A New Appreciation for Continuous Asset Monitoring
In TalkTalk aftermath, it’s time for companies to pay higher price for breaches

FLARE Script Series: Automating Obfuscated String Decoding

By Kristen Dennesen On December 28, 2015 · Add Comment

Introduction

We are expanding our script series beyond IDA Pro. This post extends the FireEye Labs Advanced Reverse Engineering (FLARE) script series to an invaluable tool for the reverse engineer – the debugger. Just like IDA Pro, debuggers have scripting interfaces. For example, OllyDbg uses an asm-like scripting language, the Immunity debugger contains a Python interface, and Windbg has its own language. Each of these options isn’t ideal for rapidly creating string decoding debugger scripts. Both Immunity and OllyDbg only support 32-bit applications, and Windbg’s scripting language is specific to Windbg and, therefore, not as well-known. The pykd project was created to interface between Python and Windbg to allow

OPML

You can grab the full OPML file here.

Follow on Twitter
Follow on Facebook

Contributors

Government Technology
“Fuhyô no Michi” (不評の道): “The path of unpopularity”
[I] Immutable Security
@"figurative";
@neirajones
/dev/null – ramblings of an infosec professional
/dev/random
27001Academy
451 Security
A Collection of Bromides on Infrastructure
A Journey through Security
A Techie's Musings
About.com

debugger scripts to be written in Python. Because malware reverse engineers love Python, we built our debugger scripting library on top of pykd for Windbg.

Here we release a library we call flare-dbg. This library provides several utility classes and functions to rapidly develop scripts to automate debugging tasks within Windbg. Stay tuned for future blog posts that will describe additional uses for debugger scripts!

String Decoding

Malware authors like to hide the intent of their software by obfuscating their strings. Quickly deobfuscating strings allows you to quickly figure out what the malware is doing.

As stated in Practical Malware Analysis, there are generally two approaches to deobfuscating strings: self-decoding and manual programming. The *self-decoding* approach allows the malware to decode its own strings. *Manual programming* requires the reverse engineer to reprogram the decoding function logic. A subset of the *self-decoding* approach is emulation, where each assembly instruction execution is emulated. Unfortunately, library call emulation is required, and emulating every library call is difficult and may cause inaccurate results. In contrast, a debugger is attached to the actual running process, so all the library functions can be run without issue. Each of these approaches has their place, but this post teaches a way to use debugger scripting to automatically self-decode all obfuscated strings.

Challenge

To decode all obfsucated strings, we need to find the following: the string decoder function, each time it is called, and all arguments to each of those instances. We then need to run the function and read out

Business Security
Acunetix » Web Application Security Blog
Adeptus-Mechanicus
Adventures in Security
Adventures in Security
Alert Logic » Blog
Amrit Williams Blog
An Analytical Approach
An Information Security Place
Andrew Hay
Anitian Blog
Anti-Malware Reviews
Application & Cyber Security Blog
ARCHIMEDIUS
Armchair Stratosphere
armoredcode.com
Articles by MIKE FRATTO
artofinfosec
Ask Marian
articles
Astaro Blog – Security Perspectives Feed
AsTech Consulting
aut disce, aut discede
Avast Blog
Avast: Keeping it safe

the result. The challenge is to do this in a semi-automated way.

Approach

The first task is to find the string decoder function and get a basic understanding of the inputs and outputs of the function. The next task is to identify each time the string decoder function is called and all of the arguments to each call. Without using IDA, a handy Python project for binary analysis is Vivisect. Vivisect contains several heuristics for identifying functions and cross-references. Additionally, Vivisect can emulate and disassemble a series of opcodes, which can help us identify function arguments. If you haven't already, be sure to check out the FLARE scripting series post on tracking function arguments using emulation, which also uses Vivisect.

Introducing flare-dbg

The FLARE team is introducing a Python project, flare-dbg that runs on top of pykd. Its goal is to make Windbg scripting easy. The heart of the flare-dbg project lies in the DebugUtils class, which contains several functions to handle:

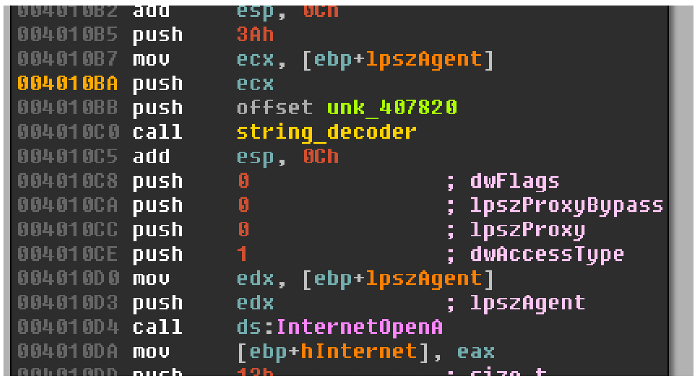
- Memory and register manipulation
- Stack operations
- Debugger execution
- Breakpoints
- Function calling

In addition to the basic debugger utility functions, the DebugUtils class uses Vivisect to handle the binary analysis portion.

Example

I wrote a simple piece of malware that hides strings by encoding them. Figure 1 shows an HTTP User-Agent string being decoded by a function I named string_decoder.

Barry Shteiman
Beast or Buddha
Bernd Marienfeldt
BlackRidge Sentinel
Blaze's Security Blog
Blog
Blog
Blog
Blog
Blog – ERPScan
Blog – Phoenix TS
BlogInfoSec.com
Blogs – Pixalert International Blog
Blue Box: The VoIP Security Podcast
Blue Ocean
Boaz Gelbord
Branden R. Williams, Business Security Specialist
BruteForce Lab's Blog
Carnal0wnage & Attack Research Blog
CGISecurity – Website and Application Security News
Check Chain Mail and Hoaxes
Checkmarx
CIO Blogs – Reasonable Doubter
Cisco Talos Blog



```
004010B2 add      esp, 0Ch
004010B5 push     3Ah
004010B7 mov      ecx, [ebp+lpszAgent]
004010BA push     ecx
004010BB push     offset unk_407820
004010C0 call     string_decoder
004010C5 add      esp, 0Ch
004010C8 push     0                 ; dwFlags
004010CA push     0                 ; lpszProxyBypass
004010CC push     0                 ; lpszProxy
004010CE push     1                 ; dwAccessType
004010D0 mov      edx, [ebp+lpszAgent]
004010D3 push     edx               ; lpszAgent
004010D4 call     ds:InternetOpenA
004010DA mov      [ebp+hInternet], eax
```

Figure 1: String decoder function reference in IDA Pro

After a cursory look at the string_decoder function, the arguments are identified as an offset to an encoded string of bytes, an output address, and a length. The function can be described as the following C prototype:

```
VOID string_decoder(PBYTE input, PCHAR output, DWORD len);
```

Now that we have a basic understanding of the string_decoder function, we test decoding using Windbg and flare-dbg. We begin by starting the process with Windbg and executing until the program’s entry point. Next, we start a Python interactive shell within Windbg using pykd and import flaredbg.

```
0:000> .load pykd
0:000> !py
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from flaredbg import flaredbg
```

Next, we create a DebugUtils object, which contains the functions we need to control the debugger.

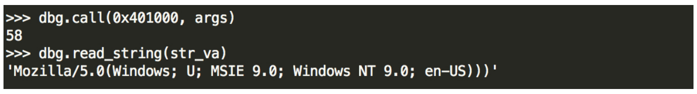
```
>>> dbg = flaredbg.DebugUtils()
```

We then allocate 0x3A-bytes of memory for the output string. We use the newly allocated memory as the second parameter and setup the remainder of the arguments.

```
>>> str_va = dbg.malloc(0x3a)
>>> args = [0x407820, str_va, 0x3a]
```

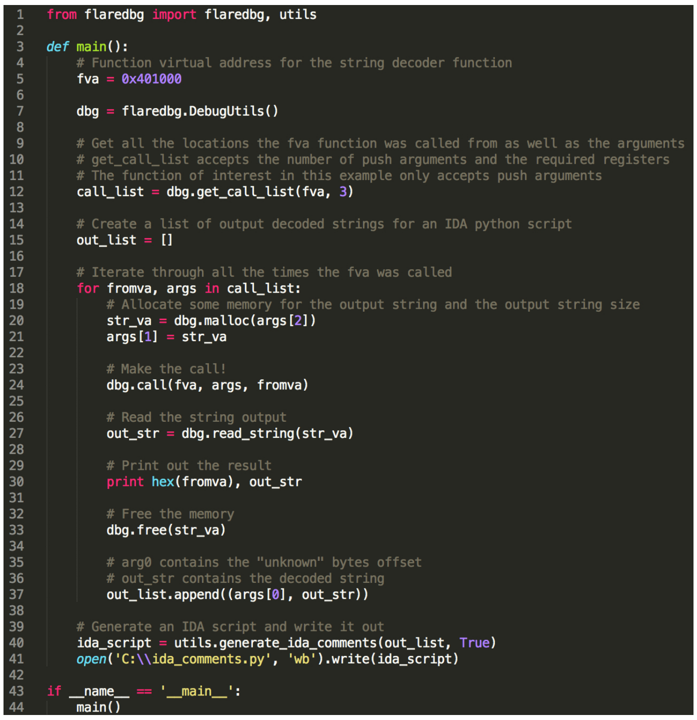
Clearswift Blog
Clerkendweller : Application Security and Privacy
Coffee To Code
Cognitive Dissidents
Comments on:
Comments on:
Commtouch Café
concise
Converging Network
Corelan Team
CQR
Cryptosmith
Cyber Trust Blog » Cloud Computing
Cyber Trust Blog » Cybersecurity
CyberCrime & Doing Time
Cymbel
Cyveillance Blog – The Cyber Intelligence Blog
Catch²² (in)secuяity / ChrisJohnRiley
Dancho Danchev's Blog – Mind Streams of Information Security Knowledge
Dark Reading
Data Security Blog | Vormetric
Defense Rests
Dennis Kudin on Security

Finally, we call the string_decoder function at virtual address 0x401000, and read the output string buffer.



```
>>> dbg.call(0x401000, args)
58
>>> dbg.read_string(str_va)
'Mozilla/5.0(Windows; U; MSIE 9.0; Windows NT 9.0; en-US)))'
```

After proving we can decode a string with flare-dbg, let's automate all calls to the string_decoder function. An example debugger script is shown in Figure 2. The full script is available in the examples directory in the github repository.



```python
from flaredbg import flaredbg, utils

def main():
    # Function virtual address for the string decoder function
    fva = 0x401000

    dbg = flaredbg.DebugUtils()

    # Get all the locations the fva function was called from as well as the arguments
    # get_call_list accepts the number of push arguments and the required registers
    # The function of interest in this example only accepts push arguments
    call_list = dbg.get_call_list(fva, 3)

    # Create a list of output decoded strings for an IDA python script
    out_list = []

    # Iterate through all the times the fva was called
    for fromva, args in call_list:
        # Allocate some memory for the output string and the output string size
        str_va = dbg.malloc(args[2])
        args[1] = str_va

        # Make the call!
        dbg.call(fva, args, fromva)

        # Read the string output
        out_str = dbg.read_string(str_va)

        # Print out the result
        print hex(fromva), out_str

        # Free the memory
        dbg.free(str_va)

        # arg0 contains the "unknown" bytes offset
        # out_str contains the decoded string
        out_list.append((args[0], out_str))

    # Generate an IDA script and write it out
    ida_script = utils.generate_ida_comments(out_list, True)
    open('C:\\ida_comments.py', 'wb').write(ida_script)

if __name__ == '__main__':
    main()
```

Figure 2. Example basic debugger script

Let's break this script down. First, we identify the function virtual address of the string decoder function and create a DebugUtils object. Next, we use the DebugUtils function get_call_list to find the three push arguments for each time string_decoder is called.

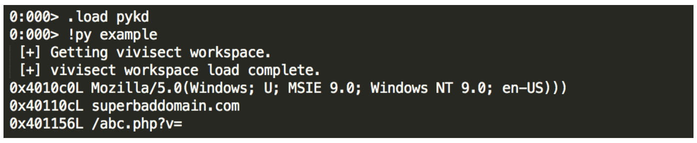
Once the call_list is generated, we iterate all calling addresses and associated arguments. In this example, the output string is decoded to the stack. Because we are only executing the string decoder function and won't have

Dome9
Dominic White
Dr Anton
Chuvakin Blog
PERSONAL Blog
DrekinnVörn
EH-Net Online
Mag
eLearnSecurity
: IT Security
Training
Elie on Internet
Security and
Performance
Emergent Chaos
EnableSecurity
Enterprise
Network
Security Blog
from
ISDecisions
Eon Security
Blog
Episteme:
Belief.
Knowledge.
Wisdom
Errata Security
eSoft Network
Security Blog
EurotrashSecurity
Fighting
In.Security
firstinitiallastname
flyingpenguin
Fortinet Blog
Frames And Bits
– The Andrew
Storms Blog
Front Line
Sentinel
GEEK Speek
GFI Labs blog
GNUCITIZEN
Government
Technology:

the same stack setup as the malware, we must allocate memory for the output string. We use the third parameter, the length, to specify the size of the memory allocation. Once we allocate memory for the output string, we set the newly allocated memory address as the second parameter to receive the output bytes.

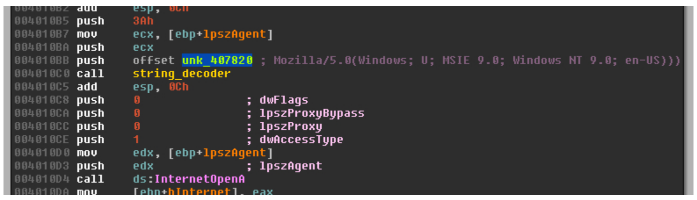
Finally, we run the string_decoder function by using the DebugUtils call function and read the result from our allocated buffer. The call function sets up the stack, sets any specified register values, and executes the function. Once all strings are decoded, the final step is to get these strings back into our IDB. The utils script contains utility functions to create IDA Python scripts. In this case, we output an IDA Python script that creates comments in the IDB.

Running this debugger script produces the following output:



```
0:000> .load pykd
0:000> !py example
 [+] Getting vivisect workspace.
 [+] vivisect workspace load complete.
0x4010c0L Mozilla/5.0(Windows; U; MSIE 9.0; Windows NT 9.0; en-US)))
0x40110cL superbaddomain.com
0x401156L /abc.php?v=
```

The output IDA Python script creates repeatable comments on all encoded string locations, as shown in Figure 3.



```
004010B2 add     esp, 0Ch
004010B5 push    3Ah
004010B7 mov     ecx, [ebp+lpszAgent]
004010BA push    ecx
004010BB push    offset unk_407820 ; Mozilla/5.0(Windows; U; MSIE 9.0; Windows NT 9.0; en-US)))
004010C0 call    string_decoder
004010C5 add     esp, 0Ch
004010C8 push    0               ; dwFlags
004010CA push    0               ; lpszProxyBypass
004010CC push    0               ; lpszProxy
004010CE push    1               ; dwAccessType
004010D0 mov     edx, [ebp+lpszAgent]
004010D3 push    edx             ; lpszAgent
004010D4 call    ds:InternetOpenA
004010DA mov     [ebp+hInternet], eax
```

Figure 3. Decoded string as comment

Conclusion

Stay tuned for another debugger scripting series post that will focus on plugins! For now, head over to the flare-dbg github project page to get started. The project requires pykd,winappdbg, and vivisect.

State & Local Government
News Articles
Graham Cluley
Hackers Center – Security Tools and Texts
Health Security Solutions
HolisticInfoSec
How is that Assurance Evidence?
HPE Security – Data Security
Hurricane Labs
I hack, therefore I am
I Smell Packets
IBM Application Security Insider
IceLock Blog
ICSA Labs – What's New for Network IPS
ICSA Labs Security Testing Blog
IDmachines
Idoneous Security
Impermium
Imperva Cyber Security Blog
Increased Visibility » SBN
Infamous Agenda
Info Loss
Information Centric Security
Information Security Blog | Perimeter E-Security
Information

SHARE → G+1 0 Tweet
Like 2 Pin it

Security is not an Oxymoron
Information Security Leaders
Information Security Strategy
Information Security Thoughts – Allen Baranov
Information Technology Enthusiast
InfoSec Daily Podcast
Infosec Events
InfoSec Mentors
Infosec Pro
Infosec Ramblings
InfoSec Resources
InfoSecPodcast.co
Infosecurity.US
Infosecurity.US
Inliniac
Insecurity
Inside The Threat Blog by Lancope
Invincea » Invincea Endpoint Security Blog | Invincea
iplosion security
iScan Online Blog
IT Best Practices and Compliance Reporting Information
IT Security &

Encryption
IT Security Blog. Mitigating Risks. Enabling Business Strategies. W. Mark Brooks
IT Security Expert Blog
IT Security, Windows Scripting and other matters
IT Security: The view from here
ITauditSecurity
iViZ Blog
J4vv4D
Jackwillk Security
JeffB’s Security Blog
JeffSoh on NetSec
Jeremiah Grossman
Jitender Arora
Joel Esler
John Melton's Weblog
Jon's Network
Jump ESP, jump!
Juniper
Kellep Charles Information Security Blog Space
Krebs on Security
Kyplex Cloud Security Blog
Last In – First Out
Latest articles from SC

Magazine The SC Magazine Awards Blog
Latest blog entries
Let's Talk Security
Linux Audit
Liquid Information
Liquidmatrix Security Digest
Lucius on Security
Mac Virus
Marble Security
Marco Ramilli’s Blog
Matt Flynn’s Security and Identity Blog
Merit Network: NetSec Blog RSS Feed
Michael Mongold’s Technology Security
Michael on Security
MichaelPeters.org
Mission Critical Global Technology Group – Mission Critical Global Technology Group
Monkey – House
MX Logic ThreatBlog
mxlab – all about anti virus and anti spam
Naked Security – Sophos

Naked Security – Sophos
Naked Security – Sophos
NETRESEC Network Security Blog
Netsparker, Web Application Security Scanner
Network Security Blog
Network World Cisco Security Expert
No Tricks
offense-in-depth
OpenDNS Security Labs
Permission Denied
Perspecsys
PhishLabs
Pipes Output
Porticor Cloud Security
Praetorian Prefect
Princess of Antiquity
psilva's prophecies
Punto 1
Radware Blog
Rainbow and Unicorn
Rainer's Blog
Random Security
Rational Survivability
Reflections on Security
Reusable

Security
Richi Jennings
Rick Holland's blog
RioSec
RiskAnalys.is
RiskPundit
RJS Smart Security
Robert Penz Blog
Room362
Rory.Blog
RSA Conference Blog
RSSC
rud.is
ryanlrussell
SANS Digital Forensics and Incident Response Blog
Schneier on Security
SecuraBit
Secure Ideas: Professionally Evil!
Secure Maryland
Secure Solutions
Secure Value
SecureAuth
SecureITExpert
SecurePuter
SecuriTeam Blogs
Security Aegis
Security and risk
Security Architecture
Security Balance
Security Blog

Security Braindump
Security Catalyst
Security Circus
Security Coin
Security Connections
Security Data Visualization
Security For All
Security For All
Security Generation
Security Heavy
Security In The Virtual World
Security Incite Rants
Security Intelligence and Big Data | raffy.ch – blog
Security Labs
Security Management at the Speed of Business – AlgoSec Blog
Security Mike’s Blog
Security Ninja
Security Products articles
Security Retentive
Security Samurai
Security Spread
Security TechWatch
Security Through Education
Security Uncorked by JJ

Security Workshop
Security x.0
Security4all – Dedicated to digital security, enterprise 2.0 and presentation skills
SecurityEverAfter
SecurityMaverick.c
SecurityOrb.com
SecurityRamblings
Securosis Complete
Seven Elements
ShackF00
Sicurezza Informatica Made in Italy
Simon PG Edwards
SIPVicious
Skeeter Spray
Small Blue-Green World
Social-Engineer.Com – Professional Social Engineering Training and Services
Solutionary Minds RSS – Your Information Security Blog
SOURCE Conference Blog
soxi
Speaking of Security – The RSA Blog and Podcast

spylogic.net
SteveHanna
Tracker
Stratum
Security
Sucuri Blog
Sucuri Blog
Sunnet
Beskerming
Security
Advisories
Suspicious
Minds
Symantec
Connect –
Security – Blog
Entries
Tales of a
Security
Professional
Talking Identity
| Nishant
Kaushik's Look
at the World of
Identity
Management
TeamSHATTER
Techdulla
Technicalinfo.net
Blog
Technomagic
Tenable
Network
Security Blog
Terebrate
Tevora Blog
The .NET n00b
The Akamai
Blog
The Ashimmy
Blog
The AVIEN Blog
The Dark Visitor
The Essential
Home Computer
Security Blog

The Evident.io Blog
The Falcon’s View
The FYRM Blog
The Grey Corner
The InfoSec Blog
The IPS Guy
The IT Security Guy
The Kai Roer Blog
The Laws of Vulnerabilities
The Next CISO
The PhishLabs Blog
The Privacy Blog
The Security Advocate
The Security Mentor
The Security Practice
The Security Pub | The Security Pub
The Security Shoggoth
The Security Shoggoth
The Security Skeptic
The State of Security
The Symplified Blog
theharmonyguy.cc
Third Defense Blog
Thoughts On The Wire
Threat Center

Live Blog
Threat Research
ThreatFire Research Blog
Through the looking glass….
Trail of Bits Blog
Trey Ford – Security Spin Control
TriGeoSphere
Trusted Computing Group Blog
trustedsignal — blog
Trusteer blogs
tssci-security
un-excogitate
Uncommon Sense Security
Undead Security
Untangle
usability | security | freedom
varlogsecurity
VASCO Data Security – Blog
Venafi Blog
Veracode Blog
View From The Bunker
Vigilant Software Blog
Vincent Arnold
Virtue Security
VisibleRisk
Vittorio Bertocci
Voice of VOIPSA
VPN Haus
We Live Security »
Languages »
English
Webantix

Webroot Threat Blog
Websense Insights
When IT Meets Politics
WhiteHat Security Blog
Windows IT Pro
wirewatcher
WP White Security
Writing Secure Software
ZDNet | Zero Day RSS
Zen One
Zscaler Research
クリスマス・サンタのお話（Christmas-Santa Clause)

About SBN Home SBN – The Feed Blogger Hall of Fame
Security Blogger Awards