ADITYA K SOOD AKA
0KN0CK

Breaking in Add-on Malwares

Difficulty

■■□

This paper covers the working functionality of Malware Add-ons. The add-ons are called Application Extension programs that enhance the functionality of a program. The web browsers use a number of Add-ons as browser helper objects. The transformations in technology have increased the incidence of Malwares.

Malwares perform rogue functioning by keeping the identity intact with systems. No doubt the front end remains the same but the working strategy is different. This paper deals specifically with Malware application extensions and its deleterious impacts on the system. Internet Explorer case study will be undertaken to dissect the internal structure of Add-ons. The practical techniques will be cited to understand the malwares effectively.

Structural View of Malware Add-ons

Add-ons are considered as tiny software components that are supplementary to web browsers or are enhancements to the previous installed software The web browser loads the Add-ons as working elements and produces information based on that extension. The main aim is to enhance the flexibility of working components to ensure versatility. Nevertheless, they may cause application instability by impacting application software (crashing of software). Thus they marginalize the security and contribute to malfunctioning of reliable software. For example Internet Explorer crashes when a rogue Malware is added as an extension in the form of *Dynamic Link Library.*

The IE fails to render the malware contents in a possible manner leading to its crash. This is due to either buffer overflow or pointers mismatch. Protection mechanisms have been introduced to prevent the issue application exploitation. The infection surface of an application is too wide and random. So it is impossible to predict the affect on system or application state. It is also hard to control the behavior of an infection element. The Add-ons work on the defined benchmarks against which they are designed. Attackers are well versed in designing Malware Add-ons for exploiting resources. Some of the definitive reasons have been illustrated below:

- Most of the web browsers allow code execution without any warning or invalidation checks which is a matter of security breach. Rogue extensions properly exploit this inherited behavior of software's application. Operating system shows warning when a Malware Add-on tries to access the system DLLS or EXES for manipulative functioning. But a single vulnerability or flaw can lead to system compromise. Attackers are much aware of these weaknesses so they design application extensions with insecure code. It is done to generate exceptions that can not be handled by exception handlers of the system. This results in system instability.
- It has been noticed that the application extension code for browsers is mainly in JavaScript or an ingrained DLL. Most of the Dynamic Link Libraries are added as Add-ons. The DLL call specific modules for relative

WHAT YOU WILL LEARN...

The user will understand the working and cause of Internet Explorer based Malware Addons. The paper also provides the semantics of malware handling.

The user will learn more about breaking in procedure for analyzing Malwares for understanding the hidden functioning.

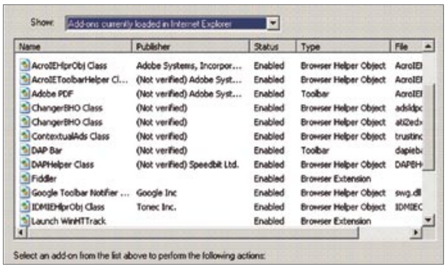
The causes of Application Crash with the Live demonstrated example of Internet Explorer.

WHAT YOU SHOULD KNOW...

A reader should be accompanied with the peripheral knowledge of Add-ons in the Internet Explorer.

Optimum knowledge of application design and extensibility.

A brief knowledge of error generation and handling mechanism.



Show: Add-ons currently loaded in Internet Explorer

Name	Publisher	Status	Type	File
AcroIEHlprObj Class	Adobe Systems, Incorpor...	Enabled	Browser Helper Object	AcroIEI
AcroIEToolbarHelper Cl...	(Not verified) Adobe Syst...	Enabled	Browser Helper Object	AcroIEI
Adobe PDF	(Not verified) Adobe Syst...	Enabled	Toolbar	AcroIEI
ChangerBHO Class	(Not verified)	Enabled	Browser Helper Object	adsldpc
ChangerBHO Class	(Not verified)	Enabled	Browser Helper Object	ati2ed
ContextualAds Class	(Not verified)	Enabled	Browser Helper Object	trustinc
DAP Bar	(Not verified)	Enabled	Toolbar	dapieb
DAPHelper Class	(Not verified) Speedbit Ltd.	Enabled	Browser Helper Object	DAPBH
Fiddler		Enabled	Browser Extension	
Google Toolbar Notifier ...	Google Inc	Enabled	Browser Helper Object	swg.dll
IDMIEHlprObj Class	Tonec Inc.	Enabled	Browser Helper Object	IDMIEC
Launch WinHTTrack		Enabled	Browser Extension	

Select an add-on from the list above to perform the following actions:

Figure 1. *Various Add-ons loaded in Internet Explorer*

functioning. The modules are designed for performing operations that affect the system. If instability occurs through web browser, the system is going to be affected in one or the other way. Internet Explorer encounters a lot of problem and is exploited many times. Some Add-ons create ActiveX Objects for performing registry related operations to temper the operating system directly. The ActiveX Objects provide direct interface with application based software. One can access Registry structure, Local File System and can perform Command Execution.

- The Add-ons can perform remote functioning too. In these cases the DLL are designed to load some Malware content in the form of XML from Malware based websites. When the content is downloaded by the browser through GET requests, the browser finds it very hard to render the XML content normally. Due to this default exception, handle is not able to handle that intrinsic error. Hence the browser crashes and vulnerable element is undertaken.
- The users show vulnerable mind set too. Rogue Add-ons are added to web browsers without verification and notification. If one looks carefully, the BUG is not inherent in the application but flaws in the third party code affects the front end applications and base software.

A snapshot of Internet Explorer is shown displaying the loaded browser helper objects.

Let's dissect the peripheral structure of Add-ons: Figure 2.

The above presented layout is the working design of a Malware Add-on. In this primarily the Add-on comprises of a number of rogue functions. These functions perform certain operations that crash the application and enable it to work as a Malware base. This not only degrades the application robustness but also affect the operating system. The working components are presented in the snapshot. The Add-on comprised of *Functional Module Rm*(1) and so on. The various *Rm* (n) are considered as functional modules which are coded to dethrone the functioning. When these functions are called by an application as a part of add-on it generates an exception. The general view of manipulated functions is presented in the diagram. Every single module adheres to a different type of operation. The operations are performed in a covert manner. These functions are operated at the back end and make your system an exploited base.

The architecture of Internet Explorer is complex. The compatibility is lowered with the addition of Add-ons that are not verified prior to load in an application. Internet Explorer mechanism of handling these add-ons is highly vulnerable as it crashes most of the time thereby creating an opportunity of exploitation. Even if exploitation vector is not so intensified, application still suffers a lot.

Let's look at the general view of browser working. In this browser inherits an interactive renderer internally. It acts as content rendering system when a browser is loading a XML based data from a remote server. After that the raw data has to be presented in a generic manner for users. This whole process is termed as Document synthesis because of the all browser supports DOM i.e. document object model. The problem arises when the content is not rendered successfully and an exception occurs.

One can see clearly the functionality of browser. Add-ons adhere to hidden functioning. When ever an Add-on is loaded into the context of browser the processing is done according to the defined benchmarks. The benchmark here refers to standard operation performed by the browser when a request is initialized. So a simple code stub in the form of Add-on exploits the browser processing in a rogue manner. Overall discussion has been done. Let's analyze the case:

Case: Internet Explorer is loaded into memory and suddenly it crashes. The

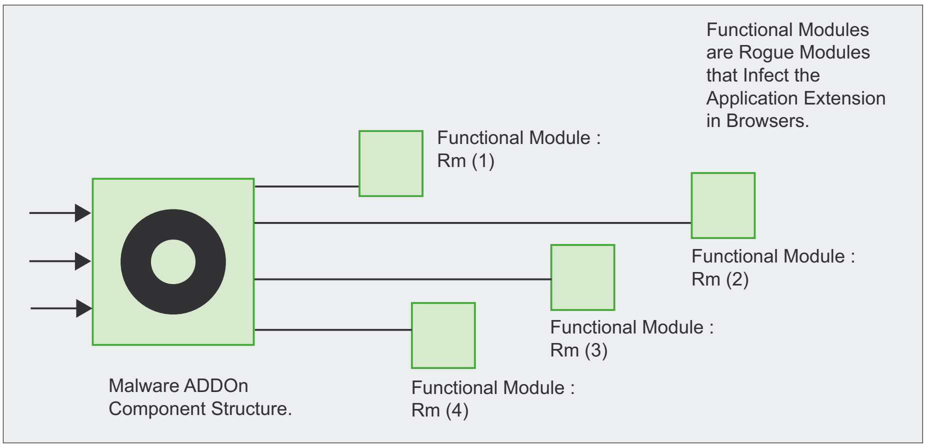

Functional Modules are Rogue Modules that Infect the Application Extension in Browsers.
Functional Module : Rm (1)
Functional Module : Rm (2)
Functional Module : Rm (3)
Functional Module : Rm (4)
Malware ADDOn Component Structure.

Figure 2. *Overview of Malware Add-ons*

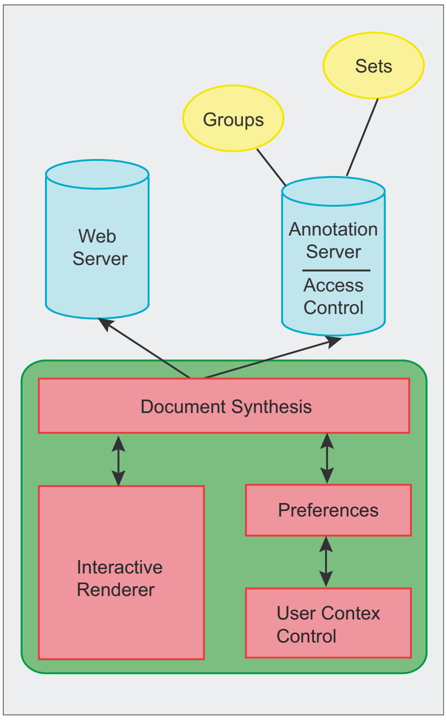


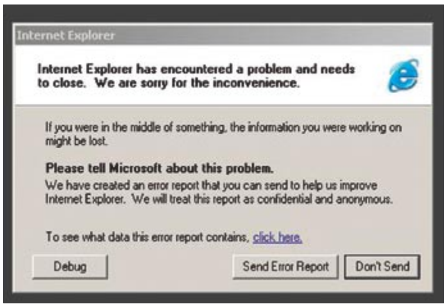
Figure 3. *Peripheral working of a Browser*

snapshots that have been presented below to look at the case exactly. After that we will analyze the internal problem by reversing and finding the root cause.

The IE is crashed. It is a standard layout of error handling by Windows Operating System. Of course the next step is to analyze the real point of infection that has triggered the instability in the application. The next layout will clear up the picture to some extent.

The system is using so many Add-ons but this situation is never encountered. But this layout says that inTru.dll Dynamic Link Library is not verified by the Internet Explorer. One question is always stated that failure in verification of Add-on leads to application crash. This is a subtle response and it projects the behavior of Intru.dll when it is applied as such for functional usage. The very first point comes to mind is the identity of this DLL. Is it a Malware? This DLL is performing certain rogue functions that are not handled by Internet Explorer. Another Debug layout is presented on Figure x. It shows the exception and debugs statistics of the IE crash due to `Intru.dll`.

A cross check is performed to analyze the effect on Internet Explorer when intru.dll is not loaded as Add-on. In order to prove this the Add-on is flagged as disabled in IE to watch the functioning.



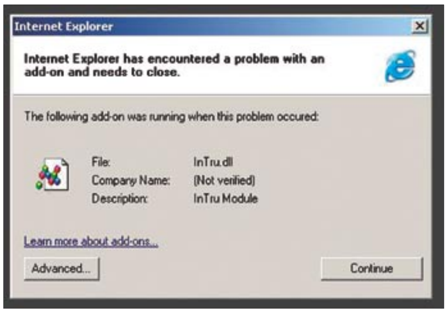
Internet Explorer has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.
We have created an error report that you can send to help us improve Internet Explorer. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

Figure 4. *Internet Explorer Crash*



Internet Explorer has encountered a problem with an add-on and needs to close.

The following add-on was running when this problem occured:

File: InTru.dll
Company Name: (Not verified)
Description: InTru Module

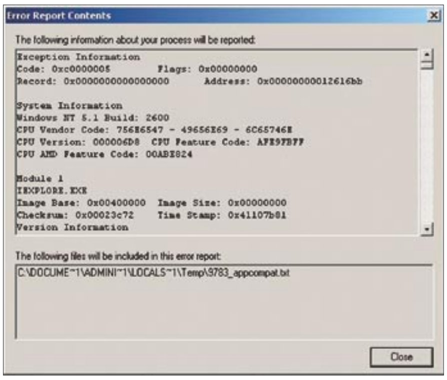
Learn more about add-ons...

Figure 5. *Cause of Crash*

There are number of add-ons presented in the above layout. The inTru.dll is set in disabled mode and is not loaded while IE is in dynamic state. There are also other Add-ons which are not verified by IE mechanism but IE does not show any vulnerable stats. Now the `InTru.dll` will be reversed for its Malware characteristics.

Analysis

The next step is to dissect the internals of this InTru.dll. One thing is sure – this DLL is loading or calling rogue functions that are not handled by IE effectively. Even if function is executed, the response is not assembled by IE in a structured manner thereby generating exceptions. The internals will reflect the components of this DLL. The term is known as *Root Cause*. It is defined as the main cause of an exceptional error. It can be analyzed by traversing an application object tree from bottom to top. Actually the functionality of objects is based on hierarchical layout in the form of tree. It is almost similar to tree data structure and object as nodes. This approach of finding the cause of error is very effective from flexibility point of view. The applications will be tested on standard benchmarks. But he question is Why IE crashes? Let's start the things.



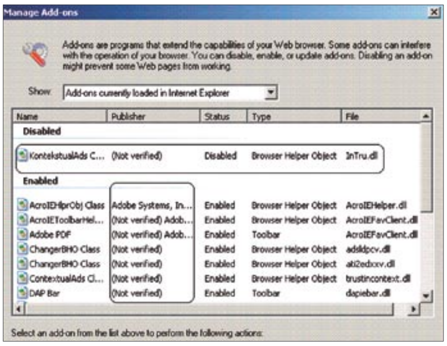
The following information about your process will be reported:

```
Exception Information
Code: 0xc0000005         Flags: 0x00000000
Record: 0x0000000000000000       Address: 0x00000000012616bb

System Information
Windows NT 5.1 Build: 2600
CPU Vendor Code: 756E6547 - 49656E69 - 6C65746E
CPU Version: 000006D8  CPU Feature Code: AFE9FBFF
CPU AMD Feature Code: 00ABE824

Module 1
IEXPLORE.EXE
Image Base: 0x00400000  Image Size: 0x00000000
Checksum: 0x00023c72    Time Stamp: 0x41107b81
Version Information
```

The following files will be included in this error report:

C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\9783_appcompat.txt

Figure 6. *Debug Response*



Add-ons are programs that extend the capabilities of your Web browser. Some add-ons can interfere with the operation of your browser. You can disable, enable, or update add-ons. Disabling an add-on might prevent some Web pages from working.

Show: Add-ons currently loaded in Internet Explorer

Name	Publisher	Status	Type	File
Disabled				
KontekstualAds C...	(Not verified)	Disabled	Browser Helper Object	InTru.dll
Enabled				
AcroIEHlprObj Class	Adobe Systems, In...	Enabled	Browser Helper Object	AcroIEHelper.dll
AcroIEToolbarHel...	(Not verified) Adob...	Enabled	Browser Helper Object	AcroIEFavClient.dll
Adobe PDF	(Not verified) Adob...	Enabled	Toolbar	AcroIEFavClient.dll
ChangerBHO Class	(Not verified)	Enabled	Browser Helper Object	adsldpcv.dll
ChangerBHO Class	(Not verified)	Enabled	Browser Helper Object	ati2edxxv.dll
ContextualAds Cl...	(Not verified)	Enabled	Browser Helper Object	trustincontext.dll
DAP Bar	(Not verified)	Enabled	Toolbar	dapiebar.dll

Select an add-on from the list above to perform the following actions:

Figure 7. *Malware inTru.dll is Disabled*

Breaking in Functions

This process is based on DLL Tracing to look for calling modules. The IDAG Functional Graph is presented in Figure 8.

The graph shows the functions that are called by this DLL with respect to

Listing 1. *Declaration of InternetOpen() and InternetConnect() function*

```
HINTERNET InternetOpen(
               LPCTSTR lpszAgent,
               DWORD dwAccessType,
               LPCTSTR lpszProxyName,
               LPCTSTR lpszProxyBypass,
               DWORD dwFlags
);
HINTERNET InternetConnect(
               HINTERNET hInternet,
               LPCTSTR lpszServerName,
               INTERNET_PORT
                       nServerPort,
               LPCTSTR lpszUsername,
               LPCTSTR lpszPassword,
               DWORD dwService,
               DWORD dwFlags,
               DWORD_PTR dwContext
);
```

Listing 2. *Declaration of HTTPOpenRequest() function*

```
HINTERNET HttpOpenRequest(
               HINTERNET hConnect,
               LPCTSTR lpszVerb,
               LPCTSTR lpszObjectName,
               LPCTSTR lpszVersion,
               LPCTSTR lpszReferer,
               LPCTSTR*
                       lpszAcceptTypes,
               DWORD dwFlags,
               DWORD_PTR dwContext
);
```

Listing 3. *Declaration of HTTPSendRequest() function*

```
BOOL HttpSendRequest(
               HINTERNET hRequest,
               LPCTSTR lpszHeaders,
               DWORD dwHeadersLength,
               LPVOID lpOptional,
               DWORD dwOptionalLength
);
```

Listing 4. *Declaration of InternetReadFile() function*

```
BOOL InternetReadFile(
               HINTERNET hFile,
               LPVOID lpBuffer,
               DWORD
                dwNumberOfBytesToRead,
               LPDWORD
                lpdwNumberOfBytesRead
);
```

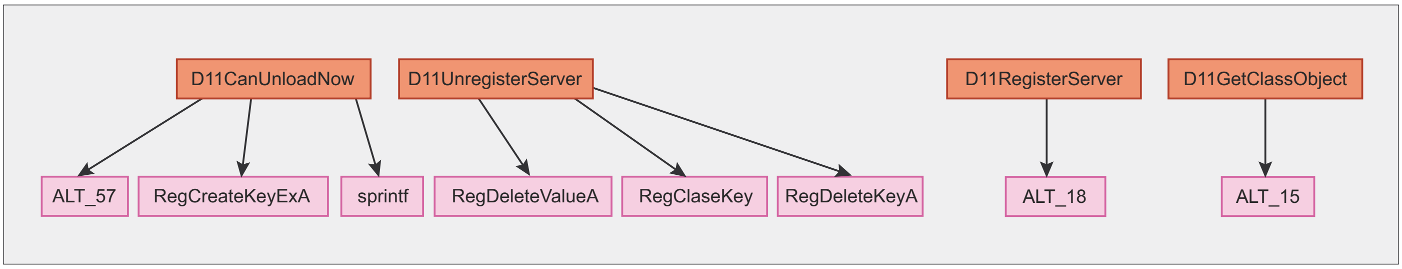


Figure 8. *Function Graph in IDAG*

DllRegisterServer, DllCanUnloadNow, DllUnRegisterServer and *DllGetClassObject*. These functions are the standard benchmarks that are followed by every single DLL to load manually into the application. You also can find Registry API's that are used by this DLL effectively. This functional graph only provides a peripheral aspect of DLL Tracing. Mostly these functions are exported. It means that these are generically designed in the code manually. Let's see the exported function windows to cross check:

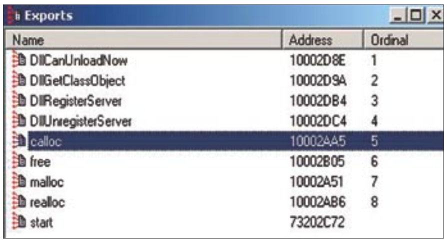
These functions are well structured. But looking at them it can be undertaken easily that no malware function is defined as such. So now we are going to knock the Import Address table for dynamically called API's. This will clear up the picture to some extent too.

The imported functions list is structured on Figures 9 and 10. The snapshot of specific functions is undertaken. It can be seen very clearly that Wininet functions have been used. This means the required Addon is using Winnet functions dynamically to perform the task it is meant for. These functions are the prime elemental modules that are used by this Add-on. One by one the usage of every single function will be discussed to understand the working statistics. The *InternetOpenA* prepares an application to start using Winnet library. The initialization parameter prepares an application for using internal data structure required for performing internet based functions. The very next imported function is *InternetConnectA*. This function is primarily used for opening protocol handles to exchange data. It includes FTP, HTTP or Gopher, etc. Actually it sets up an active session with an unique site. This process is generically termed as Session Builiding. The application mainly requests a session for transaction. It means as soon as the Intru.dll is loaded into the Internet Explorer it tries to open up an active session with some website where the malware content or manipulated content is located. Let us look at the function prototypes first: Listing 1.

The next function is *HTTPOpenRequestA*. The *InternetReadFile* function uses the handle given by this *HTTPOpenRequestA* function to download the stream of data as file. This makes the process easier because data transaction taken over the internet is in the form of a file. The settings of various flags in these functions play a critical role in determining the characteristics of data being transferred. Take a look at the *HTTPOpenRequest* function: Listing 2.

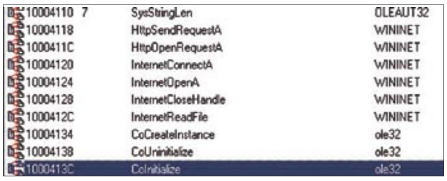
So the behavior of Add-on is getting cleared from functional point of view. The next function is *HTTPSendRequestA*. Straight forward this function sends a request to HTTP server for performing required task. Then the client specifies extra headers to send along with the request. So the functional picture is quite clear. Let's have a look at the function presented in Listing 3.

So the functions are presented. The next function is `InternetReadFileA`. This function reads a file from the website whose working handle is being created by the other wininet functions. One thing is sure that this specific add-on is reading file from some website. There must be a specific URL present in the code which provides a source of calling to these functions. In this HINTERNET handle is created by the initialization functions. And the data is retrieved by sequential process of streaming (see Listing 4).



Name	Address	Ordinal
DllCanUnloadNow	10002D8E	1
DllGetClassObject	10002D9A	2
DllRegisterServer	10002DB4	3
DllUnregisterServer	10002DC4	4
calloc	10002AA5	5
free	10002B05	6
malloc	10002A51	7
realloc	10002AB6	8
start	73202C72	

Figure 9. *Exported Functions view in IDAG*



10004110	7	SysStringLen	OLEAUT32
10004118		HttpSendRequestA	WININET
1000411C		HttpOpenRequestA	WININET
10004120		InternetConnectA	WININET
10004124		InternetOpenA	WININET
10004128		InternetCloseHandle	WININET
1000412C		InternetReadFile	WININET
10004134		CoCreateInstance	ole32
10004138		CoUninitialize	ole32
1000413C		CoInitialize	ole32

Figure 10. *Imported Functions view in IDAG*

These are the imported functions used by this Intru.dll object. Another basic part to check is how these functions are interrelated. This has been stated in previous articles that Cross Functional Analysis is to be performed for effective analysis. The code is tested on Intrinsic Calling of functions. In this cross references of the imported and exported functions are to be checked. But this Add-on does not provide any user specific references to and from the code. This relatively clears the picture that no generic cross references are used in this. The code is also equipped with Registry Keys. Let's have a look at the code (see Listing 5).

So presented code shows that registry has been tempered by the Add-on. This add-on is creating a key with a string parameter as `kontekstualAds`. At the same time registry deletes function is used. It means when ever a Add-on is dynamically loaded in the Internet Explorer a registry key is created with the defined string in the windows registry database. As soon as the DLL is unloaded when IE is closed the key is deleted leaving no traces in the database. Actually this process is used by malwares to active falsified working so that tracing is not possible. And the keys are deleted. This makes malware very dynamic from intrinsic point of view. So this possibility is cross checked by scanning registry. I simple performed a logical check with Regedit. Let's see Figure 11.

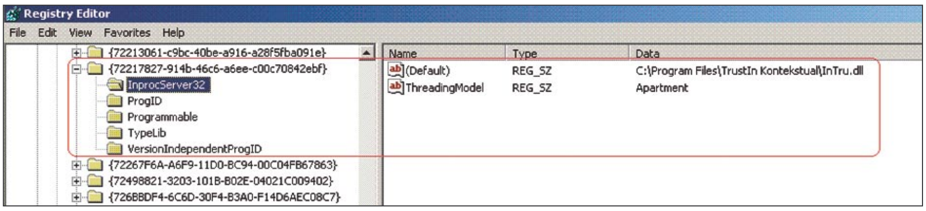
The inference is clear about the active use of registry in this. So it shows the presence of CLSID object in the registry database. Even if you see in the list of imported functions, the CoCreateInstance API is used directly for creating an instance of any Class object. The CLSID of contextual can be verified from the picture. Let's see at the API call see Listing 6.

So the class object is created by Add-on Intru.dll effectively. The next point is to dissect the code in hexadecimal layout for analysis. The process is termed as Hex Dumping of raw code. This enables the reverse engineer to look at the raw output with respect to hexadecimal codes for better understanding. Always remember the dumping of malware code in hexadecimal always yield fruitful results. We are going to analyze the hex dump of Intru.dll Add-on.

The different color codes present the subtle information that is extracted from the source for better understanding of malware. Following inferences have been analyzed as:

- The blue color code states that the buffer is padded with useless strings. Many times Malware programs are padded with useless data in order to set the efficient size of buffer to be called. So the strings used in this Malware add-on do nothing but are only used for peripheral use.
- Secondly, the red color code provides us with the information of destination URL which is used by an application to establish a session for downloading file from the source. The URL extracted is *http://www.adscontex.com/dir/Kontekstual/config.xml.Last*. It means an XML file is either fetched or used by this Add-on. This can be considered as one of the factors of crashing Internet Explorer because the contents of the file are not rendered well by Internet Explorer and hence an exception occurs. The raw code is throwing information in an efficient manner.
- The grey color projects information regarding *Registry Creation* and *Deletion*. This has been explained in previous section. The only difference one can predict about the raw structure of the registry keys in this output.
- The green color shows that JavaScript is triggered for Pop up generation when ever a session is established with the destination for malicious activities.
- The black color consists of useless buffer with *blah* strings. It also covers the Registry path and showing in `REG_SZ` string parameter in other format i.e. `software\%s`.

So overall this gives a straight forward view of the working semantics of these types of browser helper objects. Based on this inference a practical diagram have been designed to present the working



```
{72213061-c9bc-40be-a916-a28f5fba091e}
{72217827-914b-46c6-a6ee-c00c70842ebf}
    InprocServer32
    ProgID
    Programmable
    TypeLib
    VersionIndependentProgID
{72267F6A-A6F9-11D0-BC94-00C04FB67863}
{72498821-3203-101B-B02E-04021C009402}
{726BBDF4-6C6D-30F4-B3A0-F14D6AEC08C7}
```

Name	Type	Data
(Default)	REG_SZ	C:\Program Files\TrustIn Kontekstual\InTru.dll
ThreadingModel	REG_SZ	Apartment

Figure 11. *Registry Check against IE Add-on i.e inTru.dll*

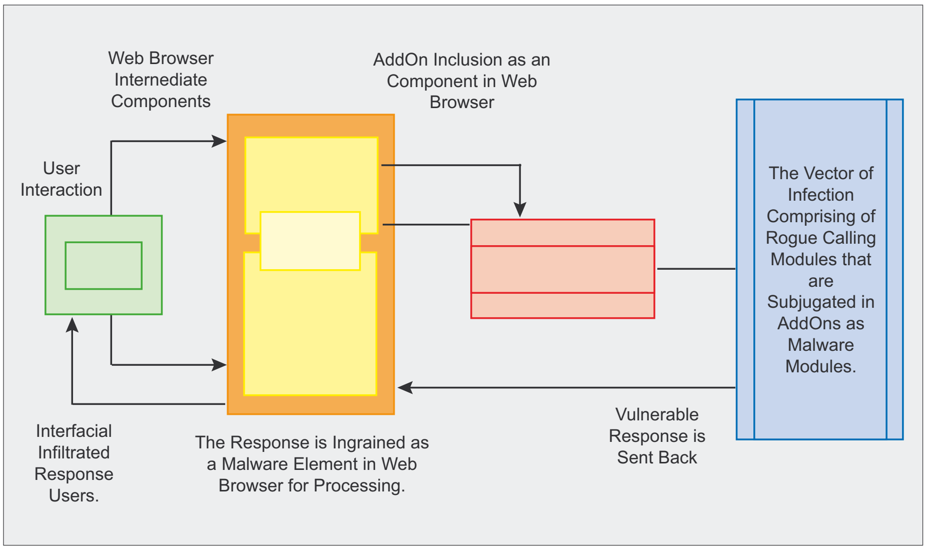

Web Browser Internediate Components
AddOn Inclusion as an Component in Web Browser
User Interaction
The Vector of Infection Comprising of Rogue Calling Modules that are Subjugated in AddOns as Malware Modules.
Vulnerable Response is Sent Back
Interfacial Infiltrated Response Users.
The Response is Ingrained as a Malware Element in Web Browser for Processing.

Figure 12. *Malware Add-on functionality diagram in detail*

Listing 5. *Registry addition by Malware*

```
mov     [ebp-4], eax
call    ds:RegCreateKeyExA
mov     edi, ds:sprintf
mov     esi, offset aKontekstualAds
push    esi
lea     eax, [ebp-404h]
push    offset aSDisplayname
push    eax
call    edi ; sprintf
mov     ebx, ds:RegDeleteValueA
add     esp, 0Ch
lea     eax, [ebp-404h]
push    eax
push    dword ptr [ebp-4]
call    ebx ; RegDeleteValueA
push    esi
lea     eax, [ebp-404h]
push    offset aSUninstallstri
push    eax
call    edi ; sprintf
add     esp, 0Ch
lea     eax, [ebp-404h]
push    eax
push    dword ptr [ebp-4]
call    ebx ; RegDeleteValueA
push    esi
push    dword ptr [ebp-4]
call    ds:RegDeleteKeyA
push    dword ptr [ebp-4]
call    ds:RegCloseKey
push    0
push    1
push    offset unk_0_10005540
call    ds:ATL_57
pop     edi
pop     esi
pop     ebx
leave
retn
```

Listing 6. *Declaration of CoCreateInstance() function*

```
       STDAPI CoCreateInstance(
REFCLSID rclsid,
LPUNKNOWN pUnkOuter,
DWORD dwClsContext,
REFIID riid,
LPVOID * ppv       );

       HRESULT CoInitialize(
           LPVOID pvReserved
           );
```

methodology of a specific Malware based browser helper object. This not only helps in understanding the hidden artifacts but also useful in information gathering (see Figure 12).

Overall the analytical phase is summarized in the snapshot provided on Figure 12. The addition of rogue component i either as Plugin or helper objects affects the system state and makes application suffer a jolt. When a Malware component is loaded into the system via application interface, it performs some backdoor manipulations. Like presented the Malware Add-on has some vectors of infection that is called remotely. As soon as it dynamically loaded, it opens a session through WinInet functions and tries to perform manipulative functions. A response is undertaken and structured back to the application for infecting the system. Sometimes the content is not handled well by the application that leads to an exceptions hard to be managed by the exception handler. As a result of it, application crashes.

Listing 7. *Hexadecimal Dump*

```
62 6C 61 68 62 6C 61 68-62 6C 61 68 62 6C 61 68 "blahblahblahblah"
62 6C 61 68 62 6C 61 68-62 6C 61 68 62 6C 61 68 "blahblahblahblah"
62 6C 61 68 62 6C 61 68-62 6C 61 68 62 6C 61 68 "blahblahblahblah"
62 6C 61 68 62 6C 61 68-62 6C 61 68 62 6C 61 68 "blahblahblahblah"
62 6C 61 68 62 6C 61 68-62 6C 61 68 62 6C 61 68 "blahblahblahblah"
62 6C 61 68 62 6C 61 68-62 6C 61 68 62 6C 61 68 "blahblahblahblah"
62 6C 61 68 62 6C 61 68-62 6C 61 68 62 6C 61 68 "blahblahblahblah"
62 6C 61 68 62 6C 61 68-62 6C 61 68 62 6C 61 68 "blahblahblahblah"
00 00 00 00 53 6F 66 74-77 61 72 65 5C 25 73 5C "....Software\%s\"
25 73 00 00 49 6E 54 72-75 00 00 00 4B 6F 6E 74 "%s..InTru...Kont"
65 6B 73 74 75 61 6C 20-41 64 73 00 69 65 78 70 "ekstual Ads.iexp"
6C 6F 72 65 2E 65 78 65-00 00 00 00 4C 61 73 74 "lore.exe....Last"
50 6F 70 75 70 00 00 00-62 65 66 6F 72 65 45 6E "Popup...beforeEn"
64 00 00 00 25 73 25 73-00 00 00 00 3C 62 72 2F "d...%s%s....<br/"
3E 3C 73 63 72 69 70 74-20 6C 61 6E 67 75 61 67 "><script languag"
65 3D 22 6A 61 76 61 73-63 72 69 70 74 22 20 64 "e="javascript" d"
65 66 65 72 3E 6B 65 79-77 6F 72 64 3D 22 25 73 "efer>keyword="%s"
22 3C 2F 73 63 72 69 70-74 3E 00 00 72 65 64 69 ""</script>..redi"
72 65 63 74 00 00 00 00-81 BF 33 29 36 7B D2 11 "rect....ü+3)6{-"
B2 0E 00 C0 4F 98 3E 60-68 74 74 70 3A 2F 2F 77 "¦.+Oÿ>`http://w"
77 77 2E 61 64 73 63 6F-6E 74 65 78 2E 63 6F 6D "ww.adscontex.com"
2F 64 69 72 2F 4B 6F 6E-74 65 6B 73 74 75 61 6C "/dir/Kontekstual"
2F 63 6F 6E 66 69 67 2E-78 6D 6C 00 4C 61 73 74 "/config.xml.Last"
43 66 67 46 65 74 63 68-00 00 00 00 43 4B 6F 6E "CfgFetch....CKon"
74 65 6B 73 74 75 61 6C-41 64 73 20 74 72 69 65 "tekstualAds trie"
73 20 74 6F 20 70 65 72-66 6F 72 6D 20 73 74 61 "s to perform sta"
72 74 20 61 63 74 69 6F-6E 73 00 00 6D 69 6E 00 "rt actions..min."
75 72 6C 00 69 67 6E 6F-72 65 73 69 74 65 73 00 "url.ignoresites."
69 67 6E 6F 72 65 00 00-70 65 72 69 6F 64 00 00 "ignore..period.."
74 68 72 65 73 68 6F 6C-64 00 00 00 66 65 65 64 "threshold...feed"
00 00 00 00 2C 3A 3B 60-27 22 2B 2D 5F 28 29 7B "....,:;`'"+-_(){"
7D 5B 5D 3C 3E 2A 26 5E-25 24 23 40 21 3F 7E 2F "}[]<>*&^%$#@!?~/"
7C 5C 3D 20 09 0D 0A 31-32 33 34 35 36 37 38 39 "|\=
123456789"
30 00 00 00 4C 6F 77 00-48 69 67 68 00 00 00 00 "0...Low.High...."
68 74 74 70 3A 2F 2F 00-47 45 54 00 57 69 6E 49 "http://.GET.WinI"
6E 65 74 20 54 65 73 74-00 00 00 00 00 00 00 00 "net Test........"
10 59 2F B6 28 65 D1 11-96 11 00 00 F8 1E 0D 0D "Y/¦(e-û..°-

"
A8 41 00 10 44 2B 00 10-6A 2E 00 10 D4 2E 00 10 "¿A.D+.j..+.."
00 00 00 00 00 00 00 00-67 2E 00 10 67 2E 00 10 "........g..g.."
AB 36 00 10 00 00 00 00-00 00 00 00 00 00 00 00 "½6............"
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 "................"
00 00 00 00 00 00 00 00-55 6E 69 6E 73 74 61 6C "........Uninstal"
6C 53 74 72 69 6E 67 00-44 69 73 70 6C 61 79 4E "lString.DisplayN"
61 6D 65 00 54 72 75 73-74 49 6E 20 4B 6F 6E 74 "ame.TrustIn Kont"
65 6B 73 74 75 61 6C 00-53 6F 66 74 77 61 72 65 "ekstual.Software"
5C 4D 69 63 72 6F 73 6F-66 74 5C 57 69 6E 64 6F "\Microsoft\Windo"
77 73 5C 43 75 72 72 65-6E 74 56 65 72 73 69 6F "ws\CurrentVersio"
6E 5C 55 6E 69 6E 73 74-61 6C 6C 5C 25 73 00 00 "n\Uninstall\%s.."
22 00 00 00 72 65 67 73-76 72 33 32 20 2F 75 20 ""...regsvr32 /u "
2F 73 20 22 00 00 00 00-25 73 5C 55 6E 69 6E 73 "/s "....%s\Unins"
74 61 6C 6C 53 74 72 69-6E 67 00 00 25 73 5C 44 "tallString..%s\D"
69 73 70 6C 61 79 4E 61-6D 65 00 00 53 6F 66 74 "isplayName..Soft"
77 61 72 65 5C 4D 69 63-72 6F 73 6F 66 74 5C 57 "ware\Microsoft\W"
69 6E 64 6F 77 73 5C 43-75 72 72 65 6E 74 56 65 "indows\CurrentVe"
72 73 69 6F 6E 5C 55 6E-69 6E 73 74 61 6C 6C 00 "rsion\Uninstall."
01 00 00 00 00 00 00 00-C0 00 00 00 00 00 00 46 " .......+......F"
46 69 6C 65 32 52 65 6D-6F 76 65 00 00 00 00 00 "File2Remove....."
```

Conclusion

The application reversing is a very effective technique to understand the working of Internet Explorer based Malwares. The practical analysis of browser helper objects can be summed up as a learning experience. Through this one can learn the parameters of technology and the stringent effects when it is not properly implemented. The Malwares can be in the form of Plugins, Add-ons etc which act as an additional interface for versatile functioning. So the dissection of these Malwares should be done effectively for understanding the hidden parameters. The procedural and practical techniques should be applied to understand the backdoor functioning of malware oriented browser helper objects. It has been rightly stated *To understand the core, you must dig in.* ●

On the 'Net

- *http://www.openrce.org*
- *http://www.openrce.org/blog/browse/aditya_ks*
- *http://www.nynaeve.net/*
- *http://home.arcor.de/idapalace/ – Index of IDAPalace*
- *http://www.exetools.com*

Aditya K Sood aka 0kn0ck
An independent security researcher and founder of SecNiche Security, a security research arena. He holds BE and MS in Cyber Law and Information Security from Indian Institute of InformationTechnology.He is a regular speaker at conferences such as XCON, OWASP, andCERT-IN. Hisother projects include Mlabs, CERA, and TrioSec. *http://www.secniche.org*