Royal HOLLOWAY SERIES 2012

Malware Armouring: The Case Against Incident-related Binary Analysis

A traditional incident response process based on binary analysis may be futile when dealing with today's armoured malware. Learn how malware is shielded from analysis and how security pros can redirect their antimalware efforts.

BY STEVE HENDRIKSE AND JOHN AUSTEN

TechTarget

Royal Holloway University of London

Malware Armouring: The Case Against Incident-related Binary Analysis

WHILE AN IMPORTANT step in the incident response process is to understand the breadth of an intrusion and its impact to the organisation, advances in malware armouring have made it virtually impossible to decipher the true nature and intent of the malware. So effective incident response should focus primarily on early detection and identification of threats, and timely and effective response using some automated eradication method.

INCIDENT RESPONSE

In a commercial enterprise the most important reason for having an active incident response process for malware infestations is to understand the impact of the infestation. At best, a complete listing of all the systems and data that the malware was in contact with will allow identification of leaked or compromised intellectual property.

The incident response process is responsible for accurately identifying the threat and its propagation methods, thus allowing for quick and effective cleaning and recovery of the corporate network. If the threat has not been accurately identified, or if the threat has taken an extended amount of time to characterise, unnecessary damage may be realised in the form of additional data loss or cost of remediation effort.

Computer Security Incident Response is made up of several interconnected phases. They form a circle where the output of each phase is fed into the next, and that from the last phase is fed back into the first:

- **Identification**
- **Containment**
- **Eradication**
- **Recovery**
- **Lessons Learned**

The organisation should be able to understand how the threat was able to bypass existing controls to gain a foothold in the corporate environment. The bypass may have resulted from weak technical controls, weak policy, or weak user awareness and training. Once these areas of weakness are identified, they can be fed back into the incident response process as opportunities for improvement.

Preparation. Depending on the organisation's response requirements, a response team will be formed of staff from departments like public relations, product management, human resources, legal and the various IT functions.

Clearly, the processes and procedures required for analysis of malware executables and an effective response should be developed and tested prior to an actual incident. In this way, the analysts in the team will have experience using the tools and can recognise valid output from the tools.

Tools for malware analysis can be classified into four families:

- **File identification and manipulation tools**
- **System delta tools**
- **System activity monitoring tools**
- **File deconstruction tools**

Identification. Once a suspicious host has been identified, the search for the malicious executable begins. In the family of system activity monitoring tools, the tools most useful are those that can identify:

1. **inconsistencies** between high level (API) calls and low level (direct) calls
2. **processes** in "run"-able state that aren't shown in the task list
3. **registry** anomalies or inconsistencies
4. **hidden** sections of disk (both allocated and not)
5. **non-standard** network port use or activity

Containment. With the confirmed identification of an executable file spawning a malicious process on the computer, the incident response process moves into the *containment phase*. Its primary goal is to keep the problem from getting worse. In the commercial enterprise, this goal is likely to include the collection and delivery of a specimen copy of the malicious executable file to the business partner responsible for creating a "detect and clean" A/V signature.

Once the specimen has been sent to the A/V partner, short-term con-

INCIDENT RESPONSE
ARMOURING TECHNIQUES
ENCODING
CONCLUSIONS

tainment efforts will be undertaken, which may include installing a network access control list (ACL) to disallow any in- or out- bound network traffic from the host, the blocking of certain executables, or disconnection of infected machines from the corporate network.

Eradication. The goal of the *eradication phase* is to get rid of the attacker's artifacts on the machine, including accounts, malicious code, pirated software, pornographic material or anything else the malicious actor has left on the machine. Once the malware has been identified and analysed, compensating controls can be applied to remove the threat of re-infection and/or spread of the infection.

Recovery. The goal of the *recovery phase* is to put the impacted systems back into production in a safe manner. Once the threat has been removed from the environment, the task of recovering work product interrupted by the infection can commence. Data can be recovered from backups and put back into production once the data has been scrubbed to ensure it is free from malicious additions or modifications.

Lessons learned. The goal of the *lessons learned phase* is to document what happened and improve the organisations' capabilities. By assessing the incident, from infection to identified impact, an organisation can evaluate the controls for their effectiveness in stopping identified threats. This evaluation should provide a list of areas that can be given priority from the perspective of compensating control implementation.

ARMOURING TECHNIQUES

The goal of all armouring techniques in malware is to delay or stop the analysis of an executable by behavioural observation and/or reverse engineering. This maximises its effective lifespan in the network—intranet or Internet—and the anonymity of both the attackers and their controlling servers.

Effective lifespan is the amount of time that the malware can exist within a system or network without being detected by mainstream detection mechanisms like those listed in above under "identification".

Organisations will have limited success in cleaning a system if they are not able to identify all of the locations in which the malware has embedded itself. Once identified, a malware specimen will be analysed for the following purposes:

1. **Development** of a compensating control against the malware threat—for detection and cleaning

2. **Quantification** of incident impact

3. **Technical** curiosity or research, which includes "borrowing" another malware writer's techniques

ENCODING

There is a family of armouring techniques that all revolve around obfuscating the human readable content of the malware from the analyst. These encoding techniques do not alter the malware's execution flow; rather they simply mask the user input/output routines under a veil of obfuscation.

String Encoding. During development any character-based English readable strings are replaced in the malware by an encoded equivalent. Without this, the strings stored within the executable bring a wealth of insight about its workings to the analyst. However, if dumping of strings yields no actionable intelligence, the malware could be opened in a debugger or disassembler and analysed to identify the string encoding and decoding functions. Once these functions are identified, each of the encoded strings, also identified in the disassembly process, can be manually decoded to identify its actual message.

Payload Encoding (Encryption). Payload encoding works much like string encoding, but is applied to the communication between victim and controller over the network. Prior to sending any communication over the network, the data is put through an encoding or encryption algorithm.

This encoding can also be applied to payloads stored on the local disk of the infected computer—like keylogger data files. In the process of media and/or network capture analysis, these encrypted/encoded payloads may be found, but no intelligence will be gleaned from them due to the encoding. As before, an analyst may be able to open the malware in a debugger or disassembler and identify the routine responsible for the encoding of the data,

Effective lifespan of armoured malware is maximised by the following capabilities:

- **Hidden methods of spread and infection**
- **Unknown capabilities**
- **Resistance to identification and classification**

but the task will be quite laborious and may even lead to the abandonment of the analysis of the malware.

Executable Packing. Malware authors use their own packers to compress an executable or DLL. Every standard Windows file based on the PE (Portable Executable) format has a loader section containing the instructions required for decompressing each section of the stored file prior to copying it to memory. Since the loader is responsible for protecting the rest of the executable, the packer implements protection against analysis, such as encryption or debugger detection.

Due to section relocations and the removal of run-time library dependency references, not all malware can be successfully unpacked into valid executable format. This often forces the analyst into manual analysis of the executable which is tedious, error prone and may lead to abandonment of the analysis.

Malware is analysed for the purpose of:

1. Developing compensating controls
2. Quantification of incident impact
3. Research into malware techniques

Virtual Machine Environment (VME) Detection. Security researchers have observed that there are multiple kernel data structures within a running system that give clues to whether the running system is within a virtual machine environment. Malware is often able to query these tables to determine if it is being run inside a virtualised machine.

In addition to kernel structure differences, most VMEs have device drivers that optimise the use of shared and/or virtualised hardware. Similarly, there are typically communication channels between the host and guest operating systems. Often these device drivers or communications channels can be discovered, which identifies the environment as being virtual.

If a piece of malware is able to detect whether the environment is virtual, it can execute differently from when the malware is executing on "bare metal" hardware. Thus hackers can use VME detection to hinder analysis by returning fake or manufactured results in an effort to divert attention away from the actual goals of the malware.

Run-time Decryption. Run-time decryption is similar in most ways to a packer, with the exception that instead of only using a compression algorithm to reduce the size of an executable, the executable is compressed,

encrypted, and then written to disk. Like a packer, for which the unpacking routine is distributed with the executable, the decryption routine for an encrypted executable is stored inside the executable and is distributed with each copy.

In much the same way as a packer can be defeated by manually unpacking the executable, an encrypted executable can be manually decrypted and returned to its native machine ready state. Since the decryption routine is attached to the executable, the analyst has all the required data needed to defeat the armouring technique.

Polymorphism and Metamorphism. One of the shortcomings of runtime-encrypted malware is that while the virus body looks different from generation to generation, the decrypter that is embedded in the virus remains constant for all generations. As a result, it is possible to detect the virus indirectly by recognising the code pattern of the decrypter. Polymorphism is a particularly robust form of runtime encryption. A polymorphic virus generally makes several changes to the default encryption settings, as well as altering the decryption code.

In contrast to polymorphism, the idea behind metamorphism is to alter the content of the virus itself, rather than hiding the content with encryption. The typical execution phases of metamorphic malware start with the capability to locate its own code within the host program, then decode, analyse, transform itself to a new entity and finally reattach to a new host.

While the self-transforming techniques for creating metamorphic malware provide great benefit to the malware author, metamorphism adds complexity and as much as 80% more code to the malware.

A strategy recently devised to offer the benefits of polymorphism and metamorphism while not adding considerable complexity to the malware itself is known as "server-side polymorphism". Instead of having the malware carry the transformation routines within it; the malware is either transformed upon distribution from the server, or the malware is updated to a transformed version of itself after initial infection.

Packing an executable offers many advantages to the malware:

- **It is protected from reverse engineering or analysis**
- **The unpacked image is never written to disk, bypassing many host based protections**
- **It is an IDS (intrusion detection system) evasion tactic, as string markers within the executable are obfuscated**
- **Reduced size is useful for covert transmission of payloads.**

The primary use for polymorphism and metamorphism is to evade detection by signature-based and heuristic-based tools. Unlike both packed and run-time encrypted malware, future generations of poly- and metamorphic malware will escape detection by signature based monitoring solutions.

Anti-Debugging. There are functions available in the Windows API that return TRUE if the current program is running from within a debugger. Unfortunately, the Windows API is quite high-level and easy to spot (and bypass) from within the debugger. In response, developers of malware move lower in the application stack and bypass the API call in order to directly query the process or thread structures and even CPU registers themselves—which is more difficult to detect.

INCIDENT RESPONSE
ARMOURING TECHNIQUES
ENCODING
CONCLUSIONS

In a polymorphic virus, the content of the underlying virus code body does not change; encryption alters its appearance only.

An *exception* is an unintended or unexpected event that occurs during the execution of a program, and requires the execution of code outside the normal flow of control. Software exceptions are initiated explicitly by applications or the operating system. A side effect of the *UnhandledExceptionFilter* function for handling exceptions can be used to signal to the malware that it is being executed from within a debugger.

Timing based detection asserts that the time difference between instructions is longer when an executable is being debugged than if the CPU is freely running the executable. If a sequence of instructions takes longer than expected, the malware can conclude it is running within a debugger and respond accordingly.

Modified code based detection is based on the developer having stored a fingerprint (or hash) of the malware itself within the malware. Then, upon each execution of the binary, the in-memory process fingerprint is compared to the fingerprint stored within the executable. If the fingerprints differ, the malware can trigger a different payload.

All of the forms of debugger detection are intended for use in hiding the true nature of machine level instructions from the analyst. If the analyst can unravel this true nature, he will be able to bypass the various types of protection intended by these instructions.

Multi-partite. Multi-partite traditionally refers to a malware threat that has components of both a boot sector virus and a rootkit. This traditional view has been updated to include the *Downloader/Dropper* family of threats. These usually save a range of files to the victim's drive, and launch them without any notification or with fake notification of an archive error, an outdated operating system version, or etc.

The goal of the multi-partite armouring technique is to minimise the footprint and separate each feature of the malware such that it is harder to detect and remove, while allowing for easier upgrading and expansion to new threat capabilities.

CONCLUSIONS

The primary reason an organisation will undertake a binary analysis of malware is to understand the breadth and impact of the attack. In the case of a malware infection, the breadth may simply be number of machines infected. The impact of the attack will be decided based on the amount of damage done by the infected machine under the control of the attacker or by the amount and type of data corrupted, deleted or stolen through the infection. Traditionally, in order to understand the damage potential of either of these, analysis of the malware was required.

Malware authors recognise that a key defence against being discovered is the ability to remain hidden from sight of traditional security controls. In this way, one of the goals of binary armouring is to ensure that the defenders of a network are unable to differentiate between normal traffic and traffic that exists as a result of an ongoing infection by malware. Additionally, the other primary goal of these armouring techniques is to block the understanding of the mechanics and capabilities of the armoured executable.

As organisations continue to rely more and more on the computing infrastructure and the interconnectedness of their business, and while the

Techniques for malware self-transformation include:

1. **Instruction substitution, which entails the replacement of an instruction or a group of instructions with an equivalent instruction or group.**
2. **Register swapping whereby the malware mutates itself by moving operands to different registers.**
3. **Adding instructions that have no effect on program outcomes in order to change the appearance of the executable.**
4. **Permuting subroutines to change the appearance of the malware.**
5. **Transposing instructions that have no dependency between them.**

proceeds of a successful intrusion continue to soar, malware will continue to flourish. This malware is created at an alarming rate, it is armoured using sophisticated techniques, it attacks organisations from every conceivable vector, and it is difficult to analyse properly. Until such time as a technology is created that will accurately identify and block malware, commercial organisations should allocate their resources to classifying and controlling access to their sensitive data, rather than trying to decipher the intent of malware.

Once your AV vendor has created a signature for the malware, your best course of action is to let the AV solution clean up and disinfect the system, rather than spending endless time trying to get to the bottom of how the malware managed to infect in the first place. ■

INCIDENT RESPONSE

ARMOURING TECHNIQUES

ENCODING

CONCLUSIONS

ABOUT THE AUTHORS:

Steve Hendrikse has a background in system administration and development and has worked in most disciplines of information security, including penetration testing, technical risk assessment, incident response and investigation, and operational security monitoring. Currently, Hendrikse works for a multinational telecommunications firm doing application vulnerability assessments and security code reviews.

John Austen is course director for the Royal Holloway Diploma in Information Security. He was the head of the Computer Crime Unit, New Scotland Yard, until September 1996. Austen was the first chairman of the Interpol Computer Crime Committee, serving from 1991 to 1996 and was responsible for the worldwide standardisation of police procedure.