Analysis of the Use of XOR as an Obfuscation Technique in a Real Data Corpus

Carolina Zarate, Simson Garfinkel, Aubin Heffernan, Scott Horras, Kyle Gorak

Chapter 9

ANALYSIS OF THE USE OF XOR AS AN OBFUSCATION TECHNIQUE IN A REAL DATA CORPUS

Carolina Zarate, Simson Garfinkel, Aubin Heffernan, Scott Horras and Kyle Gorak

Abstract The only digital forensic tools known to provide an automated approach for evaluating XOR obfuscated data are DCCI_Carver and DC3_Carver, two general-purpose carving tools developed by the Defense Cyber Crime Center (DC3). In order to determine the use of XOR as an obfuscation technique and the need to adapt additional tools, we analyzed 2,411 drive images from devices acquired from countries around the world. Using a modified version of the open source tool `bulk_extractor`, evidence of XOR obfuscation was found on 698 drive images, with a maximum of 21,031 XOR-obfuscated features on a single drive. XOR usage in the corpus was observed in files with timestamps between the years 1995 and 2009, with the majority of the usage found in unallocated space. XOR obfuscation was used in the corpus to circumvent malware detection and reverse engineering, to hide information that was apparently being exfiltrated, and by malware detection tools for their quarantine directories and to distribute malware signatures. The results indicate that XOR obfuscation is important to consider when performing malware investigations. However, since the corpus does not contain data sets that are known to have been used by malicious entities, it is difficult to draw conclusions regarding the importance of extracting and examining XOR obfuscated files in criminal, counterintelligence and counterterrorism cases without further research.

Keywords: XOR, obfuscation, steganography, `bulk_extractor`

1. Introduction

A variety of single-byte operators have been used for simple data hiding or obfuscation. A classic technique involved the use of the ROT13 operator [34] to hide off-color jokes disseminated on the early Internet. Common techniques used today include single-byte XOR and single-byte rotate operators. These operators are essentially poor encryption algorithms with 8-bit keys: they are trivial to decode, but to do so the analyst or tool must specifically probe for evidence of their use. If no detection algorithm is employed, even 8-bit encryption is sufficient to hide data.

The only digital forensic tools known to the authors that provide forensic investigators with an automated approach for finding XOR-obfuscated data are DCCI_Carver and DC3_Carver, two general-purpose forensic carving tools developed by the Defense Cyber Crime Center (DC3). Existing commercial and open source digital forensic tools largely ignore single-byte obfuscation techniques. Although it is relatively easy to modify forensic and anti-malware tools to scan for data that have been obfuscated, most digital forensic tools and malware scanners eschew steganography detection because it is computationally expensive and the use of steganography is thought to be low. However, simple obfuscation techniques are sufficient to bypass most commercial and open source malware detection techniques, rendering the lack of de-obfuscation features in tools a vulnerability.

This paper focuses on a specific obfuscation technique, which we call XOR(255). XOR is the exclusive-or binary operation. XOR binary operations are performed with "keys." A key is a series of bytes with which a source file is XORed. XOR(255) is a special case of a single-byte key with the value `0xFF`. Bytewise XOR(255) has the effect of inverting every bit in a source file. XOR(255) has the advantages of being fast (it typically executes in less than one clock cycle on modern architectures), reversible and performed in-place. XOR(255) has the additional property of leaving the file entropy unchanged, allowing processed data to remain invisible to tools that search for encrypted data using entropy techniques. We focused on XOR(255) because of its ease of use and effectiveness at obscuring data.

We created a plug-in for the open source tool `bulk_extractor` [18] that processes data with XOR(255). The structure of `bulk_extractor` is such that each scanner is applied independently to each block of data during processing. The modified tool was employed to scan a corpus of several thousand drive images extracted from used storage devices that had been purchased on the secondary market. The analysis was limited to email addresses, URLs, JPEG photographs, Windows executables, ZIP files and other kinds of easily recognizable data that had been obfuscated by XOR(255). The tool optimistically applied XOR(255) before and after every decompression or decoding step in its search for recognizable structured data.

The analysis revealed that simple obfuscation techniques were present on the drives purchased on three continents over a ten-year period, with some examples of obfuscation as recent as 2009. The use of XOR(255) was limited to a small fraction of the corpus, but the applications of obfuscation are very relevant to forensic investigations. In particular, obfuscation was used to hide signatures in malware and user-generated content that appeared to be targeted for exfiltration. Also, obfuscation was used by legitimate programs to prevent reverse engineering.

2. Prior Work

Simple data obfuscation techniques predate the modern Internet. The ROT13 operator was used in the 1980s to distribute off-color jokes and the answers to riddles on Usenet.

2.1 Use of XOR

The academic literature on XOR obfuscation is relatively sparse. Indeed, documented examples of XOR use are largely confined to online blogs of malware investigators. Mueller [29] notes that Norton Antivirus uses XOR to obfuscate its log files and *Quarantine* folder; we verify this observation in this paper. Hussey [22] suggests that XOR is sometimes used as an obfuscation technique to hide data as it is being exfiltrated; we discovered evidence of XOR used for this purpose.

Several types of malware have been reported as using simple obfuscation techniques to hide data being exfiltrated from a victim machine. Trojan.NTESSESS XORs the results of its commands, including uploaded files, with a changing nine-byte key [12]. Even advanced malware such as Stuxnet, Duqu, Flame and Red October use XOR as the basis of obfuscation algorithms to hide data [35]. Stuxnet uses a 31-byte key with XOR [16]. Duqu XORs data from its keylogger, which it sends to its server [35]. Like Duqu, Flame employs XOR obfuscation techniques on captured keystrokes, screenshots, email messages and conversations recorded via the computer microphone [35]. Red October also uses XOR obfuscation techniques when exfiltrating information from Nokia phones, iPhones and networks [35].

Simple obfuscation techniques have proven to be popular for protecting malicious code and exfiltrated data from anti-virus software and forensic investigators. The SymbOS/OpFake.A!tr.dial malware from Opera Updater is one such example. Apvrille [2] found the Opera Updater malware to contain a 91-byte XOR key as a more complicated algorithm for hiding itself. Similarly, variations of Trojan.PingBed and Trojan.NTESSESS employ XOR obfuscation and embedding techniques in PNG files to evade anti-virus scanners [11, 12]. Cannell [6] suggests that malware may contain XORed URLs as references to malicious files and executables online, creating the appearance of benign files. An example is Trojan.Win32.Patched.mc, which disguises itself as a harmless Flash file. The Trojan file `main.swf` analyzed by StopMalvertising [25] contains an XORed hexadecimal string, which allows an XORed malicious executable to be downloaded. This poses a problem for anti-virus scanners because the files appear harmless, but proceed to download files from URLs that are known to be malicious. XOR has also been used by advanced malware as a method for hiding their main code from anti-virus software – Stuxnet, Duqu, Flame and Red October all obfuscate their payloads and main executables with XOR [35]. In addition, the Storm botnet and other advanced botnets have begun to take advantage of the technique, obfuscating their traffic with XOR in order to avoid identification [24].

Previous research has examined how well anti-virus systems deal with simple obfuscation in malware. Common commercial anti-virus systems tested by Christodorescu and Jha [8] frequently did not identify obfuscated malware, demonstrating that simple obfuscation is an effective technique to defeat anti-virus scanners. The Internot Security Team [23] examined a similar case and found that simple obfuscation allowed malware to easily bypass many anti-virus scanners.

Other researchers have provided insight into techniques that could help undermine XOR obfuscation. Cannell [5] suggests that patterns inside XORed material can be used to find the XOR key. Malware normalization has also been proposed by Christodorescu, *et al.* [9] as a method for undermining obfuscation techniques by normalizing a file, enabling anti-virus scanners to detect malicious content.

2.2 Manual Analysis Tools

Although numerous digital forensic tools can de-obfuscate a region of bytes with an XOR mask or use XOR pre-processing in malware analysis, only one tool applies XOR as part of automated processing. The other tools that de-obfuscate data require the techniques to be manually invoked by the operator.

Some tools offer methods for de-obfuscating data, but leave the interpretation up to the operator. Hex editors often implement such a feature, enabling the user to view data in hex and offering de-obfuscation functions. The open source Hexplorer [13] has the ability to XOR regions before they are displayed. The `translate.py` script [31] also allows an investigator to XOR a file with a single-byte key. Other tools are specifically tailored to de-obfuscating certain components of a forensic investigation. The MemoryDump plugin [1], for instance, has an option for XORing a memory dump before outputting the data to the user.

Other tools examine data to look for XOR keys, leaving it up to the user to de-obfuscate and analyze the information. XORBruteForcer [15] uses a brute-force algorithm to determine the keys that were most likely to have been used in XORing the data. In contrast, NoMoreXOR [14] attempts to guess an XOR key using frequency analysis.

String searching is also a frequently-used feature in obfuscation analysis tools. XORSearch [30] uses a brute-force approach to search for specific strings that have been obfuscated using XOR, ROL and ROR. Similarly, XORStrings [32], an extension of the XORSearch tool, scans a given file for strings that have been XORed with a specific sequence. The `iheartxor` tool [20] offers a similar function, using brute-force to search for certain obfuscated regular expressions and plain strings.

2.3 Automated Analysis Tools

As mentioned above, DCCI_Carver and DC3_Carver are the only digital forensic tools that provide investigators with an automated approach for uncovering a wide variety of XOR obfuscated files in storage media. In addition to these tools, a literature search turned up tools that provide automated assistance in identifying obfuscated malicious files. Several programs can search for XORed executables embedded in files. Hexacorn's DeXRAY tool [21] acts as a file carver and can identify obfuscated malicious files under a single XOR layer. OfficeMalScanner [3] has a feature to scan for traces of XORed malicious data in Office documents using a brute-force method. Another document scanner tool, Cryptam [26], uses cryptanalysis to identify obfuscated embedded executable files and is effective at detecting XOR obfuscation [27].

3. Materials and Methods

We modified an open source tool to preprocess all the examined data with XOR(255). We then processed data from 2,411 forensic images and found 324,144 XOR artifacts on 698 of the images, with a maximum of 21,031 validated artifacts on a single forensic image.

The goal of the research was to identify the different ways that XOR obfuscation is used in real data in order to determine if the quantity and quality of obfuscation cases are sufficient to suggest implementing XOR de-obfuscation functions as a standard step in automated forensic processing. In particular, we analyzed a corpus of data seeking to determine the extent to which XOR(255) is implemented as an obfuscation technique.

3.1 Real Data Corpus

The Real Data Corpus [17] is a collection of several thousand digitized digital storage devices collected from countries around the world between 1998 and 2013. The corpus contains data from computer hard drives, cell phones, CD-ROMs, DVDs and thumb drives purchased from secondhand computer stores and markets. The corpus thus permits the sampling of XOR usage over a 15-year period in countries around the world, essentially representing a real-world data set.

A limitation of the Real Data Corpus is that it does not include data sets that are known to have been used by malicious entities. Thus, it would be difficult to draw conclusions regarding the importance of extracting and examining XOR obfuscated files in criminal, counterintelligence and counterterrorism investigations without further research. In addition, since the analysis was limited to email addresses, URLs, JPEG photographs, Windows executables, ZIP files and other kinds of easily recognizable data, we did not search for all the types of files that would be considered relevant to these types of investigations.

3.2 bulk_extractor

The `bulk_extractor` forensic tool can scan digital media for email addresses, phone numbers, Internet domain names and other data of interest to investigators. The tool is employed by triage and malware investigators [18]. The tool employs compiled regular expressions and hard-coded finite state machines to extract features from digital media. Features are pieces of key information such as credit cards, emails and URLs that are encountered in forensic investigations. For each feature, `bulk_extractor` determines the feature encoding and records the information in its output. The program labels features found using XOR(255) de-obfuscation with the string "XOR(255)."

The `bulk_extractor` tool tracks the features that result from the application of each scanner; this information is recorded in the program output. By using the tool in this manner, it was possible to search the entire corpus systematically for different uses of XOR obfuscation. For example, we were able to detect and distinguish the case of a JPEG photograph being XORed and then archived in a ZIP file from a series of JPEGs that were archived in a ZIP file and then XORed.

Table 1. *bulk_extractor* features used in the study.

Feature
URLs
WINPE headers
Local file headers of ZIP archives
Exif headers (from JPEGs)

The `identify_filenames.py` tool included in the `bulk_extractor` package was used to associate identified features with the files in which they were located. For each feature, the tool indicated if the feature occurred in an allocated file, a deleted file or disk sectors that could not be associated with any file. This allowed the identification of the files containing XORed information and also helped understand the purpose and extent of the obfuscation.

3.3 Feature Selection

We examined the encodings of domains, URLs, email addresses, ZIP file data structures and Windows Portable Executable (WINPE) headers in order to find examples of XOR(255) usage. The problem with looking for XORed features is to distinguish actual XORed features from "false positives" or matching data that appear by chance, but that are not actual features.

We observed a much greater percentage of false positives (i.e., significantly lower precision) for the features that were found with XOR encodings than is typical for `bulk_extractor` output (which normally has high precision). We assume that the error rate or the number of false positives detected by `bulk_extractor` is constant for random input and a large set of data. When the input contains a substantial signal, such as when running `bulk_extractor` to find plaintext features, the precision is high. However, with an XOR filter, the signal is low because the majority of the features in the data are not XORed. Since the signal is lower, but the error rate remains constant, the precision is lower.

The higher incidence of false positives complicated the analysis. We addressed the error incidence by restricting the analysis to features with significant internal structure that are automatically checked for internal consistency (Table 1). The features are, thus, self-validating; as a result, they are extracted with an extremely low false positive error rate. Self-validating features can also be easily confirmed to be legitimate at a glance, such as opening a JPEG file or clicking a URL to visit a site.

Thus, we could safely assume that most of the identified features were true positives.

Because our goal was to evaluate files containing certain XOR(255)-encoded features, we ignored features that were not being evaluated in order to restrict the data set. In particular, we removed features that were not XORed, features that were not being considered and "false positive" features that we did not recognize as legitimate features.

3.4 Analysis of Identified Files

After identifying the features that had been XOR-encoded, we proceeded to identify the files that contained them. The Sleuth Kit [7] was used to extract the files that `identify_filenames.py` associated with intriguing XORed features. We analyzed the original files and the files after decoding with XOR(255) to determine the purpose for which the XOR obfuscation was used.

3.5 Evaluation of URLs

The vast majority of the XOR-encoded features discovered were URLs. Malware files may contain some mechanisms to further infiltrate a victim, while appearing benign at the same time. For example, malware often contained URLs of malicious sites for downloading additional malicious executables and other files. We examined the XORed URL features outputted by `bulk_extractor` because it has been reported that some malware samples use XOR to obscure embedded URLs. We evaluated the XORed URLs using the Google Safe Browsing API [19] and the McAfee TrustedSource Real-Time Threat Service [28].

4. Results and Discussion

This section presents the results and discusses their implications.

4.1 XOR Obfuscation as a Watermark

We found several cases in which an innocuous URL was XOR(255)-encoded and embedded in a legitimate program. For example, drive image AE10-1023 contained a copy of Nero 7 with the byte sequence `97 8b 8b 8f c5 d0 d0 88 88 88 d1 91 9a 8b 90 d1 9c 90 92 0a` at decimal offset 15,416,790,675. This maps to byte offset 112 of the file `Program Files/Nero/Nero 7/Nero CoverDesigner/def.dat`, a 13,630-byte file with an MD5 value of `8f4b534ed6a82e1885e155541aab0940` that is reported to be part of the Nero Premium distribution [33]. The

Table 2. Cross-tabulation of the classification of the 30,684 URLs.

McAfee Risk	Google OK	Google Malware	Total
High Risk	1,938	143	2,081
Medium Risk	1,301	9	1,310
Minimal Risk	24,090	6	24,096
Unverified	2,216	5	2,221
Total	29,545	163	29,708

string was transformed to `http://www.nero.com` after being processed by XOR(255).

4.2 XORed URL Analysis

A total of 281,712 XORed URLs were found in the drive images, 30,684 of them were distinct. Google's Safe Browsing (GSB) database only checks the domains of sites and is quite conservative. For each XORed URL, GSB reports if the URL's domain is "OK" or "Malware." McAfee's Real-Time Threat Service considers the entire URL and provides better discrimination. For each URL, we recorded if McAfee considered it to be "High Risk," "Medium Risk," "Minimal Risk" or "Unverified."

Table 2 shows how the URLs were evaluated by Google's Safe Browsing API and McAfee's Real-Time Threat Service. Overall, roughly 10% of the distinct URLs found in the XORed data were tied to malware.

4.3 XOR Obfuscation in Anti-Virus Software

Confirming Mueller's observatin [29], we found many XOR-obfuscated URLs in Norton Anti-Virus log files and virus definitions. We also discovered some files in the *Quarantine* directory that had been obfuscated with XOR(255). The drive image il3-0161 contained one such collection of XORed quarantined malware. `bulk_extractor` determined that the file `05FB1F54.exe` contained an XORed WINPE header. The file was also indicated to be located under *Program Files/Norton Antivirus/Quarantine/* with an MD5 value of `a96ae9519ea968ac0089-d6b53cef9b2b`. Examination of the hex dump of the original file revealed that it had no executable code or strings. However, after the file was XORed, the entire file appeared to be malware, containing strings such as "RegDeleteKeyA," "DownloadFile" and "Download.Trojan."

Malware is often distributed via “drive-by downloads” from compromised web servers and, thus, the presence of a malicious URL is a key indicator of malware. Norton and other anti-virus scanners use this approach to identify malware. By obfuscating malware signatures and log files, anti-virus scanners avoid accidentally identifying themselves as malware.

4.4 Obfuscated URLs in Malware

We found a significant number of XORed domains and URLs in programs that were clearly associated with malware. In many cases, the URLs were verified as malicious by Google's Safe Browing API or had names that were clearly malicious. One example is the file `SENDFILE.EXE` in the drive image IN10-0145, which has an MD5 value of `3d419f96-355b93e641ba097c08121937`. The unobfuscated version of the file contained several malicious strings, including URLs linked to malware and instances that modify the registry and processes. For example, the 47-byte sequence `b7 ab ab af c5 d0 d0 ac bc be b1 b1 ba ad d1 a9 d2 a7 d2 ac bc be b1 b1 ba ad d1 bc b0 b2 d0 ac ba ab aa af d0 b9 b6 b3 ba d1 af b7 af` appears at the decimal offset 4426 in `SENDFILE.EXE`. After it was processed by XOR(255), the string became `HTTP://SCANNER.VAV-X-SCANNER.COM/SETUP/FILE.PHP`. Based on this analysis, it is clear that some examples of malware use XOR(255) obfuscation in their normal operations.

The use of XOR(255) by anti-virus scanners and malware is problematic. In our testing, many program samples containing malicious XORed URLs were not identified as malicious by any of the 46 anti-virus engines at `VirusTotal.com`. However, after the files were de-obfuscated, many of the engines were able to identify the malware. Clearly, the engines simply look for the malicious URLs (because applying XOR(255) to the entire executable would also corrupt the Windows PE header and damage the executable code). It is troubling that common anti-virus engines and professional forensic tools do not even offer this simple technique for finding obfuscated malicious data, especially given the widespread use of XOR(255) in malware that we encountered in the Real Data Corpus and in our literature search.

4.5 Anti-Reverse Engineering

In many of the drive images, we observed XOR-obfuscated variations of an email address belonging to Reznik (co-developer of the RealAudio and RealVideo algorithms [10]) in DLLs associated with software from his company. The email address and its variations appeared to have been repeated several times in blocks throughout the Real codecs. One example was found in the drive image AE10-1029 in the file `Program Files/Real/RealPlayer/converter/Codecs/erv4.dll`. The `erv4.dll` file is 483,328 bytes in size and has an MD5 value of `e8b759859b53e19c261162 783dae9869`. The byte sequence `a6 8a 8d 96 86 df ad 9a 85 91 96 94 df c3 86 8d 9a 85 91 96 94 bf 8d 9a 9e 93 d1 9c 90 92 c1 ff`, which appeared at offset 61,120, was de-obfuscated to `Yuriy Reznik <yreznik@real.com>`. We contacted Dr. Reznik and learned that his obfuscated email address was embedded in the binary as a key for decrypting code tables used by the RealVideo 8 codec.

We also found several instances of XOR(255)-encoded JPEG digital photographs that appeared to be from Adobe Flash video games. Drive image IN10-0060 contained the Battle Rush Adobe Flash game menu screen background, drive image SG1-1062 had an XORed menu screen of Gutterball 2 and drive image TH0001-0010 had many Beach Party Craze backgrounds. We believe that the JPEG pictures were encoded as an anti-reverse engineering measure.

4.6 XOR(255)-Encoded ZIP File

On one drive image, we observed remnants of multiple ZIP files in unallocated space that contained confidential information that had been archived and XOR-encoded. The disk was from a computer running Windows 95 that had been purchased on the secondary market and imaged in 2007. The most recent use of the drive was in 2006.

On the drive image, we found a total of 802 documents with timestamps in the ZIP archive ranging from 1991 through 1999, with the majority of the files from 1998 (112 files) and 1999 (623 files). No archived files were found with timestamps after 1999.

After finding these files, we proceeded to modify the ZIP scanner of `bulk_extractor` to carve the remnants into files that could be transferred to another system and analyzed. Upon reviewing the extracted files, we found spreadsheets, personal emails and other sensitive documents containing employee names, home addresses and government identification numbers. We located a batch file, `EMPACA2.BAT`, that zipped the documents to create the archive on the drive image. We were unable to find evidence of the tools that had been used to obfuscated the archive and, thus, could not prove that the obfuscation was the result of an exfiltration attempt.

There are several possible explanations for the obfuscation. The data could have belonged to the individual who was obfuscating the data; this individual could have been developing an application to protect a set of

Table 3. Processing times for `bulk_extractor` with and without the XOR scanner.

Test Image	Size	Without XOR	With XOR	Δ
`nps-2009-domexusers`	40 GB	522 sec	799 sec	+53%
`nps-2011-2tb`	2 TB	34,140 sec	58,147 sec	+70%

data or to protect his/her own data. Alternatively, the obfuscated ZIP file could be the result of software processing the data. On the other hand, the data could have been obfuscated by a rogue employee to hide the data or in an attempt to exfiltrate the data.

4.7 Performance Impact

We saw a significant increase in the processing time after adding the XOR scanner. One standard test drive image required 53% additional processing time, while another required 70% additional time (Table 3). Timing was performed on a 12-core HP Z800 with dual Intel Xenon E5645 CPUs running at 2.4 GHz with 24 GiB of RAM. The operation system was Fedora 19 Linux.

Because `bulk_extractor` recursively processes decompressed data and the XOR scanner attempts to de-obfuscate the data at each step of the pipeline, the actual increase in time is dependent on the data in a disk image. The increase in time is also proportional to the amount of data that is compressed and have to be re-processed, but it is not proportional to the amount of XOR-obfuscated data. For example, the NPS Realistic drive image nps-2011-2tb contains a large number of Adobe PDF files, each of which has multiple `zlib`-compressed regions. Applying XOR deobfuscation to these bytes quadruples the processing time required by other `bulk_extractor` scanners: the bytes were processed with the scanners as they sat in the disk image, after XOR de-obfuscation, after decompression, and after decompression and XOR de-obfuscation.

4.8 Accuracy of bulk_extractor

We were interested in substantiating the accuracy of `bulk_extractor` in detecting features obfuscated with XOR(255). Two standard forensic drive images, nps-2009-domexusers (40 GB) and nps-2011-2tb (2 TB), were used to study the accuracy of `bulk_extractor`'s XOR scanner. We took a random sample of 500 features from each type of feature from each drive image. Each feature was determined to be a true positive or false positive by hand. We found that, for each type of feature, bulk_extractor had almost all the features as true positives or all the features as false positives. There was a higher incidence of features on the 2 TB drive image that were almost completely false positives than on the 40 GB drive image.

4.9 XOR(255)-UNZIP-XOR(255) Property

We were surprised to discover several instances in which the sequence XOR(255)-UNZIP-XOR(255) applied to a fragment of a ZIP-encoded file yielded the same result that would be produced by the straightforward application of UNZIP. We hypothesize that is the result of zlib's use of an adaptive decompression algorithm. We subsequently suppressed XOR(255)-UNZIP-XOR(255) processing with a modification to the bulk_extractor XOR scanner.

5. Conclusions

The analysis of the Real Data Corpus revealed multiple cases of XOR(255) obfuscation. XOR(255) was used to encode URLs in antivirus scanner files and malware in order to avoid detection. XOR(255) obfuscation was also used to hinder the reverse engineering of RealVideo 8 codecs and several Adobe Flash video games. In one case, XOR(255) was likely used in a successful attempt to exfiltrate sensitive data.

The study of XOR obfuscation in the Real Data Corpus revealed several instances where XOR obfuscation is relevant to digital forensic investigations. However, the increase in processing time by bulk_extractor and other forensic tools may be problematic when large data sets have to be examined in a limited amount of time. Therefore, while forensic tools should implement features that would allow investigators to perform simple de-obfuscation, the de-obfuscation should be adjustable by the investigator to run in a manual or automatic mode (similar to the XOR feature in DCCI_Carver and DC3_Carver). Additionally, because the analysis has demonstrated a need for de-obfuscating malware and exfiltrated data, a forensic tool should be operated with the de-obfuscation function enabled in order to capture all the information. Likewise, XOR de-obfuscation should be enabled in child exploitation and objectionable content cases to address the so-called "botnet" or "SODDI" defense [4]. Our XOR scanner is included in the bulk_extractor 1.4 release and is enabled using the flag `-e xor`.

Our future research will focus on enhancing bulk_extractor's XOR scanner to run with little noticeable performance impact. Another line of research is to conduct a thorough survey of the prevalence and purpose of other simple obfuscation and steganography techniques such as rotate and the ROT13 algorithm, as well as, XORing with different keys. Finally, efforts should focus on enhancing anti-virus systems to detect malware that has been obfuscated; the fact that simple XOR obfuscation can successfully evade today's anti-virus systems is troubling indeed.

The opinions and views expressed in this paper are those of the authors and do not reflect the views of the Naval Postgraduate School, the U.S. Military Academy, the Department of the Navy, the Department of the Army or the U.S. Department of Defense.

Acknowledgements

We wish to thank Robert Beverly, Michael Shick and Yuriy Reznik for their help with this research. We also wish to thank Dave Ferguson from DC3, and Colonel Greg Conti, Lieutenant Colonel Matt Burrow and Lieutenant Michael Nowatkowski from the U.S. Military Academy.

References

[1] aeon, MemoryDump (`www.woodmann.com/collaborative/tools/index.php/MemoryDump`), 2009.

[2] A. Apvrille, Symbian malware uses a 91-byte XOR key, Fortinet, Sunnyvale, California (`blog.fortinet.com/symbian-malware-uses-a-91-byte-xor-key`), 2012.

[3] F. Boldewin, Frank Boldewin's `www.reconstructer.org` (`www.reconstructer.org/code.html`), 2009.

[4] S. Brenner, B. Carrier and J. Henninger, The Trojan horse defense in cybercrime cases, *Santa Clara High Technology Law Journal*, vol. 21(1), pp. 1–53, 2004.

[5] J. Cannell, Nowhere to hide: Three methods of XOR obfuscation, Malwarebytes, San Jose, California (`blog.malwarebytes.org/intelligence/2013/05/nowhere-to-hide-three-methods-of-xor-obfuscation`), 2013.

[6] J. Cannell, Obfuscation: Malware's best friend, Malwarebytes, San Jose, California (`blog.malwarebytes.org/intelligence/2013/03/obfuscation-malwares-best-friend`), 2013.

[7] B. Carrier, The Sleuth Kit (`www.sleuthkit.org/sleuthkit`), 2013.

[8] M. Christodorescu and S. Jha, Testing malware detectors, *ACM SIGSOFT Software Engineering Notes*, vol. 29(4), pp. 34–44, 2004.

[9] M. Christodorescu, J. Kinder, S. Jha, S. Katzenbeisser and H. Veith, Malware Normalization, Technical Report #1539, Deparment of Computer Sciences, University of Wisconsin, Madison, Wisconsin (`ftp.cs.wisc.edu/pub/techreports/2005/TR1539.pdf`), 2005.

[10] G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman and Y. Reznik, Video coding for streaming media delivery on the Internet, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11(3), pp. 269–281, 2001.

[11] Cyber Engineering Services, Malware obfuscated within PNG files, Columbia, Maryland (`www.cyberengineeringservices.com/malware-obfuscated-within-png-files`), 2011.

[12] Cyber Engineering Services, Malware obfuscated within PNG files > Sample 2, Columbia, Maryland (`www.cyberengineeringservices.com/malware-obfuscated-within-png-files-sample-2-2`), 2011.

[13] M. Dudek, Hexplorer (`sourceforge.net/projects/hexplorer`), 2013.

[14] G. Edwards, NoMoreXOR (`github.com/hiddenillusion/NoMoreXOR`), 2013.

[15] J. Esparza, XORBruteForcer (`eternal-todo.com/var/scripts/xorbruteforcer`), 2008.

[16] N. Falliere, L. O'Murchu and E. Chien, W32.Stuxnet Dossier, Symantec, Mountain View, California, 2011.

[17] S. Garfinkel, P. Farrell, V. Roussev and G. Dinolt, Bringing science to digital forensics with standardized forensic corpora, *Digital Investigation*, vol. 6(S), pp. S2–S11, 2009.

[18] S. Garfinkel, Digital media triage with bulk data analysis and `bulk_extractor`, *Computers and Security*, vol. 32, pp. 56–72, 2013.

[19] Google, Safe Browsing API, Mountain View, California (`developers.google.com/safe-browsing`), 2013.

[20] A. Hanel, `iheartxor` (`hooked-on-mnemonics.blogspot.com/p/iheartxor.html`), 2012.

[21] Hexacorn, DeXRAY, Hong Kong, China (`www.hexacorn.com/blog/category/software-releases/dexray`), 2012.

[22] B. Hussey, Decoding data exfiltration – Reversing XOR encryption, Crucial Security Forensics Blog (`crucialsecurityblog.harris.com/2011/07/06/decoding-data-exfiltration-%E2%80%93-reversing-xor-encryption`), 2011.

[23] Internot Security Team, Bypassing anti-virus scanners (`dl.packetstormsecurity.net/papers/bypass/bypassing-av.pdf`), 2011.

[24] B. Kang, E. Chan-Tin, C. Lee, J. Tyra, H. Kang, C. Nunnery, Z. Wadler, G. Sinclair, N. Hopper, D. Dagon and Y. Kim, Towards complete node enumeration in a peer-to-peer botnet, *Proceedings of the Fourth International Symposium on Information, Computer and Communications Security*, pp. 23–34, 2009.

[25] Kimberly, Analysis of `imm32.dll – Trojan.Win32.Patched.mc`, StopMalvertising (`stopmalvertising.com/malware-reports/analysis-of-imm32.dll-trojan.win32.patched.mc.html`), 2011.

[26] Malware Tracker, Cryptam document scanner, North Grenville, Canada (`malwaretracker.com/doc.php`), 2012.

[27] Malware Tracker, New malware document scanner tool released, North Grenville, Canada (`blog.malwaretracker.com/2012/02/new-malware-document-scanner-tool.html`), 2012.

[28] McAfee, TrustedSource – Check Single URL, Santa Clara, California (`www.trustedsource.org/en/feedback/url?action=checksingle`), 2011.

[29] L. Mueller, XOR entire file or selected text, ForensicKB (`www.forensickb.com/2008/03/xor-entire-file-or-selected-text.html`), 2008.

[30] D. Stevens, XORSearch and XORStrings (`blog.didierstevens.com/programs/xorsearch`), 2007.

[31] D. Stevens, Translate (`blog.didierstevens.com/programs/translate`), 2008.

[32] D. Stevens, New tool: XORStrings (`blog.didierstevens.com/?s=xorstrings`), 2013.

[33] Systweak CheckFileName, View Nero Premium Details (`www.checkfilename.com/view-details/Nero-Premium`), 2013.

[34] I. Venkata Sai Manoj, Cryptography and steganography, *Internal Journal of Computer Applications*, vol. 1(12), pp. 61–65, 2010.

[35] N. Virvilis and D. Gritzalis, The big four – What we did wrong in advanced persistent threat detection? *Proceedings of the Eighth International Conference on Availability, Reliability and Security*, pp. 248–254, 2013.