EXPLOIT GENERATION AND JAVASCRIPT ANALYSIS AUTOMATION WITH WINDBG

CSABA FITZL, MIKLOS DESBORDES-KORCSEV

Csaba Fitzl

- Just Google my name ☺
- Wife, 3 year old son
- Nr. 1 hobby: Hiking
- Plenty of unimportant certs, and some useful ones: OSWP, OSCP, OSCE, OSEE

Miklos Desbordes-Korcsev

- Off the grid (married, 2 kids)
- Not a cert collector, relevant here: GIAC REM
- Hobby: hiking, trying a new sport each year

PART 1: AUTOMATED EXPLOIT GENERATION

Exploit writing challenges

- Time consuming
- Heavily manual intensive process
  - Discovering memory layout
  - Finding bad characters
- While (exploit doesn't work == True):
  - Start process, attach debugger, crash, Modify exploit

Exploit writing methodology - BOF

- Standard process
  - Find EIP overwrite location
  - Examine memory layout, registries
  - Somehow jump to shellcode
  - Generate shellcode
  - Put all together

The Task

- A tool which can automate the entire exploit writing process
- From crash PoC to working Exploit
- If possible 0 manual interaction

The tool

- Written in Python
- Uses the "pykd" library to interact with WinDBG
- Will be released after the conference

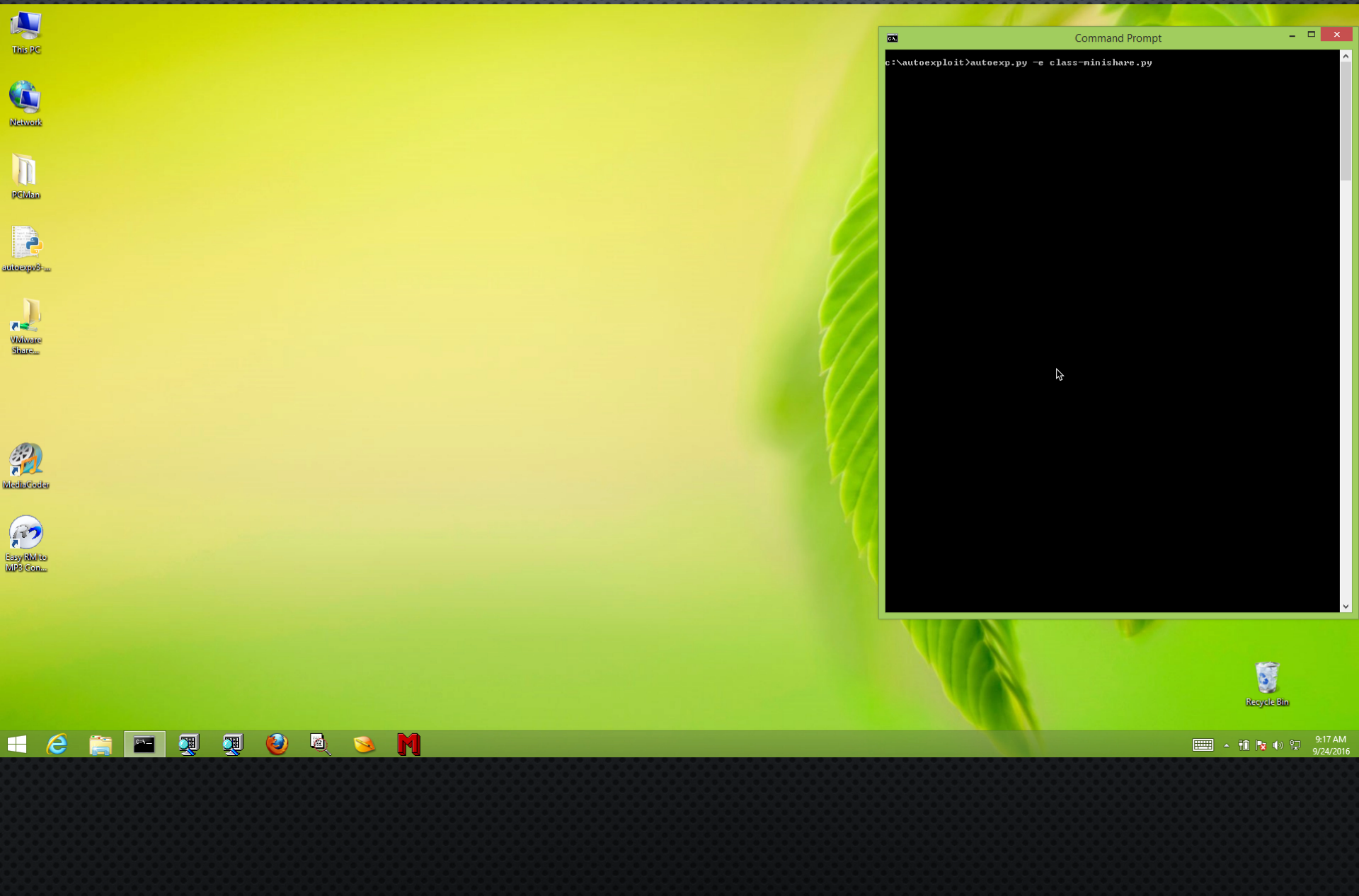
What can it do?

- Currently works for classic BoFs
- Can bypass ASLR
- Works for network and file based exploits
- Will create a successful exploit from a simple crash
- Automates the entire process (even finding bad characters!)
- No need to manually start the process / WinDBG

THE LOGIC

- Find EIP overwrite location / offset
- Find registers pointing to the buffers
- Find bad characters
- Find a way to jump to the shellcode (JMP, CALL, etc…)
- Generate shellcode
- Put it all together

DEMO #1 - MINISHARE


This PC
Network
PCMan
autoexpv3-...
VMware Share...
MediaCoder
Easy RM to MP3 Con...
Command Prompt
c:\autoexploit>autoexp.py -e class-minishare.py
Recycle Bin
9:17 AM
9/24/2016

How to use it?

- Some pre-work needs to be done
- Exploit is a Class
- Has to be populated with initial info (crash)

WHAT HAS TO BE CHANGED?

```
def exploit(self):
    """
    This function runs the actual exploit
    """
    sleep(1)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(('127.0.0.1',80))
    message = "GET " + ''.join(self.buffer) + " HTTP/1.1\r\n\r\n"
    sock.send(message)
    sock.close()
```

FUTURE?

- SEH based overflows
- More tricky jumps to shellcode
- DEP bypass

The flipside of the coin

Part 2: JavaScript analysis automation

Browser exploit reversing challenges

- Time consuming
- Challenging process
  - If analyzed with the browser's built-in debugger – finding the right breakpoints
  - If analyzed with external JS engine – no DOM
- Getting around anti-debugging, anti-reversing techniques

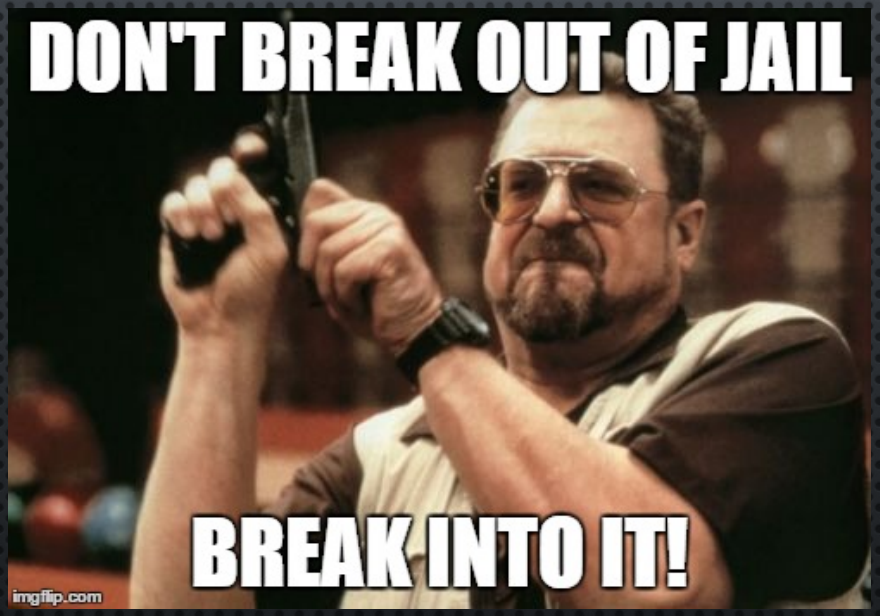
Browser Exploit Reversing Methodology

- Deobfuscate code
- Catch the functions before they execute
  - Breakpoint
  - Overload functions
- Locate the exploit code
- Understand shell code

THE TASK

- A TOOL WHICH CAN AUTOMATE THE JAVASCRIPT DEOBF
- MAKE IT IMPOSSIBLE FOR JS CODE TO FIGURE OUT IT IS B
- HAVE THE MALWARE RUN IN ITS NATURAL ENVIRONMENT
- STOP AT EXPLOIT
- MINIMUM MANUAL INTERACTION

SO?



The tool

- Harvesting the awesome power of WinDBG!
- With some Python helper code
- Uses the "pykd" extension for WinDBG
- Will be released after the conference

What can it do?

- Currently works for IE11
- Deobfuscates EVAL based code
- Stops at exploit
- Logs each session to new file
- Automates the entire process

Finding the peeking hole

- Find the right function
  - Best to use dynamic arguments
- Find the pointer to the argument
  - Chain.py to the rescue!

> !py chain find -a 0x0490AFC0 -l 1 -r esp -dw 20

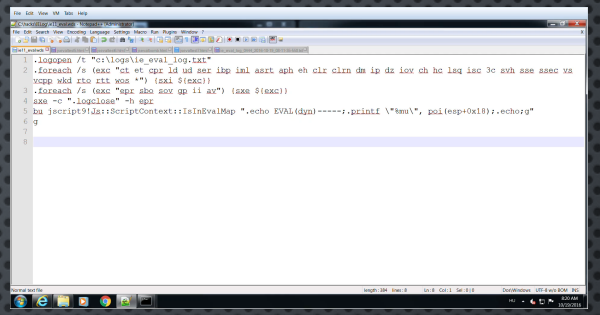
```
bu jscript9!Js::ScriptContext::IsInEvalMap
  ".echo EVAL(dyn)-----;.printf \"%mu\", poi(esp+0x18);.echo;g"
```

Automating things

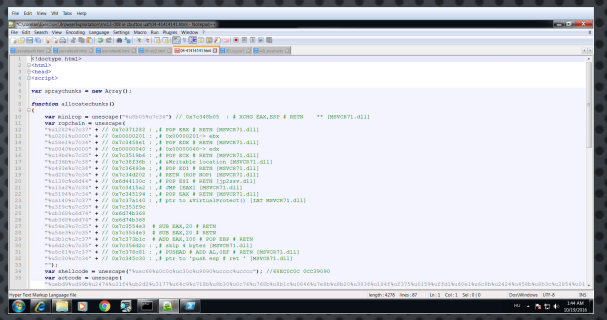
- Auto attaching windbg to browsers is a challenge
- The WDS
  - Set up a unique log file
  - Ignore unneeded breaks
  - Set up breaks for needed exception types
  - Set up a final means to close the logfile on exit
  - Set up the logging breakpoint

```
.foreach /s (exc "ct et cpr ld ud ser ibp iml asrt aph eh clr clrn dm ip dz iov
ch hc lsq isc 3c svh sse ssec vs vcpp wkd rto rtt wos *") {sxi ${exc}}
.foreach /s (exc "epr sbo sov gp ii av") {sxe ${exc}}
```

DEMO #1 – STRESS TEST OBFUSCATED JS



DEMO #2 – ACTUAL EXPLOIT



Future plans

- Document and release methodologies for MS Edge and Chrome
- Catching metasploit shellcode

SOURCE CODE

GITHUB.COM/THEEVILBIT/EXPLOIT_GENERATOR
GITHUB.COM/SZIMEUS/EVALYZER

CONTACT

- CSABA:
  - FITZL.CSABA@GMAIL.COM
  - TWITTER: @THEEVILBIT
- MIKLOS:
  - SZIMEUS@GMAIL.COM