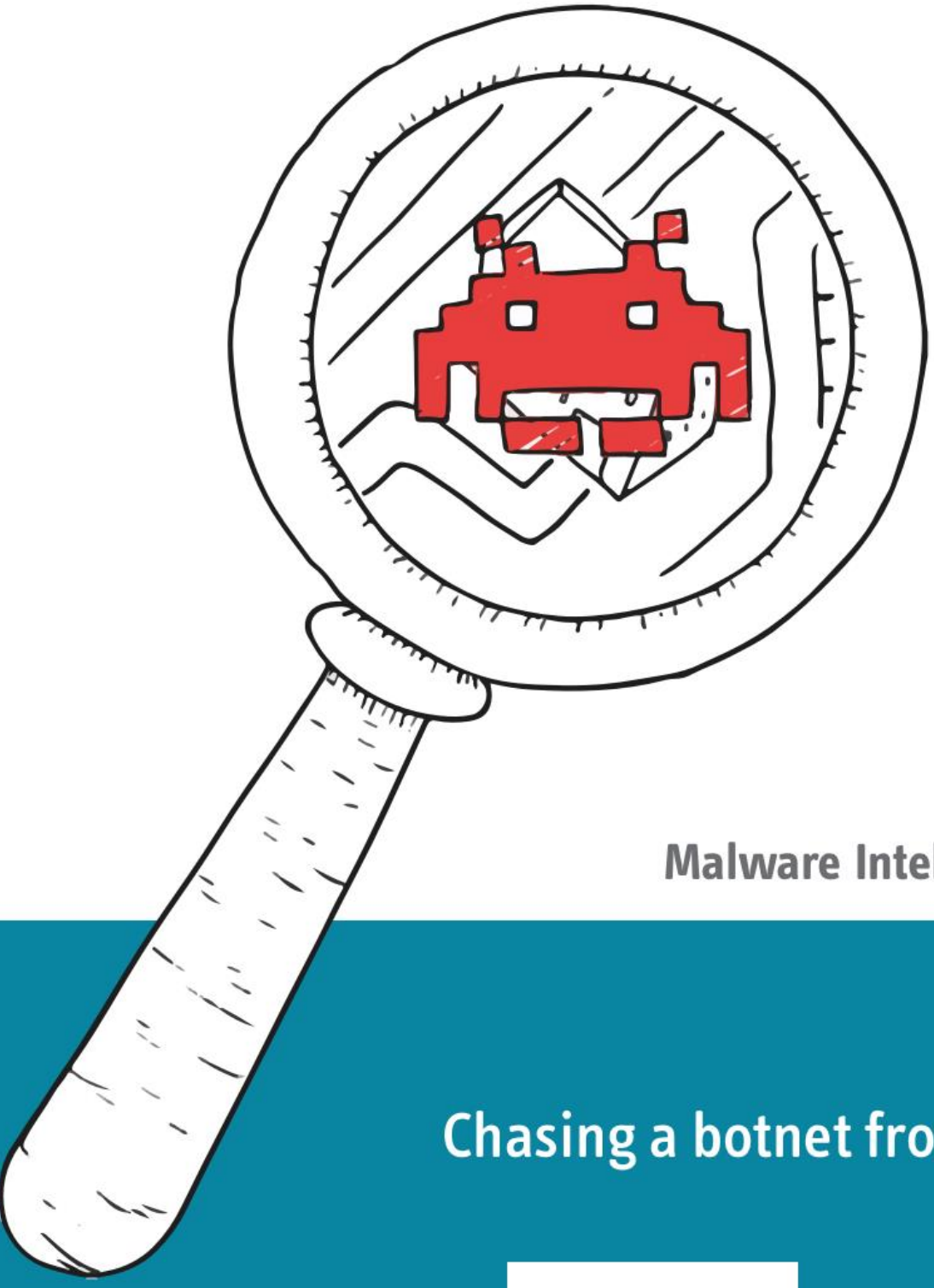
anubisnetworks™
a BITSIGHT company

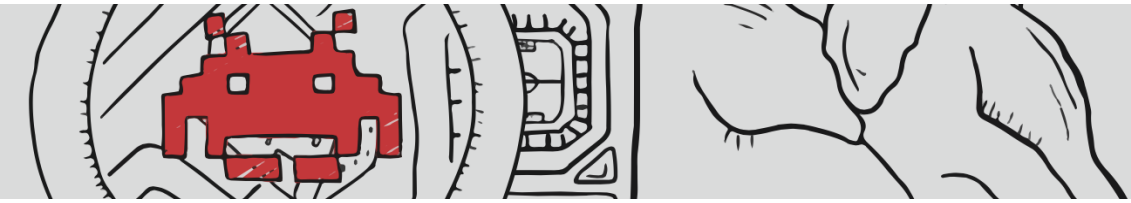


Malware Intelligence Report

Dridex

Chasing a botnet from the inside

TLP-WHITE
PUBLIC RELEASE

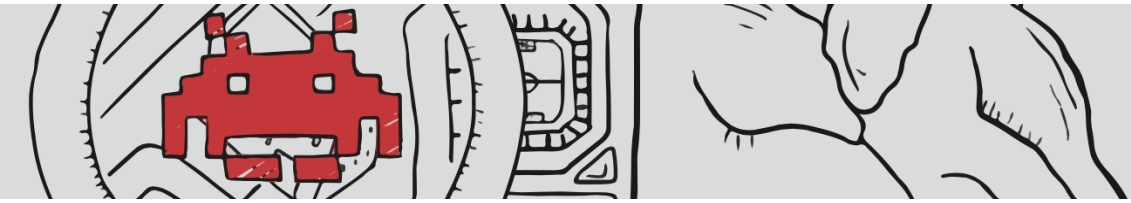


anubisnetworks™
a BITSIGHT company

Version Information

Version	Date	Description
01	2015-08-18	Initial version.
02	2015-10-14	Minor editing and audience scope.
03	2015-10-15	Public release.

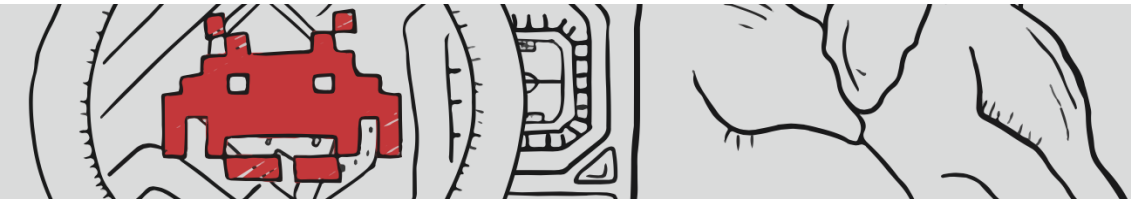
This report was prepared by AnubisNetworks Labs.



anubisnetworks™
a BITSIGHT company

Table of Contents

Contents



anubisnetworks™
a BITSIGHT company

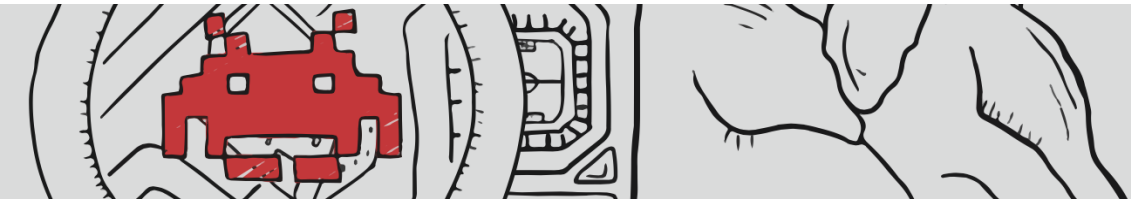
Executive Summary

In March 2015, AnubisNetworks Labs team started analyzing multiple malware samples of the Dridex family in order to:

1. Map the infections of associated botnets;
2. Understand the complexity of its communication channels;
3. Enumerate vulnerabilities that could be exploited.

From our research we have concluded the following:

1. Dridex deploys a hybrid peer-to-peer network with different layers in order to achieve resilience against a takedown;
2. The Dridex ecosystem is constituted by a small number of independent botnets, that have different geographies and financial institutions as targets;
3. Dridex is currently being used to obtain credentials from a large number of online services including several banking services of multiple countries;
4. Dridex P2P network protocols have vulnerabilities that allow interception of bot communications, and may be used to disrupt or takeover the botnet;
5. By reversing the Dridex communication protocols, we were able to deploy a rogue node that can eavesdrop on botnet communications between bots and supernodes (admin_nodes), enumerate bots and map the infection dispersion of the botnets



Scope

This paper describes in detail the communication channels of the Dridex malware once it's up and running on the infected systems, namely its P2P network, encryption methods and associated C2 infrastructure. Our intent is to focus on how the malware communicates with their command and control and to map possible vulnerabilities on the protocols it uses to support monitoring capabilities.

There is no intent to document its distribution mechanisms nor the behavior of the malware on the infected systems in great detail, except for network communications and operations that support those communications (e.g., encryption). Infection dispersion of the many botnets associated with Dridex is also out of scope of this document.

There are no attribution indicators on this document.

Dridex Overview

Dridex has been around since November 2014 and is an evolution of the malware families known as Bugat, Geodo, Feodo and Cridex.

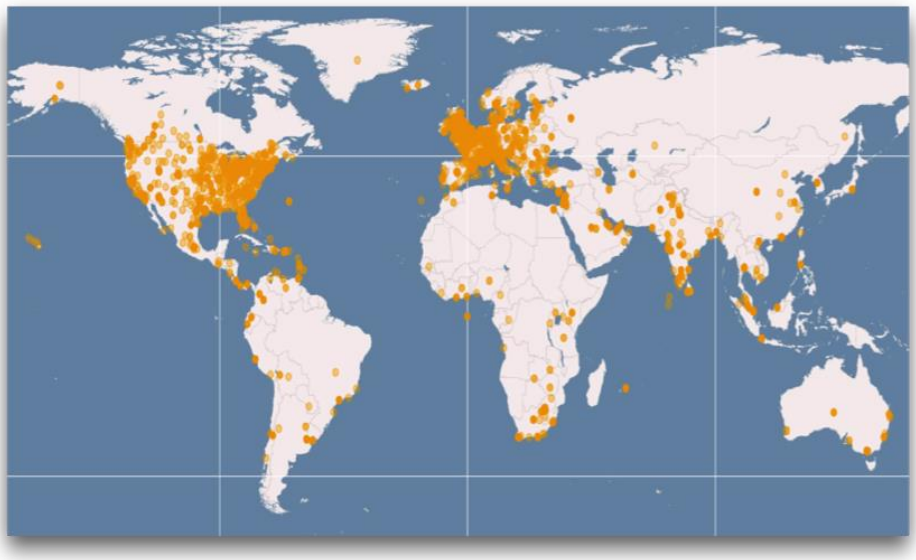
The malware is distributed via email, with a malicious Microsoft Word document as attachment, that once opened downloads a second stage payload that infects the system.

Primarily targeting home banking users, is a malware with various capabilities including man in the browser, keylogger, proxy and VNC. Features a peer-to-peer (P2P) network and uses cryptography on its communication channels.

The Dridex ecosystem is constituted by eight (8) botnets, as tagged by their operators. From those botnets, the ones identified as 120, 125, 200, 220 and 320 seem to be the main botnets and 121, 122 and 123 seem to be aliases for botnet 120, or at least segregated from 120 at a logical level only, since they shared the same command and control systems.

Currently, botnets 120, 200 and 220 are the most active, with predominant infections in Europe, North America and Asia.

Dridex botmasters are very active, launching new campaigns against different geographies and hardening the botnet infrastructure with new countermeasures and command and control systems.



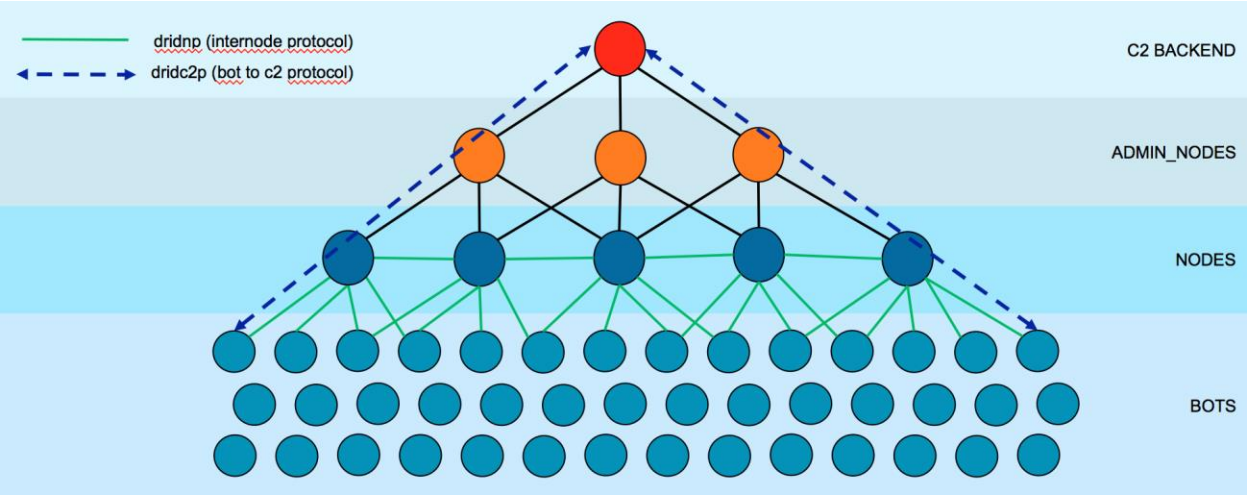
Dridex 220 distribution (sampled)

Peer to Peer Network

The botnet uses a hybrid P2P network to communicate. There are different types of nodes and different protocols that are used. This section details each of these.

Network Structure

The network has the following structure:



It is composed by the following elements:

- **bots -** Bots are the most common network element. These are the infected systems that are not directly connected to the Internet, have firewall active or NAT;
- **nodes -** Nodes are infected systems that are directly connected to the Internet have no firewall and no NAT and have been successful promoted to this layer;
- **admin_nodes -** Admin nodes appear to be composed by compromised servers. These act as a proxy layer between the C2 backend and Nodes;
- **servers -** Servers are the initial C2 communication point used by the malware. These are hardcoded in the malware files and the malware communicate with them to obtain an initial list of peers;
- **c2 backend -** The command and control backend operated by the botmasters. Notice that we have not seen this layer of the network so this component is speculated to exist.

Network Communications

The network protocols and encryption schemes have changed over time. This document only covers the most recent scheme we are aware of, which has been in use since 30th April 2015.

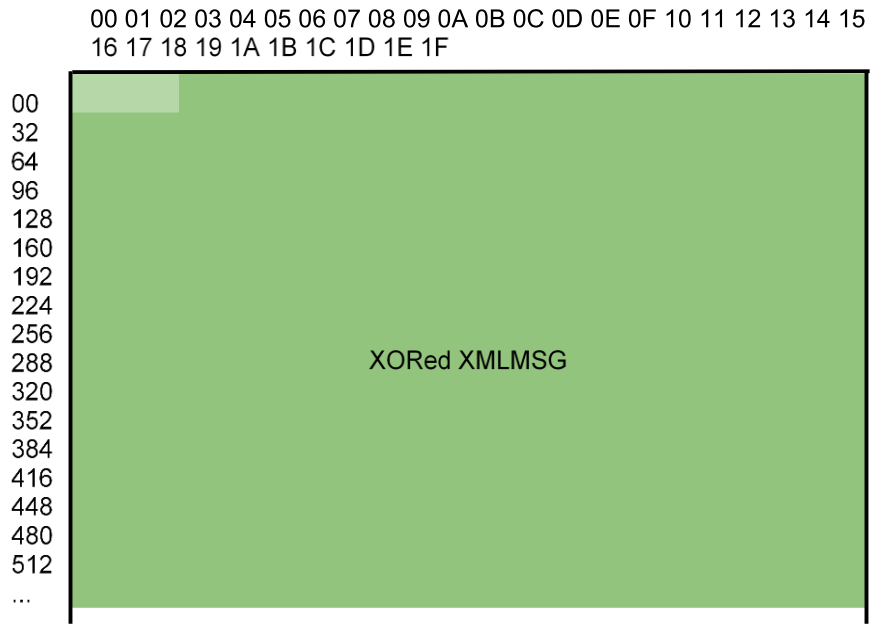
Network Protocols

Dridex uses XML messages for communication between all of its network elements. However, these messages are wrapped in different encryption protocols that are used depending on the context of the communication. The following protocols are used by Dridex when sending the XML messages:

- **Server communication protocol** - Used during the infection of a system, to obtain the initial list of nodes in the botnet;
- **Internode communication protocol -** Used when the communication does not contain information relevant to the C2, for instance when messages are used solely to maintain the structure of the network;
- **C2 communication protocol** - Used when the message needs to reach the C2 backend, for instance when sending stolen data or requesting new commands.

Server Communication Protocol

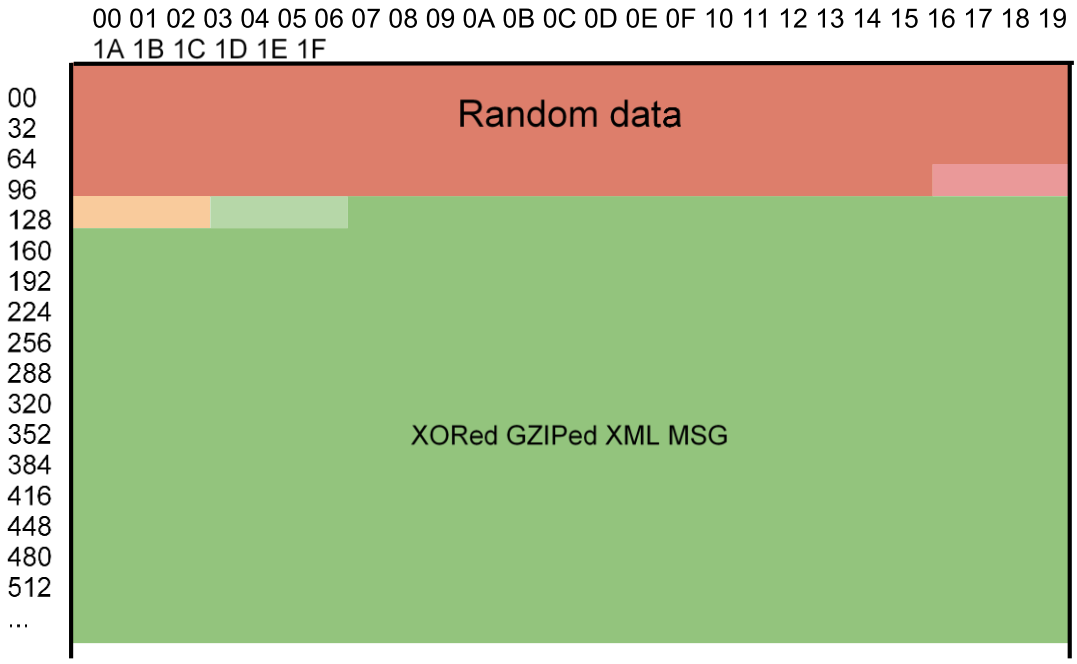
The communication to the servers is transmitted in HTTP over SSL. The body of the HTTP request contains a 4 byte XOR key and the XOR encrypted XML message. The following image shows the HTTP body structure:



- **XOR Key (4 bytes)** - The Key used to XOR encrypt the data;
- **XORed XMLMSG** - The XML Message encrypted through XORing each 4 bytes of the message with the key;

Internode Communication Protocol

The internode communication is transmitted directly through TCP sockets and uses the following structure:

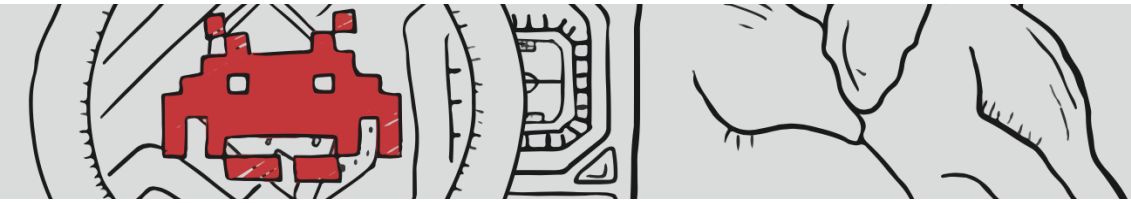


- **Random data (124 Bytes)** - Random data generated using the Windows API function CryptGenRandom;

- **Checksum (4 bytes)** - A checksum result of the random data. The checksum is the reverse of the sum of the 31 4 bytes Integer values contained in the 124 bytes of random data; The following is an example implementation in python:

```
struct.pack(">I",-uint32(sum(struct.unpack('<31I',rnd1))))[::-1]
```

- **Datalen (N bytes)** - The data length in a special format stored in two Signed Short Integer (2 byte) numbers. The first short contains the reverse of the length of the data



divided by 30000 and the second part contains the reverse of the remainder of the same division. If the result of the division is smaller than one, a random two byte that results in a negative value is used for the first short. Following is an example implementation in Python:

```
datalen=len(xoredpayload+xorkey)
hpart=datalen/30000
lpart=datalen%30000
if (hpart):
   hpartbytes=struct.pack('<h',-hpart)
else:
   hpartbytes="\x00\x00"
if (lpart):
   lpartbytes=struct.pack('<h',-lpart)
else:
   lpartbytes="\x00\x00"
lenbytes=hpartbytes+lpartbytes
```

- **XOR Key (4 bytes)** - The Key used to XOR encrypt the data. This is a random 4 byte key obtained through the Windows API CryptGenRandom function;

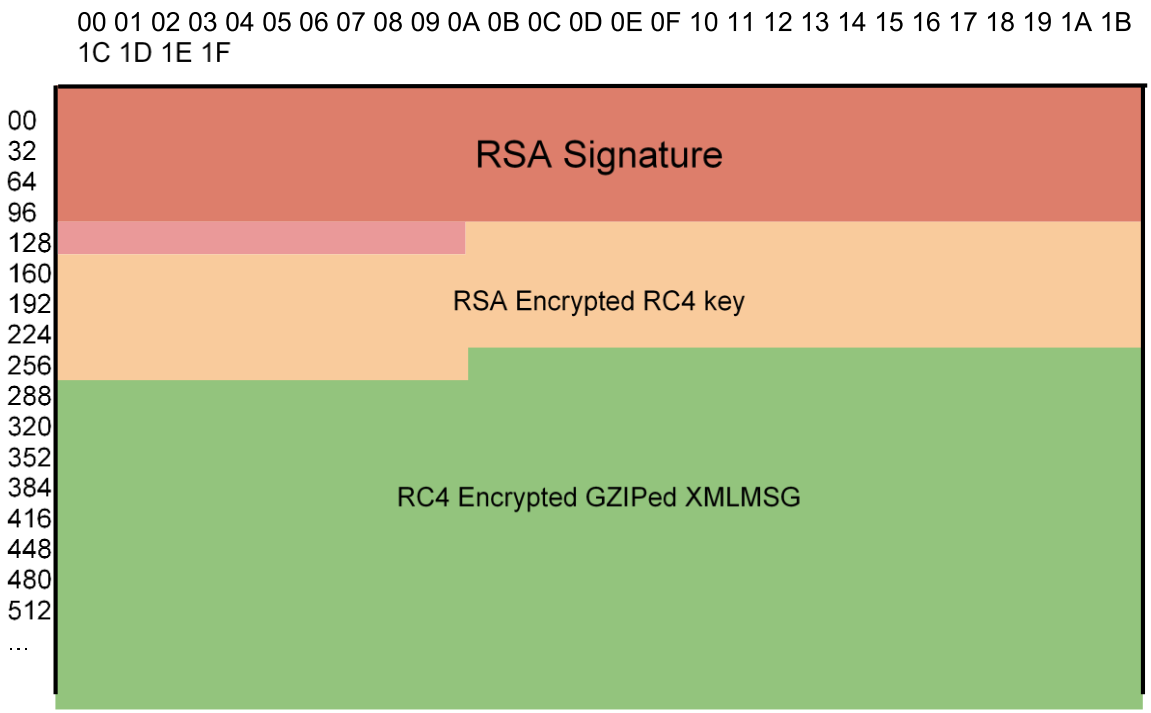
- **XORed GZIPed XMLMSG** - The XMLMSG to be sent to the node is compressed using gzip format. Each 4 bytes of the compressed data is then XORed with the XOR key.

C2 Communication Protocol

The C2 communication is transmitted through HTTP over SSL (the bots do not check the validity of the SSL certificate provided).
Whenever a bot sends a message to the C2, the HTTP request contains a special header that identifies the Unique ID of the requesting bot. The following is an example of the HTTP header of a request sent by a bot to the C2:

```
Id: ABCDEFG_12345678901234567890123456789012
```

The HTTP request body contains the XML message encrypted in the following format:



- **RSA Signature (128 bytes)** - The RSA Signature of the remaining of the HTTP request body using the Bot RSA Private Key that was generated locally;

- **Microsoft SimpleKey Blob Header (12 bytes)** - The header of the Microsoft SimpleKey Blob. This is part of the format that Microsoft Crypto Library uses to exchange encryption keys. In this case, the following bytes are used:

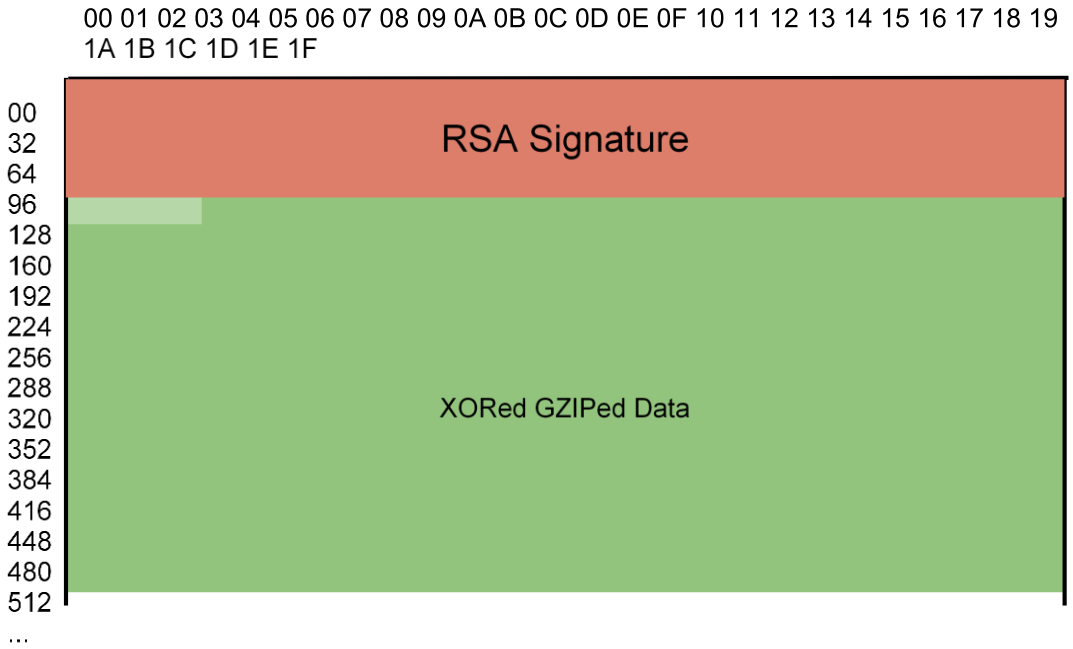
```
0x010200000168000000a40000
```

- **RSA Encrypted RC4 Key (128 bytes)** - A Random RC4 key is generated for each message being sent and encrypted using the C2 public key that is hardcoded in the malware files.

- **RC4 Encrypted Compressed XMLMSG** - The XMLMSG to be sent to the node is compressed using GZIP format. The compressed XML is then encrypted using the RC4 Key that was generated for this message;

Info Protocol

Some information is base64 encoded and sent inside the XML messages. This is key information (e.g. software modules, initial node list, Info XML Message contents) which is double encrypted and signed by the C2 private key. After decoding the base64, it's contents are stored in the following format:



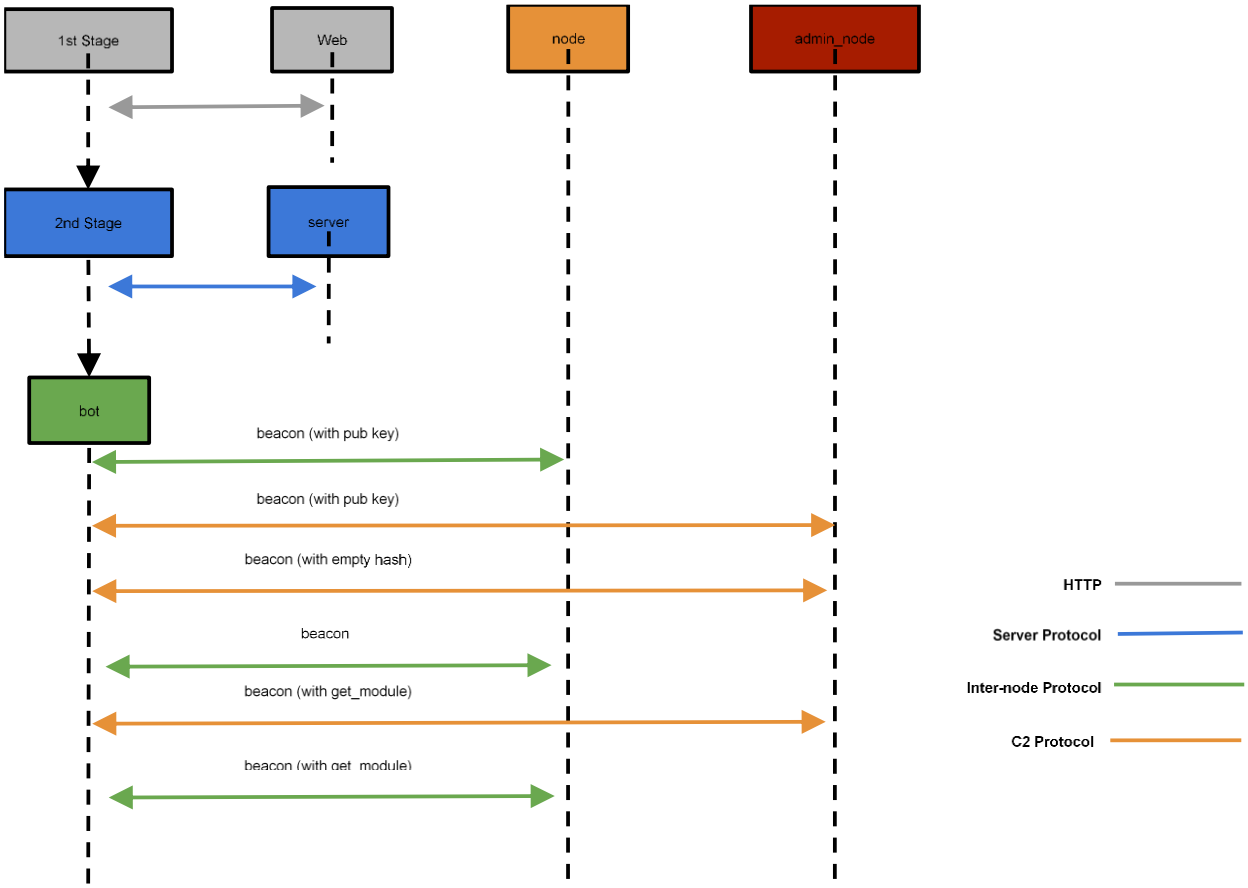
- **RSA Signature (128 Bytes)** - RSA Signature of remaining payload (XOR key and XORed payload);
- **XOR Key (4 bytes)** - The key used to XOR encrypt the data, this is a random 4 byte key;
- **XORed GZIPed data** - The data to be sent to the node is compressed using gzip format. Each 4 bytes of the compressed data is then XORed with the XOR key;

Communication Flows

This section describes the typical communication flows that occur when a bot is first infected, when it tries to escalate to node and after infection while it is periodically beaconing to the C2.

Joining the Network

The following diagram shows the sequence of communications that occur during the infection of a system until it has fully joined the Dridex peer to peer network.



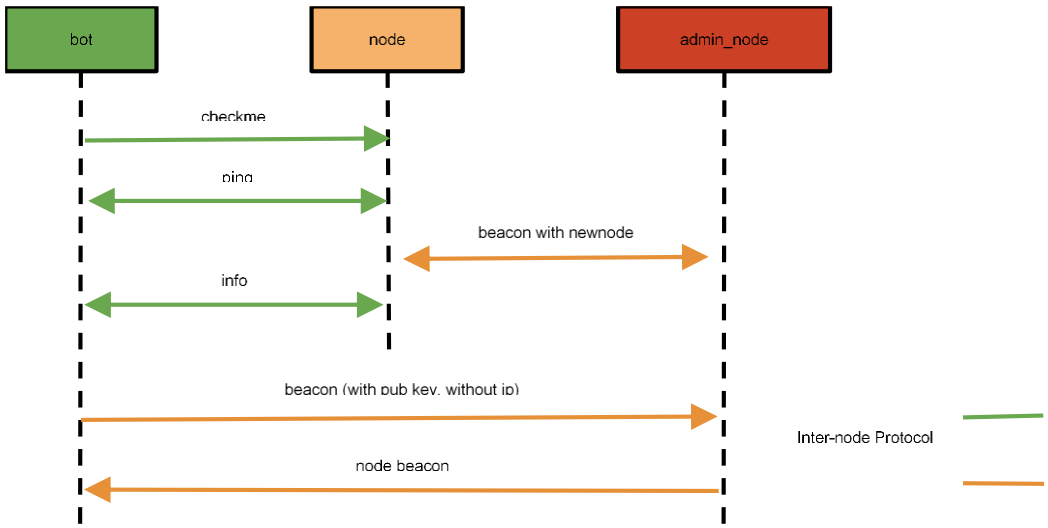
When a new system is infected it performs the following actions (the contents of the different XML messages will be detailed in the following section):

1. The first stage, a malicious Microsoft Office document, downloads via a regular HTTP request the second stage malware from a compromised web server;
2. The second stage malware connects to its hard coded servers using Dridex Server protocol (described earlier in this document) and sends a "*loader*" XML message, requesting the third stage malware/bot DLL and the initial list of nodes;
3. The bot connects to one of the nodes in order, from first to last of the list, until one responds correctly and sends a "*beacon with pub key*" XML Message that results in a response that contains the external IP address of the bot;
4. The bot sends a "*beacon with pub key*" XML message using C2 communication protocol that contains the IP address received from the node and the bot's RSA public key;

5. The bot sends a *"beacon with empty hash"* XML message to the C2 that shows an empty config file hash value. In response, the C2 sends the latest config file, the commands it wants the bot to execute, and the most recent list of nodes;
6. The bot sends a *"beacon"* XML message to the node. In response, the node replies with a list of the available modules;
7. The bot sends a *"beacon with get_module"* XML message requesting some of the available modules from the C2;
8. The bot sends a *"beacon with get_module"* XML message requesting some of the available modules from the node;

Escalate to Node

Once a bot has registered into the network (after it has successfully sent the public key to the server and received a positive reply) it starts the process of elevation to node, in parallel with the remaining requests related to obtaining the config file and the available modules. This process will allow bots that are not behind a NAT to work as nodes in the P2P network. The following diagram shows the sequence of communications that occur during the escalation process:



The following communications occur during this process:

1. The bot sends a "*checkme" XML message* to three randomly selected nodes from the node list and repeats this process until all of the nodes are tested;
2. Each node that receives the "checkme" request will try to connect to the bot in the TCP port listed in the "*checkme" XML message* and send a *"Ping" XML message*;
3. If the node receives a "*Pong" XML message*, in reply to "Ping" it will send a "beacon with newnode" XML message to the C2 warning that a new potential node was detected;

4. The C2 will respond with an appropriate "*Info" XML message* to be sent to the new node. This message will typically contain the admin_node to be used by the new node;

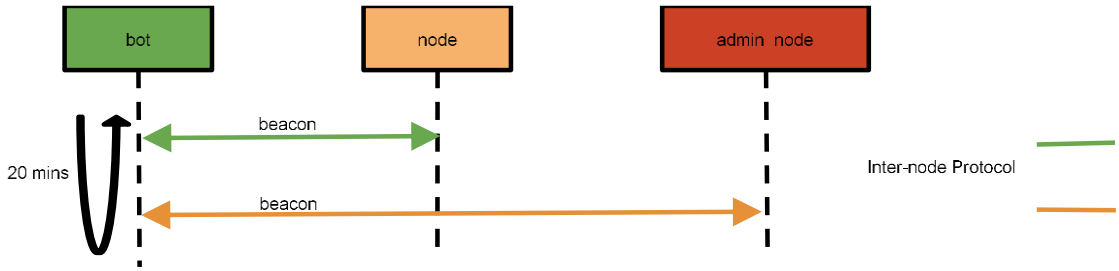
5. The node will connect to the bot and send the "info" XML message;
6. The bot will connect directly to the admin_node and send a *"beacon with pub key"* XML message, without IP, containing the bot's public key;
7. Finally, the new node will connect directly to the admin_node and send a "*node beacon*" XML message to the admin containing the type node XML field;

Periodic Checkin

After a bot successfully registers into the network, tries to escalate to node, downloads the config file and downloads all the available modules, it periodically checkin to the C2 to:

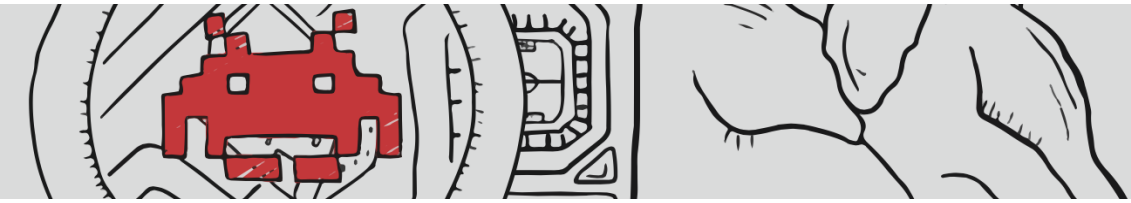
- Send any command output results;
- Send stolen data;
- Check for updates to config files;
- Check for updates to the modules.

The following diagram shows the communications that occur periodically after the infection:



Once an infected system has registered itself as a new bot, it starts the node escalation process by performing the following actions:

1. The bot sends a *"beacon"* XML message to the node. In reply it can receive updated IP address if the bot IP address has changed and new versions of modules; If a new module or new version is received the bot will send a message requesting the new module;
2. The bot sends a *"beacon"* XML message to the C2. This message can contain a <data> field containing stolen data, command outputs and debugging messages. The reply can contain updated configuration files, additional commands, updated node lists or updated admin_nodes list.



Message Formats

Dridex uses the protocols described before to transmit and receive a set of XML messages. These messages and their responses take the bots through a set of states that allow the botnet to function. These messages are divided in the following categories:

- Server messages;
- Node messages;
- C2 messages.

Most of these messages share a set of common attributes. The most relevant are:

- **unique**: The bot unique ID , built from the computer name and a hash based on a registry key value, with the following format: <PC-NAME>_<md5 hash>;
- **botnet**: The botnet where the bot belongs to. Each botnet is a separated Dridex P2P network with separate configs, modules, bots, nodes and admin nodes.
- **version**: The software version of the bot module;
- **system**: The operating system version;
- **type**: The role of the infected system (bot or node).

Server Messages

The following messages are sent between the second stage component of a Dridex install and one or more of its hardcoded servers. These will use the Server communication protocol, described earlier in this document, for encryption and transmission.

Loader

When the second stage malware is loaded into the system it connects to a list of hardcoded IP addresses (servers) and downloads the bot and the node list.

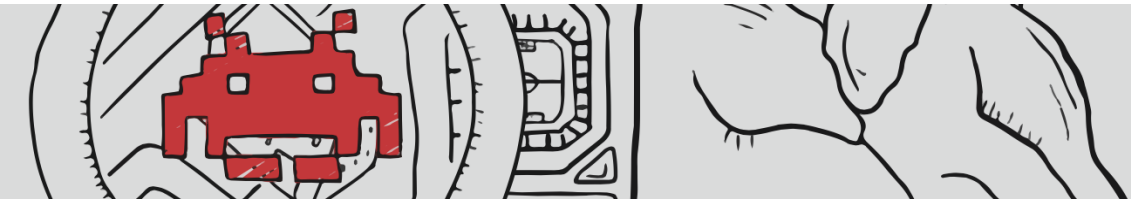
Following are the examples of the request and response:

REQUEST

```
<loader><get_module unique="ABCDEFG_12345678901234567890123456789012" botnet="200"
system="56384" name="bot" bit="32"/><soft><![CDATA[]]></soft></loader>
```

RESPONSE

```
<root><nodes>T8JR2C+UMVGlsgJ2SRVSz2FgrzM2WMfGK3hR1RiZCs4WsX2+kZtgOC4OHPfIooR9l/mJWW
iFeSZDCPB/zrIX5Bw+Y47mhGm8wrSoY5m5BMFzaKevVKkE8rSATmGuNPjAT/vXUAOaz2a91pwYbSxSreap/
YF8Ippz3V+TFxl6tcxKIV2qVapVqlWpGCUepFebWel6Lmln8jlB/yawjfhcEuqaTLx5GJ0ZRo2mE4NRzSUg
EeqkigvPOjR6c305PVkCxODSXndpHIoUIYyVRWSwvd3JcRI+fT8ZlpDXMS+BRT5XGGLiEw8lfMzBm5BofPW
```




```
ETeDw2Ci6hUmjWZlHadA7X5etcVN1dJyqVao=</nodes><module name="bot"
bit="32">iMAlP+gf5ujehA6EsHVySqflek7YqX3+QCR6xcDb1LPgus4jXEhmT0Z21y2uLLginEA8gKBFFr
GdXDTSnz3F/qZc0oV2lhf3ul4pb5HVe0spkF0FGohUnvwxXTPW20Az+tk+jHU3XtQMnm6++6jk4P1jFJRVW
noGuaFj+VPXxodNWpAAAwAAAAQAAAD//wAAuAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAADYAAAADh+6DgC0Cc0huAFMzSFUaGlzIHByb2dyYW0gY2Fubm90IGJlIHJ1biBpbiBET1MgbW9
...SNIP...
```

The request sends the bot unique ID, which will be used in all future communications with the botnet and the list of software that is installed in the system.
The response contains the nodes and the bot module for the system architecture. Both the node list and the module are encrypted using the Info protocol described earlier.

Node Messages

The following messages are sent between a bot and a node or between nodes. These will use the Internode communication protocol, described earlier in this document, for encryption and transmission.

Beacon

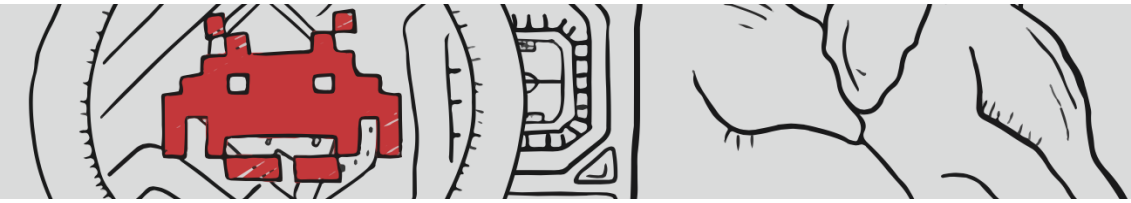
The beacon message will be sent by a bot to a node in the periodic checkin that is made every 20 minutes after the initial infection. It does not include the public key and includes the bots last known external IP address. The response will include the nodes known modules and the IP address of the bot if it is different from the one sent on the request.
Following are the examples of the request and response:

REQUEST

```
<root>
   <unique>ABCDEFG_12345678901234567890123456789012</unique>
   <botnet>220</botnet>
   <version>196637</version>
   <system>64584</system>
   <type>bot</type>
   <ip>1.2.3.4</ip>
</root>
```

RESPONSE

```
<root>
<ip>1.2.3.4</ip>
<modules>bot_x32:b7365e711500550df99f254feac9d310,bot_x64:31f5073565351e1ba825d6a3b
ceac295,socks_x32:4b56a817f184c8536ed812d31bafd61b,socks_x64:da4c598b35d30ac60c3000
28be58021d,vnc_x32:863226dc453c35cd5880fcb2a858812e,vnc_x64:3fb6f34d7cf994abbcecb6d
679cc76d7</modules>
</root>
```



Beacon with pub key

The beacon with pub key message is sent by an infected system to a node, the first time a bot communicates with a node. This message differs from the regular beacon in that it contains the bot's public key and it does not contain the bots external IP address.

This message will result in the following information being sent by the node:

1. The bot current external IP address;
2. The most recent module list known by the node.

Following are the examples of the request and response:

REQUEST

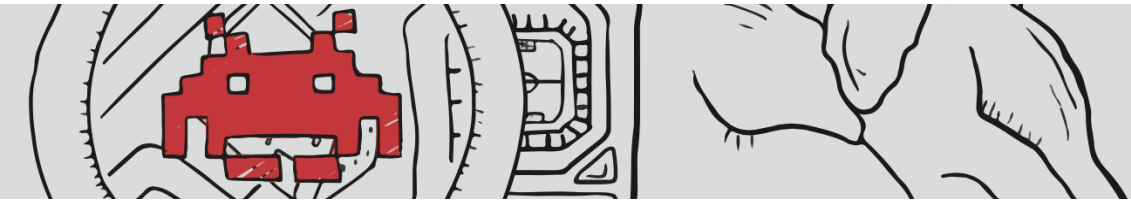
```
<root>
   <unique>ABCDEFG_12345678901234567890123456789012</unique>
   <botnet>220</botnet>
   <version>196637</version>
   <system>64584</system>
   <type>bot</type>
   <public>-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCzmIn0VTR06xaUCBc+DAOc1w/c
r9X9sTdFmhqjIfR6RpmsBnjSyvUml3bUD0F+zJCKjHs0+qfsEg5jbcUWy2pxsRZY
0P0FkFvbhAcU6XOzxu6+mfil8W7T+eqi8xPk0NlgyoNfNBDrmOTs9V/gS0xWxYRu
jTXX3qOGm+uIsyDeAwIDAQAB
-----END PUBLIC KEY-----</public>
</root>
```

RESPONSE

```
<root>
   <ip>1.2.3.4</ip>
<modules>bot_x32:b7365e711500550df99f254feac9d310,bot_x64:31f5073565351e1ba825d6a3b
ceac295,socks_x32:4b56a817f184c8536ed812d31bafd61b,socks_x64:da4c598b35d30ac60c3000
28be58021d,vnc_x32:863226dc453c35cd5880fcb2a858812e,vnc_x64:3fb6f34d7cf994abbcecb6d
679cc76d7</modules>
</root>
```

Beacon with get_module

The beacon with get_module message is used to request a software module from a node. The nodes store and transmit software modules, which allows to avoid excessive load on the C2 infrastructure with traffic relating to software module distribution.



Following are the examples of the request and response:

REQUEST

```
<root>
   <unique>ABCDEFG_12345678901234567890123456789012</unique>
   <botnet>220</botnet>
   <version>196629</version>
   <system>23128</system>
   <type>bot</type>
   <ip>1.2.3.4</ip>
   <get_module name="vnc" bit="32" />
</root>
```

RESPONSE

```
<root>
   <ip>1.2.3.4</ip>
<modules>bot_x32:b7365e711500550df99f254feac9d310,bot_x64:31f5073565351e1ba825d6a3b
ceac295,socks_x32:4b56a817f184c8536ed812d31bafd61b,socks_x64:da4c598b35d30ac60c3000
28be58021d,vnc_x32:863226dc453c35cd5880fcb2a858812e,vnc_x64:3fb6f34d7cf994abbcecb6d
679cc76d7</modules><module name="bot"
bit="32">iMAlP+gf5ujehA6EsHVySqflek7YqX3+QCR6xcDb1LPgus4jXEhmT0Z21y2uLLginEA8gKBFFr
GdXDTSnz3F/qZc0oV2lhf3ul4pb5HVe0spkF0FGohUnvwxXTPW20Az+tk+jHU3XtQMnm6++6jk4P1jFJRVW
noGuaFj+VPXxodNWpAAAwAAAAQAAAD//wAAuAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAADYAAAADh+6DgC0Cc0huAFMzSFUaGlzIHByb2dyYW0gY2Fubm90IGJlIHJ1biBpbiBET1MgbW9
kZS4NDQokAAAAAAAAACWTSP5h8iatYfImrWHyJq1GNFutYPImrUY0Xa1j8iatov17rXDyJq1h8ietBvImrW
```

Checkme

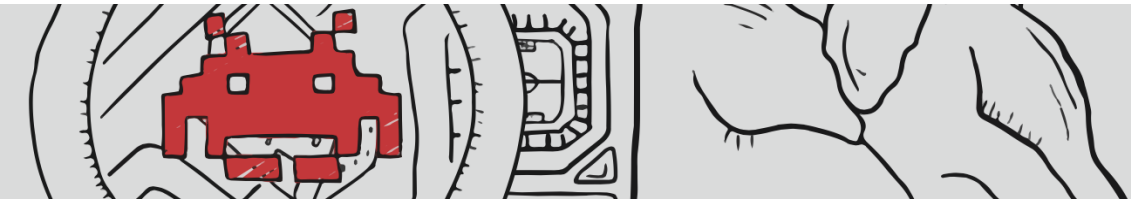
The checkme message is used by a bot, during the escalation to node process, to request an existing node to check if it has a direct internet connection (no firewall and no NAT) and is eligible to become a node.
Following are the examples of the request and response:

REQUEST

```
<root><checkme name="ABCDEFG_12345678901234567890123456789012" host=""
port="443"/></root>
```

RESPONSE

```
<404>
```



Ping

During the escalation to node process, after a node receives the checkme request from a bot, it will try to send a ping request to the port specified on the checkme message. If it receives a pong message than the bot is eligible to become a node.

Following are the examples of the request and response:

REQUEST
<root><ping name="ABCDEFG_12345678901234567890123456789012" /></root>

RESPONSE
<root><pong name="ABCDEFG_12345678901234567890123456789012" /></root>

Info

The info message is sent to new nodes informing them of the admin_nodes they should start contacting.

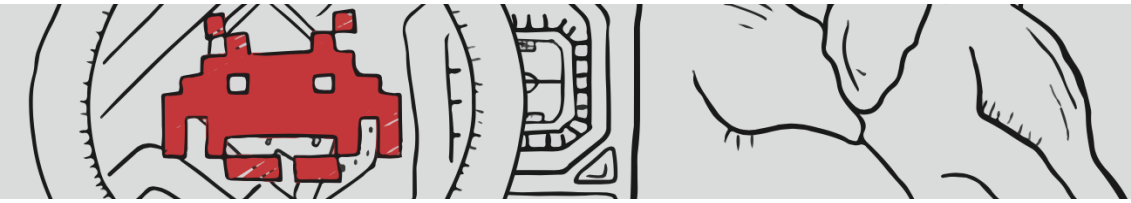
Following are the examples of the request and response:

REQUEST
<root><info>T8JR2C+UMVGlsgJ2SRVSz2FgrzM2WMfGK3hR1RiZCs4WsX2+kZtgOC4OHPfIooR9l/mJWWiFeSZDCPB/zrIX5Bw+Y47mhGm8wrSoY5m5BMFzaKevVKkE8rSATmGuNPjAT/vXUAOaz2a91pwYbSxSreap/YF8Ippz3V+TFxl6tcxKIV2qVapVqlWpGCUepFebWel6Lmln8jlB/yawjfhcEuqaTLx5GJ0ZRo2mE4NRzSUgEeqkigvPOjR6c305PVkCxODSXndpHIoUIYyVRWSwvd3JcRI+fT8ZlpDXMS+BRT5XGGLiEw8lfMzBm5BofPWETeDw2Ci6hUmjWZlHadA7X5etcVN1dJyqVao=</info></root>

RESPONSE
<200>

The body inside the info tag is a encoded using the Info protocol described earlier and contains the admin_node to be used by the bot in the following format:

```
<root><admin_node>1.2.3.4:443</admin_node></root>
```



C2 Messages

The following messages are sent between a bot and the C2. These will be sent through a node using HTTPS protocol containing a HTTP header that includes the bot unique ID and will be forwarded by the node to the admin node without being inspected. The body of the HTTP requests will be encrypted using the C2 communication protocol described earlier in this document.

Beacon

The key component of messages sent to the C2 is the beacon. After the bot initial infection and escalation process, the bots periodically check in to the C2 by sending a beacon as the following. If the C2 has a config file with a hash that is different from the hash sent by the bot, the new config file will be returned in the response.

REQUEST

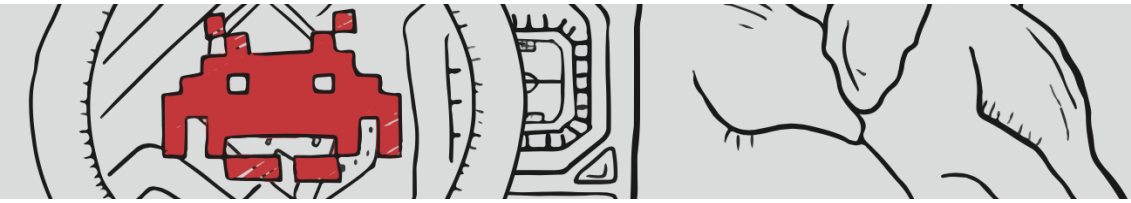
```
<root>
   <unique>ABCDEFG_12345678901234567890123456789012</unique>
   <botnet>220</botnet>
   <version>196629</version>
   <system>23128</system>
   <type>bot</type>
   <ip>1.2.3.4</ip>
   <hash><![CDATA[37d0d82c2c7b156c82156d3aee6d23bca7faf052]]></hash>
</root>
```

RESPONSE

```
<root>
</root>
```

Beacon with pub key

A bot will send the beacon with pub key message when it first communicates with the C2 during the register process. This beacon contains the public key of the bot's own key pair and does not yet contain a config file hash. In response it will receive a public key. However, the received public key does not seem to be used in any communication encryption or signature validation routines.



REQUEST

```
<root>
   <unique>ABCDEFG_12345678901234567890123456789012</unique>
   <botnet>220</botnet>
   <version>196629</version>
   <system>56392</system>
   <type>bot</type>
   <ip>1.2.3.4</ip>
   <public>-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNsnbaFc+kXX3IDA2tmGZClurw
HkM3wO7Ea9t6S3QRsQItDdvGvZofyRsNdiUnNdV3bagapJbA0LMcsOWS3LBCIgtH
Slfw4eYlcOLGMIlPyy7MqVPwQKrtlDprUGflerSpLGkzOdtM8bVqmHRi1Rm2PARy
nMelCdnmy4fQ8dashQIDAQAB
-----END PUBLIC KEY-----</public>
</root>
```

RESPONSE

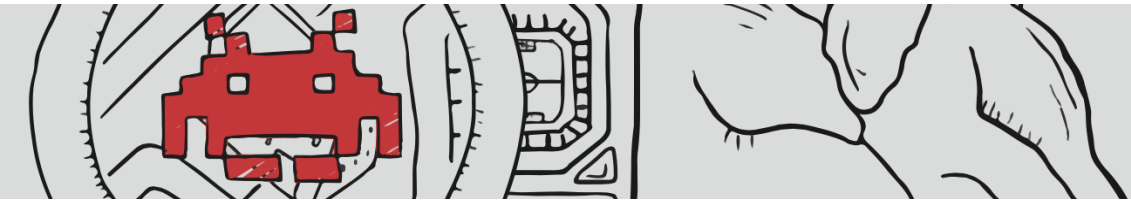
```
<root><public>-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDP5z7lwWHpYZZaBjw3iSzHOFwg
Eg3CiuqbosZqfWqVwWpBIg178X4z+erjdELboFr6u7kEu3MScNHeVrLak9SI/OK7
39jykbGp92MEgJef+5d/juUEajM5xAqI3e4Su3yvFSOW6zJqWwxQxDN8CF+tVeau
eDb7uPfcLmeGxLjmAQIDAQAB
-----END PUBLIC KEY-----
</public></root>
```

Beacon with empty hash

During the initial registration of the bot, it will send a beacon containing an empty hash value. In response, the server will send the current configuration file and an initial set of commands that the bot should execute.

REQUEST

```
<root>
   <unique>ABCDEFG_12345678901234567890123456789012</unique>
   <botnet>220</botnet>
   <version>196629</version>
   <system>23128</system>
   <type>bot</type>
   <ip>1.2.3.4</ip>
   <hash><![CDATA[]]></hash>
</root>
```



RESPONSE

```
<root><settings hash="37d0d82c2c7b156c82156d3aee6d23bca7faf052"><httpshots><url
type="deny" onget="1" onpost="1">\.(gif|png|jpg|css|swf|ico|js)($|\?)</url><url
type="deny
" onget="1" onpost="1">(resource\.axd|yimg\.com)</url><url type="allow" onget="1"
onpost="1">^https://www\.bankline\.(natwest\.com|rbs\.com|ulsterbank\.(ie|co\.uk))/
</url
><url type="allow" onget="1"
onpost="1">^https://corporate\.santander\.co\.uk/</url></httpshots><httpinjblock><u
rl type="allow">(|\.)alstats\.com</url>

...SNIP...

</pattern><replacement><![CDATA[ce:'4']]></replacement></modify></actions></httpinj
ect></httpinjects></settings><commands><cmd id="3343"
type="17"><kl>WinBacs,albacs,Albany.EFT.Corporate.Client,wpc,eSigner,StartStarMoney
,StarMoney,acsagent,accrdsub,acevents,acCOMpkcs,ac.sharedstore,jp2launcher,cspregto
ol,RegTool,translink,deltaworks,dfsvc,multibit,ArmoryQt,QtBitcoinTrader,Cortex7,OEB
MCC32,WUB,electrum,DexiaSoft,acCOMcsp,AccessPay,Suite,Entreprise,Enterprise,bitcoin
,bitcoin-
qt,Suite,Entreprise,Suite,Enterprise,argenie,argenie64,ARcltsrv,TokenManager,aetcrs
s1,csdrv32,sllauncher</kl></cmd></commands></root>
```

Beacon with get_module

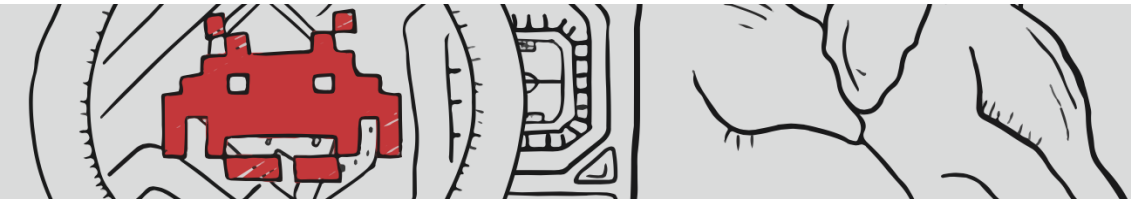
During the initial registration of the bot, or whenever the bot finds that there are new software modules available, it will request some of them from the node and some from the server. In order to do so, it will send a beacon containing the get_module tag. In response, the server will send the requested module.

REQUEST

```
<root>
   <unique>ABCDEFG_12345678901234567890123456789012</unique>
   <botnet>220</botnet>
   <version>196629</version>
   <system>23128</system>
   <type>bot</type>
   <ip>1.2.3.4</ip>
   <hash><![CDATA[37d0d82c2c7b156c82156d3aee6d23bca7faf052]]></hash>
   <get_module name="vnc" bit="32" />
</root>
```

RESPONSE

```
<root>
```




```
<module name="bot"
bit="32">iMAlP+gf5ujehA6EsHVySqflek7YqX3+QCR6xcDb1LPgus4jXEhmT0Z21y2uLLginEA8gKBFFr
GdXDTSnz3F/qZc0oV2lhf3ul4pb5HVe0spkF0FGohUnvwxXTPW20Az+tk+jHU3XtQMnm6++6jk4P1jFJRVW
noGuaFj+VPXxodNWpAAAwAAAAQAAAD//wAAuAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAADYAAAADh+6DgC0Cc0huAFMzSFUaGlzIHByb2dyYW0gY2Fubm90IGJlIHJ1biBpbiBET1MgbW9
kZS4NDQokAAAAAAAAACWTSP5h8iatYfImrWHyJq1GNFutYPImrUY0Xa1j8iatov17rXDyJq1h8ietBvImrW
```

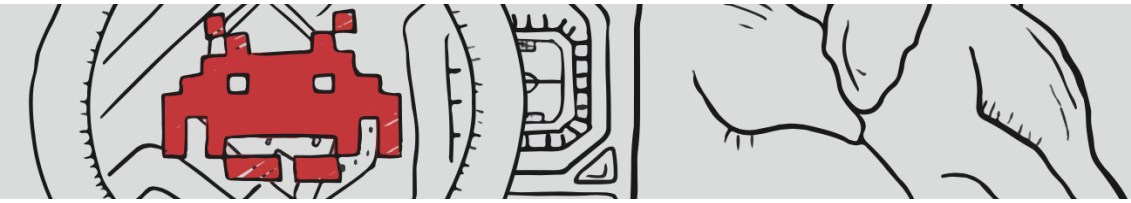
Beacon with data

Whenever the bot has some data to sent to the C2 it will send a beacon with a data tag that contains the data to be sent. The data being sent depends on the way it is collected, but it can be comprised of:

- Keylogger data
- Man In The browser data (forms, passwords, etc.)
- Screen shots
- Debugging data
- Other data

REQUEST

```
<root>
   <unique>ABCDEFG_12345678901234567890123456789012</unique>
   <botnet>220</botnet>
   <version>196629</version>
   <system>56392</system>
   <type>bot</type>
   <ip>1.2.3.4</ip>
   <data>
      <keylog session="11972477" app="jp2launcher.exe" time="2015-06-27 04:20:37"
type="shot"><![CDATA[/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDABQODxIPDRQSERIXFhQYHzMhHxwcHz
8tLyUzSkFOTUlB
SEZSXHZkUldvWEZIZoxob3p9hIWET2ORm4+AmnaBhH//2wBDARYXFx8bHzwhITx/VEhUf39/f39/
f39/f39/f39/f39/f39/f39/f39/f39/f39/f3//wAARCAMABVYDASIA
AhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQA
AAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3
ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWm
p6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEA
AwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSEx
BhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElK
...SNIP...
```

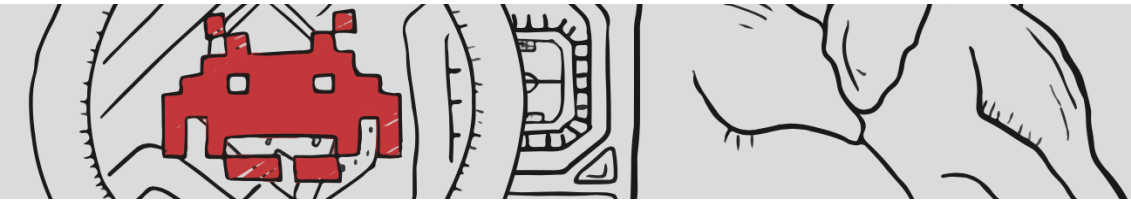


Node beacon

The node beacon is sent by the nodes to the admin node periodically (every 20 minutes) to keep alive its node status. The following is an example node beacon:

REQUEST

```
<root>
   <unique>ABCDEFG_12345678901234567890123456789012</unique>
   <botnet>220</botnet>
   <version>196629</version>
   <system>23128</system>
   <type>node</type>
</root>
```



Running a Fake Node Inside Dridex Botnets

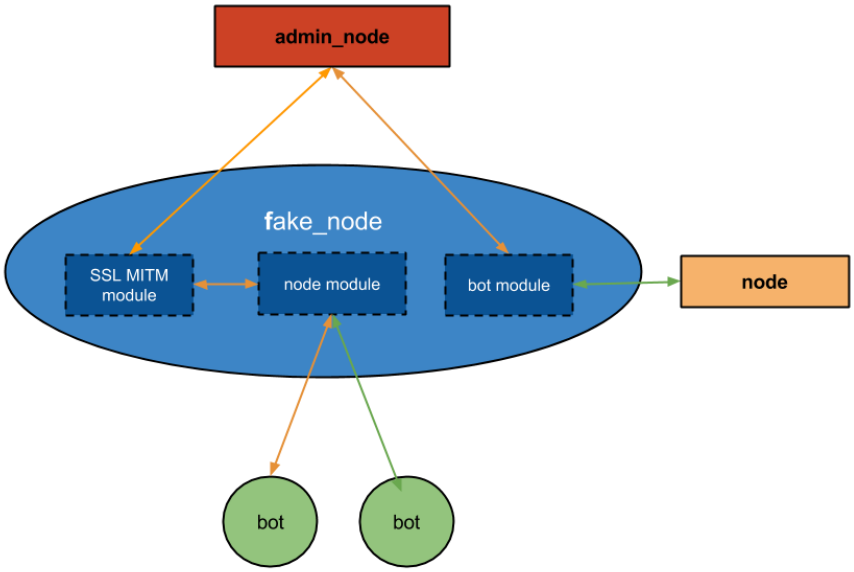
AnubisNetworks main objective while performing the reverse engineering effort that resulted in the information described in the previous sections of this document was to research a way to monitor the botnet, its infections, configuration files, modules and exfiltrated data in real time.

In order to understand more about Dridex and its operations, we decided to use the protocol knowledge we already had to develop a fake bot in Python with the following capabilities:

- **Register itself into the botnet** - This was a necessary first step. The fake bot needed to “speak” dridex protocol and be indistinguishable from the remaining bots;
- **Escalate to node** - After escalating to node, we would start receiving connections from other bots;
- **Intercept communications** - The ability to decrypt connections would allow us to understand exactly the type of information that was being extracted from the bots and possible behaviour patterns of the botmasters;
- **Receive updated configuration files** - This capability was interesting because it would enable us to monitor new targets (e.g. banking institutions) whenever they were added or removed from each of the botnets in real time;
- **Receive updated software modules** - This would allow us to monitor updates to existing modules and the appearance of new modules in the botnet;
- **Send arbitrary messages between the nodes, and spoof C2 replies** - This would allow us to further test the botnet and understand its behaviour when manipulated.

Architecture

The following diagram represents the architecture of the fake node:



The system is composed by the following components:

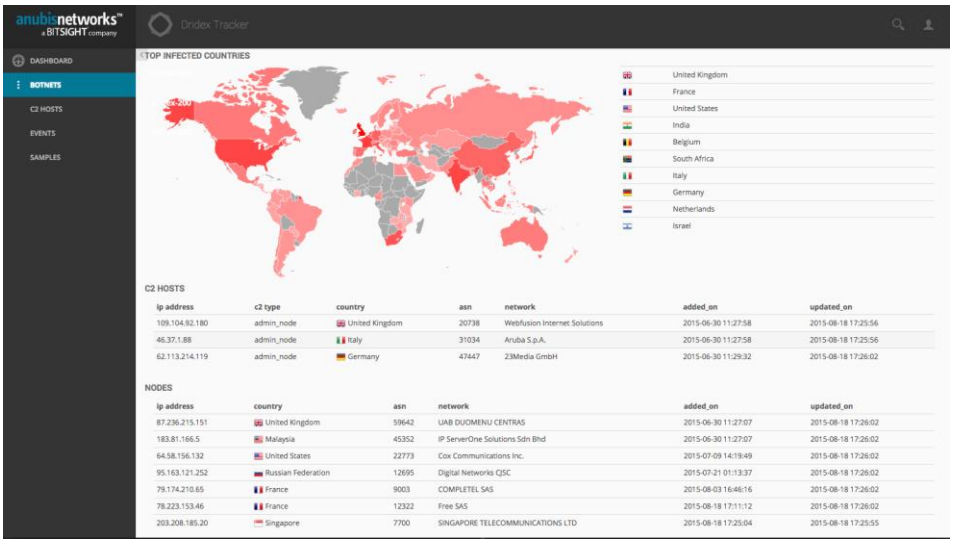
- **Bot module** - This module is responsible for registering the bot into the P2P network. It will communicate with the nodes as a regular bot, and after escalation, it will communicate with the admin_nodes as a node;
- **Node module** - This module is responsible for receiving all incoming connections and processing all internode communication. If the incoming connection is not in Internode protocol, the module forwards it to the SSL MITM module;
- **SSL MITM module** - This module is responsible for intercepting SSL communications by presenting a fake certificate to the client and reestablishing the SSL connection to the admin server. This module is also responsible for decrypting all C2 protocol communications using the C2 private key.

Deployment

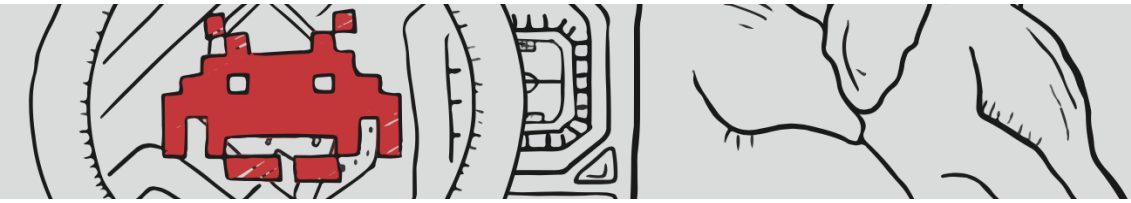
Our fake node achieved all objectives. Running it revealed to be a challenge due to the high activity level of the botnet operators and the countermeasures they deployed over time, see Countermeasures Timeline below.

The fake node was successfully used to monitor the botnet for a few months, but this approach has a high maintenance cost due to the constant evolution of the botnet. However, building this software allowed us to better understand the inner workings of the botnet and to identify a few vulnerabilities.

To track the different Dridex botnets, C2 changes and bot infections over time we have developed another application on top of the fake node that was used for collection, storing and reporting, called Dridex Tracker. The following image shows the Dridex Tracker GUI:



Screenshot of Dridex Tracker



Countermeasures Timeline

The fake node was run with all capabilities enabled from the end of April to the first half of July 2015. During this time the botnet and bot modules were updated several times and several countermeasures were added by the botmasters to prevent rogue nodes from entering the network.

The following is a chronological record of some noteworthy occurrences:

- **22nd April** - Initial efforts to join the botnet were successful. The fake node escalated to node and started to receive connections from other bots;
- **30th April** - Dridex protocol changed and we were no longer able to join the botnet. The new protocol was reversed and the fake node code was improved;
- **20th May** - The botmasters started to block rogue nodes by removing them from the node list and disabling the IP address to register into the network. We opted to use the rogue node only for one botnet at a time;
- **27th May** - The botmasters added a feature when blocking a rogue node IP that sent 127.0.0.1 as the IP address of the admin nodes and nodes, causing the rogue nodes to enter an endless loop;
- **30th May** - The botmasters started to try to disable rogue nodes by sending them a “killer module” that erased a system MBR, and caused a reboot;
- **23rd June** - The botmasters started blocking rogue nodes more frequently, several times a day;
- **10th July** - The node escalation process seems to be controlled manually or in some other way that prevents the automated immediate node escalation.

Results

Infection Dispersion

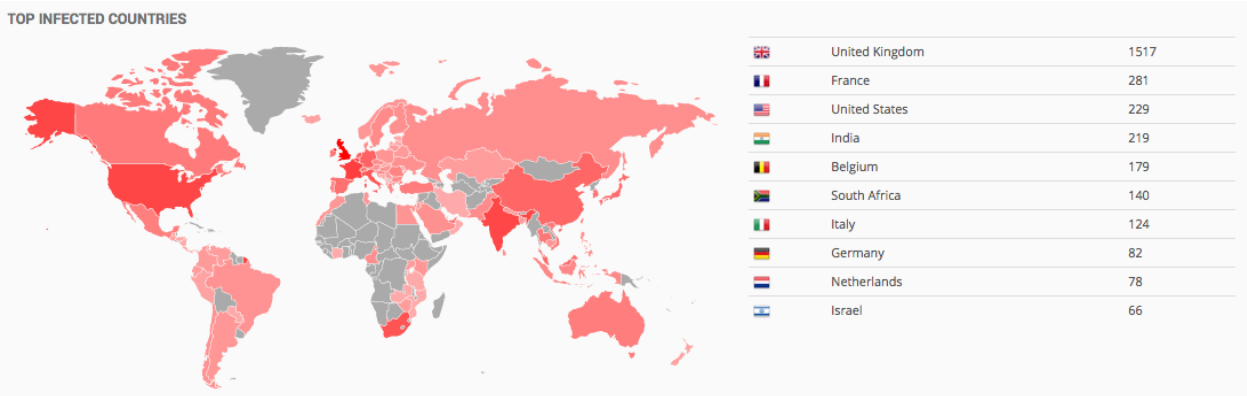
Dridex botnets don't have the same size in terms of infected systems when compared with other botnets in the wild but represent a very serious threat. That is unusual, given the high rate of email distribution campaigns.

In April 2015, for the main Dridex botnets we counted the following unique bots:

Botnet	Unique Bots
Dridex-120	5850
Dridex-200	750
Dridex-220	9650

In July we observed a drop on those numbers. The following counters are related to unique bots for the main Dridex botnets that our fake node intercepted in the time window between 29th June and 10th July 2015.

Dridex-220



Dridex-220 botnet geo dispersion and top 10 infected countries (unique bots: 3770)

Dridex-120

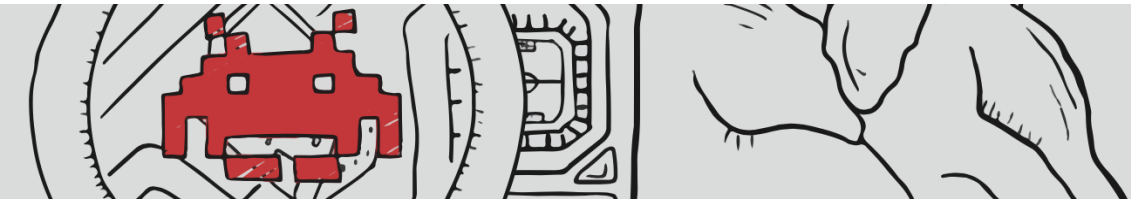
Dridex-120 botnet geo dispersion and top 10 infected countries (unique bots: 995)

Targets

The main targets of Dridex were banks from several countries. During our analysis, we collected several versions of the config files for the main botnets (120, 200 and 220) in real time as they were updated. The following are the hashes of the config files. The number of files reflects the number of updates and shows the high level of activity of the botnet operators.

Dridex-120:

```
a2297e5943b67baea4981492d130e832e477385e
46fb662c14139fe9896fac7ae446da1cfec267a4
a1f77caef817bf37e9106a0c404a34933c44fbea
1f257aa466fafc0fb7ee86500d12971934e56774
01af94bec88baf6487c1ad7dd4df1da76a851a86
0e4770e4d920bc6d96cebf1790211dc19f0ea79a
2bd2401726c43d424628877f4d1d13f94084fad1
556b30c88b78e0df7d5a456085c1159ec312cf91
68d86c793fe06b5a41a65bd7626750201ec61b18
a0a36d2ec157515af30b49e14494be43201c2fcc
c44a63e98920e8e2d07e3a2f94c26bbd878bbdb5
caba93b3a74620fb747748931834e181aaf6d557
aafc829063b25e70e11052e9995095253471586c
de85d6246ad7951f1af306bd65a2f82555dc5bc8
```



Dridex-200:

```
9df3eb9f7a404afc66b7a91877255ca1692959c7
cba023ce14c2502a31be47c1e4be1f4a21e730c1
1751c8359d81eee60437ef2c9d91b2b53ec1598e
```

Dridex-220:

```
24877bb165bc0d66723be7dca1491b73f533fa1f
2a2e3c51d4a05c8e8c5bb76cb756eb87fa6b0dea
37d0d82c2c7b156c82156d3aee6d23bca7faf052
3c24b82d5e241f520d362d1f6a2713b5ca7011eb
44450edfefa88b92e1f5d0ef8e38c7b78cddfce7
79f9220f966c0de08e2b5e9d37064ed6ae39ccf3
830deb5d060789d353a1e5507669c4fda155ac67
8e94867d2ff752a0da55a9ab1f6046a915316908
a04ab4391ca6c78871f366198c90e201b40b23e1
a4e43c67156ed562b3f08358f520a21b0ae69463
a60ba9a229caba4daf591cfed67517e711ed7e22
a8fbd1b6fab99d20362edec7d2937cc35eb3248b
aa3dc12da7bf2b030a3b1310b38e168298b6c92b
bf008392dfdf36f77492a52eb3ab60aec586bcd6
bf7201be41ba82d7f7ae521e5171b6e7d2d4b80e
c2700de864cc9fb90f0450b6dcda52519389ab4f
d1c36a0addb61b31fb52bccf0024a1acf589a6dd
e09de57a7e946654d0db8d8911ccc3909d1689a1
edcd41096f3678170e38c63968d7a1bec99ebe9e
f37ff046e73a532a4064098e8d217a173d947c73
```

These config files contained the webinject settings for several hundred URL patterns that mostly belonged to online banking applications and other financial related . Besides the webinject data, the bots were also instructed to keylog data from the following applications:

```
WinBacs,albacs,Albany.EFT.Corporate.Client,wpc,eSigner,StartStarMoney,StarMoney,acs
agent,accrdsub,acevents,acCOMpkcs,ac.sharedstore,jp2launcher,cspregtool,RegTool,tra
nslink,deltaworks,dfsvc,multibit,ArmoryQt,QtBitcoinTrader,Cortex7,OEBMCC32,WUB,elec
trum,DexiaSoft,acCOMcsp,AccessPay,Suite,Entreprise,Enterprise,bitcoin,bitcoin-
qt,Suite,Entreprise,Suite,Enterprise,argenie,argenie64,ARcltsrv,TokenManager,aetcrs
s1,csdrv32,sllauncher
```

One interesting fact was that Dridex not only collected information information from banking sites but also from many other websites that are not directly included in the config files webinject settings. These included login credentials for corporate extranets, VPN, webmails

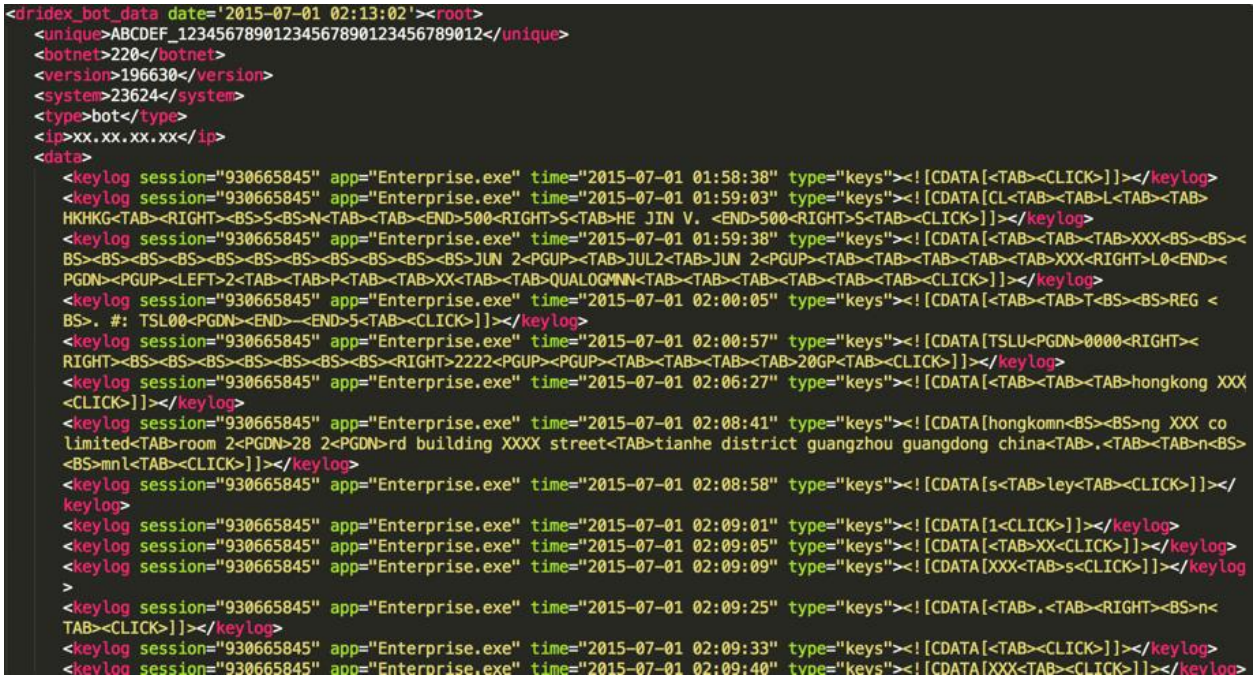
and several online services. While running our fake node on botnet we observed a total of 2069 unique domains whose information was stolen from user's browsers transmitted to the C2.

Stolen information

Our fake node was able to decrypt communications as they were being sent from the bots to the C2 Server. This allowed us to gain a better understanding of the type of information that was being harvested, and although there were banking credential being stolen, a large part of it was not directly related to online banking. We observed mainly three types of submitted data:

Keylogger data

This included data from a set of applications that were configured to be keylogged. Also because it included generic software such as jp2launcher.exe (the process for java applets), several chat applications and other applet based interfaces were being logged. The following image shows an example of a keylogging session being transmitted:



```
<dridex_bot_data date='2015-07-01 02:13:02'><root>
  <unique>ABCDEF_12345678901234567890123456789012</unique>
  <botnet>220</botnet>
  <version>196630</version>
  <system>23624</system>
  <type>bot</type>
  <ip>xx.xx.xx.xx</ip>
  <data>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 01:58:38" type="keys"><![CDATA[<TAB><CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 01:59:03" type="keys"><![CDATA[CL<TAB><TAB>L<TAB><TAB>
    HKHKG<TAB><RIGHT><BS>S<BS>N<TAB><TAB><END>500<RIGHT>S<TAB>HE JIN V. <END>500<RIGHT>S<TAB><CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 01:59:38" type="keys"><![CDATA[<TAB><TAB><TAB>XXX<BS><BS><
    BS><BS><BS><BS><BS><BS><BS><BS><BS><BS>JUN 2<PGUP><TAB>JUL2<TAB>JUN 2<PGUP><TAB><TAB><TAB><TAB><TAB>XXX<RIGHT>L0<END><
    PGDN><PGUP><LEFT>2<TAB><TAB>P<TAB><TAB>XX<TAB><TAB>QUALOGMNN<TAB><TAB><TAB><TAB><TAB><TAB><CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:00:05" type="keys"><![CDATA[<TAB><TAB>T<BS><BS>REG <
    BS>. #: TSL00<PGDN><END>-<END>5<TAB><CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:00:57" type="keys"><![CDATA[TSLU<PGDN>0000<RIGHT><
    RIGHT><BS><BS><BS><BS><BS><BS><BS><RIGHT>2222<PGUP><PGUP><TAB><TAB><TAB><TAB>20GP<TAB><CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:06:27" type="keys"><![CDATA[<TAB><TAB><TAB>hongkong XXX
    <CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:08:41" type="keys"><![CDATA[hongkomn<BS><BS>ng XXX co
    limited<TAB>room 2<PGDN>28 2<PGDN>rd building XXXX street<TAB>tianhe district guangzhou guangdong china<TAB>.<TAB><TAB>n<BS>
    <BS>mnl<TAB><CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:08:58" type="keys"><![CDATA[s<TAB>ley<TAB><CLICK>]]></
    keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:09:01" type="keys"><![CDATA[1<CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:09:05" type="keys"><![CDATA[<TAB>XX<CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:09:09" type="keys"><![CDATA[XXX<TAB>s<CLICK>]]></keylog
    >
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:09:25" type="keys"><![CDATA[<TAB>.<TAB><RIGHT><BS>n<
    TAB><CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:09:33" type="keys"><![CDATA[<TAB><CLICK>]]></keylog>
    <keylog session="930665845" app="Enterprise.exe" time="2015-07-01 02:09:40" type="keys"><![CDATA[XXX<TAB><CLICK>]]></keylog>
```

Dump of a chat session sent to Dridex C2

Web Inject Data

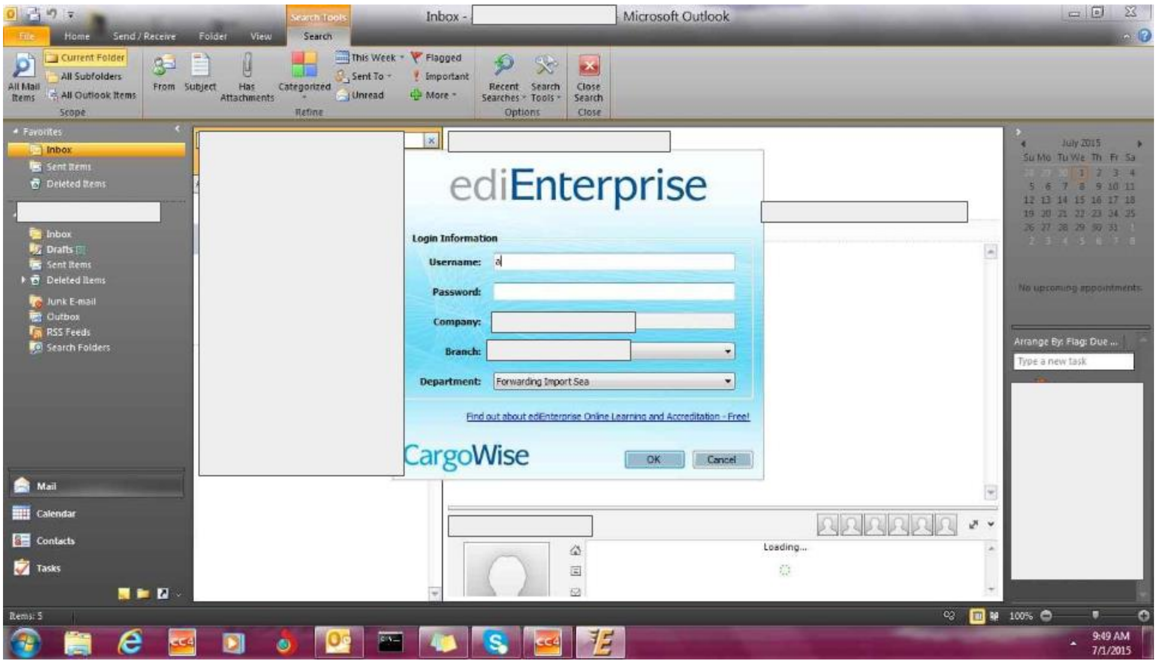
Web injects were able to extract specific HTTP submitted data from a large number of different domains. Following is an example of a set of credentials for an online application being sent to the C2:


```
  <data>
    <http time="2015-06-30 12:25:15" crc32="2362269086">
      <url>https://login.XXX.com/</url>
      <useragent>Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko</useragent>
      <userinput><![CDATA[haXXXXXX8@htotmail.com XXXXXXX ]]></userinput>
      <data>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
</data>
```

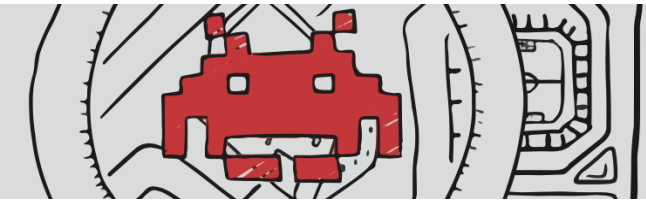

Login credentials been sent to Dridex C2

Screenshots

Screen shots were also taken for a given set of applications. Following is an example of a screenshot image that was transmitted to the C2:



Captured screenshot from Dridex infected system sent to C2

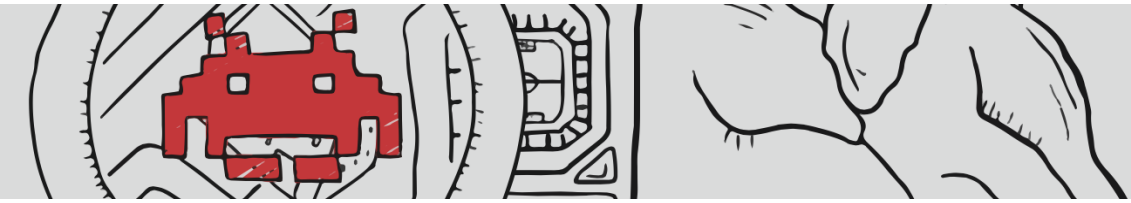


Indicators of Compromise

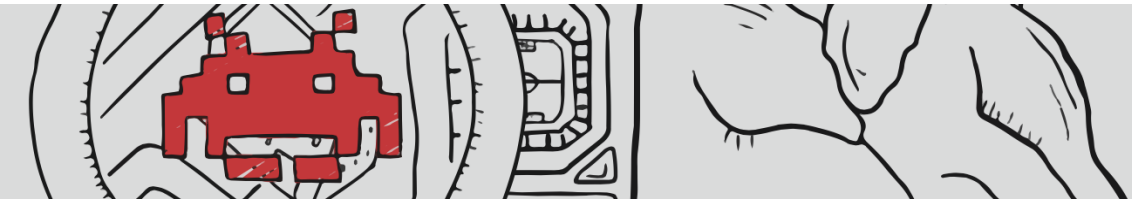
Analyzed Samples

Some of the samples analyzed during our research, referenced by their SHA-256 hashes:

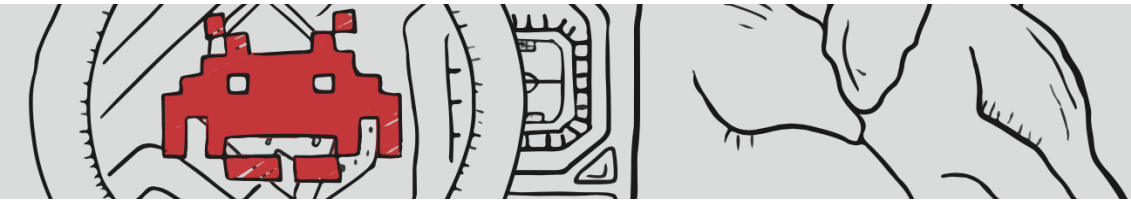
```
00c571f017d487ebe781dcafc469fe6f872c563baccef48a48275a2f3f897bf8
03385cbccae28797e0f6b8c1f9b55e767dad487fb652162def9f8eb7a86b29a7
044b887de7fcbe126825a16174fe7dd9ef7753f313f4775ca489b92e54be5bf5
053f9ea7ad120ae759dd71a1da27cfd8e8cb44496e24e9f11a6ce132dab10294
05f5c678ee84532613dc60464eab3ce862f3999eb5dad96d2cb5f47254cffdcb
076da91896b0ee6269644a1ab0afa42de278c551bcf2b0933ed92d269b377dd7
0791f50d933a90b53b96be52d91aa35e12b31e57dc55e43e4e6131cb94c3bc1a
083aa942388416bb9688b1a0520ab15c910448c75b75d90a7b6f404f2e105a38
17f5e836f2a817bb0f977dbddfc58b03cb235e0fd27c4865de44fd37f249fb36
1c14399e361fd7d3c66d65c9a79d4896e03ce977bed4caf7e68b8741828b00e4
201f37cb8c33330b72f73b6a4b5a4699b01d7529f7a2e372dc2ff94e068c2241
2069ab8ed9245560ebb2219df37e4b740e6a0ed6931ed71fa31dd580cf0ed834
24aff09953c8301e82217137001b9ed45a84a3f3a04d6ce1a875d2d36eb275fb
252bfae5a124882c38c7727e25d3e637161bcc9e03f31b9922756f1868466d19
253e7b09431c1a9473008b49480cf702baa65eabd79c44d8d7be1560af6017de
288e528e15297b2ed1b7850650f9ac72e4fb7ba50412bfc3375af7a2fb092a7e
2d95c36352bc110f2c74d7ccc83a3003df0fbf2c62268f446ae4954852db8448
34fc383d51cbe08ab998e986e2cf1ee1535a3130bb0165e3b6fa94a67b0be0ec
38011cb19ccccad55d331ff725f208eb164fd7fa7418cd50a8ad2c026716112d
3a4322466c019d662b33b6e08db1c1e42dce08c80a5b8ffee402e14bb7af8584
3acdb80ecb3cb2dd9203691cd59bb9e0750e5d9eecb726e5de1b33cd29609993
3f8d6700ee02836dae168b6bda36dde73a43b77eedb722f766e6455204fa126d
3ff2a363fd9c3f63b519f3ee51f18cf4b395001bfa5a954ec47d9849e1b050d7
4120279ad7d80943896f8e6e15cf9ffc951e78e8f79ebf1d254473faceb8ccec
41d14fd287593204b179a90a9d0c83dc78ef2e6dcf90ffc087fbfbdfb3392f95
475295a3c2c8e19fa4a24ad66318957c4031d0ef4464ddf0b3665ec9ee23c10e
4fdd1e4ec1fe53c5ec347c1956367059a7318bcedea0e0969508923b186afcb8
52104187baa76905d8793861d0214f3c60124697c6c85d0bef27815310dc182b
53937c1c877f79f5ae1e5d3e7b2c177a6ba36ed1e222430177833cc47bfe524c
5ac0c18b4743626c3c49492cd7470c1b4060c553705bb49612a5d1b2be0c2fb5
5b35ccc23990f88b412e1b8abc04ba930abc0131a9a23bd43c27a85faf2304d1
5cb5ca44a45dcf5c03b26a4b3fbbf52937f78651d5cf51bba3732f17433916d8
5daa5dc22bacc1857ff2b17f2572b86de2cbbfbb83016fefb4352e7e497d68e9
5e1626e985ef11cae40587e715df23cf6ef048f11b2ce3405da899209beee05a
5e98a73e1a377d75a8363672adf3c3f490d0bc01d7a104e8fd28e3cdf6457e44
605e2d3f4c0789147e63885eedbf93a5c6ec381170fe00f14ff558c6d7973c83
608daddf175bdf12babc4df2bff093b9506c6cd87f800a1bb95cc0005e9f85e5
6287776f22b19e409efe6342900d31b499c1077cd36fd8a4d988e3bfe1710d49
637dcd114f945c7e25a649023e309f6b0165ba5a2d8e54395b426dad5526ce36
66cf9503474113671d048185899933f383ee96b48638b44d6a12796c974aa7cf
673e98d480871eece9374a35de386768c4318fb94e93be0da727e7fa0d5523bc
69140fdbfd44d30286c188ce3489ba43a606eec8b7803d664e3d054f259ca2f2
6b596c71af43f1433de928f454e132eefbb1a0e5ea42d41a629fef08d59247fd
6e67206bafe0cea4d7e2229d6014b8cc6824135eddffa6db243f527c93f58486
70c5b1ab923037b517236e9d7d901f56bbc9903b5fbe5cb49c94ac24f0146b88
78d84f8368b44692b74500f6071788170bc2fef27363586b6d569d745d1cf2a2
7907e5919d22fbd25b082cfd33dbb8c99d42fc1258767a4de178388b3eca9679
7bcb0abcfbea20ecfe31d8dd65146b8b1ffd0d81479d11dc329b2f99e263bd78
7f159c34b8d8e662405dd403df17a03f16a634b454677fb47e0ece7e277e2315
```



803f63c24dc2e9b2a266adcf27950e247d83bb9b2a84065e030e68574cecaf9d
818d07c8b0f082dab108dc999265bb2f4f443ac93487efaf54c66a23e00482f2
84c9c2d16e31ce35c5ebc8ef2a1d185fe6b365d9a90ffe6ee93c52f9ad0f2911
855646fda14da9fbcaad3e92f1f7f84162c6744b5cba55524e21fd651033f159
88aaa24a15840ef0e4d3fd73379a455073d1699ba63a74e0faa7b31600521dbc
8862de75728637f852fee06dee15258a2f4af6ea12e80f98a9614629c8956d5d
8bb1ccc07e79bc6818c865343da2e8bdc33a83cc93c3e1bf1f2c5f69d7a40387
8c79bff0c302a0c1762fc59ab7001001a9293506197579213e087868493b756e
8e6bb148ffc0e18c0450a89f7b0ba729a28eb22da12fd3f69d18daa85fd09024
9014c90c29945043eb94ba1d5119b6961c1595f5af319b2796b9850f6c0c8b4f
93242b25aaf2aea8ecef7327cbfc40cb8f8fda0eecd4618674062f81186ed8b2
93989989f4811b942491b63f848964237ab135d4c0c53b9039dbd1c5afbf703d
939bfb99c45aef07a26e0a3e214bc028f3f4455c9d6f871b033348e51c1e6491
93aff0e300e8ec913b9405528506b875f8fe4dc9944c570ab95bdd86160f4574
950d3059ddee96b7178b2f360fb0c1ed4454ada78528659ff70dc4b538ccd448
96ebf5fef0cbe6c7e5335e35f0c81177c3531393cdd10f33ab30b45f1694c546
971403cb96c22acecd030aeb7b25da08fd10af4a85877d8d2c49a8ac26a90796
982eda0ae68525e7cb212ff81f59e9887a361df7bce5321f6f110c2210f42cf8
9a6dbb718f05748f8e8e6d9ac75748244e41fba4eef6a04f9dfb242012a2cd29
9c1394c8bc24313c76e7fe113657875455d009e9b24f30b4208628d93ca9c7fc
9f10dc38f9e284ad4993a0579e1f732a2da1eb69d04aba5babcb2d4458b5a7da
9f2e357c83fdfb6c560322b3357f394f61ea6d5b1c8a22a7d3e9a6160633b20d
9fb3a5b2ef49ee304ae0fb75fcc2d951718d28687d2721a529b592d0719de580
a4e8e089097301769e27982f55561fbe5240dc695ca50b9b6a801d4e4b609b50
a60bc90e9e8281321bdb8595c8734688a67982864fcce1a20347112a663439ff
a7ffc9851184437d694d85e77ef0eebbaaddbd40d40b0a22bb2cb51deee32305
ab9cb59a12a10a06a5fede0cebf9184f2a533bd135f14beb9d7c9864001cf20a
aca8a777c51c537d23fbe8ca181bb49fa0eb250e0a2e1551ea2e4fad9a746fc7
ad7a6f5a99d3678475d136a06fd710da1c4cac00cb4f4109e44e14f574f3ff88
c1731a1f5142190cd397057dfa1a0de77025eb2786b8ec1007724ed9fe705084
c25f09ac747e1eedf75ca88bb74c4b660b4611c789f2f3030754c8140801ecb8
c56a46575f00e527844ea393c50aa58500dda94088c34489559b610200ba756b
c577a5ddbdf85ab2a168223d80981cf1d835f15dbf0437cc43b5801cc37010a6
c6725b674fc0c20a696e4b6df95ced81c912a7b11dcadfd7f8b81c6106b556cd
c6db0d4bfb2a909ce0767269bbe18f9190e75260eccddab8990563ac8b862cac
c6e78a167bd5277bca73acf9164ca611cb2999621a63ff380a9ccf440202c31d
c94e5b273aa0c65c144e74586de4507f2e807b9f2a0246a76faf8cc9f7f921f6
cb77b8792d5d117b420304e3b6e4fc72076550ea4f1c9bb9a13870c8e0699269
cd88c3d4a1a8779edae8ff706b7fdad6598c85af8b0b5d87e1f33d93be5c4315
d06b515db7f8579da3c5cc21b667c98463bd26b4e77e4d85a1dbf9326f060c63
d1b57efcaf5d061e387638ceb1d8fe16729e37c3c2a7bcf8e5131598463d2684
d2865aa1193fb32e5604ab03e9cd709dacc0a114b5c611c5b3b5470b1af4f32f
d31efa8255e1eb4319f0713a817c8148fa35771a01d47fe009bfc3dff60eab4f
d40d05316db62ea3d27830f87892aaeab3ff22b63babadb755aad10dcba85a7d
d57dc5d12825f9dacceff29738ce4797bcc68b2bf67baf328c59651ebbdd5f8c
d57f99ad25e12386ca27c4a1b621c72243380573d8eab2fb2d9ff08671a926d0
d73ded7b08da92491bd2f2564fa070e820912fbe6ae253795245e8f29539398b
ddf5a6e438082b8523bf5df34bd885b092eb4f49c91af7fa5a56bb44914a048a
de483957fdb99fefcc645c8d116cd4ac3d41bcb8b2060010039cbdaba75f4f93
e036c4d6364f94287144a68a815eaadf23e40ad169ac780a7ac7a2aad3b36b32
e70d529938102d1fca9a55beb00fe4b33985cc7a6ca9d1ba7f5998954091a3e0
e75d9851b45c83b7112546c9006d627326da820c3322002cc66249c4d57abfb1
e90cc9975fae547bb31f48c265a54718207df57cf67782576dfee2294ded7ae2
e9f72a82d897d57219e1a9a3837f089edec92f58c01c62743c025dfea1be4d17
eabc6e91742d9ad251a40c92e7185b10b8ef812311afde5ca551def564485a03
eb885ffd162131493af6ec76465045abbb1081f4fee55ad538ab9e4f826e54de
ebf680dee76110bff21166800de38b1940f9ee795f190a447d4a4765758aa127
ecad6974598a1efb6cc7d4e4012b8aab0259323ef45a9eda24c27de82ed5566d




```
ed23794c19b93c3b62707a8ca8d81256c56f588ea8061c3c990e54a0adf2cc18
f2328ad463d584ba06cba3338d73b1ee2ba772401d51cf0c88c51aec53bd3623
f3a359c36d63466abb988d8b4aa96dd8fef6fd4450dd09c123c6aa8f513d1bba
f3a70107ff94c33e5c5ed76b06a1ddc8d68d04a3f2b8b9027ad2d0d02e54c467
f4360ba1a956280eace36bf3b8a9eb1961f269f4c855e89d4d4d490278c7c653
fa252e501332b7486a972e7e471cf6915daa681af35c6aa102213921093eb2a3
fb0e2bba53ddc68e9f1046de0b5b240669b4f3f83be63d3d0f575c38a7ae80b0
fead2bf5b5b9a76c8267c675bb64ad8de5b9f85faf7c87cb72e890aca3253117
fecac80db9b28cb2eaa0de509af4f113784d3cab3be3c3751b3330c8b165f043
ff87f367bb4ac340964743e8ef4e346f96546cec5fda5c0c9d7956237520b31f
5ee065c1ff85512a9219e58cdcd1173e9c8106d860c8752b202db679631f477d
```



About Cyberfeed

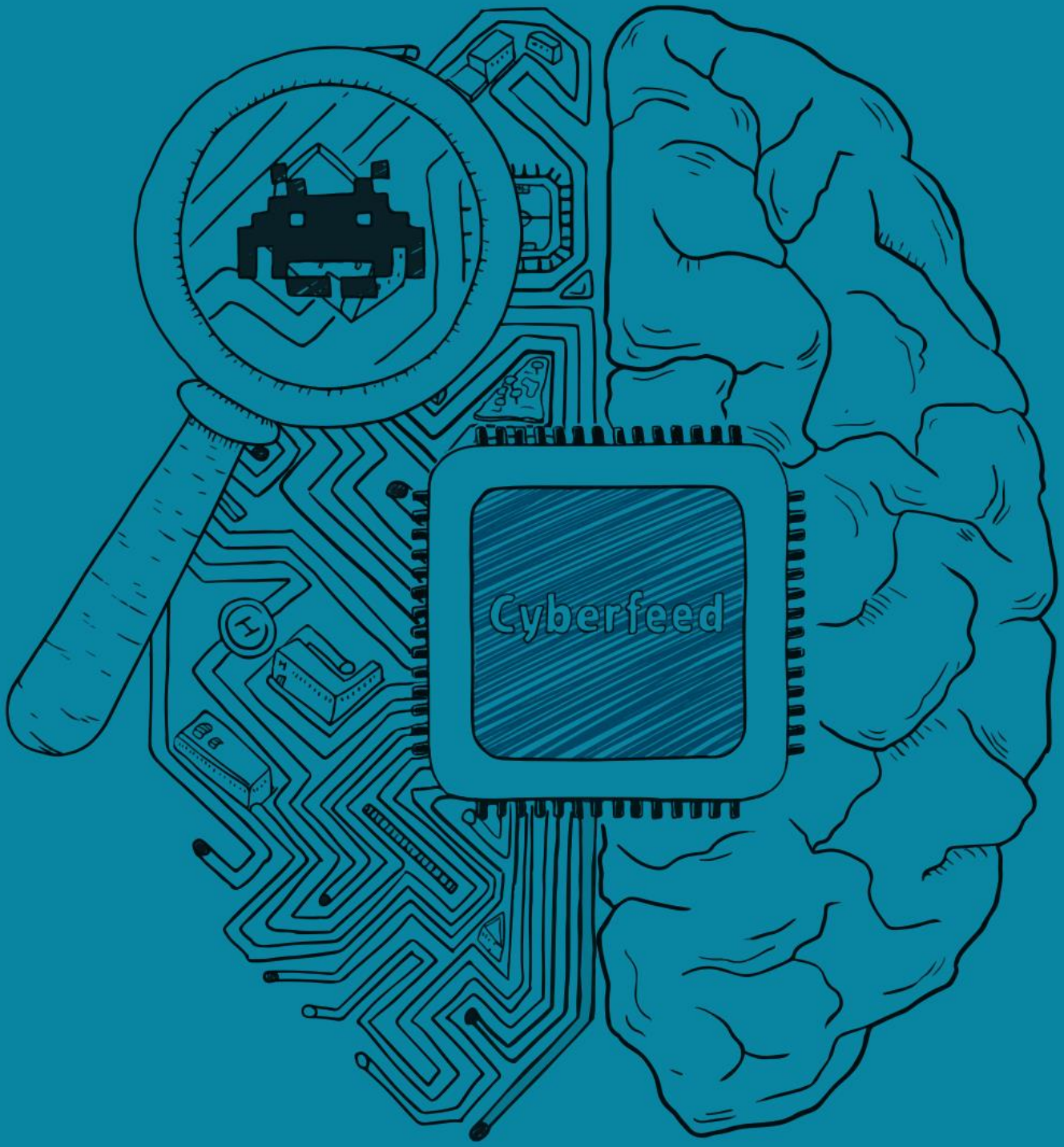
This investigation was developed based on the data provided by Cyberfeed, AnubisNetworks Threat Intelligence offer. Cyberfeed provides a unique view in the early detection of cyber-threats. The combination of real-time security monitoring, context, and "smart eyeballs" empowers organizations and government agencies to have a new and powerful approach to fighting cybercrime. What makes Cyberfeed stand out?

- More than 25 000 events per second collected and streamed to security feeds as they happen.
- AnubisNetworks' Threat Intelligence offering provides real-time data on cyber-attacks, allowing organizations to mitigate risks as they arise.
- Since the information is processed in real-time, users do not need to invest in data storage, making this a lean and very light service.
- Turning data into actionable intelligence is the cornerstone of AnubisNetworks' Threat Intelligence offering. Events collected from disparate sources, including botnets only tracked by AnubisNetworks, are transformed and enriched into meaningful knowledge to help stop cyber threats and attacks.
- It dynamically provides full context of what is happening and is not a snapshot of the past, enabling cybersecurity analysts to obtain increased visibility into a situation and enable informed decision-making.
- Cyberfeed API flexibility allows customizing and processing data feeds in real-time, including measuring, filtering and de-duplicating events on the fly. Cybersecurity analysts can slice and dice data to better fit their requirements and build the necessary knowledge to stop threats.

How can AnubisNetworks play a role?

AnubisNetworks is an IT Security Company specialized in real-time threat Intelligence for B2B.

With an established reputation for delivering innovative and effective solutions, AnubisNetworks has worked with multiple service providers, risk and security organizations, telcos, major banks, media groups and large corporations delivering Real Security in Real-Time against Real Threats.



CONTACTS

PORTO
Address
UPTEC, Rua Alfredo Allen 455/461
EC.3.12, 4200-135 Porto, Portugal
Phone: +351 220 993 873

LISBON
Address
Av. Quinta Grande 53, Edifício Prime
5A,
Alfragide, 2610-156 Amadora, Portugal
Phone: +351 217 252 110

BOSTON
Address
125 CambridgePark Drive
Suite 204, Cambridge, MA 02140l
Phone: +1 617 245 0469

FOLLOW US

twitter.com/anubisnetworks

linkedin.com/company/anubisnetworks

youtube.com/anubisnetworks

ON.2 O NOVO NORTE
PROGRAMA OPERACIONAL REGIONAL DO NORTE

POR LISBOA

QREN QUADRO DE REFERÊNCIA ESTRATÉGICO NACIONAL

UNIÃO EUROPEIA
Fundo Europeu de Desenvolvimento Regional

anubisnetworks™
a BITSIGHT company