Poster: HTTP Botnet Resilient to Takedown

Chia-Mu Yu (Faculty)
Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan

I. Motivation

Botnets have been the most dangerous threat on the Internet. Recently, advanced botnets Flashback, Makadocs, and Vernot evade the detection by taking advantage of public web services Twitter, Google Docs, and Evernote, respectively. However, the collaboration between the government and industry partners may still disrupt the botnet communications [2].

Here, to study the possible trends in evolution of botnets, we play the devil's advocate by using the vulnerabilities of cloud storage services (CSSs) to design a stealthy and robust C&C channel. This is a vital step in developing a truly effective countermeasure to the potential exploitation of CSSs by the botnet C&C.

II. Covert Channel via Data Deduplication

Chunk-based cross-user client-side data deduplication (C3D2) has been widely used in CSSs to achieve storage savings and reduce the need for bandwidth between the client devices and the storage servers. The file to be uploaded is first partitioned into chunks. For a chunk x, the user first calculates and transmits a hash value $\bar{h}(x)$ to the server. Upon receiving the hash value, the server checks whether it has a copy of $\bar{h}(x)$. If so, the user keeps x locally, saving the bandwidth of sending x to the server, and the server maps $\bar{h}(x)$ to the current user, saving the storage space for another copy of x. If not, the user is asked to transmit x to the server.

If C3D2 is used in the CSS, the client may know the chunk existence status (CES) in the CSS by looking at the hash response. As a result, a covert channel is established because when one uploads (deletes) a specific chunk x, another party gets a positive (negative) hash response and can be thought of as "receiving" a bit 1 (0) [1].

III. Security Assumption

Though the design of our botnet uses cloud storage as a communication medium, we assume that the CSS administrator will provide full support to the defender except for ethical and economic conflicts. On the other hand, though code obfuscation is usually adopted, by reverse-engineering the bot binary from the malware sample, the defender is assumed to be able to obtain the full malware code and access the credentials. These two assumptions could be critical; in this sense, a fast-flux botnet is no longer robust since the DNS administrator can unregister the suspicious domain name extracted from the bot binary. Hence, the botmaster needs to confront a situation where the botnet can hardly survive.

IV. Botnet Resilient to Takedown

Basic Idea. To strengthen botnet robustness, we introduce a notion of *undeletability*, which means that, although the commands are placed on the CSS, no one, including the defender and CSS administrator, can erase the commands. This can be the strongest notion of botnet robustness, to the best of our knowledge. The rationale behind the undeletability is that C3D2 enables different users to share the same data chunks. Because the botnet commands are possibly comprised of benign users' data, the defender and the CSS administrator cannot simply erase the commands. In this sense, the botmaster sees the benign users' data as hostages.

In addition, we introduce a notion of *automatic recovery* as a complement to the undeletability. More precisely, the defender can still add noises to disrupt the botnet command albeit the command cannot be erased. The automatic recovery ensures that the command can only be disturbed partly within a time period and C&C will be recovered automatically unless the CSS administrator keeps paying economic penalties.

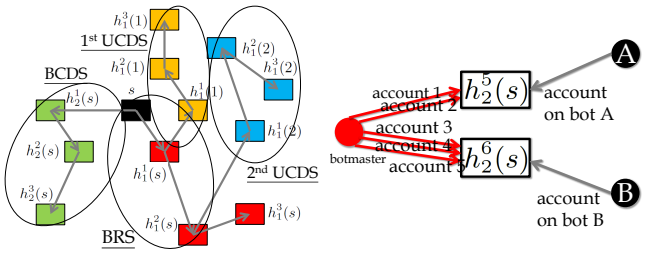
In the following, we consider that the malware is hardcoded with a random seed s and pseudorandom function $h(\cdot)$. The bot spontaneously registers a CSS account and performs the hash query/uploading, as described in Sec. II. Define $h_i^1(x) = h(i||x)$ and $h_i^j(x) = h_i(h_i^{j-1}(x))$, for $i, j \geq 1$.

Bot Registration. A *bot registration sequence* (BRS) is defined as a hash chain of infinite length: s, $h_1^1(s)$, $h_1^2(s)$, ... The characteristic of a BRS is that, for each consecutive pair of hash values, the lower-order hash value can be thought of as the data chunk to be queried, while the higher-order hash value serves as the hash value of the data chunk to be queried. For example, one may send $h_1^2(s)$ to query the CES of the chunk $h_1^1(s)$.

When joining a botnet, a new bot finds out an index w such that the chunks $h_1^0(s) = s$, $h_1^1(s)$, ..., and $h_1^{w-2}(s)$ have already been in the CSS but $h_1^{w-1}(s)$ is not. More specifically, a new bot initially queries $h_1^1(s)$ to see whether there has already been a copy of s. If so, then the bot keeps querying the CES of subsequent chunks in BRS until there is a negative response from the CSS. Given such a w, the bot sees w as its bot ID and uploads $h_1^{w-1}(s)$ to the CSS to finish the bot registration.

To detect the bot registration, the botmaster does not need to monitor all of the hash values in the BRS. Instead, the botmaster constantly queries only $h_1^w(s)$ to sense the new bot registration because of its knowledge of all the registered bots. Once the botmaster senses an update of $h_1^{w-1}(s)$, it knows that a new bot has been recruited.

Command Dissemination. A *broadcast command dissemi-*



(a) BRS, BCDS, and UCDSs (b) Multi-account approach

Fig. 1: BRS, BCDS, UCDSs, and multi-account approach

nation sequence (BCDS) is defined as a hash chain of infinite length: s, $h_2^1(s)$, $h_2^2(s)$, ... Consider that the botmaster wants to disseminate a command b to the entire botnet. The *chunk set* $\mathcal{C}_{s,h}^z(b)$ of a bit string $b = b_1 b_2 \dots b_\rho$, where ρ is the number of bits required to represent a command, is defined as a set of chunks $\{h^{i-1}(s)|b_i = z, 1 \leq i \leq \rho\}$. We claim that a bit string b can be encoded and represented by the CES of $\mathcal{C}_{s,h_2}^0(b)$ and $\mathcal{C}_{s,h_2}^1(b)$ because one who queries the CES of $\mathcal{C}_{s,h_2}^0(b)$ and $\mathcal{C}_{s,h_2}^1(b)$ may receive the corresponding negative and positive responses, which can be interpreted as 0 and 1, respectively. Note that in the case where each command has the same length ρ, and the bot reads different subsequences of BCDS depending on different time periods. In other words, if the next command is to be issued, different chunk set needs to be used but we omit the notational details to ease the presentation.

For example, a copy of the chunk $h_2^3(s)$ in the server implies the positive reponse of the hash query $h_2^4(s)$ and can be interpreted as $b_4 = 1$, while $b_6 = 0$ corresponds to the inexistence of the chunk $h_2^5(s)$ in the server. The botmaster can disseminate a command b to the entire botnet through BCDS by uploading the chunks in $\mathcal{C}_{s,h_2}^1(b)$ to the server via multiple CSS accounts spontaneously registered by the botmaster and doing nothing for the chunks in $\mathcal{C}_{s,h_2}^0(b)$. An illustration of BRS and BCDS is shown in Fig. 1a, where both BRS and BCDS starts with s.

V. Discussion

Undeletability. During the command dissemination, the goal of the botmaster uploading the chunks in $\mathcal{C}_{s,h_2}^1(b)$ via multiple accounts is to create an illusion that the chunks are owned by multiple users. In particular, the simplest form of such a *multi-account approach* is that for each chunk in $\mathcal{C}_{s,h_2}^1(b)$, the botmaster spontaneously registers a number of CSS accounts and then uploads the chunk to the CSS via these accounts. An example can be found in Fig. 1b, where the botmaster creates an illusion that three users own $h_2^5(s)$ and four accounts share $h_2^6(s)$.

In a C3D2-based CSS, there is always a possibility that the chunks in $\mathcal{C}_{s,h_2}^1(b)$ are also uploaded by legitimate users. Furthermore, the above approach always results in the situation where each chunk in $\mathcal{C}_{s,h_2}^1(b)$ is owned by different users. One can observe that even when the defender knows that the commands have been placed on BCDS, the chunks in $\mathcal{C}_{s,h_2}^1(b)$ still cannot be removed. Otherwise, it might jeopardize the data integrity of the benign users.

A more aggressive way of C&C disruption is to simply delete the chunks in $\mathcal{C}_{s,h_2}^1(b)$ because one may argue that the chances of them colliding with a real user's chunk could be rather small. In other words, if a chunk is the output of a bot's pseudorandom function, then it may safely be regarded as malicious content and deleted. However, despite the small chance, the CSS administrator should not take a risk of deleting the benign user's data. Otherwise, the accounts created by the botmaster and used to upload chunks only once may claim that it is in a legitimate use, its chunks are no longer available, and demands an indemnity. Such a dispute would be difficult to resolve.

Automatic Recovery. Instead of removing the chunks in $\mathcal{C}_{s,h_2}^1(b)$, an alternative attempt to disrupt the command dissemination is for the CSS administrator to fill all the positions of BCDS. This is in some sense similar to adding a huge amount of noises to BCDS, and can temporarily disrupt the C&C, at the expense of the increased use of the storage space. Notice that this only achieves temprary C&C disruption because as time goes by, different parts of BCDS will be used. Thus, the CSS administrator needs to keep filling the chunks in $\mathcal{C}_{s,h_2}^1(b)$, during the whole botnet's lifetime. Otherwise, the BCDS-based C&C channel will be recovered automatically because no noise is added to it and the command can be extracted easily.

Noises on BRS. The benign user's data chunks might happen to be equivalent to the hash values in the BRS before they are used for the bot registration. Although having the impact on counting the bots, this in fact does not compromise the bot registration.

Noises on BCDS. The benign user's data chunks might happen to be equivalent to the hash values in the BCDS before they are used for the command dissemination. This may destroy the command integrity. A simple solution is to apply an error correction code to the command first and then upload the chunks in $\mathcal{C}_{s,h_2}^1(\mathcal{E}(b))$, where $\mathcal{E}(b)$ denotes the encoded command.

Unicast Command Dissemination. Since the bot ID w can represent a specific bot, by defining *unicast command dissemination sequence* (UCDS) as a hash chain: w, $h_2^1(w)$, $h_2^2(w)$, ..., the botmaster can send the command to the individual bot in a way similar to broadcasting the command. In essence, we also claim that a bit string b can be encoded and represented by the CES of $\mathcal{C}_{w,h_2}^0(b)$ and $\mathcal{C}_{w,h_2}^1(b)$ because one who queries the CES of $\mathcal{C}_{w,h_2}^0(b)$ and $\mathcal{C}_{w,h_2}^1(b)$ may receive the corresponding negative and positive responses, which can be interpreted as 0 and 1, respectively. An illustration of UCDSs is shown in Fig. 1a

References

[1] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services, the case of deduplication in cloud storage. *IEEE Security & Privacy Magazine*, 2010.

[2] Microsoft, the FBI, Europol and industry partners disrupt the notorious ZeroAccess botnet. http://www.microsoft.com/en-us/news/press/2013/dec13/12-05zeroaccessbotnetpr.aspx