# Accepted Manuscript


Title: A survey of similarities in banking malware behaviours

Author: Paul Black, Iqbal Gondal, Robert Layton

PII: S0167-4048(17)30202-X
DOI: https://doi.org/doi:10.1016/j.cose.2017.09.013
Reference: COSE 1209

To appear in: *Computers & Security*

Received date: 30-8-2016
Revised date: 25-8-2017
Accepted date: 21-9-2017

Please cite this article as: Paul Black, Iqbal Gondal, Robert Layton, A survey of similarities in banking malware behaviours, *Computers & Security* (2017), https://doi.org/doi:10.1016/j.cose.2017.09.013.

# A Survey of Similarities in Banking Malware Behaviours

Paul Black*, Iqbal Gondal†, Robert Layton‡
Internet Commerce Security Lab, Federation University
Email: *paulblack@students.federation.edu.au, †iqbal.gondal@federation.edu.au, ‡robertlayton@gmail.com

Paul Black is studying a PhD in Information Security at the Internet Commerce Security Lab (ICSL) at Federation University. His PhD topic is Techniques for the Reverse Engineering of Banking Malware. Paul has a Masters of Computing, his research topic was the reversing of Zeus malware. Paul started his career as a programmer in 1981 and has worked in banking, defence, law enforcement and malware analysis.

Dr. Iqbal Gondal is a leading researcher in the area of condition monitoring, sensor information processing, wireless communication and cyber security. Currently he is Director of Internet Commerce Security Lab (ICSL), Federation University Australia. ICSL conducts research in the application of advance analytics techniques for cybersecurity and condition monitoring and provides innovative Cybersecurity solutions to the industry. In the past, he was director of ICT strategy for the faculty of IT in Monash. He has served in the capacity of Director of Postgraduate studies for six years, member faculty board and member of Monash academic board. He is Fellow of Engineers Australia.

Dr. Robert Layton is a Data Scientist working with text problems in a number of domains. His research focuses on the methods used to build cybercrime attacks and the analysis of the outcomes. He is an Honorary Research Fellow at Federation University Australia and the inaugural Federation University Young Alumni of the Year in 2014.

***Abstract*—Banking malware are a class of information stealing malicious software that target the financial industry. Banking malware families have become persistent with new versions being released by the original authors or by others using leaked source code. This paper draws together a fragmented and industry based literature to provide a coherent description of major banking malware families, their variants, relationships and source code leakages. The concept of malware behaviour is well established in the research**

**literature. However, the literature has not settled on an identification of key malware behaviours. Malware behaviours are defined by existing standards, but they are broad in scope and some individual behaviours are not well defined. This paper identifies a set of malware behaviours that are present in the selected banking malware families. The conceptual distance between the low level detail of Application Programming Interface (API) calls and a high level understanding of malware behaviour is known as the semantic gap. This paper assembles a dataset of malware behaviours and then shows experimental use of the Pharos Framework to bridge this semantic gap by providing automatic identification of malware behaviour using static methods.**

***Index Terms*—malware analysis, malware components, banking malware, malware similarity, malware capabilities, malware behaviours, zeus, citadel, dridex, dyre, carberp, rovnix, vawtrak**

**Keywords:** malware analysis, malware components, banking malware, malware similarity, malware capabilities

## I. Introduction

This paper uses the term "banking malware" to refer to information stealing malware developed by criminal groups, which are used to attack the financial industry. Banking malware are known to attack Microsoft Windows and other operating systems. An industry report states that in 2016 approximately 70 percent of malware samples targeted the Windows Operating System (OS) [1]. A significant body of research, analysis, and reporting exist for malware attacking the Windows OS. The malware considered within this paper is specific to the Windows OS, however the method described in this paper is applicable to malware targeting any OS.

Current literature describing malware families is largely produced by commercial organizations. Problems which arise from reliance on a commercial literature is that data may be lost due to the abandoning of aging webpages, quality is variable and reports may not be complete as they are driven by commercial interests. Malware research has focused on techniques for malware detection, classification, clustering and malware similarity [2]. Prior research has not focused on specific malware families and the evolution of these malware families. This may be due to a view that malware instances are ephemeral and are soon replaced by new and different malware instances. This paper shows that these banking malware families have become persistent, are evolving and that knowledge of these malware families is relevant to

academic and industry based malware researchers.

Dynamic analysis has been widely used to obtain an instruction trace or a trace of API calls. A limitation of dynamic analysis is that it is rarely possible to obtain full execution coverage of a malware sample. Static analysis can be used to extract a graph of API calls from unpacked malware samples. Static analysis has the advantage of providing full coverage of the malware sample but may have difficulties determining API calls made using obfuscation techniques. The API calls details obtained using either dynamic or static analysis methods have previously been used for malware detection and malware classification.

Although many different banking malware families exist, they share the same high-level goals, that is to enable information stealing and financial crime. The aim of this paper is to examine similarity in the behaviours of this selected set of malware which serve similar purposes but have different and somewhat unknown origins. This paper provides an example of the use of the Pharos Framework to identify behaviours which are present in a set of banking malware samples. This automatic classification of the behaviours present in a malware sample presents a representation of malware capability at a higher level of abstraction than has previously been available.

The requirement for the identification and comparison of malware behaviours is different from the comparison of similarity between malware functions. To establish similarity between functions it is necessary to show that the actions performed by two functions are equivalent. To compare the behaviours of two malware samples, it is necessary to show that the aggregate behaviour of a group of functions in each sample is equivalent.

A malware family comprises all the variants of a malware containing common malicious behaviours [3]. To understand the relationship between banking malware and other types of malware, attempts have been made to build a malware taxonomy [4]. However these categorisation schemes have been outgrown by the rapid proliferation of new types of malware.

The structure of this paper is as follows, the literature review examines the concept of malware behaviour, then a review of the selected malware families is given. This is followed by an overview of the different ways that the selected malware behaviours may be implemented. The implementation details of the behaviours of each malware family is given. Finally, an example of the static identification of malware behaviours is provided by the use of the Pharos Framework.

This paper provides a survey of the behaviours of the following selection of recent Microsoft Windows banking malware families: Zeus V2, Citadel, Carberp, Vawtrak, Dridex, Dyre and Rovnix malware. These malware families were selected based on prevalence. Table I shows a timeline of banking malware families, the inheritance relationships between variants and the dates of source code leaks.

### *A. Evolution and Persistence*

Once created, banking malware families have become persistent and have continued to evolve, despite opposition from law enforcement. This persistence is due to the malware author's reuse of source code, the incorporation of leaked malware source code and a desire to build on existing malware products rather than re-inventing the wheel. An example of the persistence of banking malware families is provided by the Zeus malware. Zeus version 1 was detected in 2006. Zeus version 2 was detected in 2010. Gameover Zeus was detected in 2011. The Zeus V2 source code was leaked in April 2011 [5] [6]. Following the Zeus source code leak Citadel, ICE-IX, Kins, Panda and Chthonic Zeus variants were developed in a period spanning from 2011 untill 2016. Leaked source code provides an opportunity for a malware analyst to increase the insight of their analysis, however leaked source code also provides an opportunity for malware authors to improve their products.

### *B. Contribution*

The contributions of this paper are as follows:

- Assembling a coherent description of major banking malware families, their variants, relationships and source code leakages.
- Identification that when created banking malware families are persistent with new versions being released by the original authors, or by others using leaked malware source code.
- Identification of a set of key behaviours that allow a high-level comparison of the techniques used to implement banking malware families.
- Identification of the techniques that are used to implement the identified behaviours.
- Demonstration of the Pharos Framework which can provide behaviour identification by static analysis.

## II. Related Work

An OS provides a finite number of API calls to allow application programs to access OS

services. Consequently, the set of all possible operations that can be performed by a finite program using OS services is finite. Malware operates by manipulating OS behaviour; therefore, malware operations can be represented as a finite set of API calls [7]. The monitoring of malware API calls using dynamic analysis results in a detailed log. This log of malware API calls needs to be converted into a higher level representation that would be more easily understood by an analyst. The conceptual distance from a low level log of API calls to a higher level representation is known as a semantic gap. Martignoni [7] bridges this semantic gap by mapping the high-level goals of the malware into finite sets of API calls. The set of high-level goals is referred to as the malware's behaviour [7].

Dynamic analysis of malware behaviour has been used to extract specifications of malware behaviour [8]. The specifications are derived from API call profiles obtained from dynamic analysis. These malware specifications are used for the detection of malware. Malware capabilities labelling, using Linear Temporal Predicate Logic (LTPL) operating on a dynamic trace of file system access, has been used to identify malware capabilities as part of a malware detection system [9].

Singh (2002) [10] proposes the modelling of the malicious behaviours of malware by using tuples of Subject, Object, Action, and Function, where subject represents active system entities, object is a system resource which is used to store information, action is the malicious behaviour which is initiated by a trigger, and function is the outcome of the action performed by a subject on an object.

Grégio proposes a set of malware behaviours in order to build an extensible malware taxonomy [4]. These behaviours are grouped into classes. The classes are, Evasion, Disruption, Modification, and Stealing. The Evasion class contains behaviours pertaining to the removal of evidence, removal of registries, anti-verus engine termination, and firewall termination. The Disruption class contains behaviours for the scanning of known vulnerable services, email sending, IRC/IM known port connection, and IRC/IM unencrypted commands. The Modification class contains behaviours pertaining the creation of a new binary, creation of unusual mutex, modification of name resolution file, modification of browser proxy settings, modification of browser behaviour, persistence, download of known malware, download of unknown files, and driver loading. The Stealing class contains behaviours pertaining to the theft of system/user data, theft of credentials or financial data, and process hijacking [4]. It is noted that the behaviours

identified by Grégio are not complete, for example, gaining control of the command line or desktop (Backconnect) could be considered to be a member of the Modification class.

## III. Malware Behaviours

The MAEC language is provided to permit the sharing of structured information about malware [11]. The MAEC language defines low level actions, mid-level behaviours, and high-level capabilities in relation to malware. Low level actions are those actions that can be performed by malware through OS API calls. Mid-level behaviours aim to organise and describe the intention behind low level actions. Behaviours describe the operation of a malware instance at an abstract level and consist of groups of low level actions [12]. Capabilities provide a description of the full range of behaviours of malware. MAEC provides a non-exhaustive list of malware capabilities which are shown in Table II [13]. The MAEC language not only provides a language for exchanging information about malware, it formalises the concepts of malware low-level actions, behaviours, and capabilities. Some of the default MAEC behaviours are broad in scope and may be difficult to map into low level actions, e.g the Fraud behaviour.

Banking malware employs the following key capabilities: command and control, data theft, spying, integrity violation, data exfiltration, fraud, and persistence. This paper uses the following non-exhaustive set of behaviours: Persistence, Configuration, Process Injection, Information Stealing and Injection, Network Communications, Backconnect, Screenshot and Video Capture, and Anti-Analysis. These behaviours were identified as they represent the core behaviours of the selected banking malware families. A comparison of the implementation of these behaviours is used to demonstrate the degree of similarity between banking malware families. A comparison of MAEC capabilities to the behaviours used in this paper is shown in Table II. As the MAEC capabilities represent all malware families, they are a superset of banking malware behaviours.

The Application Program Interface (API) calls referred to in this paper are assumed to be Windows OS API calls, the details of which are provided in Microsoft developer documentation [14].

The remainder of this section discusses how each of the key malware behaviours might be implemented in the case of the Windows OS. The following section will then review the implementation used by each malware in our selection.

### *A. Persistence*

Modern OS's have a function to automatically start programs, including malware. In the Windows OS, these are known as Auto-Start Extensibility Points (ASEP) [15] [16]. When malware is installed, it uses the OS ASEP capability to ensure that it is started and to ensure its persistence [9]. The `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` registry entry is often used for malware start-up. The registry entry is commonly referred to as the registry run key or the run key [17] [18] [16].

The `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs` registry entry contains a list of Dynamic Link Libraries (DLLs) loaded into every GUI process when it is started [15] [19] [20] [16]. The winlogon program creates events for actions such as logon, logoff, start-up, shutdown, and lock screen [15]. The `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon` registry entry contains the name of the DLL that will be called to handle a specific Windows event. This registry entry may be used for malware purposes.

The Windows OS runs system tasks as services, which are DLL files that are run by the svchost program. It is common to see several svchost instances running on a Windows OS. The regsvr32 program is used to register DLL files as Windows services. Malware DLLs are commonly run as a service using the regsvr32 command or by adding an entry to `HKLM\SYSTEM\CurrentControlSet\Services` [17] [18] [16]. A malware installer may patch (trojanize) an OS program. The patch inserts malware code and then returns control to the OS program [17]. There are many other registry locations that are used for malware start-up. The Autoruns program displays the programs that automatically run when the Windows OS is started [17]. A malware program can protect its persistence with the use of a thread or a process that is used to recreate the persistence mechanism when it is removed from the OS start-up locations [21].

The Carberp and Rovnix malware families make use of a bootkit (contraction of "boot loader" and "rootkit"). The bootkit allows malware code to load an unsigned driver and maintain control through the boot process. Bootkit techniques allow the malware developer to circumvent the Windows signed driver policy and to maintain persistence when the OS is re-installed [22]. The malware driver may be used to create a hidden, encrypted Virtual Filesystem (VFS). I/O Request Packet (IRP) filtering is used by the malware driver to prevent the encrypted filesystem

from being read or overwritten. The malware driver is used to inject malware code from the hidden filesystem into a target process [22] [23].

*B. Configuration*

Banking malware uses a configuration [24] that provides details of the financial institutions that are being targeted, data to be displayed to the victims, and it may contain scripts to automatically transfer funds from the victim account. The configuration may contain webinjects data or include URL links to webinjects scripts. A webinject consists of HTML/Javascript which is injected into a webpage. Webinjects may be present in the configuration or the configuration may link URLs where the webinjects can be downloaded. Webinjects are used to defeat the security measures of a specific bank. Webinjects may be used for a variety of purposes including automatic funds transfer [25] [26] [27] [28]. The configuration stored on the victim's computer is encrypted to avoid detection. The encryption used by the banking malware includes AES, TwoFish, RC2, RC4, RC6 and custom encryption. Custom encryption functions are developed by malware authors using bit and arithmetic operators. Custom encryption systems do not provide strong encryption, however they are novel and may cause existing analysis tools to fail [29] [30] [31] [32] [33] [23] [34].

Banking malware configuration is generally implemented as a static configuration within the malware sample, while a dynamic configuration is downloaded from the attacker's Command and Control (C2) server. The static configuration contains basic details such as the IP addresses or hostnames of the C2 server, a customer identifier and possibly a cipher or encryption key [26] [35]. The dynamic configuration may contain details of websites to ignore, websites to target and the actions to be performed when a specified website is visited. The use of a dynamic configuration gives the attacker flexibility in controlling the attack, thereby allowing configuration changes in response to bank countermeasures and changing targeting requirements [26] [36].

A criminal group running a large network of compromised computers (a botnet) may partition the botnet so that their criminal "customers" have access to data stolen from a specified section of the botnet. This is performed by embedding a customer ID into the configuration of the banking malware.

*C. Process Injection*

Process injection is a technique commonly used by malware in order to hide the malware

code by running it within a benign system process. Due to the complexity of process injection, the method used by a malware family may be considered to be invariant [37]. Process injection allows the injected malware to utilise the access level of the target program. Malware code can be injected into a web browser process in order to obtain internet access [17]. Malware process injection can be performed using a number of techniques including DLL injection, direct injection, process hollowing, and Asynchronous Procedure Call (APC) injection.

DLL injection is a common method to cause a process to execute malicious code. DLL injection operates by injecting code into a remote process. When this code is executed, it calls the `LoadLibrary` API with the name of the malicious DLL. The OS then loads the malicious DLL and begins execution of the `dllmain` function. The Windows APIs most often used in DLL injection are: `VirtualAllocEx`, `WriteProcessMemory`, `GetModuleHandle`, `GetProcAddress`, and `CreateRemoteThread` [17] [38] [39] [40].

Direct injection uses APIs similar to those in DLL injection, except it is not necessary to force the target process to load a malicious DLL. The Windows APIs commonly used in direct injection are: `OpenProcess`, `VirtualAlloc`, `OpenProcess`, `WriteProcessMemory` and `CreateRemoteThread`. Calls are made to `VirtualAllocEx` and `WriteProcessMemory` in order to copy the data and code of the malicious program into the target process. When the injected code begins execution, it obtains the addresses of the required APIs using `LoadLibrary` and `GetProcAddress` calls or by using custom functions. Direct Injection requires a higher skill level on the part of the malware author [17] [38].

Process hollowing starts by creating a benign process in a suspended state. `GetThreadContext` is used to obtain a thread in the process and `ReadProcessMemory` is used to read the memory of the process. `ZwUnmapViewOfSection` is used to deallocate the memory associated with each of the program sections. `VirtualAllocEx` is used to allocate memory and malware code is copied into the newly allocated memory using the `WriteProcessMemory`. `SetThreadContext` and `ResumeThread` are used to start process execution. An anti-virus program running on the victim's machine will see the name and PE header of the benign process and may not detect the executing malicious code [17].

The APC facility, provided by the Windows OS allows a function to be called in response to an asynchronous event. Normal execution of the program is interrupted and the called function executes as a thread in its own context. APC's may be issued for kernel mode or for user mode

code [41] [17]. The PowerLoader injection method is an example of APC injection that is used to target the Windows explorer process. In this technique, a pointer in the taskbar's window handling procedure is overwritten. A handle for the taskbar is obtained using the `FindWindow` API. The function pointer for the taskbar's window handling procedure is located at the start of the extra window memory. Extra window memory is accessed using the `GetWindowLong` and `SetWindowLong` APIs. A `GetWindowLong` call is made to obtain the address of the taskbar's window handler function. A `SetWindowLong` call is made to set the handler function pointer to the injected malware function. Finally, a call is made to the `SendNotifyMessage` API to cause execution of the malware function [42] [43].

*D. Information Stealing and Injection*

A key characteristic of banking malware is to lower OS integrity in order to steal user data and to modify data presented to the user. The lowering of OS integrity is performed by hooking API calls to alter flow control [44]. The hooking of API calls can be performed in several places in non-privileged (user-mode) and privileged execution (kernel mode). User-mode hooking techniques include the hooking of the Import Address Table (IAT) and inline hooking. User-mode hooking is performed on a per process basis. However, system mode hooks are applied to all processes. System mode hooking is more demanding because it is not well documented and errors in the hooking software can cause system failure [9] [40]. System mode hooking techniques include the hooking of the System Service Dispatch Table (SSDT), API call patching, Direct Kernel Object Modification (DKOM) and the addition of a malware file system driver.

There is one IAT per process and each IAT contains pointers to functions contained in the DLLs that are used by the process. The IAT is populated by the OS loader. In IAT hooking, the malware first saves the original API call function pointer and replaces it with a call to a malware function. Before the malware function exits, the original API call is made [39].

In inline hooking, the first five bytes of an API call are saved and then overwritten to jump to a malware function. The saved bytes are then executed at the end of the malware function in order to call the API call. As the Intel 32 bit architecture uses variable length instructions, a disassembler function can be used in the hooking process in order to determine instruction lengths to ensure that overwriting does not corrupt instructions and cause a system crash. The inline hooking approach is also used by the Microsoft Detours library [39] [45] [46].

When an API call is made by a user-mode program, the API call provides an interface to the underlying system code operating in privileged mode. An API call dispatch ID is passed to the `SYSENTER` instruction. The `SYSENTER` instruction is used to enter ring 0 or privileged mode execution. The API call dispatch ID is used as an index to the SSDT to obtain a function pointer for the required API call. Overwriting of a function pointer in the SSDT allows hooking of an API call. Unlike non-privileged hooking methods, SSDT hooking provides hooking of an API call for all processes [9] [40].

System code may be patched and patching can be performed on disk or in memory. The patching of the system code requires a higher skill level and is harder to detect than user-mode hooking. Patching can be used to disable tests, provide inline malware code, or to jump to malware code in a separate memory allocation [9].

DKOM provides methods to access and modify kernel data structures relating to processes and drivers, in order to hide processes or drivers. Windows processes and drivers are represented as data structures on double linked lists. A process may be hidden by removing the corresponding data structure from the process list. One difficulty with the DKOM approach is that much of the Windows OS kernel internals are not documented and may be changed by patches or by the release of OS updates [9] [40].

Windows provides a layered device driver architecture. In this architecture, IRPs that represent driver requests are passed between the objects at different layers of the device driver stack. The insertion of a malware device driver can be used to log keystrokes or to hide directories and files by dropping IRPs [9].

The research by Liang [47] resulted in the HookFinder program. This program uses dynamic analysis to identify hooking behaviours in malware samples without prior knowledge of hooking mechanisms.

*E. Network Communications*

Banking malware uses encryption of network traffic in order to avoid detection by intrusion detection systems (IDS) and intrusion prevention systems (IPS). TLS/SSL, RC4, AES, and custom encryption schemes are commonly used.

Stolen victim credentials and malware statistics are valuable information for the attacker. These details are passed back to a C2 server controlled by the attacker. This process is known as exfiltration [36]. Custom protocols are generally used by malware for loading a dynamic

configuration or exfiltrating stolen data. These protocols frequently use common TCP/IP ports (HTTP, HTTPS and DNS) as firewalls are often configured to pass data that uses these ports [48].

Banking malware can use a Domain Generation Algorithm (DGA). A DGA is an algorithm that produces domain names that may be based on date. The generated domain names are registered in advance by the attacker. The generated domain names are used as a fallback measure in cases where hardcoded C2 servers are not available [24]. A DGA can generate a large number of domain names. This may also be an attempt to hinder analysis [49]. Replication of date based DGA algorithms can be used to identify malware related domains. DGA domain names generated by replication may be used to identify malware instances and may be used in remediation [50].

URL redirection can be used to prevent the victim from accessing an anti-virus vendor's website. It can also be used to redirect the victim to a phishing website. URL redirection may be performed by adding extra entries to the `hosts` file, by setting up a proxy autoconfiguration file, or by hooking API calls [4].

*F. Backconnect*

Backconnect provides an attacker with access to the command line or desktop of an infected computer. Desktop access on the victim's computer allows banking transactions to be made by the attacker while the victim is connected to an internet banking session. A backconnect session is initiated when a command is sent from the attacker's C2 server to the victim's computer. Malware installed on the victim's computer responds by opening a connection to the attacker's C2 server. Initiating the connection from the victim's computer to the attacker's C2 server bypasses connection problems due to firewall or Network Address Translation (NAT) connections. A backconnect facility can be implemented using SOCKS, FTP, HTTP, Virtual Network Computing (VNC) or Remote Desktop Protocol (RDP) protocols [51] [52]. An advantage of using the VNC protocol is that VNC allows the victim and attacker to be simultaneously connected to a compromised computer. If an attacker using the RDP protocol connects to a compromised computer while the victim is using the computer, then the victim will be logged out [53].

A SOCKS or HTTP proxy provides the attacker with the ability to login to an internet banking website which, through the lens of bank security software, appears as though the

banking session is originating from the IP address of the victim's computer.

*G. Screenshot and Video Capture*

The screenshot behaviour allows an attacker to capture images of the desktop, showing details such as the use of a virtual keyboard to enter credentials for a banking session. Video capture allows recording of a video of the users banking session, providing the attacker with a better understanding of the victim's banking session [31].

*H. Anti-Analysis*

Strings in malware are commonly encrypted in order to delay analysis and prevent simple techniques being used to identify the malware. Simple custom encryption methods using bit or arithmetic operations, base64 encoding, and substitution ciphers are commonly used for string obfuscation [54] [34] [55] [17] [56]. Strings in malware may also be encrypted with stronger encryption algorithms [57].

API calls are commonly obfuscated in malware in order to delay analysis and to prevent simple methods from being used to understand the malware. Commonly used API call obfuscations include the use of an API call handler function that may be passed integer or hash values representing the API to be called. Another technique is to copy API code to a new memory allocation within the malware program [58] [53] [55].

There are a number of well-known virtual machine (VM) detection methods. These methods include Microsoft Virtual PC detection using custom instruction codes, VMWare detection using the control port, Interrupt Descriptor Table (IDT) register testing, checking of system services, checking the MAC address of the network card, checking for VM specific hardware devices, filesystem checking, use of the `cpuid` instruction to test for hypervisor execution, checking for VM specific registry keys and testing the number of CPU cores [59] [60] [61]. The Rovnix malware creates a system crash by sending an IoControlCode to a Sysinternals driver that is installed as part of the Rovnix installation. The aim of this is to halt automated analysis in a VM [21].

Malware anti-analysis features may prevent installed malware samples from being collected and then executed on a computer different from the installation computer. These techniques operate by generating data from a feature of the installation computer and patching the installed malware program with this data. When the installed program is executed, the feature data is checked against the current execution environment. An example of this is the use of the

hard disk volume identifier by Zeus malware [30]. When the Citadel malware detects that it is executing in a VM, it attempts to connect to a randomly generated C2 address. This is an attempt to mislead analysis into concluding that the sample is not active [62] [31].

## IV. Malware Families

This section gives details of the following banking malware families: Zeus and the Citadel variant, Vawtrak, Dridex, Dyre, Carberp and Rovnix. Descriptions of the operation of the above malware families are based on research papers, industry reports, and web pages. These sources describe potentially different versions of malware and, in some instances, present conflicting details of malware operation.

### *A. Zeus Malware*

The first version of the Zeus malware (Zeus V1) also known as Zbot was detected in 2006 [63] [6]. Zeus was the first banking malware kit. It was sold publicly and little technical knowledge was required to create a Zeus botnet. Zeus version two (Zeus V2) was first detected in 2010. Zeus V2 provided the facility to have multiple instances of Zeus running on one computer and the use of RC4 encryption in Electronic Code Book (ECB) mode [64]. Zeus V2 malware has the following capabilities:

- Rule based stealing of user data from HTML forms,
- Rule based injection of data into a websites HTML,
- URL redirection to websites controlled by the attacker,
- Capture of the HTML of targeted websites,
- Stealing of cookies,
- Deletion of cookies,
- Stealing of mail and FTP account credentials,
- Download and execution of additional programs and
- Built-in virtual network computing (VNC) console.

Zeus malware achieves persistence by adding the Zeus program to the `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` registry entry [30].

The Zeus V2 program contains a `xor` encrypted static configuration that includes a dynamic-configuration-URL. The dynamic configuration includes the URL where stolen data can be uploaded, trigger conditions for screenshots, the URL where the updated versions of the malware can be downloaded, URL-masks to control logging, URL redirection pairs, IP and

hostname pairs for URL redirection, and webinjects. When the dynamic configuration is decrypted, a recursive `xor` procedure is performed to decode the decrypted data. This procedure is known as "visual decrypt". The decoded data is then decompressed using UCL (NRV2B) decompression [30] [5]. The encrypted Zeus configuration data is stored in a randomly named registry key [30] [65]. Stolen data is encrypted using a second RC4 key. This data is temporarily stored into a file before being sent to the C2 server. Zeus V2 malware uses three threads: a thread for receiving network data, a thread that downloads the configuration data, and a thread that checks the malware's start-up key in the Windows registry and replaces it if it is removed [30].

Zeus uses a direct injection method to inject its malware code into other processes. Zeus iterates through processes using the `CreateToolhelp32Snapshot`, `Process32FirstW`, and `Process32NextW` APIs. The system and the Zeus processes are skipped. A mutex is then created using a `CreateMutexW` call. Each selected process is opened with an `OpenProcess` call. An access token for this process is obtained by an `OpenProcessToken` call with a requested access of TOKEN_QUERY. If the access token is not available, iteration continues with the next process. The process is then opened using an `OpenProcess` call. The access mode specified on the `OpenProcess` call checks for full access to the target process. If the current process is accessible, then memory is allocated in the target process using a `VirtualAllocEx` call. The Zeus malware is copied to the target process using a `WriteProcessMemory` call. A handle to the previously created mutex is passed using a `DuplicateHandle` call. The injected Zeus code is then started using a `WriteProcessMemory` call. The current process and the generated mutex are then closed by `CloseHandle` calls [65].

When a Backconnect command is sent from the attacker to the compromised computer, a network connection is opened from the victim's computer to a server controlled by the attacker. This allows network connections to a firewalled machine [66]. The Backconnect facility provides command line access and allows the use of the RDP, Socks, and FTP protocols [66] [67].

Zeus contains a Virtual Network Computing (VNC) based facility that allows the attacker to connect to the Graphical User Interface (GUI) of the victim's computer and perform banking transactions from the victim's IP address and allows access to hardware based authentication mechanisms used by the victim [68].

Zeus contains a screenshot facility. The trigger conditions to take screenshots are provided in the configuration. An example would be taking a screenshot of the banking balances of an internet banking session. The trigger condition would be the URL used by a specific bank to display balance information. Screenshots can be taken by a left mouse click on the virtual keyboards used by banking websites [68] [63].

Zeus provides a URL redirection facility that is used to take the victim from the desired website to an attacker controlled website [68].

An anti-analysis feature of the Zeus malware involves creating a GUID from the volume identifier of the hard disk on which the malware is installed. This GUID data is then written into the installed malware program. When the Zeus malware is executed, an action is taken to check the internal GUID data against a GUID created from the volume identifier of the hard disk from the execution environment. If the two GUIDs do not match, the malware terminates [30].

Zeus uses inline hooking of API calls [63]. Zeus V1 hooks the `NtQueryDirectoryFile` API call to hide its files and uses a simple fixed key encryption algorithm [30]. Zeus V2 does not hide its files [30]. A weakness with Zeus V2 is that its configuration data could be directly downloaded by security researchers and decrypted using the configuration RC4 key contained within the malware sample.

The source code for the Zeus V2 malware was leaked online in April 2011 [62] [5] [6]. The public availability of the Zeus source code led to the development of a number of Zeus variants including Citadel and ICE-IX. Another Zeus variant was the Zeus Peer to Peer (Zeus P2P) malware, also known as Gameover Zeus (GOZ).

### *B. Citadel Malware*

The Citadel malware was derived from the leaked Zeus V2 source code. Citadel was first detected in January 2012 [31]. Citadel is based on the Zeus source code, so there are significant similarities between the Zeus malware and the Citadel malware. The facilities of the Citadel malware are similar to those of the Zeus malware with the addition of the following improvements:

- Modified RC4 algorithm,
- Video capture,
- Sandbox detection,
- Support for Google Chrome,

- Securing of the configuration file,
- AES cryptography,
- VM detection,
- Automated DOS command line injection and
- Distributed denial of service attack capability [31] [62].

Citadel maintains persistence by adding the Citadel program to the `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` registry entry [62]. Citadel configuration data is stored in the registry in the same manner as Zeus. Citadel samples contain a 32 byte "hex as ASCII" value that is referred to as the login key. In Citadel version 1.3.4.5 the RC4 algorithm was modified by the addition of `xor` operations [31] [62] [69]. Citadel uses AES 128 bit encryption in ECB mode [31]. The AES encryption key is created by taking an MD5 hash of the login key and then using an RC4 key to encrypt this MD5 hash [31]. Version 1.3.4.5 uses a modified RC4 algorithm. The modification consists of adding extra `xor` operations to the RC4 encryption function. In version 1.3.5.1, the AES function was modified by the use of an additional 128 bit `xor` key. This key is used to `xor` each block of data prior to AES encryption or after AES decryption. Citadel uses UCL compression as does Zeus. Citadel communicates with the C2 server using the HTTP protocol with RC4 encrypted data [31].

The security of the Citadel configuration was improved by requiring an encrypted request for the configuration. To create a request to download a Citadel configuration, a new RC4 key is generated using a modified RC4_init function and static values embedded within the Citadel program. The new RC4 key is used to encrypt a configuration request data structure prior to sending to the Citadel C2 server [62] [70]. Citadel uses the same direct process injection technique as Zeus [37].

Citadel uses the same backconnect techniques as Zeus. Citadel contains a screenshot facility and has a video capture plugin to provide a live video recording of operations on the victim's screen. Citadel contains a VNC server and a SOCKS server [31] [71].

Citadel performs VM detection by checking strings in the resources section of each process running on the system. When a VM is detected, Citadel attempts to connect to a randomly generated URL. This is an attempt to deceive analysis and to indicate that the sample is no longer active [62] [31]. The Citadel malware contains a list of anti-virus websites which it prevents to by using DNS redirection [62].

### C. *Vawtrak Malware*

Vawtrak malware belongs to the Gozi malware family and is also known as Neverquest or Snifula [34] [72] [73]. Gozi malware was first detected in 2007 [74]. Gozi was a competitor to Zeus malware and used a "Crimeware As A Service" business model [72]. Vawtrak malware was first detected in August 2013 [75]. A new version of the Vawtrak malware was detected in October 2015 [76]. The following description refers to the Vawtrak variant from 2013. Vawtrak has the following facilities:

- Keystroke logging and credential stealing [34],
- VNC server, to allow remote control of the victim's computer [34] [72],
- SOCKS server, to allow the attacker to utilize communications with the IP address of the victim's computer [34] [72],
- Screenshot and video capture facilities [34] [72],
- A debugging feature that may also be used by malware analysts [72] [72],
- Ability to hinder anti-virus software by setting a Software Restrictions Policy (SRP) in the Windows registry. This SRP setting restricts the privileges of anti-virus processes [34] [72],
- Ability to disable the IBM Trusteer Rapport security program by hooking the VirtualProtect API [34],
- Support for information stealing and injection in Internet Explorer, Firefox and Chrome web browsers [34].

Vawtrak uses the `RegCreateKey` and `RegSetValueEx` API calls to create `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` registry entry to automatically start the Vawtrak malware DLL using the regsvr32 program [77] [72] [34].

Vawtrak uses an RNG function that is similar to the Linear Congruential Generator (LCG) employed in Visual C++ [72]. The LCG seed is stored in the header of encrypted data. The stream of pseudorandom numbers generated by the LCG function are used to perform `xor` encryption.

Communications with the C2 server use the HTTP protocol [34]. Some Vawtrak C2 servers are accessed using The Onion Router (TOR) protocol. TOR C2 hostnames are generated using a DGA with hardcoded seed values [77]. Data sent to the C2 server is obfuscated by a custom encoding function [78], and in later versions, this data is also base64 encoded. Vawtrak

configuration data from the C2 server is encrypted using LCG based encryption and is compressed using aPLib compression [79] [34] [80]. Initial C2 addresses are contained in the embedded static configuration. When an updated C2 server list is downloaded, signing is verified with an embedded RSA public key [34]. Encrypted and compressed configuration data is stored in a randomly named registry key [72] [34].

The Vawtrak configuration contains a list of target URLs and the associated webinjects. Following this is a list of social networking, entertainment, and shopping website URLs. When these URLs are encountered, credentials are extracted and sent back to the C2 server. These credentials are used to further spread the Vawtrak malware. Following the URL list is a list of banking related phrases. When these phrases are detected, the contents of the webpage are sent to the C2 server and are used to facilitate the development of webinject scripts to target additional organisations [72]. The Vawtrak configuration includes a Project ID that is used to partition the botnet and is specific to each Vawtrak customer [34].

On a 32 bit OS, Vawtrak injects its malware into processes using direct injection [34]. On a 64 bit OS, APC injection [72] using the PowerLoader technique is used to target the explorer process [73]. Vawtrak malware is injected into all processes at the security level of the injecting process. The injected code terminates if it is running in a system process [34].

Vawtrak uses inline hooking of user-mode API calls and contains a basic disassembler which is used to check the length of instructions prior to overwriting [34].

Vawtrak contains a VNC capability that provides the attacker with access to the desktop of the compromised computer. Vawtrak supports communications from the network address of the compromised computer using a built in SOCKS server [34] [72].

*D. Dridex Malware*

Dridex malware was first detected in 2014 [81] [82]. Dridex is also known as Bugat or Geodo [83] [84] [85]. Dridex is a variant of Cridex [84]. Cridex was first detected in 2012 and included a self-propagation capability which is not present in the Dridex variant [86]. In order to make the botnet more resistant to takedown efforts Dridex implemented a P2P communications topology in November 2014. Dridex communications are multi-layered. The outermost layer is comprised of compromised computers. Compromised computers which are not behind NAT firewalls may be used as peer nodes which form the second communications layer. The third layer is a frontend which handles communications with the C2 computers which form the fourth

layer [87] [86]. Unlike malware which is sold as crimeware kits, Dridex is a single botnet which is partitioned into subnets which divide the botnet based on attacker requirements. Dridex subnets are identified by numeric identifiers e.g. botnet 120, botnet 220 [86].

Dridex uses the following modules:

- Loader module,
- Main module,
- SOCKS module,
- VNC module,
- MOD4 server module,
- MOD6 server module,

The Dridex loader module is responsible for downloading the main Dridex module. The loader module contains a static configuration which includes the IP address to download the main module and the malware configuration. These items are requested using a RC4 encrypted XML message [86].

The main module provides the following functions:

- Backconnect,
- Cookie stealing,
- Credential theft,
- File operations,
- Command line processing,
- Information stealing and injection,
- Keystroke logging,
- Peer node server,
- Stealing of user information from HTML forms,
- Screenshots,
- HTTP redirection,
- Attacks Chrome, Firefox and Internet Explorer web browsers,

A remote graphical interface to the compromised computer is provided by the VNC module. Command line execution and file system operations are provided by the SOCKS module. Execution of new processes is provided by the mod4 module. Spamming using the contact list of the compromised computer is provided by the mod6 module. Dridex provides 32 bit and 64 bit

modules [86].

Dridex is started by the creation of a registry run key [87]. Dridex uses DLL injection to inject the main module into the explorer process [88]. When the Dridex loader runs, it deletes the Dridex run key from the registry and deletes the Dridex loader from the file system. The Dridex run key and loader are re-written on system shutdown [88] [85]. Dridex uses obfuscation of strings and API calls. Dridex uses an encrypted XML based protocol which operates underneath TLS communications. Dridex contains a hardcoded copy of the C2 public key. At initialization, Dridex generates a new RSA keypair, the new public key is encrypted with the C2s public key and sent to hardcoded layer 3 server. The Dridex modules and a node list are then requested. All subsequent communications are made through the peer nodes. The Dridex configuration is then requested. The Dridex protocol uses RC4 and RSA encryption. An RC4 key is encrypted using the generated RSA private key, the remainder of the message is RC4 encrypted. Dridex uses `xor` based custom encryption in the protocol for downloading modules. Binary data in the protocol is base64 encoded. Following module download, Dridex communications are directed to the Dridex nodes [86] [87]. The Dridex configuration is stored in a randomly named registry key using custom encryption and aPLib compression [89].

*E. Dyre Malware*

Dyre malware was first detected in 2014 [90]. Dyre is also known as Dyreza, Dyzap or Dyranges [91]. Dyre malware has the following capabilities:

- Man in the middle attack,
- DGA,
- Invisible Internet Project (I2P) tunnelling support,
- Built-in VNC server,
- 32 bit and 64 bit payloads [92] [91],
- AES encryption,
- Single partitioned botnet,
- Attacks Chrome, Firefox and Internet Explorer [92],

The first section of the injected Dyre payload contains position independent code that rebuilds the Import Address Table (IAT) [93]. The Dyre installer writes a randomly named executable file to the “Application Data” directory [94]. A mutex is created to check whether the malware is already installed. Persistence is achieved by creating a run key in

`HKCU\Software\Microsoft\CurrentVersion\Run` [92] [91] [94]. A call is made to `IsWow64Process` to determine whether it is running on a 64 bit system. The Dyre Installer contains resources incorporating 32 bit and 64 bit versions of the Dyre payload [91]. The Dyre installer runs and creates a service named "Google Update Service", which injects the Dyre payload into the target process, and then terminates [93] [95] [96].

The `NtMapViewofSection`, `VirtualAloc`, `NtQuerySystemInformation`, `OpenThread` and `NtQueueApcThread` API calls are used to perform APC injection of the Dyre payload into an svchost or explorer process [93] [95] [91] [94]. Dyre uses inline hooking of system API calls [94]. Dyre contains a table consisting of the compilation timestamps for the versions of the wininet DLL and the corresponding hooking offset. This table is used to identify the wininet version prior to patching. Unknown copies of wininet.dll are sent back to the C2 server [94].

The static configuration is stored in a randomly named program resource. The Dyre payload reads the configuration data and writes an encrypted configuration to a file in the "Application Data" directory. The configuration is in an XML format that differs from the commonly used Zeus configuration format [94].

If the infected computer uses a Network Address Translation (NAT) internet connection, then Dyre uses Session Traversal Utilities for NAT (STUN) to determine the IP address of the infected computer [97] [91]. Dyre performs a man in the middle attack by redirecting website requests through an attacker controlled proxy server. Webinjects are added when the website response is received. Storing the webinjects on an attacker controlled server has the advantage of hindering efforts to access the webinjects for analysis [96].

Dyre connects to Invisible Internet Project (I2P) nodes to establish peer to peer tunnelling connections. When the network connections are established, Dyre uses the `CreateToolhelp32Snapshot`, `Process32FirstW` and `Process32NextW` APIs to locate the web browser processes. The Dyre malware DLL is injected into the Internet Explorer, Firefox and Chrome web browsers. Named pipes are used to pass commands and data between the Dyre malware and the Dyre DLL running in the web browser [94].

If the Outlook email client is installed, then Outlook is hijacked to send emails containing the Dyre installer as attachments. The email recipients are obtained from the C2 server [98] [96].

Dyre has a modular architecture, a credential stealing module operates by capturing

HTTP POST requests which are then sent to the C2 server. The stolen data is in a plain text format [99] [90] [91] [97]. Dyre uses a VNC module [99], which has three exported functions, these are ClientSetModule, VncStartServer and VncStop Server [94]. These functions are present in the VNC module in the Carberp source code. It is possible that leaked Carberp source code may have been used by the Dyre author to create the Dyre VNC facility [94].

Incoming Dyre data is SSL encrypted. After SSL decryption, the data is AES encrypted. Configuration data and malware plugins are RSA signed [91] [97]. Dyre generates a bot-id comprised of the computer name, the OS version and a 32 byte unique identifier [99] [90].

Dyre malware contains a Domain Generation Algorithm (DGA) that uses the date as a key to generate the C2 server's IP address and port pairs [93]. When the Dyre malware is initialised, the number of processor cores is checked. If this number is less than two, the Dyre malware terminates. This test is an anti-VM test to hinder security researchers [100].

*F. Rovnix Malware*

The Rovnix malware was first detected in 2011 [101]. Rovnix makes use of a 64 bit bootkit that is similar to the bootkit used by the Carberp malware. The source code of the Carberp bootkit was leaked as part of the Carberp source code leak in 2013 [102].

Rovnix achieves persistence by infecting the Volume Boot Record (VBR) of the active drive. When a computer running a Windows OS is started, the Basic Input/Output System (BIOS) reads the first sector of the bootable hard disk. This sector is known as the Master Boot Record (MBR). The MBR contains start-up code and a partition table. The MBR code scans the partition table and locates the partition with the bootable flag set. Then the code in the first sector (VBR) of the bootable partition is executed. File system specific partition start-up code is stored in the VBR. Rovnix malware infects the VBR of NTFS boot partitions. At installation, Rovnix compresses the original VBR code, replaces the original VBR with the Rovnix start-up code, and appends the compressed VBR start-up code. When the computer is started, the Rovnix start-up code obtains control. After the Rovnix start-up code has been executed, the original VBR code is decompressed and executed. The Rovnix start-up code in the VBR is polymorphic in order to avoid anti-virus detection [103] [101] [104]. In real mode, BIOS functions are accessed using the INT instruction. BIOS hard disk services are requested using INT 13h [104] [105]. The Rovnix VBR start-up code performs hooking of the BIOS INT 13h handler. Hooking of the INT 13H allows patching of the ntloader/bootmgr, thereby enabling control to be maintained after the boot

manager is loaded. When the boot manager is loaded, the original VBR code is decompressed and executed. During the system start-up process, the start-up code must switch from real mode to protected mode. In order to maintain control in the transition from real mode to protected mode, Rovnix makes use of the IDT. The IDT is a table that is used in protected mode and contains descriptors that provide access to interrupt and exception handlers [106] [107] [101] [108]. The Rovnix malware copies itself into an unused section of the IDT. The malware then hooks the INT 1 protected mode debug handler [108] [101]. It sets hardware breakpoints using the CPU's debugging registers dr0-dr7 in order to maintain control during Windows start-up [101] [109].

The use of debugging registers allows the malware to gain control at specific points during the kernel loading process. This VBR infection technique is complex and requires a skilled developer. Rovnix malware operates on 32 bit and 64 bit versions of the Windows OS and allows an unsigned Rovnix driver to be loaded into the Windows OS. [101].

Versions of Rovnix from 2012 used disk sectors near the end of the partition to create an RC6 encrypted Virtual File System (VFS). In this version, the configuration was stored in the VFS [23]. A newer version of Rovnix from 2014 uses RC4 encryption and stores the VFS in a binary file [110].

A filtering driver is provided that prevents read or write access of the VFS file. If the Rovnix installer detects that full disk encryption software is installed on the victim's computer then the bootkit is not installed and the Rovnix banking malware is installed in the filesystem of the victim computer [21]. The `NtCreateFile` and `NtDeleteFile` API calls are hooked to protect the VFS [110].

If the bootkit cannot be installed due to insufficient storage, it will install and run a copy of the Sysinternals Contig tool to defragment storage [21].

The Rovnix installer has the facility to either download the latest version of the Rovnix banking trojan or to install a copy of the Rovnix banking trojan from program resources. The Rovnix installer contains 32 bit and 64 bit versions of the Rovnix banking trojan. The stored copies of the banking trojan are compressed with a modified version of aPLib compression. [21]. The Rovnix trojan is initialised by a registry run key or by injection via the unsigned malware driver. If the Rovnix bootkit was not able to be installed, then a registry run key is created to start the Rovnix malware. The version of Rovnix banking trojan that is started from the filesystem

contains a thread that protects the registry run key [33].

Rovnix uses a DGA to generate the C2 domain names. The DGA uses month as the key [111]. The Rovnix malware downloads additional plugins and external programs from the C2 server [103] [33].

The plugins that are downloaded by Rovnix include a banking trojan known as ReactorDemo [33], a TOR client, a password stealer, and a bitcoin stealer. Rovnix plugins are identified by the Cyclic Redundancy Code (CRC32) value of the plugin. The downloaded plugins are RC2 encrypted DLLs. The decrypted data contains a RSA signature. The hashing algorithm used in this signature is not documented. Disassembly of a Rovnix sample showed the signature hashing algorithm to be SHA1. The TOR plugin contains a DGA that can be used as an alternative to TOR communications. The password stealer plugin may have been taken from the Zeus VM (KINS) partial source code leak [33].

The Rovnix banking trojan runs a server process on the infected computer. This process acts like a proxy server. The Rovnix banking trojan is injected into the Internet Explorer, Firefox, and Chrome browser processes. Target processes for injection are identified by comparing a CRC of the process name with a set of hardcoded CRC values within the Rovnix malware. The injected malware hooks a number of functions within the browser process. The hook routines perform communication with the Rovnix proxy server process. RC2 encrypted web-injects are downloaded from the C2 server. The proxy server process parses the webpages visited by the victim and performs HTML injection from the web-injects. The Symantec Anti-Virus company identifies Rovnix malware as a Carberp variant due to the shared bootkit and similar programming techniques [33].

*G. Carberp Malware*

Carberp malware was first detected in June 2010 [112] [32]. The Carberp source code was leaked in June 2013 [102]. Carberp uses a bootkit that is similar to the Rovnix bootkit [32] [113]. It is possible that the same malware developer created both the Rovnix and Carberb bootkits [32]. The bootkit allows the Carberp malware to persist after the re-installation of the Windows OS [113]. Carberp has the following facilities:

- Modular design, which is able to download and execute new modules from the C2 server. Carberp modules have an executable decryptor and are `xor` encrypted [55],
- Screenshot module,

- Credential stealing module,
- VNC module to allow desktop access,
- A module to remove other banking malware,
- A module to disable anti-virus products,
- A module for generating traffic to perform Distributed Denial of Service (DDOS) attacks, [55] [114].
- A capability to attack Internet Explorer and Firefox web browsers.

A Carberp variant was also produced that targeted the Android platform [114].

Carberp provides a general mode of attack which is independent of specific targeting. When a victim logs into an SSL website, Carberp intercepts the POST request and copies the victim's credentials. The stolen credentials are sent back to the C2 server. Carberp also supports a targeted attack when specific URLs are used to initiate information stealing [115]. During the installation process, Carberp attempts to run privileged mode code to remove SSDT hooks from a number of operating system API calls. This is done to prevent anti-virus programs from interfering with the Carberp installation process [55]. The Carberp installer also attempts privilege escalation by exploiting several vulnerabilities [32].

A kernel mode driver is loaded by the Carberp bootkit. This driver injects a malicious DLL into the user-mode address space of processes running on the system. The bootkit is used in order to allow loading of an unsigned 64 bit driver on 64 bit Windows OS, thereby bypassing the driver signing policy of Windows 64 bit OS [32]. The Carberp bootkit supports a hidden RC6 encrypted VFS that is allocated in unused storage. The VFS is a modified FAT16 file system. A filter driver is provided that prevents read or write access of the hidden virtual file system. The bootkit only supports NTFS partitions. The malicious DLL and other files are stored in the VFS. The malicious DLL is injected into the target process at system start-up [116]. The Carberp bootkit generator uses metamorphism in the creation of the boot code. This provides a unique signature for the boot code for each Carberp malware user [117] [116]. Carberp uses function name hashing to obfuscate API calls. [55].

Carberp injects malware into every process and hooks the `NtQueryDirectoryFile` API call in order to hide the directory containing the Carberp program and configuration [55]. Carberp uses direct injection and APC injection. Carberp uses inline hooking of API calls [115].

Carberp uses the HTTP protocol for C2 server communications [115]. There are two

variants of Carpberp malware making use of RC2 or RC4 encryption. The variant using RC4 encryption was discontinued after an arrest in 2012 [114]. The leaked Carberp source code uses RC2 encryption. Carberp also uses MD5 hashing, base64 encoding, and RC6 encryption [112] [32] [116].

## V. Analysis of Malware Behaviours

The Pharos Static Binary Analysis Framework [118] is an open source program for the static analysis of malware. Pharos is based on the Rose compiler infrastructure [119] which is used for disassembly, control flow analysis and instruction semantics. A major new version of the Pharos Framework was released in June 2017 and this research is based on the facilities provided by this new version. The ApiAnalyzer component of the Pharos Framework provides the capability to identify malware behaviours by use of user supplied rules that contain sequences of API calls and the relationships between the API calls. The behaviours which are identified by ApiAnalyzer correspond to low level actions in the MAEC model. ApiAnalyzer uses data flow analysis to verify the relationships between the API calls, i.e. that the calls are operating on the same handle.

A limitation of the current version of the Pharos Framework is that API analysis can only be performed on programs which use an import table to make API calls. The current version of the Pharos Framework contains limited configuration data. It was necessary to create a configuration containing the stack offset and the API ordinal number for each API call used by each malware sample. The stack offset is the number of bytes passed on the stack when making a specific API call.

A limited number of example rules are provided with the Pharos Framework [120]. A rule is provided to search for `CreateToolhelp32Snapshot calls`, where the handle created by this call is passed to `Process32FirstW` and `Process32NextW` API calls. This rule is used to search for a behaviour where malware is iterating through running processes.

The Zeus, Citadel and Zeus P2P samples were unpacked using a static unpacker. The Vawtrak and Dyre samples were unpacked and the import tables rebuilt using OllyDbg and the Ollydmp plugin. The Dridex and Carberp malware families use hashed API obfuscation and are not suitable for use with the current version of ApiAnalyzer. A ReactorBot sample was dumped using Ollydmp, however this malware is able to defeat import table reconstruction. Therefore, the malware families used in this analysis are Zeus, Citadel, Vawtrak and Dyre.

In the experiments conducted for this paper, the unpacked malware samples were first examined with the IDA disassembler to ensure that API call sequences form behaviours which correspond to the rule being tested.

*A. Process Injection Rule 1*

The first rule tested was the example rule provided with the Pharos Framework [120]. This rule, shown in Figure 1 searches for all occurrences of the `CreateToolhelp32Snapshot` API and uses dataflow analysis to locate all `Process32FirstW` and `Process32NextW` API calls which use the handle created by the `CreateToolhelp32Snapshot` call.

The results of using the process iteration rule against the selected malware samples is shown in Table III.

It is noted that the process iteration rule located the process iteration behaviour in the Citadel malware but not in the Zeus malware, despite the fact that Citadel is a Zeus variant and the process iteration functions are very similar. The Vawtrak sample contains a process iteration behaviour corresponding to the rule, but it was not detected. Discussion with the Pharos Framework developers indicates that the design philosophy of the program is to avoid false positives, however some false negatives are considered to be acceptable. The development of the program is in progress and the program design currently contains approximations which may not work for all malware families.

The rule in Figure 2 is used to scan an unpacked malware sample and to locate instances of direction process injection. The results of using the process injection rule against the selected malware samples is shown in Table IV. The Vawtrak sample contained a process injection behaviour which is not detected. This sample uses Fastcall parameter passing [17] which is supported by the Pharos Framework, further work is required to determine the cause of this problem.

The Message Digest 5 (MD5) hashes of the samples used for experiments with ApiAnalyzer are shown in Table V.

*B. Future Work*

The open source Pharos Framework currently locates API calls through the static import table. It would be advantageous to add support for dynamic API loading into the Pharos Framework. Dynamic API loading is performed by `LoadLibrary` and `GetProcAddress`

API calls. Support for obfuscated API loading, where the API is represented by a hash value and an in-memory DLL is parsed to obtain the API virtual address, would be valuable.

The utility of the Pharos Framework could be improved by the identification of cryptographic and compression operations. This would allow rules to be created which could express common behaviours such as “data downloaded, decrypted and decompressed, then written to the registry”.

## VI. Conclusion

This paper provides a survey of seven banking malware families and draws together a fragmented and industry based literature to provide a coherent description of major banking malware families, their variants, relationships and source code leakages.

Banking malware were selected as the basis for this paper based on their prevalence, persitence, financial damage and on the lack of literature providing a good description of their internal operations.

A challenge for malware analysis is that while existing analysis techniques provide volumes of low level details from malware samples, there is a need to produce a high level understanding of malware behaviour. This task of bridging the semantic gap from low level detail into a high level understanding lies at the core of malware reverse engineering and is largely a manual process.

Although not all malware families use exactly the same behaviours, malware authors draw from a constrained set of techniques to build their malware. This allows malware families to be be described in terms of the techniques which are used to implement these behaviours. Dynamic analysis techniques have been used to generate profiles of malware behaviour, however dynamic analysis has difficulty in achieving full code coverage.

This paper uses the example of the ApiAnalyzer program from the Pharos Framework to provide the static identification of malware behaviours. The use of static analysis techniques to automatically identify malware behaviours presents a representation of malware capability at a higher level of abstraction than has previously been available.

## Acknowledgement

This research was funded in part through the Internet Commerce Security Laboratory (ICSL), a joint venture between Westpac, IBM and Federation University Australia.

## References

[1] AV-Test, "Av-test security report 2016/2017," Tech. Rep., 2017. [Online]. Available: https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf

[2] A. Mohaisen, O. Alrawi, and M. Mohaisen, "Amal: High-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, 2015.

[3] M. Christodorescu, S. Jha, S. Seshia, D. Song, R. E. Bryant *et al.*, "Semantics aware malware detection," in *Security and Privacy, 2005 IEEE Symposium on*. IEEE, 2005, pp. 32–46.

[4] A. R. A. Grégio, V. M. Afonso, D. S. Fernandes Filho, P. L. de Geus, and M. Jino, "Toward a taxonomy of malware behaviors," *The Computer Journal*, vol. 58, no. 10, pp. 2758–2777, 2015.

[5] A. Baumhof and A. Shipp, "Zeus trojan update new variants based on leaked source code," 2011. [Online]. Available: http://www.tidos-group.com/blog/2011/09/30/zeus-trojan-update-new-variants-based-on-leaked-zeus-source-code/

[6] M. Riccardi, R. Di Pietro, and J. A. Vila, "Taming zeus by leveraging its own crypto internals," in *eCrime Researchers Summit (eCrime), 2011*. IEEE, 2011, pp. 1–9.

[7] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. C. Mitchell, "A layered architecture for detecting malicious behaviors," in *Recent Advances in Intrusion Detection*. Springer, 2008, pp. 78–97.

[8] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in *Proceedings of the 1st India software engineering conference*. ACM, 2008, pp. 5–14.

[9] J. Mankin, "Classification of malware persistence mechanisms using low-artifact disk instrumentation," Ph.D. dissertation, Northeastern University Boston, 2013.

[10] P. K. Singh, "A physiological decomposition of virus and worm programs," Master's thesis, University of Louisiana at Lafayette, 2002.

[11] I. Kirillov, D. Beck, and P. Chase, "The maec language, overview," 2014. [Online]. Available: http://maecproject.github.io/documentation

[12] A. Lee, V. Varadharajan, and U. Tupakula, "On malware characterization and attack classification," in *Proceedings of the First Australasian Web Conference-Volume 144*. Australian Computer Society, Inc., 2013, pp. 43–47.

[13] I. Kirillov, D. Beck, and P. Chase, "Maec default vocabularies specification, version 4.1,"

2014. [Online]. Available: https://maec.mitre.org/language/version4.1/MAEC_Vocabs_Spec_v1_1.pdf

[14] Microsoft, "Developer resources: Api index," Tech. Rep., 2016. [Online]. Available: http://msdn.microsoft.com/library/windows/desktop/develop

[15] B. Blunden, *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. Jones & Bartlett Publishers, 2012.

[16] Y.-M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski, "Detecting stealth software with strider ghostbuster," in *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*. IEEE, 2005, pp. 368–377.

[17] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012.

[18] H. Carvey, "The windows registry as a forensic resource," *Digital Investigation*, vol. 2, no. 3, pp. 201–205, 2005.

[19] ——, *Windows Forensic Analysis DVD Toolkit*. Syngress, 2009.

[20] Y.-M. Wang, R. Roussev, C. Verbowski, A. Johnson, M.-W. Wu, Y. Huang, and S.-Y. Kuo, "Gatekeeper: Monitoring auto-start extensibility points (aseps) for spyware management." in *LISA*, vol. 4, 2004, pp. 33–46.

[21] Malware Digger, "Rovnix dropper analysis (trojandropper:win32/rovnix.p)," 2015. [Online]. Available: http://www.malwaredigger.com/2015/05/rovnix-dropper-analysis.html

[22] D. E. Rodionov, A. Matrosov, and D. Harley, "Bootkits: Past, present & future," in *VB Conference*, 2014. [Online]. Available: https://www.virusbulletin.com/uploads/pdf/conference/vb2014/VB2014-RodionovMatrosov.pdf

[23] E. Rodionov and A. Matrosov, "Defeating anti-forensics in contemporary complex threats," ESET, 2012. [Online]. Available: http://go.eset.com/us/resources/white-papers/RodionovMatrosov-VB2012.pdf

[24] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda, "Inspector gadget: Automated extraction of proprietary gadgets from malware binaries," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 29–44.

[25] J.-I. Boutin, "The evolution of webinjects," 2014. [Online]. Available: https://www.virusbulletin.com/conference/vb2014/abstracts/evolution-webinjects

[26] 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 *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*. IEEE, 2010, pp. 31–38.

[27] J. Forrester, "An exploration into the use of webinjects by financial malware," Master's thesis, Rhodes University, 2014.

[28] L. Kharouni, "Automating online banking fraud," Technical Report, Trend Micro Incorporated, Tech. Rep., 2012.

[29] C. Rossow, C. Dietrich, and H. Bos, "Large-scale analysis of malware downloaders," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2012, pp. 42–61.

[30] J. Wyke, "What is zeus?" Sophos Labs, Tech. Rep., 2011.

[31] J. Milletary, "Citadel trojan malware analysis," 2012. [Online]. Available: http://botnetlegalnotice.com/citadel/files/Patel\ Decl\ Ex20.pdf

[32] A. Matrosov, E. Rodionov, D. Volkov, and D. Harley, "Win32/carberp when you're in a black hole stop digging," 2011. [Online]. Available: www.eset.com/us/resources/white-papers/carberp.pdf

[33] Malware Digger, "Rovnix payload analysis," 2015. [Online]. Available: http://www.malwaredigger.com/2015/06/rovnix-payload-and-plugin-analysis.html

[34] J. Kroustek, "Analysis of banking trojan vawtrak," AVG, Tech. Rep., 2015. [Online]. Available: http://now.avg.com/wp-content/uploads/2015/03/avg_technologies_vawtrak_banking_trojan_report.pdf

[35] J. Wyke, "Breaking the bank(er): Automated configuration data extraction for banking malware," Technical Report, Sophos, Tech. Rep., 2015.

[36] A. Al-Bataineh and G. White, "Analysis and detection of malicious data exfiltration in web traffic," in *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*. IEEE, 2012, pp. 26–31.

[37] T. Barabosch, S. Eschweiler, and E. Gerhards-Padilla, "Bee master: Detecting host-based code injection attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2014, pp. 235–254.

[38] H.-M. Sun, Y.-T. Tseng, Y.-H. Lin, and T. Chiang, "Detecting the code injection by hooking system calls in windows kernel mode," in *2006 International Computer Symposium, ICS*, 2006.

[39] J. Berdajs and Z. Bosnić, "Extending applications using an advanced approach to dll injection and api hooking," *Software: Practice and Experience*, vol. 40, no. 7, pp. 567–584, 2010.

[40] K. Kasslin, M. Ståhlberg, S. Larvala, and A. Tikkanen, "Hide'n seek revisited–full stealth is back," in *Proceedings of the 15th Virus Bulletin International Conference*, 2005.

[41] J. Alexander, "Ghost in the shell: A counter-intelligence method for spying while hiding in (or from) the kernel with apcs," 2012.

[42] Author Unknown, "Powerloader injection - something truely amazing," 2013. [Online]. Available: http://www.malwaretech.com/2013/08/powerloader-injection-something-truly.html

[43] C. Dietrich, "Through the window: Creative code invocation," 2014. [Online]. Available: https://www.crowdstrike.com/blog/through-window-creative-code-invocation

[44] M. Garcia-Cervigon and M. M. Llinas, "Browser function calls modeling for banking malware detection," in *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*. IEEE, 2012, pp. 1–7.

[45] W. Tsaur and Y.-C. Chen, "Exploring rootkit detectors' vulnerabilities using a new windows hidden driver based rootkit," in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 842–848.

[46] G. Hunt and D. Brubacher, "Detours: Binary interception of win32 functions," in *Usenix Windows NT Symposium*, 1999, pp. 135–143.

[47] Z. Liang, H. Yin, and D. Song, "Hookfinder: Identifying and understanding malware hooking behaviors," *Department of Electrical and Computing Engineering*, p. 41, 2008.

[48] K. Born, "Browser-based covert data exfiltration," *arXiv preprint arXiv:1004.4357*, 2010.

[49] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou II, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of dga-based malware." in *USENIX Security Symposium*, 2012, pp. 491–506.

[50] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A comprehensive measurement study of domain generating malware," in *USENIX Security Symposium*. USENIX, 2016, pp. 263–278.

[51] J. Dos Santos, "Troyan citadel backconnect vnc server manager," 2012. [Online]. Available:

http://laboratoriomalware.blogspot.de/2012/12/troyan-citadel-backconnect-windows.html

[52] A. Sood and R. Enbody, *Targeted Cyber Attacks: Multi-staged Attacks Driven by Exploits and Malware*. Syngress, 2014.

[53] A. Cherepanrov and R. Lipovsky, "Hesperbot: A new, advanced banking trojan in the wild," 2013. [Online]. Available: www.eset.com/us/resources/white-papers/Hesperbot_Whitepaper.pdf

[54] Trend Micro, "File-patching zbot variants zeus 2.0 levels up," Trend Micro, Tech. Rep., 2010.

[55] M. Giuliani and A. Allievi, "Carberp - a modular information stealing trojan," 2010.

[56] J. Cannell, "Obfuscation: Malware's best friend," 2015. [Online]. Available: https://blog.malwarebytes.org/threat-analysis/2013/03/obfuscation-malwares-best-friend/

[57] I. Aronov, "An example of common string and payload obfuscation techniques in malware," 2015. [Online]. Available: https://securityintelligence.com/an-example-of-common-string-and-payload-obfuscation-techniques-in-malware/

[58] M. Suenaga, "A museum of api obfuscation on win32," in *Proceedings of 12th Association of Anti-Virus Asia Researchers International Conference, AVAR*, 2009.

[59] M. Brand, "Analysis avoidance techniques of malicious software," Ph.D. dissertation, Edith Cowan University, 2010.

[60] B. Lau and V. Svajcer, "Measuring virtual machine detection in malware using dsd tracer," *Journal in Computer Virology*, vol. 6, no. 3, pp. 181–195, 2010.

[61] E. Kovacs, "Dyre banking trojan counts processor cores to detect sandboxes," SecurityWeek, Tech. Rep., 2015.

[62] A. Rahimian, R. Ziarati, S. Preda, and M. Debbabi, "On the reverse engineering of the citadel botnet," in *Foundations and Practice of Security*. Springer, 2014, pp. 408–425.

[63] M. Ligh and S. S. Corporation, "[prg] malware case study," Secure Science Corporation, Tech. Rep., 2006.

[64] K. Selvaraj, "A brief look at zeus/zbot 2.0," 2010. [Online]. Available: http://www.symantec.com/connect/blogs/brief-look-zeuszbot-20

[65] R. Alvarez, "Same zeus, different features," 2013. [Online]. Available: http://blog.fortinet.com/post/same-zeus-different-features

[66] N. Falliere and E. Chien, “Zeus: King of the bots,” 2009.

[67] Zeus Author, “Zeus source code,” 2011. [Online]. Available: http://www43.zippyshare.com/v/81908671/file.html

[68] K. Stevens and D. Jackson, “Zeus banking trojan report,” 2010. [Online]. Available: http://www.secureworks.com/cyber-threat-intelligence/threats/zeus/

[69] AhnLab, “Malware analysis: Citadel,” Tech. Rep., 2012. [Online]. Available: http://seifreed.es/docs/Citadel\%20Trojan\%20Report eng.pdf

[70] J. Wyke, “The citadel crimeware kit - under the microsope,” Sophos Labs Naked Security Blog, Tech. Rep., 2012.

[71] A. Sood and B. Rohit, “Prosecting the citadel botnet - revealing the dominance of the zeus descendent: Part one,” 2014. [Online]. Available: https://www.virusbulletin.com/virusbulletin/2014/09/prosecting-citadel-botnet-revealing-dominance-zeus-descendent-part-one

[72] J. Wyke, “Vawtrak - international crimeware-as-a-service,” 2014.

[73] Kimberly, “Analysis of vawtrak,” 2014. [Online]. Available: http://stopmalvertising.com/malware-reports/analysis-of-vawtrak.html

[74] D. Jackson, “Gozi trojan,” 2007. [Online]. Available: http://www.secureworks.com/cyber-threat-intelligence/threats/gozi/

[75] Trend Micro, “Banking malware vawtrak now uses malicious macros, abuses windows powershell,” 2015.

[76] D. Huss and M. Matthew, “In the shadows: Vawtrak aims to get stealthier by adding new data cloaking,” 2015. [Online]. Available: https://www.proofpoint.com/us/threat-insight/post/In-The-Shadows

[77] R. Alvarez, “Nesting doll: Unwrapping vawtrak,” 2015. [Online]. Available: https://www.virusbtn.com/virusbulletin/archive/2015/01/vb201501-Vawtrak

[78] D. Kilman, “Decoding vawtrak neverquest traffic,” 2014. [Online]. Available: http://cybersecuritymave-techie.blogspot.com.au/2014/07/decoding-vawtrakneverquest-traffic.html

[79] P. Asinovsky, “Neverquest malware analysis,” 2014. [Online]. Available: https://devcentral.f5.com/articles/neverquest-malware-analysis

[80] J. Ibsen, “aplib v1.1.1 - compression library,” 2014. [Online]. Available:

http://ibsensoftware.com/products_aPLib.html

[81] M. Sanghavi, "Dridex and how to overcome it," 2016. [Online]. Available: https://www.symantec.com/connect/blogs/dridex-and-how-overcome-it

[82] R. A. Certeza, "Dealing with the mess of dridex," 2015. [Online]. Available: http://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/3147/dealing-with-the-mess-of-dridex

[83] R. Thadani, "Report: The dridex trojan is back," 2016. [Online]. Available: http://blogs.quickheal.com/report-the-dridex-trojan-is-back

[84] abuse.ch, "Cridex, feodo, geodo dridex, whats next?" 2014. [Online]. Available: https://www.abuse.ch/?p=8332

[85] Kimberly, "Analysis of dridex / cridex / feodo / bugat," 2014. [Online]. Available: http://stopmalvertising.com/malware-reports/analysis-of-dridex-cridex-feodo-bugat.html

[86] D. O'Brien, "Dridex: Tidal waves of spam pushing dangerous financial trojan," 2014. [Online]. Available: www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/dridex-financial-trojan.pdf

[87] Blueliv, "Chasing cybercrime: Network insights of dyre and dridex trojan bankers," 2015.

[88] L. Rocha, "Malware analysis - dridex loader - part 2," 2016. [Online]. Available: https://countuponsecurity.com/2016/08/28/malware-analysis-dridex-loader-part-2

[89] M. Su, "Dridex in the wild," 2015. [Online]. Available: https://www.virusbulletin.com/virusbulletin/2015/07/dridex-wild

[90] A. Shulman and H. Dorfman, "Dyre financial malware internals," 2015. [Online]. Available: https://devcentral.f5.com/d/dyre-malware-internals

[91] B. Stone-Gross and P. Khandhar, "Dyre banking trojan," 2014. [Online]. Available: http://www.secureworks.com/cyber-threat-intelligence/threats/dyre-banking-trojan/

[92] Kimberley, "Introduction to dyreza, the banker that bypasses ssl," 2014. [Online]. Available: http://stopmalvertising.com/malware-reports/introduction-to-dyreza-the-banker-that-bypasses-ssl.html

[93] A. Chiu and A. Villegas, "Threat spotlight: Dyre/dyreza: An analysis to discover the dga,"

2015. [Online]. Available: http://blogs.cisco.com/security/talos/threat-spotlight-dyre

[94] A. Hanel, "Dyre infection analysis by alexander hanel," 2014. [Online]. Available: http://hooked-on-mnemonics.blogspot.com.au

[95] Symantec Security Response, "Dyre: Emerging threat on financial fraud landscape," 2015. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/dyre-emerging-threat.pdf

[96] J. Kuhn, L. Mueller, and L. Kessem, "The dyre wolf: Attacks on corporate banking accounts," 2015. [Online]. Available: https://portal.sec.ibm.com/mss/html/en_US/support_resources/pdf/Dyre_Wolf_MSS_Threat_Report.pdf

[97] Trend Micro, "A closer look at dyre malware, part 1," 2014. [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/a-closer-look-at-dyre-malware-part-1/

[98] M. Marcos, "New dyre variant hijacks microsoft outlook, expands targeted banks," 2015. [Online]. Available: http://http://blog.trendmicro.com/trendlabs-security-intelligence/new-dyre-variant-hijacks-microsoft-outlook-expands-targeted-banks

[99] Kimberley, "Analysis of dyreza - changes & network traffic," 2014. [Online]. Available: http://stopmalvertising.com/malware-reports/analysis-of-dyreza-changes-network-traffic.html

[100] R. Lemos, "Dyre malware developers add code to elude detection by analysis tools," 2015. [Online]. Available: http://www.eweek.com/security/dyre-malware-developers-add-code-to-elude-detection-by-analysis-tools.html

[101] D. Harley, "Hasta la vista, bootkit: Exploiting the vbr," ESET, 2011. [Online]. Available: www.welivesecurity.com/2011/08/23/hasta-la-vista-bootkit-exploiting-the-vbr

[102] P. Kruse, "Carberp source code confirmed leaked," 2013. [Online]. Available: https://www.csis.dk/en/csis/news/3961/

[103] A. Matrosov, "Rovnix bootkit framework updated," 2012. [Online]. Available: http://www.welivesecurity.com/2012/07/13/rovnix-bootkit-framework-updated/

[104] B. Carrier, *File System Forensic Analysis*. Addison-Wesley Reading, 2005.

[105] L. J. Scanlon, *Assembly Language Programming for the IBM PC AT*. Brady

Communications Company, 1986.

[106] D. Quist, V. Smith, and O. Computing, "Detecting the presence of virtual machines using the local data table," *Offensive Computing*, 2006.

[107] US Air Force, "Analysis of the intel pentiums ability to support a secure virtual machine monitor," in *Proceedings of the... USENIX Security Symposium*. USENIX Association, 2000, p. 129.

[108] B. Blunden, *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. Jones & Bartlett Publishers, 2013.

[109] B. B. Brey, "Intel microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, pentium, pentium proprocessor, pentium ii, iii, 4," 2005.

[110] C. Feng, "The evolution of rovnix: New virtual file system (vfs)," 2014. [Online]. Available: http://blogs.technet.com/b/mmpc/archive/2014/05/05/the-evolution-of-rovnix-new-virtual-file-system-vfs.aspx

[111] Bitdefender, "Tracking rovnix," 2014. [Online]. Available: http://www.malwaredigger.com/2015/06/rovnix-payload-and-plugin-analysis.html

[112] R. Dolmans and W. Katz, "Rp1: Carberp malware analysis," 2013.

[113] Tigzy, "Carberp bootkit : How self protection is effective," 2013. [Online]. Available: http://www.adlice.com/carberp-bootkit-how-self-protection-is-effective/

[114] P. Kalnai, "Banking trojan carberp: An epitaph?" 2013. [Online]. Available: https://blog.avast.com/2013/04/08/carberp_epitaph/

[115] Trusteer, "Carberp under the hood of carberp: Malware & configuration analysis," 2010. [Online]. Available: landing2.trusteer.com/sites/default/files/Carberp_Analysis.pdf

[116] E. Maor, "Carberp source code for sale - bootkit included!" 2013. [Online]. Available: http://securityintelligence.com/carberp-source-code-sale-free-bootkit-included/#.VamduZP5s_s

[117] B. Grill, C. Platzer, and J. Eckel, "A practical approach for generic bootkit detection and prevention," in *Proceedings of the Seventh European Workshop on System Security*. ACM, 2014, p. 4.

[118] SEI CMU, "Pharos static binary analysis framework," 2017. [Online]. Available: https://github.com/cmu-sei/pharos

[119] D. Quinlan and C. Liao, "The rose source-to-source compiler infrastructure," in *Cetus*

*users and compiler infrastructure workshop, in conjunction with PACT*, vol. 2011, 2011, p. 1.

[120] SEI CMU, "Static identification of program behavior using sequences of api calls," 2016. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2016/04/static-identification-of-program-behavior-using-sequences-of-api-calls.html

**Fig. 1:** Rule To Identify Process Iteration

**Fig. 2:** Rule To Identify Process Injection

**TABLE I:** Banking Malware Relationships

| Malware | Detected | AKA | Variant | Source Leak |
|---|---|---|---|---|
| Atmos | 2015 | | Zeus | |
| Banjori | 2013 | Multibanker<br>BankPatch | | |
| Bedep | 2014 | Rozena | | |
| Bolek | 2016 | Kbot | | |
| Buhtrap | 2014 | | | 2015 |
| Carberp | 2010 | | | 2013 |
| Carbanak | 2015 | Anunak | Carberp | |
| Chthonic | 2014 | | Zeus | |
| Citadel | 2012 | | Zeus | |
| Corebot | 2015 | | | |
| Cridex | 2011 | Feodo Bugat | | |
| Dridex | 2014 | Bugat Geodo | Cridex | |
| Dyre | 2014 | Dyreza<br>Dyzap<br>Dyranges | | |
| Fobber | 2015 | | Tinba | |
| Gameover<br>Zeus | 2011 | Zeus P2P | Zeus | |
| Gozi | 2007 | Snifula<br>Ursnif | | 2010 |
| GozNym | 2016 | | Gozi<br>Nymaim | |
| ICE IX | 2011 | | Zeus | |
| ISFB | 2015 | Ursnif<br>Papras | Gozi | 2015 |
| Kins | 2011 | VM Zeus<br>Zberp Zeus | | |

| | | | | |
|---|---|---|---|---|
| Kronos | 2014 | | | |
| Panda | 2016 | | Zeus | |
| Qadars | 2013 | | | |
| Qakbot | 2009 | Qbot | | |
| Ramnit | 2010 | | | |
| Ranbyus | 2012 | | | |
| Rovnix | 2011 | | Carberp | |
| Shiz | 2006 | Shifu | | |
| Tinba | 2012 | Tinybanker<br>Zusy | | 2014 |
| Trickbot | 2016 | TrickLoader | | |
| Urlzone | 2009 | Bebloh<br>Shiotob | | |
| Vawtrak | 2013 | Neverquest | Gozi | |
| Zeus | 2006 | Zbot | | 2011 |

**TABLE II:** MAEC capabilities and the behaviours used in this paper

| **MAEC Capability** | **Behaviour Name** |
|---|---|
| Command and Control | Configuration |
| Remote Machine manipulation | |
| Privilege escalation | |
| Data theft | Info Stealing, Injection |
| Spying | Screenshot, Video Capture |
| Secondary Operation | |
| Anti-detection | Anti-Analysis |
| Anti-code analysis | Anti-Analysis |
| Infection/Propagation | |
| Anti-behavioural analysis | Anti Analysis |
| Integrity violation | Process Injection |
| Data Exfiltration | Network Communications |
| Probing | |
| Anti-removal | Persistence |
| Security degradation | Info Stealing, Injection |
| Availability violation | |
| Destruction | |
| Fraud | Configuration, Info Stealing, Injection |
| Persistence | Persistence |
| Machine access/control | Backconnect, Network Communications |

**TABLE III:** Malware Process Iteration Methods

| **Malware** | **API Sequence Present** | **Detected** |
|---|---|---|
| Zeus | Yes | - |
| Citadel | Yes | Yes |
| Vawtrak | Yes | - |
| Dyre | Yes | Yes |

**TABLE IV:** Malware Process Injection Methods

| **Malware** | **API Sequence Present** | **Detected** |
|---|---|---|
| Zeus | Yes | Yes |
| Citadel | Yes | Yes |
| Vawtrak bit | Yes | - |
| Dyre | - | - |

**TABLE V:** Sample Hashes

| **Malware** | **MD5 Hash** |
|---|---|
| Zeus | 306dd8c10a19c5998e88c7b1de520e2f |
| Citadel | 24547d8e6028b77a5a62b3babc9264ad |
| Vawtrak 32bit | 19f8bc63e882fbe7affccd814602638b |
| Dyre | cc0d5b95b8b30f99c1092b87c869c74c |