Highly Resilient Peer-to-Peer Botnets Are Here: An Analysis of Gameover Zeus

Dennis Andriesse[1], Christian Rossow[1], Brett Stone-Gross[2], Daniel Plohmann[3], and Herbert Bos[1]

[1]*VU University Amsterdam, The Netherlands, {d.a.andriesse,c.rossow,h.j.bos}@vu.nl*
[2]*Dell SecureWorks, bstonegross@secureworks.com*
[3]*Fraunhofer FKIE, Bonn, Germany, daniel.plohmann@fkie.fraunhofer.de*

Abstract

Zeus is a family of credential-stealing trojans which originally appeared in 2007. The first two variants of Zeus are based on centralized command servers. These command servers are now routinely tracked and blocked by the security community. In an apparent effort to withstand these routine countermeasures, the second version of Zeus was forked into a peer-to-peer variant in September 2011. Compared to earlier versions of Zeus, this peer-to-peer variant is fundamentally more difficult to disable. Through a detailed analysis of this new Zeus variant, we demonstrate the high resilience of state of the art peer-to-peer botnets in general, and of peer-to-peer Zeus in particular.

1 Introduction

Since its first appearance in 2007, Zeus has grown into one of the most popular families of credential-stealing trojans. Due to its popularity, previous versions of Zeus have been extensively investigated by the security community [6, 18]. The internals of the first two versions of Zeus, which are based on centralized Command and Control (C2) servers, are well understood, and C2 servers used by these variants are routinely tracked and blocked.[1]

In May 2011, the source code of the second centralized version of Zeus leaked into the public domain. This has led to the development of several centralized trojans based on Zeus, such as ICE IX [16], and the more successful Citadel [14]. In September 2011, a peer-to-peer (P2P) mutation of centralized Zeus appeared, known as *P2P Zeus* or *GameOver*. Due to its lack of centralized C2 servers, P2P Zeus is not susceptible to traditional anti-Zeus countermeasures, and is much more resilient against takedown efforts than centralized Zeus variants. In this paper, we perform a detailed analysis of the P2P Zeus protocol to highlight how it achieves its resilience. Our insights also shed light on the resilience potential of peer-to-peer botnets in general.

Centralized Zeus variants are sold as builder kits in the underground community, allowing each user to build a private Zeus botnet. Interestingly, this is no longer supported in P2P Zeus, which is based on a single coherent P2P network. The main P2P network is divided into several virtual *sub-botnets* by a hardcoded sub-botnet identifier in each bot binary. While the Zeus P2P network is maintained and periodically updated as a whole, the sub-botnets are independently controlled by several botmasters. Bot enumeration results from our previous work indicate that the Zeus P2P network contains at least 200.000 bots [11].

The Zeus P2P network serves two main purposes. (1) Bots exchange binary and configuration updates with each other. (2) Bots exchange lists of *proxy bots*, which are designated bots where stolen data can be dropped and commands can be retrieved. Additionally, bots exchange neighbor lists (*peer lists*) with each other to maintain a coherent network. As a backup channel, P2P Zeus also uses a Domain Name Generation Algorithm (DGA) [1], in case contact with the regular P2P network is lost.

Our results are based on Zeus samples collected from the Sandnet analysis environment [13] between February 2012 and July 2013. When we began our analysis, no detailed information on the Zeus P2P protocol was available from related work. We verified the correctness of our results through prototype poisoning and crawling attacks against Zeus, which are described in our previous work [11].

Our contributions are as follows.

1. We reverse engineer and detail the entire Zeus P2P protocol and topology, highlighting features that increase the botnet's resilience to takedown attempts.
2. We show that P2P Zeus has evolved into a complex bot with attack capabilities that go beyond typical banking trojans. Particularly, we find that P2P Zeus is used for activities as diverse as DDoS attacks, malware dropping, Bitcoin theft, and theft of Skype and banking credentials.
3. Reports from academia and industry have long warned of the high resilience potential of peer-to-peer botnets [4, 5, 7, 19, 20]. Through our analysis of the communication protocol and resilience mechanisms of P2P Zeus, we show that highly resilient P2P botnets are now a very real threat.

[1] `http://zeustracker.abuse.ch`

2 Network Topology

The Zeus network is organized into three disjoint layers, as shown in Figure 1. At the bottom of the hierarchy is the *P2P layer*, which contains the bots. Periodically, a subset of the bots is assigned the status of *proxy bot*. This appears to be done manually by the botmasters, by pushing a cryptographically signed *proxy announcement* message into the network. The details of this mechanism are explained in Section 3. The proxy bots are used by harvester bots to fetch commands and drop stolen data. Aside from their special function, proxy bots behave like harvester bots.

The proxy bots act as intermediaries between the P2P layer and a higher layer, which we call the *C2 proxy layer*. The C2 proxy layer contains several dedicated HTTP servers (not bots), which form an additional layer between the proxy bots and the true root of the C2 communication. Periodically, the proxy bots interact with the C2 proxy layer to update their command repository, and to forward the stolen data collected from the bots upward in the hierarchy.

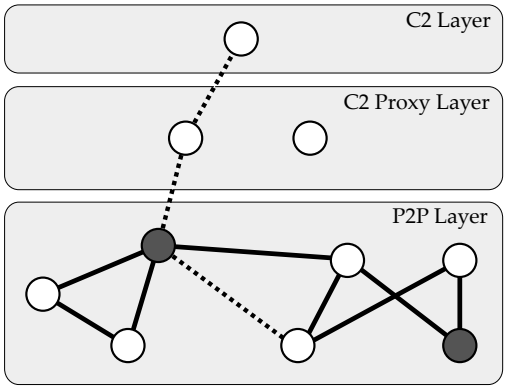

Figure 1: Topology of P2P Zeus. Shaded nodes represent proxy bots. The dotted line shows the information flow between a harvester bot and the C2 layer.

Finally, at the top of the hierarchy is the *C2 layer*, which is the source of commands and the end destination of stolen data. Commands propagate downward from the C2 layer, through the C2 proxy layer to the proxy bots, where they are fetched by harvester bots. Similarly, data stolen by harvester bots is collected by the proxy bots, and periodically propagated up until it ultimately reaches the C2 layer.

As mentioned in Section 1, the main P2P network is divided into several virtual *sub-botnets* by a hardcoded sub-botnet identifier in each bot binary. Since each of these sub-botnets is independently controlled, the C2 layer may contain multiple command sources and data sinks.

3 P2P Protocol

This section describes our analysis results on the Zeus P2P communication protocol. The results are based on Zeus variants we tracked between February 2012 and July 2013. In that time, several changes were made to the protocol by the Zeus authors. The results presented here apply to all recent P2P Zeus versions, except where noted differently.

We first provide a high level overview of the Zeus P2P protocol in Section 3.1. Next, we describe the encryption used in Zeus traffic in Section 3.2. Sections 3.3 and 3.4 provide a detailed overview of the Zeus message structure. Finally, Section 3.5 describes in detail how the Zeus P2P protocol operates.

3.1 Overview

As mentioned in Section 1, the Zeus P2P network's main functions are (1) to facilitate the exchange of binary and configuration updates among bots, and (2) to propagate lists of *proxy bots*. Most normal communication between bots is based on UDP. The exceptions are Command and Control (C2) communication between harvester bots and proxy bots, and binary/configuration update exchanges, both of which are TCP-based.

Bootstrapping onto the network is achieved through a hardcoded bootstrap peer list. This list contains the IP addresses, ports and unique identifiers of up to 50 Zeus bots. Zeus port numbers range from 1024 to 10000 in versions after June 2013, and from 10000 to 30000 in older versions. Unique identifiers are 20 bytes long and are generated at infection time by taking a SHA-1 hash over the Windows *ComputerName* and the Volume ID of the first hard-drive. These unique identifiers are used to keep contact information for bots with dynamic IPs up-to-date.

Network coherence is maintained through a push-/pull-based peer list exchange mechanism. Zeus generally prefers to push peer list updates; when a bot receives a message from another bot, it adds this other bot to its local peer list if the list contains less than 50 peers. Bots in desperate need of new peers can also actively request them. In principle, the peer pushing mechanism facilitates peer list poisoning attacks against Zeus. However, as we will see in Sections 3.2, 3.5.1 and 4, Zeus includes several resilience measures which severely complicate poisoning attacks.

Zeus bots check the responsiveness of their neighbors every 30 minutes. Each neighbor is contacted in turn, and given 5 opportunities to reply. If a neighbor does not reply within 5 retries, it is deemed unresponsive, and is discarded from the peer list. During this verification round, every neighbor is asked for its current binary and configuration file version numbers. If a neighbor has an update available, the probing bot spawns a new thread to download the update. Updates are signed using RSA-2048, and are applied *after* the bot has checked that the update's embedded version number is higher than its current version. Thus, it is impossible to force bots to "update" to older versions.

The neighbor verification round is also used to pull peer list updates if necessary. If the probing bot's peer list contains less than 25 peers, it asks each of its neighbors for a list of new neighbors. The returned peer lists can contain up to 10 peers. The returned peers are selected by minimal Kademlia-like XOR distance to the requesting bot's identifier [10]. However, we note that the Zeus P2P network is *not* a Distributed Hash Table, and apart from this XOR metric the protocol bears no resemblance to Kademlia.

In case a Zeus bot finds all of its neighbors to be unresponsive, it attempts to re-bootstrap onto the network by contacting the peers in its hardcoded peer list. If this also fails, the bot switches to a DGA backup channel, which can be used to retrieve a fresh, RSA-2048 signed, peer list. Additionally, in recent variants of Zeus, the DGA channel is also contacted if a bot is unable to retrieve updates for a week or longer. This is a very important resilience feature, as it allows the botnet to recover from peer list poisoning attacks. The DGA mechanism is described in more detail in Section 4.

As mentioned, one of the most important functions of the Zeus P2P network is to propagate lists of proxy bots. These proxy bots are periodically selected from the general bot population, and are contacted by bots which need to fetch commands and drop stolen data. Like the peer list exchange mechanism, the proxy list mechanism is also push-/pull-based. When a new proxy bot is appointed by the botmasters, an RSA-2048 signed push message is disseminated through the network to announce it.

Bots are commanded in two ways. (1) As mentioned before, harvester bots can contact proxy bots to retrieve commands. (2) Configuration file updates can also be used to convey commands to the bots.

3.2 Encryption

Until recently, bot traffic was encrypted using a rolling XOR algorithm, known as "visual encryption" from centralized Zeus [18], which encrypts each byte by XORing it with the preceding byte. Since June 2013, Zeus uses RC4 instead of the XOR algorithm, using the recipient's bot identifier as the key. Rogue bots used by analysts to infiltrate the network typically use continuously changing bot identifiers to avoid detection [11]. The new RC4 encryption is a problem, because a rogue bot may not always know under which identifier it is known to other bots, thus preventing it from decrypting messages it receives. In addition, RC4 increases the load on botnet detection systems which rely on decrypting C2 traffic [12].

Zeus uses RSA-2048 to sign sensitive messages originating from the botmasters, such as updates and proxy announcements. In all P2P Zeus variants we studied, update exchanges and C2 messages feature RC4 encryption over an XOR encryption layer. For these messages, either the identifier of the receiving bot or a hardcoded value is used as the RC4 key, depending on the message type.

3.3 Message Structure

This section describes the structure of Zeus network messages. Zeus messages vary in size, but have a minimum length of 44 bytes. The first 44 bytes of each message form a header, while the remaining bytes form a payload concatenated with padding bytes. The Zeus message structure is illustrated in Figure 2. The following message structure diagrams are to scale. Shaded areas do not represent part of the message structure itself, but serve to align the fields in the figures.

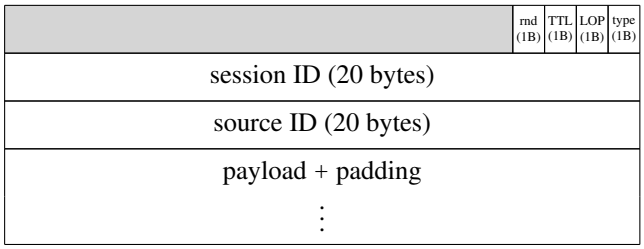

Figure 2: The Zeus message structure.

3.3.1 rnd (random)

In Zeus versions which use the XOR encryption, this byte is set to a random value. This is done to avoid leaking information, since the XOR encryption leaves the first byte in plaintext. In Zeus versions which use RC4 for message encryption, this byte is set to match the first byte of the session ID, so that it can be used to confirm that packet decryption was successful. Backward compatibility with older bots is achieved by falling back to the XOR encryption if RC4 decryption fails.

3.3.2 TTL (time to live)

The TTL field is usually unused, in which case it is set to a random value, or to the second byte of the session ID for variants using RC4 encryption. However, for certain message types, this field serves as a time to live counter. A bot receiving a message using the TTL field forwards it with a decremented TTL. This continues iteratively until the TTL reaches zero.

3.3.3 LOP (length of padding)

Zeus messages end with a random amount of padding bytes. This is most likely done to confuse signature-based intrusion detection systems. The length of padding field indicates the number of padding bytes appended to a message.

3.3.4 type

This field indicates the type of the message. The message type is used to determine the structure of the payload, and in certain cases the meaning of some of the header fields, such as the TTL field. Valid Zeus message types are described in Section 3.4.

3.3.5 session ID

When a Zeus bot sends a request to another bot, it includes a random session ID in the request header. The corresponding reply will include the same session ID, and incoming replies with unexpected session ID values are discarded. This makes it more difficult for attackers to blindly spoof Zeus messages.

3.3.6 source ID

This field contains the 20 byte bot identifier of the sending bot. The source ID field facilitates the push-based peer list update mechanism, where a bot receiving a message adds the sender of the message to its peer list in case the peer list contains less than 50 peers.

3.3.7 payload

This is a variable-length field which contains a payload dependent on the message type. The structures of relevant message payload types are described in detail in Section 3.4.

3.3.8 padding

This field contains a random number of random (non-zero) padding bytes. The number of padding bytes is specified in the length of padding field in the message header.

3.4 Payload Structure

In this section, we describe the structure and usage of the most relevant Zeus message types. Each of these message types is communicated over UDP, except for C2 messages and updates, which are exchanged over a TCP connection.

3.4.1 Version request (type `0x00`)

Version request messages are used to request a bot's current binary and configuration file version numbers. These messages usually contain no payload, but may contain a payload consisting of a little endian integer with value 1, followed by 4 random bytes. Such a payload serves as a marker to indicate that the requesting peer wants to receive a type `0x06` proxy reply message (see Section 3.4.7).

3.4.2 Version reply (type `0x01`)

A version reply contains the version numbers of the binary and configuration files of the sender. The binary version indicates the sender's Zeus version, while the configuration file version indicates the sender's configuration file version. A TCP port is also sent, which may be contacted to download the updates via TCP, although some Zeus variants also support using UDP for this (see Sections 3.4.5 and 3.4.6). Version replies end with 12 random bytes. The reply structure is shown in Figure 3.

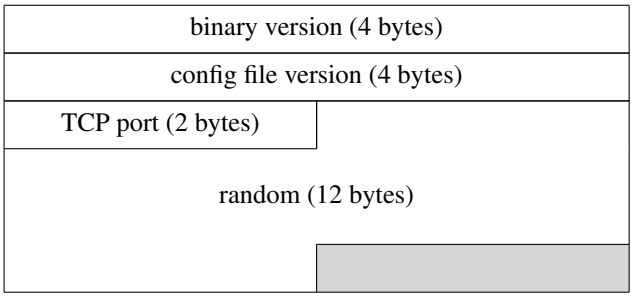

Figure 3: Version reply payload (22 bytes).

3.4.3 Peer list request (type `0x02`)

Peer list requests (Figure 4) are used to request new peers from other bots. Zeus only sends active peer list requests if its peer list is becoming critically short (less than 25 peers). Otherwise, bots typically rely on storing the senders of incoming requests.

The payload of a peer list request consists of a 20 byte identifier, followed by 8 random bytes. The responding peer will return the peers from its own peer list that are closest to the requested identifier.

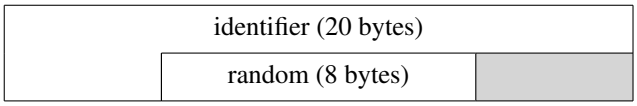

Figure 4: Peer list request payload (28 bytes).

3.4.4 Peer list reply (type `0x03`)

Peer list replies contain 10 peers from the responding peer's peer list which are closest to the requested identifier. If the responding peer knows fewer than 10 peers, then as many peers as possible (potentially zero) are returned, and any remaining peer slots are zeroed out. For each returned peer, the payload format is as shown in Figure 5. Zeus supports both IPv4 and IPv6, but in practice we have observed very few IPv6 peers. The IP type field indicates whether the peer is reachable via IPv4 (set to 0) or IPv6 (set to 2). The remaining fields contain the peer's identifier, IP address and UDP port. Any unused fields are randomized.

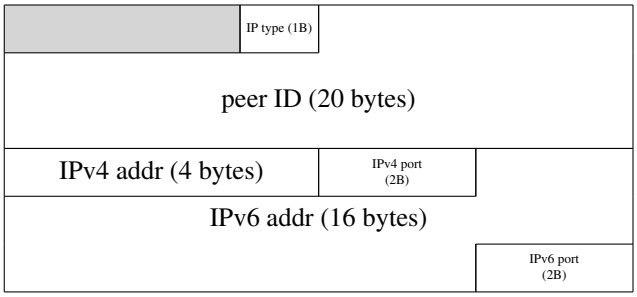

Figure 5: Peer struct (45 bytes).

3.4.5 Data request (type 0x04/0x68/0x6A)

A UDP data request payload, shown in Figure 6, starts with a single byte indicating the kind of requested data. This byte is set to 1 for a configuration file download, or to 2 for a binary update. The offset field indicates the word offset into the data at which transmission should start, and the size field specifies how many data bytes should be sent. TCP data requests consist of a message header with type 0x68 for a binary request, or type 0x6A for a configuration request.

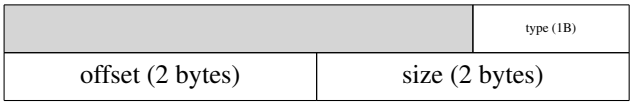

Figure 6: Data request payload (5 bytes).

3.4.6 Data reply (type 0x05/0x64/0x66)

UDP data transfers (type 0x05) are sent in chunks of 1360 bytes, until no more data is available. If a bot receives a data reply containing less than 1360 data bytes, it assumes that this is the last data block, and ends the download. If a data reply takes longer than 5 seconds to arrive, the download is aborted, and the maximum total size of any download is 10MB. These constraints mean that it is not possible to launch "tarpit" attacks, where bots are tied up by very slow and never ending downloads.

Each data reply (Figure 7) starts with a 4 byte randomly chosen file identifier, followed by the requested data. The transmitted files end with an RSA-2048 signature over the MD5 hash of the plaintext data, and are encrypted with RC4 using a hardcoded key on top of an XOR encryption layer. Before applying an update, Zeus checks that the version number contained in the update is strictly higher than its current version number. This means that it is not possible to make Zeus bots revert to older versions.

TCP data transfers start with a message header of type 0x64 for a binary update, or 0x66 for a configuration update, followed by the RC4 encrypted data.

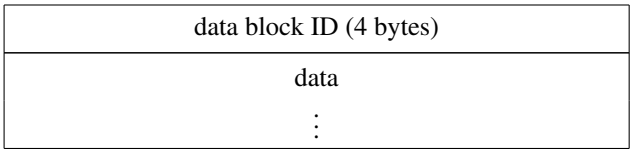

Figure 7: Data reply payload (length varies).

3.4.7 Proxy reply (type 0x06)

Proxy replies return proxy bots in response to version requests carrying a proxy request marker. A proxy reply can contain up to 4 proxy bot entries, each of which is RSA-2048 signed. Each proxy entry is formatted as shown in Figure 8. The format is similar to that used in peer list replies, except that the IP type field is 4 bytes long, and there is an RSA signature at the end of each proxy entry.

3.4.8 Proxy announcement (type 0x32)

Proxy announcements are similar to proxy replies, but are actively pushed through the Zeus network by bots which are appointed as proxies by the botmasters. Newly appointed proxy bots announce themselves to all their neighbors, which pass on the message to all their neighbors, and so on. This continues until the TTL field (Section 3.3) reaches zero. The TTL field has an initial value of 4 for proxy announcements. Thus, proxy announcements propagate very rapidly, although they cannot reach NATed bots directly. Proxy announcements contain a single proxy entry of the same format used in type 0x06 messages, as shown in Figure 8.

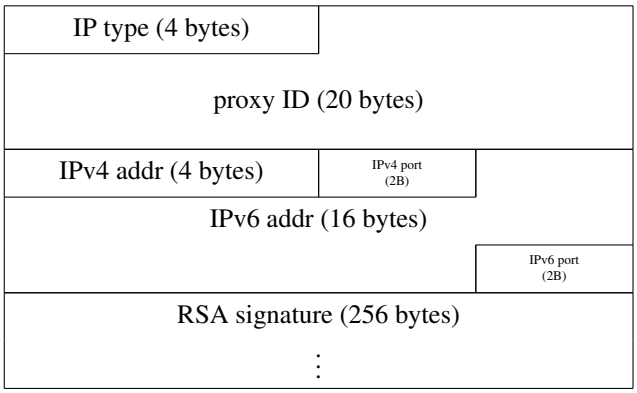

Figure 8: Proxy struct (304 bytes).

3.4.9 C2 message (type 0xCC)

Unlike most message types, C2 messages are only exchanged between harvester bots and proxy bots, and are exchanged over TCP. C2 messages are used as wrappers for HTTP messages. Because of this, we suspect that the

communication between proxy bots and the C2 proxy layer is HTTP-based. The HTTP-based C2 protocol is heavily based on the C2 protocol used in centralized Zeus [2, 6]. An example C2 HTTP header for a command request is shown in Figure 9. The X-ID field specifies the sub-botnet for which a command is being requested.

```
POST /write HTTP/1.1
Host: default
Accept-Encoding:
Connection: close
Content-Length: 400
X-ID: 100
```

Figure 9: C2 HTTP header.

The HTTP header is followed by an HTTP payload, which consists of several, optionally zlib-compressed, data fields. The payload begins with a header specifying the payload size and flags, and the number of data fields that follow. The payload header ends with an MD5 hash of the combined data fields, which is used to verify message integrity.

Each data field is XOR encrypted, and starts with a header specifying the field type, flags, and compressed and uncompressed sizes. After the header comes the actual data, the structure of which is dependent on the field type.

C2 request messages typically contain several status and information fields about the requesting bot. Typical fields included in C2 requests are shown in Table 1. Note that the type numbers of data fields are completely independent from Zeus message type numbers.

Type	Content
0x65	System name and volume ID
0x66	Bot identifier
0x67	Infecting spam campaign
0x6b	System timing information
0x77	Stolen data

Table 1: Typical C2 request fields.

The most important data field contained in a C2 response is the command field, which has type 0x01. It contains an MD5 hash used to verify integrity of the command, followed by the command itself in the form of an ASCII-string. Notable command strings are listed in Table 2.

Command	Meaning
user_execute	Execute file at URL
user_certs_get	Steal crypto certificates
user_cookies_get	Steal cookies
ddos_url	DDoS a given URL
user_homepage_set	Set homepage to URL
fs_pack_path	Upload local files to botmaster
bot_bc_add	Open VNC server

Table 2: Notable C2 command strings.

As can be seen from Table 2, Zeus supports a diverse set of commands, which goes far beyond that of a typical banking trojan. The supported commands include dropping files, launching DDoS attacks, providing remote access to the botmasters, and stealing a plethora of credentials. Aside from banking credentials, we have observed Zeus stealing Skype and MSN database files, as well as Bitcoin wallets.

3.5 Communication Patterns

Each Zeus bot runs a passive thread, which listens for incoming requests, as well as an active thread, which periodically generates requests to keep the bot up-to-date and well-connected. We describe the behavior of each of these threads in turn.

3.5.1 Passive thread

Every Zeus bot listens for incoming messages in its passive thread. A Zeus bot receiving an incoming request attempts to handle this request as described in Section 3.4. The sender of any successfully handled request is considered for addition to the receiving bot's peer list. This is the main mechanism used by *externally reachable* Zeus bots to learn about neighbors, and it is also how new bots introduce themselves to the network. If the receiving bot has fewer than 50 neighbors, then it always adds the sender of the request to its peer list. Additionally, if the identifier of the sender is already present in the peer list, then the corresponding IP address and port are updated. This is done to accommodate senders with dynamic IPs and discard stale dynamic IPs. If the identifier of the sender is not yet known, but the peer list already contains 50 peers or more, then the sending peer is stored in a queue of peers to be considered for addition during the next neighbor verification round (see Section 3.5.2).

Before adding a new peer to the peer list, a number of sanity checks are performed. First, only peers which have a source port in the expected range are accepted. NATed bots may make it into the peer lists of other bots, if they happen to choose a port in the valid range. Additionally, only one IP address per /20 subnet may occur in a bot's peer list at once. This defeats peer list poisoning attempts which use IP ranges within the same subnet. Recent versions of Zeus also include an automatic blacklisting mechanism, which blacklists IPs that contact a bot too frequently in a specified time window. This mechanism further complicates efficient crawling and poisoning of the network.

When a type 0x32 proxy announcement arrives, its signature is first checked for validity. If the message passes the check, the TTL field is decremented and the message is forwarded to all known neighbors if the TTL is still positive. Furthermore, new proxy bots which pass verification

are considered for addition to the receiving bot's proxy list. The proxy list is similar to the peer list, but is maintained separately. If the identifier of the new proxy is already in the proxy list, then the corresponding IP address and port are updated. Otherwise, if a proxy list entry is found that is over 100 minutes older than the new proxy, this entry is overwritten with the new proxy (this is not done for type `0x06` proxy replies). In any other case, the new proxy is added to the end of the proxy list. Finally, the proxy list is truncated to its maximum length of 10 entries, effectively discarding the new proxy if the proxy list was already 10 entries long.

3.5.2 Active thread

The Zeus active communication pattern consists of a large loop which repeats every 30 minutes. The function of the active communication loop is to keep Zeus itself, as well as the peer list and proxy list, up to date.

In each iteration of the loop, Zeus queries each of its neighbors for their binary and configuration file versions. This step serves to keep the bot up to date, and to check each neighbor for responsiveness. If Zeus knows fewer than 4 proxy bots, it piggybacks a proxy request marker with each version request. Each peer is given 5 chances to respond to a version request. If a peer fails to answer within the maximum number of retries, Zeus checks if it has working Internet access by attempting to contact `www.google.com` or `www.bing.com`. If it does, the unresponsive peer is discarded. If the peer responded and is found to have an update available, the update is downloaded in a separate thread.

After version querying all peers in its peer list, Zeus proceeds to handle any pending peers which were queued from incoming requests (see Section 3.5.1). Pending peers are only handled if the peer list contains fewer than 50 peers, and the procedure is stopped as soon as the peer list reaches length 50. Each pending peer is sent a single version request, and is added to the peer list if it responds.

Finally, if the peer list contains fewer than 25 peers, the bot will actively send peer list requests to each of its neighbors until the peer list reaches a maximum size of 150 peers. This is only done once every 6 loop cycles (3 hours), and is an emergency measure to prevent the bot from becoming isolated. If, despite this effort, a bot does find itself isolated, it will attempt to recover connectivity by contacting its hardcoded bootstrap peer list. If this also fails, the bot will enter DGA mode, as further described in Section 4.

4 Domain Name Generation Algorithm

As mentioned in Section 3.1, Zeus contains a Domain Generation Algorithm, activated if all of a bot's neighbors are unresponsive, or the bot cannot fetch updates for a week. The DGA generates domains where Zeus can download a fresh RSA-2048 signed peer list. The DGA is a very potent backup mechanism, which makes long term poisoning or sinkholing attacks against Zeus very difficult [11].

4.1 Algorithm Details

The Zeus Domain Generation Algorithm generates 1000 unique domains per week. A bot entering the DGA starts at a random position in the current week's domain list and sequentially tries all domains until it finds a responsive domain. The DGA uses top-level domains taken from the set {`biz`, `com`, `info`, `net`, `org`, `ru`}. The Zeus DGA bears some resemblance to the DGA of Murofet, a malware known to be related to centralized Zeus [8].

```
for(i = 0; i < 1000; i++) {
    S[0] = (year + 48) % 256;   S[1] = month;
    S[2] = 7 * (day / 7);       *(int*)&S[3] = i;

    /* convert hash to domain name */
    name = "";  hash = md5(S);
    for(j = 0; j < len(hash); j++) {
        c1 = (hash[j] & 0x1F) + 'a';
        c2 = (hash[j] / 8) + 'a';
        if(c1 != c2 && c1 <= 'z') name += c1;
        if(c1 != c2 && c2 <= 'z') name += c2;
    }

    /* select TLD for domain */
    if(i % 6 == 0) name += ".ru";
    else if(i % 5 != 0) {
        if(i & 0x03 == 0) name += ".info";
        else if(i % 3 != 0) {
            if((i % 256) & 0x01 != 0) name += ".com";
            else name += ".net";
        } else name += ".org";
    } else name += ".biz";

    domains[i] = name;
}
```

Figure 10: The P2P Zeus Domain Name Generation Algorithm.

The Zeus DGA is shown in C-like pseudocode in Figure 10. The code shown generates all 1000 domains for a given week. The generation of a domain name starts by taking the MD5 hash over the concatenation of (transformations of) the year, month, day, and domain index. The MD5 hash is then used to generate a domain name of at most 32 lower case alphabetic characters. Finally, the domain is completed by selecting one of the six top-level domains and concatenating it to the domain name.

5 Related Work

Early insights on P2P Zeus were provided by Lelli [9] and `abuse.ch` [17]. Special attention to the lifecycle of Zeus has been given by Stone-Gross [15].

The most recent previous account of P2P Zeus that we know of is given in a technical report by CERT.pl [3]. The research of CERT.pl took place independently from, but

concurrently with, our own research. While CERT.pl has focused on the P2P Zeus malware as a whole, we provide a more in-depth account focused specifically on the peer-to-peer protocol and its resilience. Additionally, the CERT.pl report predates the Zeus protocol change of June 2013, and thus does not include information on the new protocol features and encryption mechanism.

Our previous work has provided a comparison of the Zeus P2P protocol to other P2P botnet protocols [11]. Our current work differs in that we provide a much more detailed insight into the functionality and resilience of P2P Zeus in particular. To the best of our knowledge, our work is the most detailed account of P2P Zeus to date.

6 Conclusion

P2P Zeus is a significant evolution of earlier Zeus variants. Compared to traditional centralized versions of Zeus, P2P Zeus is much more resilient against takedown attempts. Potential countermeasures against P2P Zeus are complicated by its application of RSA-2048 signatures to mission critical messages, and rogue bot insertion is complicated by the Zeus message encryption mechanism which makes the use of random bot identifiers impossible. Poisoning attempts are forced to use widely distributed IPs due to a per-bot IP filter which only allows a single IP per /20 subnet. The network's resilience against takedown efforts is further increased by its use of a Domain Generation Algorithm backup channel, and by an automatic blacklisting mechanism. P2P Zeus demonstrates that modern P2P botnets represent a new level of botnet resilience, previously unseen in centralized botnets.

Acknowledgements

We would like to thank Tillmann Werner for the collaboration on reversing the Zeus peer-to-peer protocol. We also thank Christian J. Dietrich and Tomasz Bukowski for sharing their insights with us. This work was supported by the European Research Council Starting Grant "Rosetta" and the EU FP7-ICT-257007 SysSec project.

References

[1] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Proceedings of the 21st USENIX Security Symposium (USENIX Sec'12)*, Bellevue, WA, USA, 2012.

[2] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the Analysis of the Zeus Botnet Crimeware Toolkit. In *Proceedings of the 8th Annual Conference on Privacy, Security and Trust (PST'10)*, Ottawa, Ontario, Canada, August 2010.

[3] CERT.pl. Zeus P2P Monitoring and Analysis, 2013. Technical Report. `http://www.cert.pl/PDF/2013-06-p2p-rap_en.pdf`.

[4] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC'07)*, 2007.

[5] D. Dittrich and S. Dietrich. P2P as Botnet Command and Control: A Deeper Insight. In *Proceedings of the 3rd International Conference on Malicious and Unwanted Software (MALWARE'08)*, October 2008.

[6] N. Falliere and E. Chien. Zeus: King of the Bots, 2009. Technical Report, Symantec.

[7] R. Hund, M. Hamann, and T. Holz. Towards Next-Generation Botnets. In *Proceedings of the 4th European Conference on Computer Network Defense (EC2ND'08)*, 2008.

[8] K. Itabashi. How Trojan.Zbot.B!inf Uses the Crypto API, 2010. Technical Report, Symantec. `http://www.symantec.com/connect/blogs/how-trojanzbotbinf-uses-crypto-api`.

[9] A. Lelli. Zeusbot/Spyeye P2P Updated, Fortifying the Botnet, 2012. Technical Report, Symantec. `http://www.symantec.com/connect/blogs/zeusbotspyeye-p2p-updated-fortifying-botnet`.

[10] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[11] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. Dietrich, and H. Bos. P2PWNED: Modeling and Evaluating the Resilience of Peer-to-Peer Botnets. In *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P'13)*, San Francisco, CA, USA, May 2013.

[12] C. Rossow and C. J. Dietrich. ProVeX: Detecting Botnets with Encrypted Command and Control Channels. In *Proceedings of the 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'13)*, July 2013.

[13] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. van Steen, F. C. Freiling, and N. Pohlmann. Sandnet: Network Traffic Analysis of Malicious Software. In *Proceedings of the 1st ACM EuroSys Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS'11)*, 2011.

[14] R. Sherstobitoff. Inside the World of the Citadel Trojan, 2013. Technical Report, McAfee.

[15] B. Stone-Gross. The Lifecycle of Peer-to-Peer Zeus, 2012. Technical Report, Dell SecureWorks. `http://www.secureworks.com/cyber-threat-intelligence/threats/The_Lifecycle_of_Peer_to_Peer_Gameover_ZeuS/`.

[16] D. Tarakanov. Ice IX: Not Cool At All, 2011. Technical Report, Kaspersky Lab. `http://www.securelist.com/en/blog/563/Ice_IX_not_cool_at_all`.

[17] `abuse.ch`. Zeus Gets More Sophisticated Using P2P Techniques, 2011. Technical Report. `http://www.abuse.ch/?p=3499`.

[18] J. Wyke. What is Zeus?, 2011. Technical Report, SophosLabs.

[19] G. Yan, S. Chen, and S. Eidenbenz. RatBot: Anti-enumeration Peer-to-Peer Botnets. In *Lecture Notes in Computer Science, vol. 7001*, 2011.

[20] T.-F. Yen and M. K. Reiter. Revisiting Botnet Models and Their Implications for Takedown Strategies. In *Proceedings of the 1st Conference on Principles of Security and Trust (POST'12)*, 2012.