Feature-Distributed Malware Attack: Risk and Defence

Byungho Min and Vijay Varadharajan

Advanced Cyber Security Research Centre
Department of Computing, Macquarie University, Sydney, Australia
{byungho.min,vijay.varadharajan}@mq.edu.au

Abstract. Modern computing platforms have progressed to more secure environments with various defensive techniques such as application-based permission and application whitelisting. In addition, anti-virus solutions are improving their detection techniques, especially based on behavioural properties. To overcome these hurdles, the adversary has been developing malware techniques including the use of legitimate digital certificates; hence it is important to explore possible offensive techniques in a security-improved environment.

In this paper, first we propose the new technique of *feature-distributed* malware that dynamically distributes its features to multiple software components in order to bypass various security mechanisms such as application whitelisting and anti-virus' behavioural detection. To evaluate our approach, we have implemented a tool that automatically generates such malware instances, and have performed a series of experiments showing the risks of such advanced malware. We also suggest an effective defence mechanism. It prevents loading of malicious components by utilising digital certificates of software components. We have implemented a Windows service that provides our defence mechanism, and evaluated it against the proposed malware. Another useful characteristic of our defence is that it is capable of blocking general abuse of legitimate digital certificates with dynamic software component loading.

Keywords: Security, Feature-Distribution Malware, Software Component.

1 Introduction

Modern computing platforms have progressed to more secure environments with various defensive techniques. For example, all the mainstream platforms such as Windows 8, Mac OS X 10.9, iOS and Android deploy an application-based permission model that determines which application is allowed to access which system services such as network activity, disk access, other hardware access and location service. Furthermore, on Mac OS X and iOS, each permission decision is made by the user at run time right after each permission is requested by the application, making them more secure than Android, where the list of all the

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part II, LNCS 8713, pp. 457–474, 2014.

requested permissions of an application is presented at installation time, and allowed at once when a user decides to install the application.[1] As a result of this progress, malware or compromised user applications on the modern platforms may not be able to achieve their goals. However, attackers have always developed new ways to overcome security hurdles. In recent years, there are at least two outstanding trends in malicious software. First, most malware instances, even mobile ones, have become modular [1, 2]. For instance, an initial module is responsible for installation, a networking module is in charge of command and control (C&C) communications, and a rootkit module starts other modules on every system boot so that they can perform respective functionalities. This strategy lowers the possibility of detection as well as minimises exposure of the malware and the attack operation in case of detection. Another trend in malicious software is the use of legitimate certificates [3–6] so malware is allowed to run and access system services, and not blocked or detected by security solutions like anti-virus. Therefore, new malware that bypasses the security restrictions imposed by the application-based permission model can emerge in the foreseeable future, and it is important to measure the risks of such malware attack and develop effective defence measures.

In this paper, first we present *feature-distributed* malware that can bypass not only the application-based security model, but also other security schemes such as application whitelisting and Egress filtering. Although many real world malware samples use various malicious component loading techniques [3–7], we take a different perspective on malicious dynamic component loading. Since no matter what technique is used, the most important objective for malware is to remain undetected and perform its malicious activities. And commodity PCs typically have several applications installed on them from web browser to media player, and many such applications use diverse (open or closed source) software components. We have categorised popular applications, such as Firefox and iTunes, and the libraries loaded by them, and then split malware functionalities to several components. Similar to other malicious component loading attacks [8], various attacks including remote ones are possible with our approach as described in Section 2. In addition, a more capable form of modular malware has been achieved due to the *dynamic feature distribution*. To evaluate the risks of our technique, we have implemented a tool that automatically generates feature-distributed malware using three malicious component types, and showed that wide range of malicious activities is possible, even under the application-based permission model and with the protection of security tools such as anti-virus, application whitelisting and Egress filtering.

Exploring the risks of the feature-distributed malware enabled us to devise *a novel defensive mechanism*. In essence, a host process (i.e. original main executable file of an application that loads other components) checks the signer information of components to be loaded when it verifies the validity of their digital signatures to prevent the feature-distributed malware. This mechanism

[1] This has made Android malware that requires several permissions to use social engineering techniques to acquire the permissions.

turned out to be effective in blocking most real world malware attacks that use stolen legitimate certificates. We have implemented this mechanism as a Windows service so that any application can integrate our APIs during component loading.

The remainder of this paper is structured as follows. Section 2 describes the background on malware features and the design of feature-distributed malware with its attack vectors. Implementation details of the malware are discussed in Section 3. Section 4 presents the evaluation of our offensive technique, including characteristics and risks of the malware attack. In Section 5, we present our general defensive technique to prevent the feature-distributed malware attack, and evaluate its effectiveness and performance impact. We conclude the paper with a discussion of future work in Section 6.

2 Feature-Distributed Malware

This section describes major features of modern malware, design of our feature-distributed malware, remote attack examples applied from other malicious component loading studies [8], and the feature distribution strategy.

2.1 Malware Features and Application-Based Permission

Extraction of modern malware characteristics should be conducted ahead of distributing features. For this purpose, we have surveyed technical reports on recent malware instances including Stuxnet, Duqu, Flame, Gauss, Shamoon, Red October and Careto [3–7, 9] and analysed several samples of them. Common features of them is summarised below, and full details of the analysis are available in our technical report [2]:

1. Local activity
   (a) Data collection: file list, files, OS account credentials, Bluetooth sniffing, microphone recording, hardware and software information, web credentials and browser cookies (via malicious browser plugin), instant messaging (IM, e.g. Skype) recording, keylogging, emails, screenshots, network shares, connected device list, information from connected devices (e.g. SMS and contacts), saved passwords from web browsers and FTP clients cyber assets (e.g. intellectual property)
   (b) Propagation: USB infection
   (c) Backdoor: Windows service installation or modification, account creation
   (d) Supporting: payload generation (e.g. autorun.inf for USB infection), initialisation (e.g. module listing and loading), filename generation (only these interesting files are collected), malware activity logging, security tool monitoring, uninstallation
2. Remote activity
   (a) Data collection: External IP address, network scanning

(b) Propagation: account login attempt (using created or stolen credentials), serving man-in-the-middle attack, serving remote exploits such as MS08-067
(c) Networking: several commands including update, run, and uninstall, data exfiltration, heart beating

As modern platforms such iOS, Android and Mac OS X 10.9 are incorporating application-based permission model, some of the above activities cannot be performed by standalone malware or infected software. For example, microphone recording and Bluetooth sniffing require access to respective hardware access, and contact information collection requires access to Contacts application, which has to be explicitly approved by the user. Considering that most of recent malware consists of multiple modules (at application, service, or driver level) and their data collection and networking modules are usually running at user-level[2], this is a new hurdle for attackers to overcome.

2.2 Feature-Distributed Malware: Concept

Although the application-based permission model is strong enough to prevent current malware threats, it is still possible to bypass it if malware is properly implemented because there are still user-approved applications that have access to hardware and software resources. In particular, malware can perform its functionalities by dynamically distributing them to user-approved or system-approved applications. For example, a networking module that has migrated into an email application may not be able to dump OS password hashes, while a local data collection module that has migrated into an anti-virus service running under SYSTEM account can. On the other hand, the data collection module may not be allowed to communicate with its C&C server due to Egress filtering, whereas the networking module can perform the data exfiltration. This is the concept of our *"feature-distributed"* malware; malware modules cooperate in order to overcome application-based permission model and other security mechanisms such as application whitelisting and Egress filtering. In addition, the following advantages are achieved with the feature-distributed malware:

1. *More Adaptive*: when a victim installs a new application, the feature-distributed malware can further distribute its feature set to the new application without any attacker intervention. In addition, it performs self-recovery as long as there is at least one active malicious component as explained in Section 3, even when an application is updated (and so are its components).
2. *More Stealthy*: trusted and approved application hosts malicious modules. No new process for Windows services or applications are installed and executed; on the contrary, modern malware usually installs its modules as Windows services or drivers. Furthermore, malware that is detected by anti-virus becomes undetectable by distributing malicious features to multiple

[2] Kernel-level rootkit drivers are normally backdoors for persistent compromise.

applications (i.e. processes) as shown in Section 4. This is because anti-virus solutions determine maliciousness of a file or a process based on its static and dynamic characteristics; analysing all the interactions of multiple processes and components is yet to come since it can result in a serious performance issue and raise false positive rate.

3. *More Capable*: bypassing security mechanisms such as application whitelisting and Egress filtering is achieved.

2.3 Remote Attacks – Examples

Although any malware attack from drive-by download to USB infection is possible with the feature-distributed malware, we describe the attack vectors that are closely related to the concept of our proposed malware attack. These vectors assume that the malware has been delivered to a victim by an initial attack such as client-side attack or spear phishing, which is also assumed by most related work [10–18]. In addition, many real world malware attack operations including Careto, Red October, Gauss and Flame started as spear phishing.

Archive with component attack. The adversary can make an archive file that contains a normal document file and a malicious component file with an absolute path and overwrite option. Once the victim opens the archive, not only the files the victim expects to see, but also the malware components are extracted and placed in the designated path. For example, malicious `sqlite3.dll` can be extracted to the Apple iTunes' installation path. Next time iTunes starts up, the malware module is loaded by iTunes and performs its activities from feature distribution to malicious functionalities.

Carpet Bomb-based attack. The Carpet Bomb attack [19] can lead to remote code execution in connection with the feature-distributed malware. This attack happens, for instance, when Safari web browser accesses a malicious web page, and arbitrary files that can be the feature-distributed malware file are downloaded to the victim system without user consent. In particular, when the location of downloaded file corresponds to the malware's target software component, the malware will be loaded by its target application after the Carpet Bomb attack.

2.4 Feature Distribution Strategy

Since each application has different permission and privilege depending on its required functionalities, malware features discussed in Section 2.1 should be properly distributed. We categorise application types and discuss the pros and cons of each type as a malware module. An application can be included in more than one category. For example, Google Drive is a startup as well as a networking application. We have tested several of the below application types with the proposed malware concept, and have evaluated them in Section 4.

Startup (persistent). Most startup applications are automatically executed on every boot and they keep running. Examples include cloud storage like Google Drive and Dropbox, IM such as Skype, and anti-virus software. This type of

applications runs all the time while the system is up (unless the user explicitly terminates it). However, they usually have user privilege (neither administrative nor SYSTEM), and may not be allowed for networking.

- *Appropriate Module*: initialisation (self-recovery and feature re-distribution that are explained in Section 3)

Startup (update checker). Some startup applications are executed on every system boot, but terminated after they finish required tasks. Examples are update checker of various applications such as Java and Flash plugins. They can connect to the Internet, and may have administrative privilege so that they can update the relevant software. But they run for a limited time, typically several seconds.

- *Appropriate Module*: heart beating module that reports a successful compromise and access maintenance.

Networking. Web browsers, email, IM, (S)FTP and other network clients, online games, and cloud storage applications are allowed for network activity. But they normally have user privilege, and run for a limited time (longer than update checker).

- *Appropriate Module (web browsers)*: "worm" (propagation) module since C&C communication pattern that is used for worm detection is randomised by the user; only when the user surfs the Internet, worm module performs its communication, and each individual person has different Internet use pattern.

- *Appropriate Module (the others)*: data exfiltration because (1) IM and email clients tend to run all the time, and (2) network clients can transfer a large amount of data compared to other networking software.

Security Solutions. Security tools such as anti-virus usually have the highest privilege (i.e. SYSTEM on Windows), and cannot be killed by the user and user level applications. However, it is much harder to load a malicious module in the context of these applications as they protect themselves from malware so that even the user cannot modify their components or configurations. In addition, some executables like local file scanner may not be allowed for networking.

- *Appropriate Module*: local data collection, logging malware activities, and rootkit module that maintains access, updates modules, and uninstalls all the modules when required.

Productivity. Productivity applications are varied from office suites to personal information manager such as password manger 1Password. Even though these applications are executed under user privilege, and do not run all the time, they are guaranteed to have access to relevant information including office documents to password database.

- *Appropriate Module*: local data collection module specialised for each application.

Media Management and Playback. Similar to productivity applications, this type of applications can access to their contents, which include personal photos, videos and voice memos.

- *Appropriate Module*: local data collection module specialised for each application.

Table 1. Open Source Libraries Tested with Feature-Distributed Malware

Library Name	Example of Applications
SQLite	AVG, iTunes, Adobe Reader, Google Drive
OpenSSL	Open Office, Mobogenie, Mumble
Network Security Services (NSS)	Chrome, AIM, Pidgin, Firefox, Thunderbird

Device management. iTunes and Mobogenie are two device management applications respectively for iOS and Android device. These applications have access to device information and contents such as contacts and photos.

- *Appropriate module*: data collection module specialised for each application.

3 Implementation

To realise our proposed feature-distributed malware, we have developed an automatic malware generator on Windows, which is based on our implementations of (1) three malicious component types and (2) malware features such as malware initialisation, feature distribution, and common malicious activities. In this section, we discuss implementation details and considerations of the generator and other techniques.

3.1 Three Malicious Component Types

As discussed in Section 2, a feature-distributed malware file (i.e. a software component or an archive) is placed at a certain path after it is delivered to a victim (e.g. via email attachment or network share) and opened. Then the file is loaded by target applications such as Firefox, which means that the newly placed malicious component has to provide all the functionalities of the original software component. On Windows, exported functions of DLL are the functionalities the malware has to provide. We have implemented three types of file-based malicious component that satisfies this requirement.

Source code modified component. One of the best ways of providing an entire functionality of a software component is to build them from source code. Many popular applications use open source libraries as given in Table 1, and this makes it feasible to implement malicious components using open source libraries. Open source libraries are also an attractive target for feature-distributed malware because (1) there is no need for trampolines (described below), thus no additional file is produced, (2) several popular open source libraries are used by many common applications in various application types, and (3) building a multi-platform malicious component is possible for many open source libraries. For example, Careto [3] is a multi-platform malware that uses an open source software for Mac OS X backdoor module. However, building an entire library can increase the size of malware, and it takes longer than making a dummy trampoline-style component. We have tested the proposed malware attack with three widely used

libraries: SQLite, Network Security Services (NSS), and OpenSSL. They are being adopted by a huge number of applications, and a few representative examples are shown in Table 1. We edited the source code of these libraries so that they can be passed to the malware generator as an input and be merged with our malware feature implementation (Figure 1).

Trampoline-style Component. Software vendors or individual developers may get an open source library, and then edit some of its exported functions. Also, there are still various popular closed source libraries such as Microsoft C Runtime (CRT) library and Microsoft Foundation Class (MFC). In these cases, we have taken two approaches, and trampoline-style dummy library is one of them. Trampoline is a dummy function that finds its original export using its ordinal value, and jumps to it when it is invoked. Such trampolines must be implemented for all the exports of a target library so that the hosting application starts and functions correctly. We have implemented trampolines mainly in assembly code, and tested this approach with CRT and MFC libraries.

Even though the size of trampoline-style component is smaller than that of open source-based one, it cannot replace the original library, since it does not provide any actual functionality. In the current implementation, the feature-distributed malware searches for CRT libraries, and if required, renames (not replaces) them and copies the trampoline-style dummy library in the place of the original one. As a result, there comes an additional file introduced to the victim system when this type of component is used. Also, it cannot be used as an initial component because renaming can happen only when at least one malicious component is active.

Binary modified component. Another way of developing a malware component based on source-modified or closed source libraries is to manipulate them at the binary level. We have implemented an on the fly binary modification routine in our malware feature set that performs the following:

1. Add a new Portable Executable (PE) section.
2. Write binary shellcode that is built from our assembly and C code.
3. Modify Entry Point (EP) so that our shellcode is executed when the library is loaded.

When the modified component is loaded by an application, and the shellcode is executed, it loads its modifier library then jumps back to the original entry point (OEP) so that it can function as intended. For instance, if SQLite of iTunes modifies the CRT of Firefox, this CRT's shellcode loads the SQLite library of iTunes for malware functionalities. Because the newly loaded SQLite is loaded by a component of Firefox, it has the context of Firefox, not of iTunes. This enables the implementation of our shellcode to be concise and reliable, with the size of 408 bytes. We verified this approach with the two libraries tested for trampoline-style. Its limitation as a malicious component is similar to trampoline-style; it cannot be used as an initial module since it is an implemented feature and used in the malware, which should be used by an active malicious component. However, no additional file is introduced in the victim system since this approach modifies binaries on the fly.

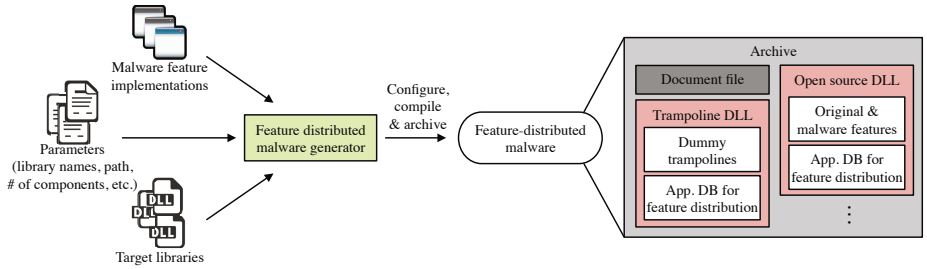

Fig. 1. Feature-distributed malware generator for archive with component attack

Anti-virus (AVG's case). As discussed in Section 2, major anti-virus solutions protect themselves, and AVG is not an exception. AVG anti-virus provides "AVG Self protection" that is composed of two filter drivers. When a user or a process (even with the highest privilege) tries to delete or modify those files, then the two drivers block such I/O requests. Similarly, new files cannot be added to the AVG folder. Therefore, it is much harder to replace or modify AVG's files [20]. After analysing AVG anti-virus, however, we found out that AVG fails to protect the self-protection filter drivers, even though it must block any unauthorised attempt to unload or detach them. As a result, AVG's two filter drivers can be detached by the user or malware using Filter Manager Control. After the detachment, arbitrary files belonging to AVG can be replaced or modified, which can be used in the feature-distributed malware.

3.2 Automatic Malware Generation

The final form of the feature-distributed malware is one single archive or software component that will be placed at a target path and then loaded by user applications. We have implemented a malware generator that automates the process of configuring, packaging and making the final component or archive file as depicted in Figure 1. The generator takes three inputs. First, user specifies several parameters such as libraries to be included, application database, final form to be generated (either an archive or a DLL file), absolute paths where the generated component is to be placed, and other malware parameters given in Table 3. Second, implemented malware features such as local password hash dump, C&C communication and the live binary modification routine are passed. These are embedded to the specified open source libraries at source code level. Lastly, source code of open source target libraries such as SQLite and OpenSSL are passed so that the generator can merge the malware features with the libraries. When merging them, the generator adds the malware features as a separate thread created in `DllMain()` so that it can run as long as the library is loaded in memory. When a trampoline-style library such as CRT is specified in the user input, relevant dummy trampoline DLL is generated and added to the final archive file with a path to Windows Temporary folder. The final archive is constructed with proper absolute paths and overwrite option using RAR or

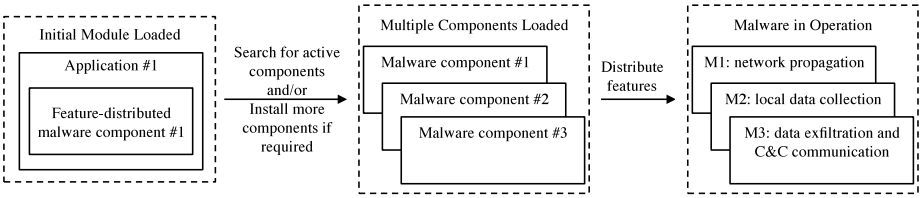

Fig. 2. Overview: initialisation and operation of the feature-distributed malware

TAR format according to the user parameter. This is possible because files of applications are usually not protected by the system, unlike files of security tools or critical system files that are protected by operating system feature such as Windows Resource Protection (WRP).

Among the input parameters, application database contains information about target applications including their categories, allowed permissions and privileges. This information is embedded in the malware so that it can initialise and dynamically distribute its features based on this information. Real world attackers behind advanced threat persistent operations usually profile their target systems, and then carefully install persistent malware for cyber espionage. Therefore, the adversary can customise application database based on initial profiling.

3.3 Malware Operation: Initialisation and Feature Distribution

Once a feature-distributed malware (i.e. a software component) is loaded by an attack vector described in Section 2, the initialisation process is started as shown in Figure 2. The component searches for other active components; remote procedure call (RPC) is used for inter-component communications. Depending on the number of active components and the configuration specified by the attacker, the first component searches for additional target libraries and replaces them with itself. For instance, a malicious `sqlite3.dll` loaded by iTunes searches for other instances of SQLite library. Preferences on libraries are specified in the malware parameter. When other instances of SQLite used by Open Office and AVG are found, the malware replaces the newly found components with itself so that they become a part of the feature-distributed malware. In the case of binary modification, target library is modified on the fly. After sufficient numbers of components become active, they dynamically distribute malware features based on their hosting process and the application database. In this example, Thunderbird component takes the networking role, Open Office module collects documents and other data, and AVG records keystrokes and logs malware activities. From this point on, the malware components perform their respective features.

When replacing a file that is currently open, the malware first renames the target, and then copies itself to the path because renaming is allowed for locked files on Windows. The renamed original file is deleted when the malicious DLL is

Table 2. Applications installed on the evaluation system

Application Name	Application Type	Component Name
Thunderbird	Networking (email client)	`mozsqlite3.dll`
Firefox	Networking (web browser)	`msvcr100.dll` (CRT)
Adobe Reader	Productivity	`sqlite3.dll`
Open Office	Productivity	`ssleay32.dll` (OpenSSL)
iTunes	Media player, iOS device manager	`sqlite3.dll`
Mobogenie	Startup, Android device manager	`ssleay32.dll` (OpenSSL)
AVG Free Anti-virus	Startup, Security	`avgntsqlitex.dll` (SQLite)

loaded. In the case of anti-virus software, nullification of self-protection feature is preceded the file replacement as explained in this section. In addition, if there are multiple candidates found after the initial search, the malware selects target components according to the following rule:

1. Search for open source libraries (no additional file resulted).
2. Search for trampoline-style target components if the number of malware components is less than user-specified value, rename the original files and copies the dummy trampoline file to the target paths located at Windows Temporary folder (additional file resulted).
3. Search for binary modification targets and modify them if the number of components is still less than three (live DLL modification performed, no additional file resulted).

Collected information is stored in the temporary folder and sent back to the attacker by the networking module such as Thunderbird's SQLite. Lastly, supporting features (Section 2.1) such as basic C&C and uninstallation function have also been implemented.

4 Evaluation

In this section, we first describe how the feature-distributed malware performs its activities, thus fulfilling usual malware requirements. Then we show how the malware can bypass security mechanisms such as application whitelisting, Egress filtering and anti-virus' behavioural detection. In the evaluations, Windows 7 Ultimate edition was used as the target system, and the applications shown in Table 2 were installed on the system.[3] These selected applications use at least one open source libraries including SQLite, NSS and OpenSSL. And the malware instances evaluated in this section were generated by our tool with the parameter values shown in Table 3.

[3] Some applications use more than one target libraries, even though only actual target library is specified. For example, Thunderbird loads NSS in addition to SQLite as shown in Figure 3.

Table 3. Common parameters for feature-distributed malware generation

Parameter	Value
Libraries used	SQLite (Section 4.1)
	SQLite, OpenSSL (Section 4.1)
	SQLite, OpenSSL, CRT (Section 4.4)
Library priority	SQLite > OpenSSL > NSS
Final format	RAR archive
Bait file	Annual report.docx
Path #1 (SQLite)	iTunes folder
Path #2 (OpenSSL)	Open Office folder
Path #3 (CRT)	Windows Temporary folder
Min. # of components	3 (varied from 1 to 4 in Section 4.4)
Dummy trampoline	Yes
Live binary modification	No
Attack AVG (when possible)	No (Yes/No in Section 4.4)
Application database	Application category, privilege, etc.

4.1 Malware Operation

This section describes two attacks that show fundamental operations of the proposed feature-distributed malware. Both attacks are archived with component attacks described in Section 2.3.

Attack with SQLite library. A RAR self-extracting (SFX) file delivered to a victim contains a normal document and a malicious SQLite library. When the victim opens the file, SQLite is extracted to iTunes folder and overwrites the existing one as specified in the RAR. Next time iTunes is launched, the replaced SQLite is loaded and the initialisation process begins as follows:

1. Collect basic system information (OS version, installed applications, etc.), and send when possible. iTunes uses network from time to time, so the iTunes component informs the attacker about this first compromise. However, other network activities such as file transfer are performed by networking component described below.
2. Start RPC and search for other active components.
3. After a predefined time (1 minute in the current implementation), check the number of active components. As the number is less than configured one (3), the SQLite component starts searching for other instances of SQLite.
4. When SQLite files of Thunderbird and AVG are found, and identified as not-infected, iTunes component replaces those instances with itself. In the case of AVG, disarming AVG is preceded as explained in Section 3.1. If the SQLite files found and are identified as infected, then the iTunes component waits until they become active, i.e. their hosting applications are launched.

After the three components become active, they split malware features according to their hosting applications (i.e. iTunes, Thunderbird, and AVG), and start performing their respective activities as follows:

1. iTunes: local data collection to Windows Temporary folder (files on the system including voice memos imported from iOS devices)
2. Thunderbird: network activity for C&C communication (e.g. archive and send collected data to the attacker)
3. AVG: local data collection to Windows Temporary folder (keystrokes & screenshots)

Attack with Multiple Libraries. This attack evaluates the feature-distributed malware in a more complex form. SQLite and OpenSSL are included in a RAR SFX archive with the normal document file, and extracted to relevant paths as specified in the parameter.[4] Initialisation process is similar to the above case, except that now there are two libraries and so preference comparison is performed when the two components become active. Because SQLite has a higher priority, iTunes component searches for an additional instance, and replaces Thunderbird's SQLite. If it cannot find any SQLite instance, OpenSSL of Open Office looks throughout the system for other instances of OpenSSL. Malware activities are logged and transferred back to the adversary so that it can adjust malware parameters. For instance, if the initial component(s) could not find enough number of target components, then the attacker can reduce this parameter value based on the collected system information.

4.2 Bypassing Application Whitelisting

Application whitelisting is more and more deployed, especially on specific-purpose systems such as SCADA and POS that do not need to be general purpose machines. In particular, ENISA recommends the use of whitelisting solutions, which restrict the execution of non-approved software and code [21].

We used AppLocker for application whitelisting. It is a security tool built in Windows 7 that allows only the configured applications to run on the system. Because (1) the feature-distributed malware consists of software components (not applications), and (2) all of its hosting applications (iTunes, Thunderbird, and Open Office in this example) are allowed in the configuration of AppLocker, we could verify each component performed respective roles without being blocked. This may look trivial since the malware compromised whitelisted applications from the start, but it apparently shows its importance, considering most modern malware instances including most advanced ones such as Stuxnet, Gauss and Careto have their own processes, which are blocked by application whitelisting mechanisms.

From this evaluation, we can infer how the feature-distributed malware can bypass application-based permission model as well. Application whitelisting can be thought as a more strict form of application-based permission model, which

[4] We have experimented another way of building the initial component such as SQLite so it contains other libraries such as OpenSSL and NSS in its PE sections. This reduces the number of files included in the archive file, but it increases the size of the initial component.

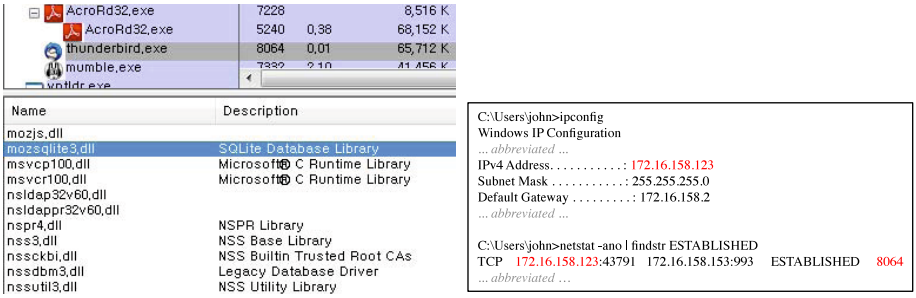

Fig. 3. Process ID (8064) of Thunderbird loading the malicious SQLite (left) and successful network connection by SQLite component under the Egress filtering environment (right)

allows only whitelisted applications to run on the system. Therefore, bypassing application-based permission policy can be done in a similar way.

4.3 Bypassing Egress Filtering

Egress filtering is increasingly being used and/or needed to block malware communications. It is required for compliance with Payment Card Industry Data Security Standard (PCI-DSS), Section 1.2.1 and 1.3.5 of version 2.0 [22]. Also, Egress filtering is recommended by US-CERT as a malware mitigation mechanism [23]. However, even under this network activity restriction, this evaluation shows that the feature-distributed malware bypasses Egress filtering rules, and connects back to the attacker.

Thunderbird is an email client, and hence it must be allowed in outgoing traffic rules. In this example, it tries a connection to the attacker since it is responsible for networking. We verified that the malicious SQLite component of Thunderbird bypassed Egress filtering firewall rules forced by Comodo Firewall as shown in Figure 3. Thunderbird (`thunderbird.exe`, process ID: 8064) is loading the malware component (`mozsqlite3.dll`) in Figure 3 (lefthand-side). And its process (8064) is making a network connection from the victim system (172.16.158.123) to the attacker (172.16.158.153) via the IMAP email protocol port (993) (righthand-side of Figure 3).

4.4 Bypassing Anti-virus

In the anti-virus bypassing evaluations discussed below, what we have tested is bypassing *behavioural* detection, not signature-based detection, because (1) our evaluation goal is to show the advantage of feature distribution, and (2) making a binary that passes signature-based detection is relatively easier to achieve using various code obfuscation tools.

Feature distribution and detectability. In this first evaluation, we first built a single component malware that passes AVG's signature-based detec-

Table 4. Anti-virus detection test

Detection	Application compromised (# of comp.)	Library used
Yes	iTunes (1)	SQLite
No	iTunes, AVG (2)	SQLite
No	iTunes, Firefox (2)	SQLite, CRT
No	iTunes, AVG, Adobe Reader (3)	SQLite
No	iTunes, AVG, Firefox (3)	SQLite, CRT
No	iTunes, AVG, Thunderbird (3)	SQLite
No	iTunes, Thunderbird, Open Office (3)	SQLite, OpenSSL
No	iTunes, Adobe Reader, Firefox (3)	SQLite, CRT
No	iTunes, Firefox, Mobogenie (3)	SQLite, OpenSSL, CRT
No	iTunes, AVG, Adobe Reader, Thunderbird (4)	SQLite
No	iTunes, AVG, Adobe Reader, Firefox (4)	SQLite, CRT
No	iTunes, Adobe Reader, Firefox, Thunderbird (4)	SQLite, CRT
No	iTunes, Adobe Reader, Open Office, Mobogenie (4)	SQLite, OpenSSL
No	iTunes, Firefox, Open Office, Thunderbird (4)	SQLite, OpenSSL, CRT

Table 5. Anti-virus (2014 versions) bypassed by the feature-distributed malware

Anti-virus name
AVAST Free, Pro, Internet Security
Avira Free, Commercial, Internet Security
Bitdefender Internet Security, Antivirus Plus
Kaspersky Anti-Virus, Internet Security
McAfee Antivirus Plus, Internet Security, Total Protection
NOD32 (ESET) Antivirus, Smart Security
Norton (Symantec) 360, Internet Security, AntiVirus

tion, but is detected by AVG's behavioural engine as a "Threat: General behavioural detection" when it starts its operation. Then we tested multiple malware instances that are functionally identical but have multiple components. As summarised in Table 4, none of the feature-distributed malware instances was detected during their operation once features are distributed. This is because anti-virus solutions determine behavioural maliciousness of a process based on the behaviours of this particular process, and each process hosting the feature-distributed malware performs only a portion of malware features.

Other anti-virus solutions. We have tested the feature-distributed malware's operation against several anti-virus solutions. The malware instance used for this experiment has four components, namely iTunes, Firefox, Open Office and Thunderbird (the last one in Table 4). Eighteen (18) solutions of seven antivirus vendors have been tested (AVG is excluded as it has been already discussed above), and Table 5 shows our feature-distributed malware performed its malicious activities without being detected by behavioural and signature-based engines of the tested anti-virus solutions.

4.5 Limitations

C&C communication of the current implementation is based on sbd that is a Netcat-clone, designed to be portable and offer strong encryption. Therefore, bypassing Egress filtering can be impossible if deep packet inspection is deployed on the host or the target network infrastructure. However, it can be easily overcome by serious attackers, since it is well-known that encapsulating any communication protocol inside another one such as actual HTTP or IMAP.

The implementation can be detected by file integrity monitoring solution. However, the probability is lower than most modern malware instances like Careto and Stuxnet. This security solution's major purpose is monitoring system's critical files and configurations or confidential folders [24]. In other words, application binaries are not major objects of monitoring. Moreover, some applications are frequently updated (e.g. three times in six weeks in the case of Firefox), thus not adequate for real-time integrity monitoring. As a result, these folders or files are often excluded from such integrity monitoring [25]. On the contrary, many of real world malware instances change critical system configurations for persistent compromise (e.g. installing a new service or driver with additional files or modifying existing service's configuration).

5 Defence against Feature Distributed Malware

This section proposes a new mechanism to prevent the feature-distributed malware attacks. We have implemented it as a Windows service with public APIs so that any application can use it. Thunderbird was used for the evaluation of its effectiveness and performance.

5.1 Proposed Defence Mechanism

Our suggestion is that an application should check the validity of the digital signature and *signer* information of a library before loading and using it, which is very easy to adopt from vendors' perspective. Typically, such verification can be performed prior to calling `LoadLibrary()` API that loads a library in the context of an application. Then, even though the malicious component has the full functionality of the original library, it cannot be loaded because it fails to pass the origin verification. Similarly, verifying both the digital signature and the signer can help most modern malware attacks that use stolen legitimate certificates. Nowadays, they are bypassing many security tools' detection and restriction because they have a *valid* digital signature. However, if software vendors develop their application according to our suggestion, malware instances signed with a stolen key cannot be loaded since the signer is different from the vendor's. In order to demonstrate our concept, we have implemented a Windows service rather than implementing it into individual applications. On the application side, it calls our service API before any invoke of `LoadLibrary()`. Then, the service extracts signer information from the digital signatures of the application binary and the library file to be loaded, and compares it. Only when the

digital signature of the library is valid and *it is signed by the same entity as the application*, the service returns true. Although we have not added error handling code to each application, several recovery processes from simply excluding the modified library (if the application can still work with limited functionalities) to updating the modified component with a clean one are possible.

5.2 Evaluation of the Proposed Defence

Among the applications targeted in the attack evaluation (Section 4), we have selected Thunderbird for the evaluation of our defence mechanism. Code that uses the service API was added to both applications. Then the same attack scenario was launched on the victim system. The SQLite library file was successfully replaced by the archive file, but it was never loaded by Thunderbird since the replaced SQLite library is not signed by Mozilla Corporation who signed the Thunderbird executable file.

We have also conducted performance evaluations using Thunderbird. Even though Thunderbird loads more than 20 libraries, the application startup time delay was under $20ms$ on a Windows 7 machine with Intel Core i7 2.4 GHz CPU and 1GB of RAM. This result shows that our defence mechanism is practical and can be effectively applied to real world applications such as Thunderbird.

6 Concluding Remarks

In this paper, we have proposed a new advanced malware that distributes its features to multiple software components in order to bypass various security policies such as application whitelisting and security tools like anti-virus. A tool that automatically generates such malware has been developed, and malware instances generated by this tool have been evaluated, showing the risks of the proposed malware. We have also suggested an effective defence mechanism that utilises digital signatures of component files to prevent loading of malicious components. To evaluate the proposed solution, a Windows service has been implemented and tested.

Although we have focused on the development and evaluation of feature-distributed malware on Windows, the underlying principle is general and can also be applied to other client platforms including Mac OS X and Android because component-based software engineering is widely used on these platforms. We are especially interested in exploring the application of our feature-distributed malware on mobile platforms. The number of mobile malware has been surging [1], and application-based permission model is already prevalent on modern mobile platforms such as Android, where it is important to prepare appropriate defences against the emerging threat.

References

1. Zhou, Y., Jiang, X.: Dissecting android malware: Characterization and evolution. In: IEEE S&P, San Francisco, CA, USA (2012)
2. Min, B., Varadharajan, V.: Deep analysis on recent malware incidents. Technical report (2012)

3. Kaspersky Lab: Unveiling "Careto" - The Masked APT. Technical report (February 2014)
4. Kaspersky Lab: Gauss: Abnormal Distribution. Technical report (August 2012)
5. Anity Labs: Analysis Report on Flame Worm Samples. Technical report (July 2012)
6. Falliere, N., Murchu, L.O., Chien, E.: W32.Stuxnet dossier. Technical report (2011)
7. Chien, E., Murchu, L.O., Falliere, N.: W32.Duqu The precursor to the next Stuxnet. Technical report (November 2011)
8. Kwon, T., Su, Z.: Automatic detection of unsafe component loadings. In: ISSTA, Trento, Italy (2010)
9. Tarakanov, D.: Shamoon the Wiper in details (August 2012), http://www.securelist.com/en/blog/208193795/Shamoon_the_Wiper_in_details
10. Murad, K., Shirazi, S.N.-u.-H., Zikria, Y.B., Ikram, N.: Evading Virus Detection Using Code Obfuscation. In: Kim, T.-h., Lee, Y.-h., Kang, B.-H., Ślęzak, D. (eds.) FGIT 2010. LNCS, vol. 6485, pp. 394–401. Springer, Heidelberg (2010)
11. O'Kane, P., Sezer, S., McLaughlin, K.: Obfuscation: The Hidden Malware. IEEE Security & Privacy 9(5), 41–47 (2011)
12. Rad, B.B., Masrom, M., Ibrahim, S.: Camouflage in Malware: from Encryption to Metamorphism. International Journal of Computer Science and Network Security 12(8), 74–83 (2012)
13. Oberheide, J., Bailey, M., Jahanian, F.: PolyPack: an automated online packing service for optimal antivirus evasion. In: Proceedings of the 3rd USENIX Workshop on offensive technologies, Montreal, Canada (2009)
14. Alvarez, S., Zoller, T.: The Death of AV Defense in Depth? - revisiting Anti-Virus Software. In: CanSecWest, Vancouver, B.C., Canada (2008)
15. Alvarez, S.: Antivirus (In) Security. In: CCC (Chaos Communication Camp), Finowfurt, Germany (2007)
16. Jana, S., Shmatikov, V.: Abusing File Processing in Malware Detectors for Fun and Profit. In: IEEE Symposium on Security and Privacy (S&P) 2012, San Francisco, CA, USA, pp. 80–94 (2012)
17. Porst, S.: How to really obfuscate your PDF malware. In: ReCon, Montreal, Canada (July 2010)
18. Bilge, L., Dumitras, T.: Before we knew it: an empirical study of zero-day attacks in the real world. In: CCS 2012, Raleigh, NC, USA (October 2012)
19. Apple: About the security content of Safari 3.1.2 for Windows (April 2012), http://support.apple.com/kb/HT2092
20. Min, B., Varadharajan, V., Tupakula, U.K., Hitchens, M.: Antivirus security: naked during updates. Software: Practice and Experience (April 2013) (accepted)
21. ENISA: Appropriate security measures for smart grids. Technical report (December 2012)
22. PCI Security Standards Council: Payment Card Industry (PCI) Data Security Standard. Technical report (October 2010)
23. US-CERT: Malware Threats and Mitigation Strategies. Technical report (May 2005)
24. Tripwire: Assure system integrity, best of breed file integrity monitoring (2014), http://www.tripwire.com/it-security-software/scm/file-integrity-monitoring
25. Arnold, M.: Tripwire Policy (May 2010), http://www.razorsedge.org/~mike/docs/tripwire.html