaster* Certificate in order to check the authenticity of the command (checking the command signature) and avoid attacks against *botnet*. Otherwise, any infiltrator could send a command to attack a specific and protected target where he can listen the messages and trace all the sources.

3.2.1 Peer-to-peer traffic obfuscation

Two main problems of Peer-to-Peer communications are the traffic generated and the message delay. In order to prevent traffic detection, each bot is listening at well-known port like *HTTP*, *DNS* or *FTP*. Usually all firewalls have the *TCP*:80 *HTTP* port open so he can exchange packets through that port.

All traffic exchange is ciphered to prevent detections by Intrusion Detection Systems (IDS).

Typically a hardcoded shared secret and block cipher algorithm like AES or Blowfish does this. Sharing a secret between millions of nodes is not a good policy. Alternatively we could use a SSL/TLS like sharing a session secret key with either peers certificates, or use *Diffie-Hellman* key exchange to provide symmetric encryption with individual sessions.

Because the 3-way SSL/TLS handshake is CPU and network intensive and requests a Certification Authority (CA) to authenticate each peer, increasing this way the network traffic, to avoid this we opted for a simple and more efficient protocol.

When a peer wants connect to a destination, he sends a message with his public key and the destination replies with the destination public key. After this exchange, peers can send cyphered messages to each other. We are aware that this creates a liability to Person-In-The-Middle attacks but since the routing between two peers is not persistent, it is hard for a third malicious peer to execute a Person-In-The-Middle attack. Peer authentication is impossible because there is no certification authority but we don't need to authenticate the peers because the message is signed and has a nonce. If the message is tampered, the sign is invalid. If the message is duplicated, the nonce is discarded. If the message is not send, the node is removed from Accomplice list

The command message is cyphered using the destination public key. This could look slow and inefficient but since we just want send one small message, is more efficient and discrete exchange just 2 messages and cipher with public key.

To avoid more keys exchange between the same peers, we establish session with the other peer keeping the destination public Key and a *nonce* to avoid reply attacks.

4. Peer-to-peer Trust System

In order to prevent attacks against "The Godfather", 1 Key, Credits, TimeLastMsgReceived>. These credits are volatile, i.e. if the node is shutdown or timeout, all his credits are lost because we don't know if the node has been compromised.

Figure 1 (Attachment) shows that peers, which are online more time, also tend to be online more time. Then our *Accomplices list* has a limited size and is sorted by peer credits. If an attacker tries to flood a peer with messages to fill the *Accomplices list* with *traitor* bots, these requests are rejected because they are new. This way, old peers have priority and the attacked bot can boot again.

4.1 Accomplice and Sessions

To avoid Public Key exchange all time, we forward the received messages to our *Accomplice* because we suppose that they will attack too. If our *Accomplice list* has been compromise, the *Accomplice Squad* can just forward to us and not to other nodes. All botnet would be compromised. So we send the message to random Node ID's and to our *Accomplice list.*

To earn credits, *Accomplice* must forward commands to us. If the command is signed by botmaster and this is a new command for us, then the command is valid.

If the signature is not valid, the node will earn negative points. If the same node sends more than 3 invalid messages, he is a traitor and we punish him by taking him out of our *Accomplices List.*

A valid command is not enough because an attacker can emulate hundreds of bots at same machine and forward them always to the same nodes, earning points with each other, this is called "Front-Men attacks". To avoid this situation, we create a proof-of-work.

4.2 Proof-of-Work

To avoid threats like *Sybil* Attacks against our network, we decided to implement a Proof-of-Work system, which gives the opportunity of peers to proof they are valid independent machines by having to resolve a *criptopuzzle* in a certain amount of time.

This doesn't avoid the crawling of DHT but makes it more difficult and slow it down. Bigger is our DHT more difficult is crawling it.

We propose a *criptopuzzle* message exchange to create a proof-of-work. The graphical representation is figure2 of attachment.

Before sending the command to a new peer, the relay peer asks for a proof-of-work before sending the bootstrap peer list. For that it sends a *criptopuzzle* to the new bot (A). First it generate a random string with 53 bytes (424 bits) which we call X:

$X = \{Timestamp\} \mid SHA\text{-}1(K_{PubA}) \mid x$

Where x is a random nonce and the timestamp is the current time signed. This string is hashed using SHA-1 to form a 160-bit string T. This result is signed using $K_{privA}(T)=S$. The k lower ordered bits of x are set to zero to form x'. Then we sent to botmaster:

$\{Timestamp\}, K_{PubA}, x', T, R, k,$

To solve this puzzle, A concatenate TS| $SHA\text{-}1(K_{PubA}) = T$ and starts attempting x'' which$SHA\text{-}1(T|x'') = S$

When this hash is correct, he sends back to relay peer:

Timestamp,R,S,x''

Relay peer will just check if S, if the elapsed time from puzzle generation and solution is valid and if *SHA-1(TS | MD5(*K_{PubA}*) | x'')=S* then A as proven to be a valid bot.

However this way we prevent being attacked by hackers with low resources, which makes the most part of hackers in general.

When peer Sam wants add Tom to his *Accomplice* List, Tom must show to Sam that he works to "Mafia" and the opposite. Our theoretical model needs to exchange 2 messages before exchange the 2 Public Key message. So we implement a new approach. The message "*KeyMsg*" has a criptopuzzle.

5 Monetize model – Botnet renting

The name "*The Godfather*" came from the classic Mafia movie. With the help of asymmetric cryptography, a *botmaster* (Godfather) can take on the role of a trusted certificated authority which provides an efficient way to rent the botnet to third parties, in parts or as a whole for a variable amount of time and for certain purposes. Let us assume some company wants to use the botnet to promote their services using spam mails or attack a competitor causing their services to lose Quality of Service with *DoS* attacks. The *botmaster* can sign a new time and permissions bounded certificate with his private key. This certificate contains the lessee's public key, a timeframe of validity and a list of allowed command classes.

From this point on, the renting party can inject its own self-signed commands into the network via an arbitrary peer. Any bot, which receives such a message, first verifies the included certificate using the loaded *botmaster* certificate. In case of validity, it verifies the distributed command using the lessee's public key obtained from the certificate. If the command belongs to a command class specified in the validated certificate and the timeframe has not yet passed, the command is scheduled or executed and the message bundle (signed command + issuer's certificate) is forwarded.

To protect against malicious lessees, we use a certificate expiration timer and a *blacklist,* which saves all expired certificates and all certificates not signed by *botmaster*. This blacklist is spread to entire network using gossip.

6. Conclusions and Future Work

6.1 Future Work:

We suggest analyzing traffic patterns on a small size botnet using research networks like Planet-Lab. The traffic pattern can show when an attack will happen and that is not good. To hide this pattern, we can keep a constant command flow. One other hand, this flow will be easier to detect by firewall and IDS mechanisms.

6.2. Conclusions

We experimentally investigated several methods to establish a secure and reliable communication between bots, DHT Peers. Due to huge amount of data, the future is Distributed Networks. This overlay networks, like Pastry DHT, must allow secure connections between peers and reliable information sharing.

We realized that attacks in itself, like *DoS*, are easy but having the computational power, network *bandwidth* and security mechanisms to defeat this attack is impossible. We can just mitigate the attackers.

The reason beyond the name "*The Godfather"* for our botnet is that there is a lot of resemblance between mafia and the way a botnet organizes, typically there are the street guys, the bots, which are disposable, who serve the Godfather, the *botmaster*, which we've to make sure to make is identity secret and not associate him with any of the crimes.

Creating a secure way to disseminate command messages to bots, without being caught is archived using signed commands and exchange them between peers using cipher connections.

Because these connections are ciphered, Intrusions Detection System cannot detect them. The signature doesn't show how create this data but insure that no one can change it.

We cannot trust on all the peers that join the DHT. So we must make hard to crawl the system because of time limitation imposed by the *criptopuzzle*. There are lots of mechanisms to attack botnets, such as Sybil attacks. We used mechanisms to counter them, like proof-of work. We study the *criptopuzzle* creation and solving as well alternative mechanisms to join a DHT even part of it was been compromised.

Our implementation provide: bootstrap list, support to generic attack commands, Accomplice System (Credits), Public Key Exchange, Asynchronous message communication, Cipher Communication Layer, Signed Messages, Criptopuzzle, Twitter bootstrap, X509 Certificate Generation, Asymmetric Key Cipher and signature and Symmetric Key Cipher.

Peer-to-Peer botnets like "The Godfather" will be even more usual on future. These new communication architectures are harder to defeat. A proposed botnet like one we stated above can be deployed on real world to perform attacks. The authors and owners of this document and “The Godfather” project will not provide this code to people with bad intentions, is just a research.

Attachments

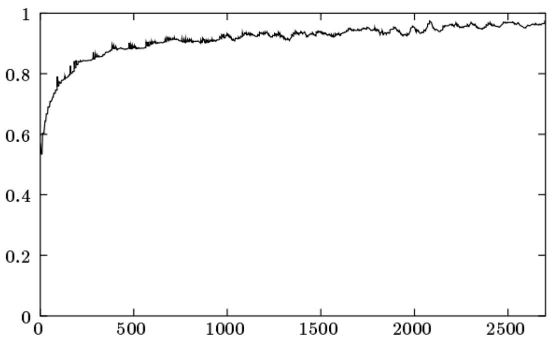


Figure 1.Probability of remaining online another hour as function of uptime. The x-axis represents minutes. The y axis shows the fraction of nodes that stayed online at least x minutes that also stayed online for x+60minutes

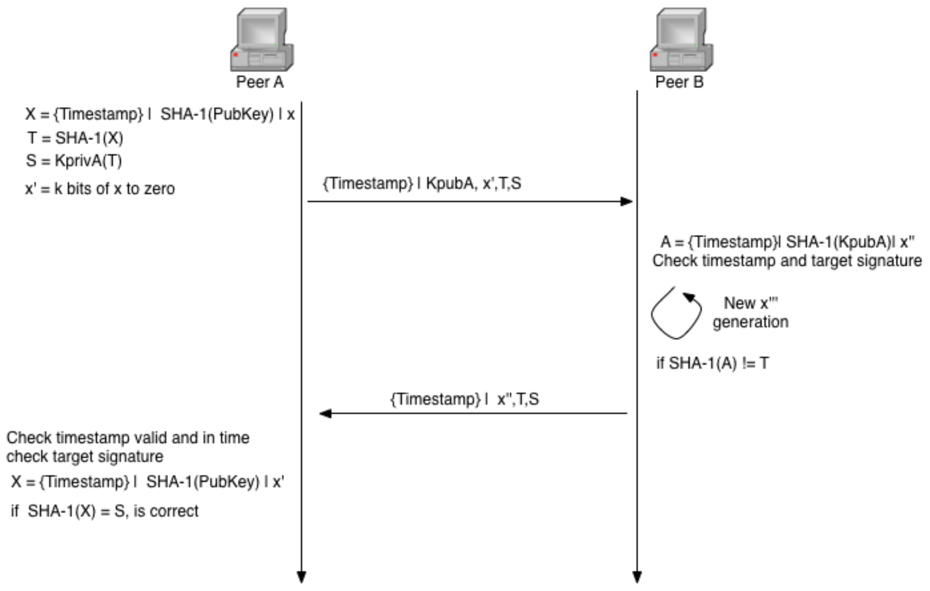

Peer A
Peer B
X = {Timestamp} I SHA-1(PubKey) I x
T = SHA-1(X)
S = KprivA(T)
x' = k bits of x to zero
{Timestamp} I KpubA, x',T,S
A = {Timestamp}I SHA-1(KpubA)I x''
Check timestamp and target signature
New x'''
generation
if SHA-1(A) != T
{Timestamp} I x'',T,S
Check timestamp valid and in time
check target signature
X = {Timestamp} I SHA-1(PubKey) I x'
if SHA-1(X) = S, is correct

Figure 2.Messages exchanged during prof-of-work

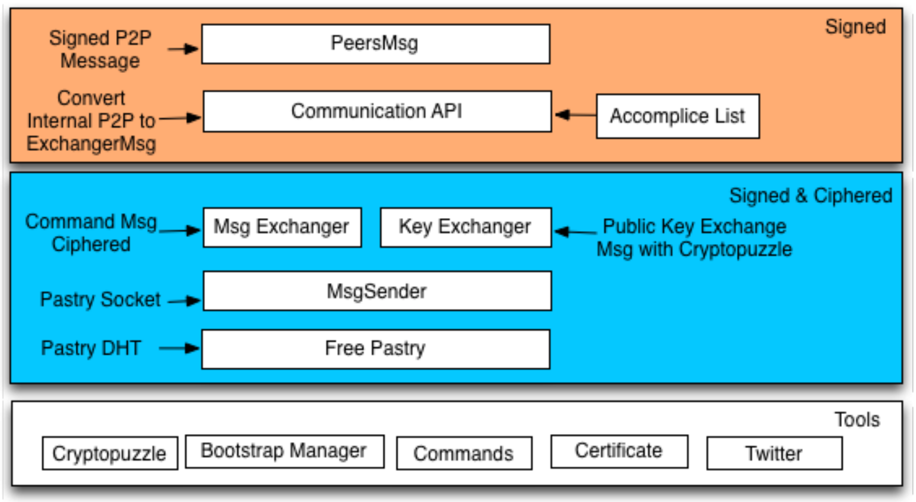
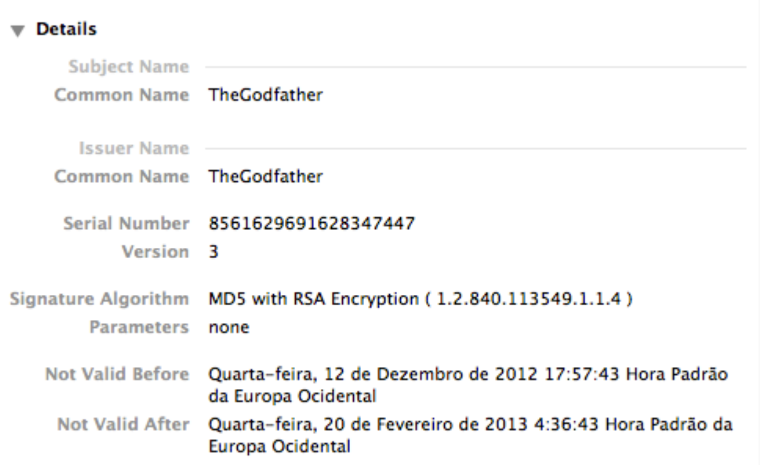


Figure 3.Software Architecture



Details

Subject Name	
Common Name	TheGodfather
Issuer Name	
Common Name	TheGodfather
Serial Number	8561629691628347447
Version	3
Signature Algorithm	MD5 with RSA Encryption (1.2.840.113549.1.1.4)
Parameters	none
Not Valid Before	Quarta-feira, 12 de Dezembro de 2012 17:57:43 Hora Padrão da Europa Ocidental
Not Valid After	Quarta-feira, 20 de Fevereiro de 2013 4:36:43 Hora Padrão da Europa Ocidental

Figure 4.Signed Root Certificate

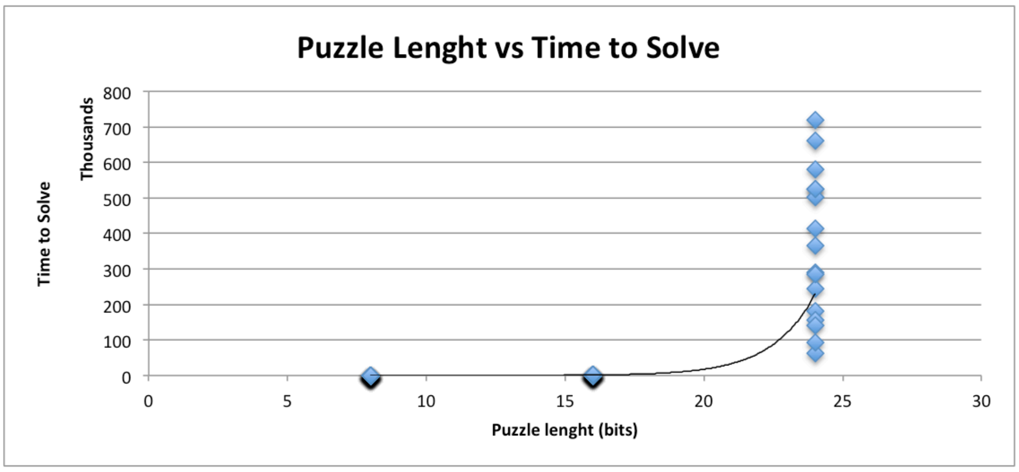


Figure 5. Time to solve a criptopuzzle