Detection-Based Cybersecurity: Signature-Based Detection

CS 459/559: Science of Cyber Security
14th Lecture

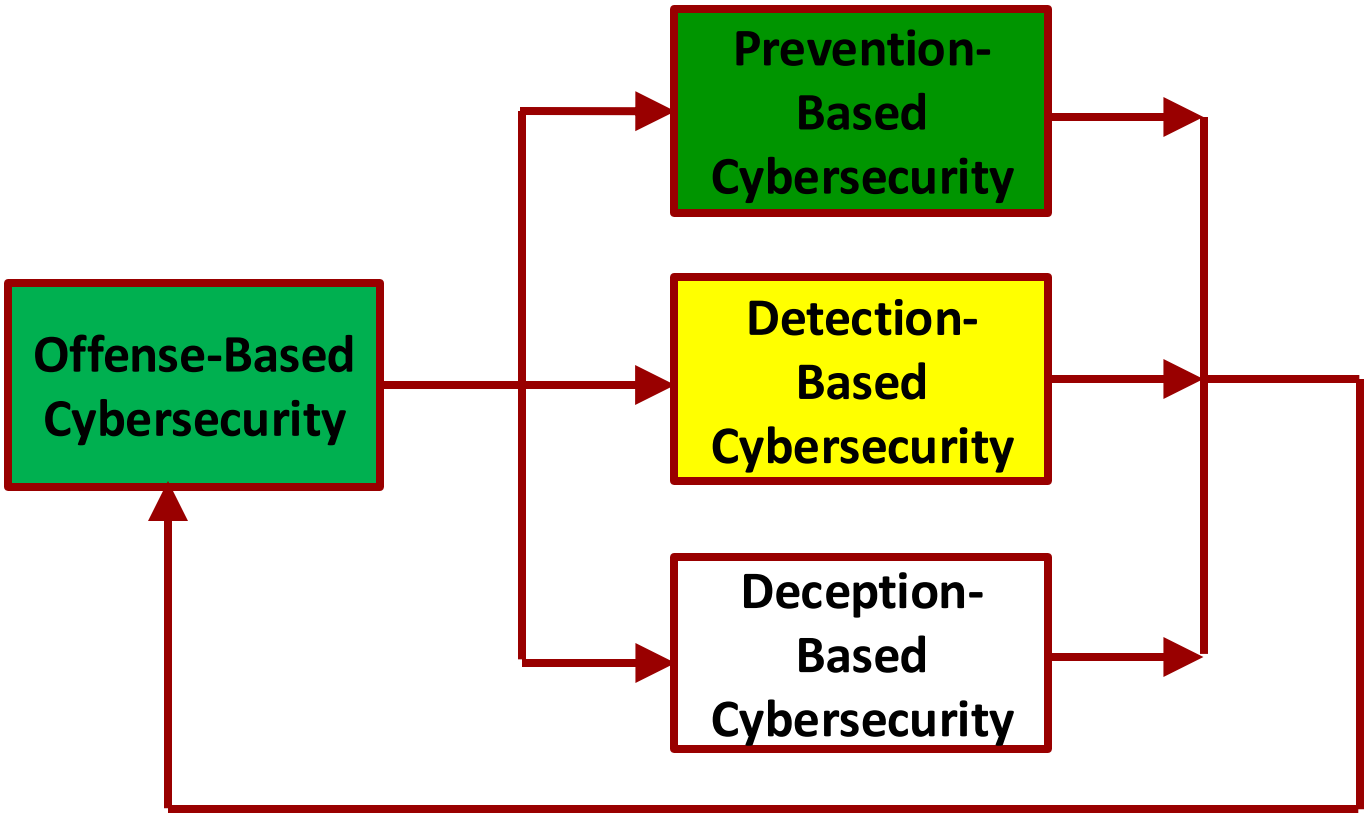
Instructor:

Guanhua Yan

Agenda

- ~~Quiz 1: September 29 (closed book)~~
- ~~Project 1 (offense): October 10~~
- Presentations: 11/17, 11/19, 11/24, 12/1, 12/3
- Quiz 2: November 12
- CTF competition: November 26
- Project 2 (defense): December 5
- Final report: December 15

Course structure



Outline

- **How to evaluate attack detection systems?**
- **What is signature-based detection?**
- **Example signature-based detection tools**
  - Yara: malware detection
  - Snort: network-based intrusion detection

How to evaluate attack detection systems?

Evaluation of attack detection system

Predicted Class

Actual Class	Class = Positive	Class = Negative
Class = Positive	True Positive (TP)	False Negative (FN)
Class = Negative	False Positive (FP)	True Negative (TN)

Confusion Matrix

- False positive (alarm) rate: FP / (FP + TN)
- False negative rate: FN / (TP + FN)
- Detection rate: TP / (TP + FN)

Example

- n = 1000 packets
- k = 250 real attacks
- I = 360 attacks (according to the detection system)
- 125 *real* attacks detected

- What's the false positive rate?
- What's the false negative rate?
- What's the detection rate?

TP, FN, FP, and TN

- n = 1000 packets
- k = 250 real attacks
- l = 360 attacks (according to the detection system)
- 125 *real* attacks detected

Confusion Matrix

Predicted Class

Actual Class	Class = Positive	Class = Negative
Class = Positive	True Positive (TP)	False Negative (FN)
Class = Negative	False Positive (FP)	True Negative (TN)

- TP = 125, FN: 250 – 125 = 125
- FP = 360 – 125 = 235, TN = 750 – 235 = 515

False positive rate

- n = 1000 packets
- k = 250 real attacks
- l = 360 attacks (according to the detection system)
- 125 *real* attacks detected

- TP = 125, FN: 250 – 125 = 125
- FP = 360 – 125 = 235, TN = 750 – 235 = 515

- What's the false positive rate? FP / (FP + TN) = 31.3%

False negative rate

- n = 1000 packets
- k = 250 real attacks
- l = 360 attacks (according to the detection system)
- 125 *real* attacks detected

- TP = 125, FN: 250 – 125 = 125
- FP = 360 – 125 = 235, TN = 750 – 235 = 515

- What's the false negative rate? FN / (TP + FN) = 50.0%

Detection rate

- n = 1000 packets
- k = 250 real attacks
- l = 360 attacks (according to the detection system)
- 125 *real* attacks detected

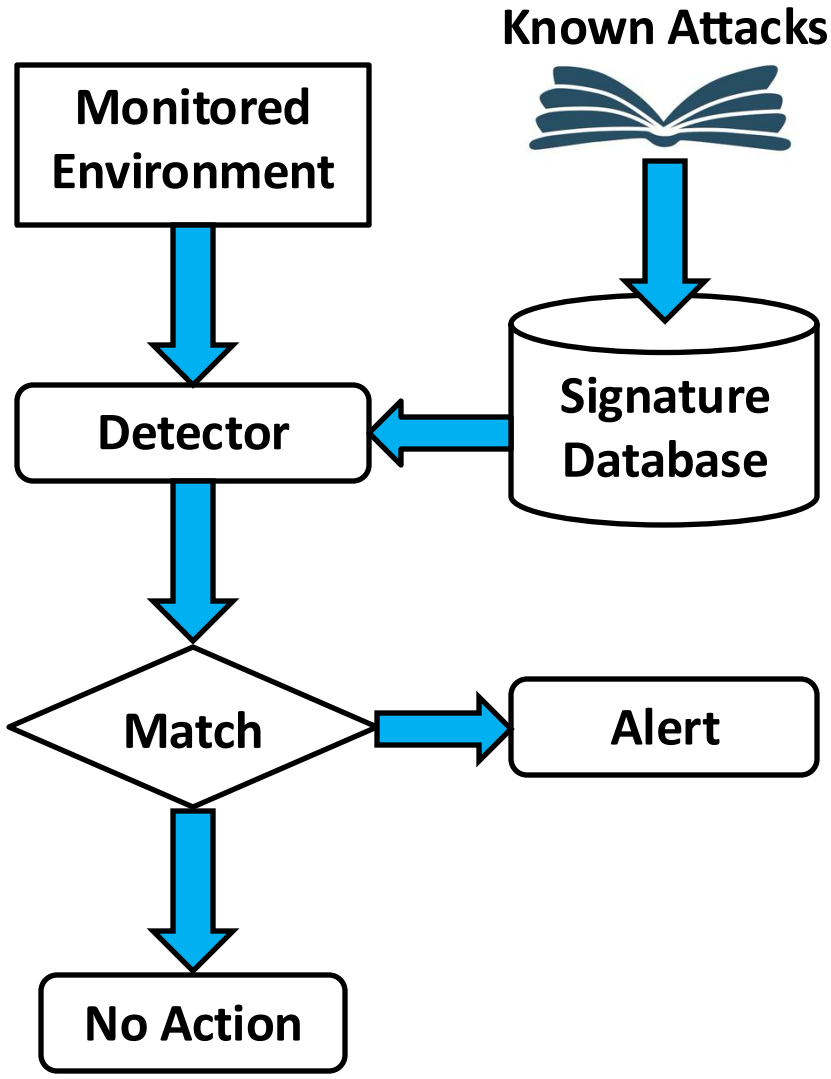
- TP = 125, FN: 250 – 125 = 125
- FP = 360 – 125 = 235, TN = 750 – 235 = 515

- What's the detection rate? TP / (TP + FN) = 50.0%

What is signature-based detection?

Signature-based detection

- **Signature-based detection is a security method that identifies known threats by comparing files, network traffic, or behaviors against a database of predefined "signatures" or patterns.**
- **This is like a digital fingerprint for known attack methods, allowing security systems to quickly detect and block them.**



Types of signatures

- **Hash-based: This technique uses a unique hash value (a fixed-size output) for a known malicious file. If a file's hash matches one in the database, it is flagged as a threat.**
- **String-based: This method scans for specific sequences of characters/bytes associated with known malware, but it can be easily evaded by attackers who obfuscate or encrypt their code.**
- **Behavioral patterns: Some signatures are based on the behavioral patterns of malware, such as repeated failed login attempts or unusual data transfers.**
- **Network signatures: Used by Intrusion Detection Systems (IDS), these signatures identify malicious network activity, such as specific command structures or unusual traffic patterns.**

Types of signatures

- Hash-based: This technique uses a unique hash value (a fixed-size output) for a known malicious file. If a file's hash matches one in the database, it is flagged as a threat.
- String-based: This method scans for specific sequences of characters/bytes associated with known malware, but it can be easily evaded by attackers who obfuscate or encrypt their code.
- Behavioral patterns: Some signatures are based on the behavioral patterns of malware, such as repeated failed login attempts or unusual data transfers.
- Network signatures: Used by Intrusion Detection Systems (IDS), these signatures identify malicious network activity, such as specific command structures or unusual traffic patterns.

Types of signatures

- Hash-based: This technique uses a unique hash value (a fixed-size output) for a known malicious file. If a file's hash matches one in the database, it is flagged as a threat.
- String-based: This method scans for specific sequences of characters/bytes associated with known malware, but it can be easily evaded by attackers who obfuscate or encrypt their code.
- Behavioral patterns: Some signatures are based on the behavioral patterns of malware, such as repeated failed login attempts or unusual data transfers.
- Network signatures: Used by Intrusion Detection Systems (IDS), these signatures identify malicious network activity, such as specific command structures or unusual traffic patterns.

Types of signatures

- Hash-based: This technique uses a unique hash value (a fixed-size output) for a known malicious file. If a file's hash matches one in the database, it is flagged as a threat.
- String-based: This method scans for specific sequences of characters/bytes associated with known malware, but it can be easily evaded by attackers who obfuscate or encrypt their code.
- Behavioral patterns: Some signatures are based on the behavioral patterns of malware, such as repeated failed login attempts or unusual data transfers.
- Network signatures: Used by Intrusion Detection Systems (IDS), these signatures identify malicious network activity, such as specific command structures or unusual traffic patterns.

Advantages

- High accuracy for known threats: Because it relies on an exact match, signature-based detection is very effective at identifying previously documented threats, resulting in a low false-positive rate.
- Speed: For known threats, the scanning and matching process is quick and efficient, enabling rapid detection and response.
- Low resource consumption: Compared to other more complex detection methods, signature-based detection requires fewer computational resources.
- Ease of implementation: The concept is simple and can be easily integrated into existing security infrastructures.
- Easy to share: repositories of signatures

Advantages

- High accuracy for known threats: Because it relies on an exact match, signature-based detection is very effective at identifying previously documented threats, resulting in a low false-positive rate.
- Speed: For known threats, the scanning and matching process is quick and efficient, enabling rapid detection and response.
- Low resource consumption: Compared to other more complex detection methods, signature-based detection requires fewer computational resources.
- Ease of implementation: The concept is simple and can be easily integrated into existing security infrastructures.
- Easy to share: repositories of signatures

Advantages

- High accuracy for known threats: Because it relies on an exact match, signature-based detection is very effective at identifying previously documented threats, resulting in a low false-positive rate.
- Speed: For known threats, the scanning and matching process is quick and efficient, enabling rapid detection and response.
- Low resource consumption: Compared to other more complex detection methods, signature-based detection requires fewer computational resources.
- Ease of implementation: The concept is simple and can be easily integrated into existing security infrastructures.
- Easy to share: repositories of signatures

Advantages

- High accuracy for known threats: Because it relies on an exact match, signature-based detection is very effective at identifying previously documented threats, resulting in a low false-positive rate.
- Speed: For known threats, the scanning and matching process is quick and efficient, enabling rapid detection and response.
- Low resource consumption: Compared to other more complex detection methods, signature-based detection requires fewer computational resources.
- Ease of implementation: The concept is simple and can be easily integrated into existing security infrastructures.
- Easy to share: repositories of signatures

Advantages

- High accuracy for known threats: Because it relies on an exact match, signature-based detection is very effective at identifying previously documented threats, resulting in a low false-positive rate.
- Speed: For known threats, the scanning and matching process is quick and efficient, enabling rapid detection and response.
- Low resource consumption: Compared to other more complex detection methods, signature-based detection requires fewer computational resources.
- Ease of implementation: The concept is simple and can be easily integrated into existing security infrastructures.
- Easy to share: repositories of signatures

Limitations

- **Vulnerable to new threats: The primary weakness of signature-based detection is its inability to detect new or "zero-day" threats that do not yet have a signature in the database.**
- **Reactive, not proactive: This method only detects threats that have already been discovered and analyzed. It cannot protect against novel attacks.**
- **Evasion techniques: Cyber attackers can use polymorphic or metamorphic malware, which changes its code to evade detection, rendering static signatures useless.**
- **Dependence on updates: Security is only as good as the database of signatures. Systems must be constantly updated to keep pace with emerging threats.**

Limitations

- Vulnerable to new threats: The primary weakness of signature-based detection is its inability to detect new or "zero-day" threats that do not yet have a signature in the database.
- Reactive, not proactive: This method only detects threats that have already been discovered and analyzed. It cannot protect against novel attacks.
- Evasion techniques: Cyber attackers can use polymorphic or metamorphic malware, which changes its code to evade detection, rendering static signatures useless.
- Dependence on updates: Security is only as good as the database of signatures. Systems must be constantly updated to keep pace with emerging threats.

Limitations

- Vulnerable to new threats: The primary weakness of signature-based detection is its inability to detect new or "zero-day" threats that do not yet have a signature in the database.
- Reactive, not proactive: This method only detects threats that have already been discovered and analyzed. It cannot protect against novel attacks.
- Evasion techniques: Cyber attackers can use polymorphic or metamorphic malware, which changes its code to evade detection, rendering static signatures useless.
- Dependence on updates: Security is only as good as the database of signatures. Systems must be constantly updated to keep pace with emerging threats.

Limitations

- Vulnerable to new threats: The primary weakness of signature-based detection is its inability to detect new or "zero-day" threats that do not yet have a signature in the database.
- Reactive, not proactive: This method only detects threats that have already been discovered and analyzed. It cannot protect against novel attacks.
- Evasion techniques: Cyber attackers can use polymorphic or metamorphic malware, which changes its code to evade detection, rendering static signatures useless.
- Dependence on updates: Security is only as good as the database of signatures. Systems must be constantly updated to keep pace with emerging threats.

Use of signature-based detection

- **Malware detection**
  - Scan suspicious files to identify malware attacks
- **Network-based IDS**
  - Monitor network traffic and detect new intrusions

Malware Scanning: Yara

Introduction
What is YARA?

- „The pattern matching swiss knife for malware researchers (and everyone else)“
- Hosted on GitGub
  http://plusvic.github.io/yara/
- **Pattern matching**:
  - strings (ASCII, UCS-2)
  - regular expressions
  - binary patterns (hex strings)
- **Classification**:
  - on input: combination of strings
  - on output: tags, metadata

Introduction
What is YARA?

```
rule my_example : tag1 tag2 tag3
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = { 6A 40 68 00 30 00 00 6A 14 8D 91 }
        $b = /[0-9a-f]{32}/
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or ( $b and $c)
}
```

Introduction
What is YARA?

```
rule my_example : tag1 tag2 tag3
{
    meta:
```

- **Curly brackets `{}` :** Indicate that the enclosed content is a hexadecimal string, not a plain text string.

```
    strings:
        $a = { 6A 40 68 00 30 00 00 6A 14 8D 91 }
        $b = /[0-9a-f]{32}/
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or ( $b and $c)
}
```

The signature is a plaintext string

Introduction
What is YARA?

```
rule my_exan
{
    meta:
        descripti
        threat _le
        in_the_w

    strings:
        $a = { 6A 40 68 00 30 00 00 6A 14 8D 91 }
        $b = /[0-9a-f]{32}/
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or ( $b and $c)
}
```

Breakdown of the regular expression

- `/ ... /` : In YARA, regular expressions are enclosed in forward slashes.
- `[0-9a-f]` : This is a character class that matches a single character. It will match any digit from 0 through 9 or any lowercase letter from a through f. This character set defines the standard characters used in hexadecimal notation.
- `{32}` : This is a quantifier that specifies the character class must be repeated exactly 32 times.

Introduction
How can YARA help me?

- A „better grep“
- Use cases:
  - Finding interesting entries on pastebin.com ...
  - Triage data
  - Preprocess files to direct reverse engineering efforts
- Integrate it into your projects:
  - C library
  - Python bindings
    https://github.com/plusvic/yara/tree/master/yara-python
  - Ruby bindings
    https://github.com/SpiderLabs/yara-ruby

Introduction
How can YARA help me?

- YARA rules are supported by security products and services
  - FireEye appliances
  - Fidelis XPS
  - RSA ECAT
  - Volatility
  - ThreadConnect threat intelligence exchange
  - VirusTotal Intelligence
  - ...

Writing YARA Rules

Identify executable files
A simple specification for PE files

- Task: To find any files in Portable Executable („PE“) format
- Simple specification: File must contain the strings „MZ“ and „PE“

```
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 c8 00 00 00  |................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000080  65 cd 43 c7 21 ac 2d 94  21 ac 2d 94 21 ac 2d 94  |e.C.!.-.!.-.!.-.|
00000090  21 ac 2c 94 25 ac 2d 94  e2 a3 70 94 24 ac 2d 94  |!.,.%.-...p.$.-.|
000000a0  c9 b3 26 94 23 ac 2d 94  52 69 63 68 21 ac 2d 94  |..&.#.-.Rich!.-.|
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 03 00  |........PE..L...|
```

Identify executable files
Adding the condition

- A portable executable file MUST contain both strings. So, add the proper condition:

```
rule PE_file
{
        strings:
                $mz = "MZ"
                $pe = "PE"
        condition:
                $mz and $pe
}
```

- Test your rule file:

```
$ yara -r executable.yara /yara/malware
```

Identify executable files
Refining the condition

- More constraints:
  - „MZ“ at offset 0
  - UInt32 at offset 0x3c points to „PE“
- Refine your condition section:

```
condition:
        ($mz at 0) and
        ($pe at (uint32(0x3c)))
```

- Test your rule file again:

```
$ yara -r executable.yara /yara/malware
```

Identify executable files
The final rule

- This is how your rule should look like:

```
rule PE_file
{
!       strings:
!       !           $mz = "MZ"
!       !           $pe = "PE"

!       condition:
!       !           ($mz at 0) and
!       !           ($pe at (uint32(0x3c)))
}
```

Obfuscation: Move Single Byte

- Can you spot the registry key name?

```
00415393  C6 45 CC 53 C6 45 CD 6F C6 45 CE 66 C6 45 CF 74  .E.S.E.o.E.f.E.t
004153A3  C6 45 D0 77 C6 45 D1 61 C6 45 D2 72 C6 45 D3 65  .E.w.E.a.E.r.E.e
004153B3  C6 45 D4 5C C6 45 D5 4D C6 45 D6 69 C6 45 D7 63  .E.\.E.M.E.i.E.c
004153C3  C6 45 D8 72 C6 45 D9 6F C6 45 DA 73 C6 45 DB 6F  .E.r.E.o.E.s.E.o
004153D3  C6 45 DC 66 C6 45 DD 74 C6 45 DE 5C C6 45 DF 57  .E.f.E.t.E.\.E.W
004153E3  C6 45 E0 69 C6 45 E1 6E C6 45 E2 64 C6 45 E3 6F  .E.i.E.n.E.d.E.o
004153F3  C6 45 E4 77 C6 45 E5 73 C6 45 E6 5C C6 45 E7 43  .E.w.E.s.E.\.E.C
00415403  C6 45 E8 75 C6 45 E9 72 C6 45 EA 72 C6 45 EB 65  .E.u.E.r.E.r.E.e
00415413  C6 45 EC 6E C6 45 ED 74 C6 45 EE 56 C6 45 EF 65  .E.n.E.t.E.V.E.e
00415423  C6 45 F0 72 C6 45 F1 73 C6 45 F2 69 C6 45 F3 6F  .E.r.E.s.E.i.E.o
00415433  C6 45 F4 6E C6 45 F5 5C C6 45 F6 52 C6 45 F7 75  .E.n.E.\.E.R.E.u
00415443  C6 45 F8 6E                                     .E.n
```

Obfuscation: Move Single Byte

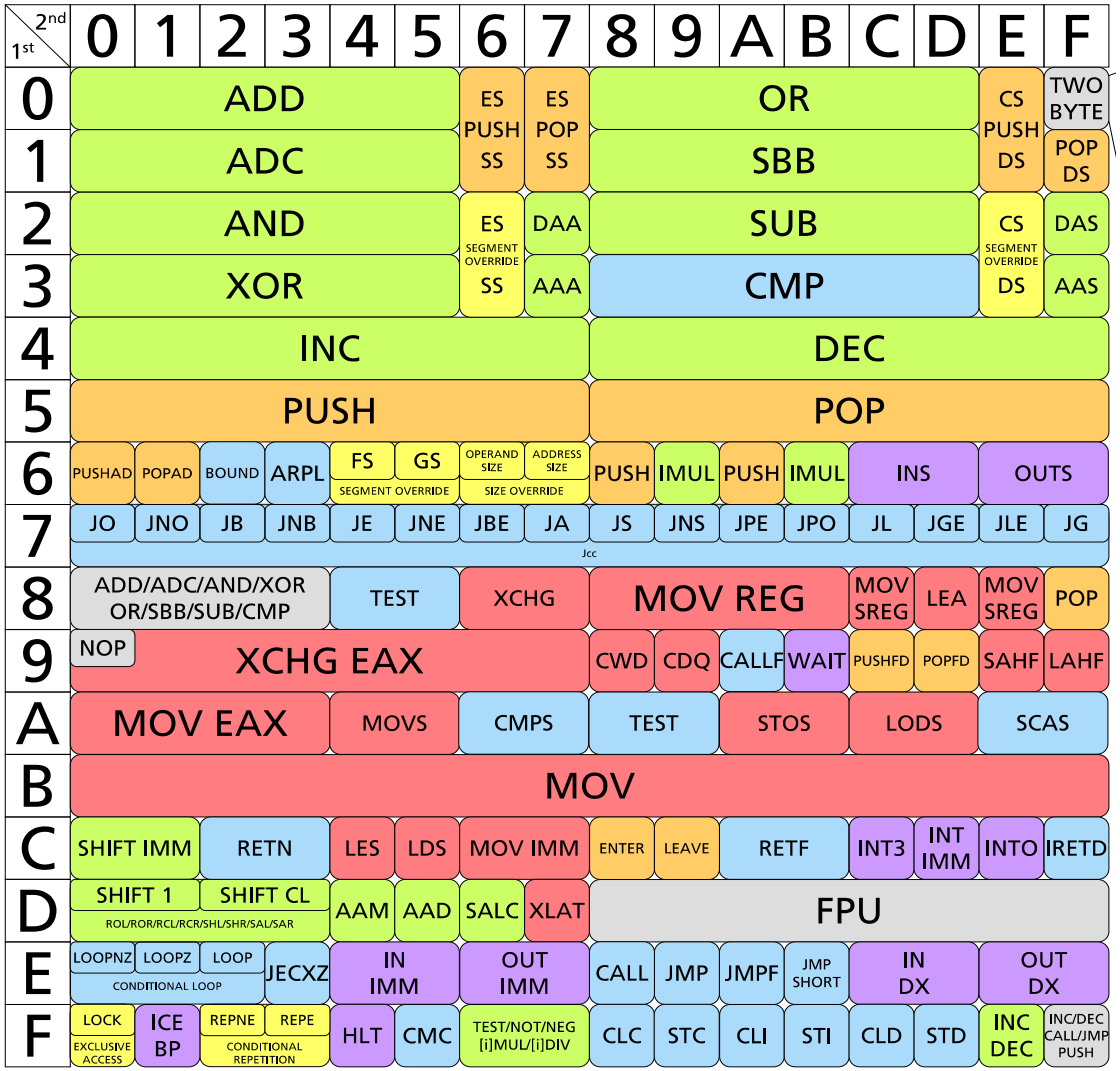
- Can you spot the registry key name?

```
00415393  C6 45 CC 53 C6 45 CD 6F C6 45 CE 66 C6 45 CF 74   .E.S.E.o.E.f.E.t
004153A3  C6 45 D0 77 C6 45 D1 61 C6 45 D2 72 C6 45 D3 65   .E.w.E.a.E.r.E.e
004153B3  C6 45 D4 5C C6 45 D5 4D C6 45 D6 69 C6 45 D7 63   .E.\.E.M.E.i.E.c
004153C3  C6 45 D8 72 C6 45 D9 6F C6 45 DA 73 C6 45 DB 6F   .E.r.E.o.E.s.E.o
004153D3  C6 45 DC 66 C6 45 DD 74 C6 45 DE 5C C6 45 DF 57   .E.f.E.t.E.\.E.W
004153E3  C6 45 E0 69 C6 45 E1 6E C6 45 E2 64 C6 45 E3 6F   .E.i.E.n.E.d.E.o
004153F3  C6 45 E4 77 C6 45 E5 73 C6 45 E6 5C C6 45 E7 43   .E.w.E.s.E.\.E.C
00415403  C6 45 E8 75 C6 45 E9 72 C6 45 EA 72 C6 45 EB 65   .E.u.E.r.E.r.E.e
00415413  C6 45 EC 6E C6 45 ED 74 C6 45 EE 56 C6 45 EF 65   .E.n.E.t.E.V.E.e
00415423  C6 45 F0 72 C6 45 F1 73 C6 45 F2 69 C6 45 F3 6F   .E.r.E.s.E.i.E.o
00415433  C6 45 F4 6E C6 45 F5 5C C6 45 F6 52 C6 45 F7 75   .E.n.E.\.E.R.E.u
00415443  C6 45 F8 6E                                      .E.n
```

X86 opcode???

Obfuscation: Move Single Byte

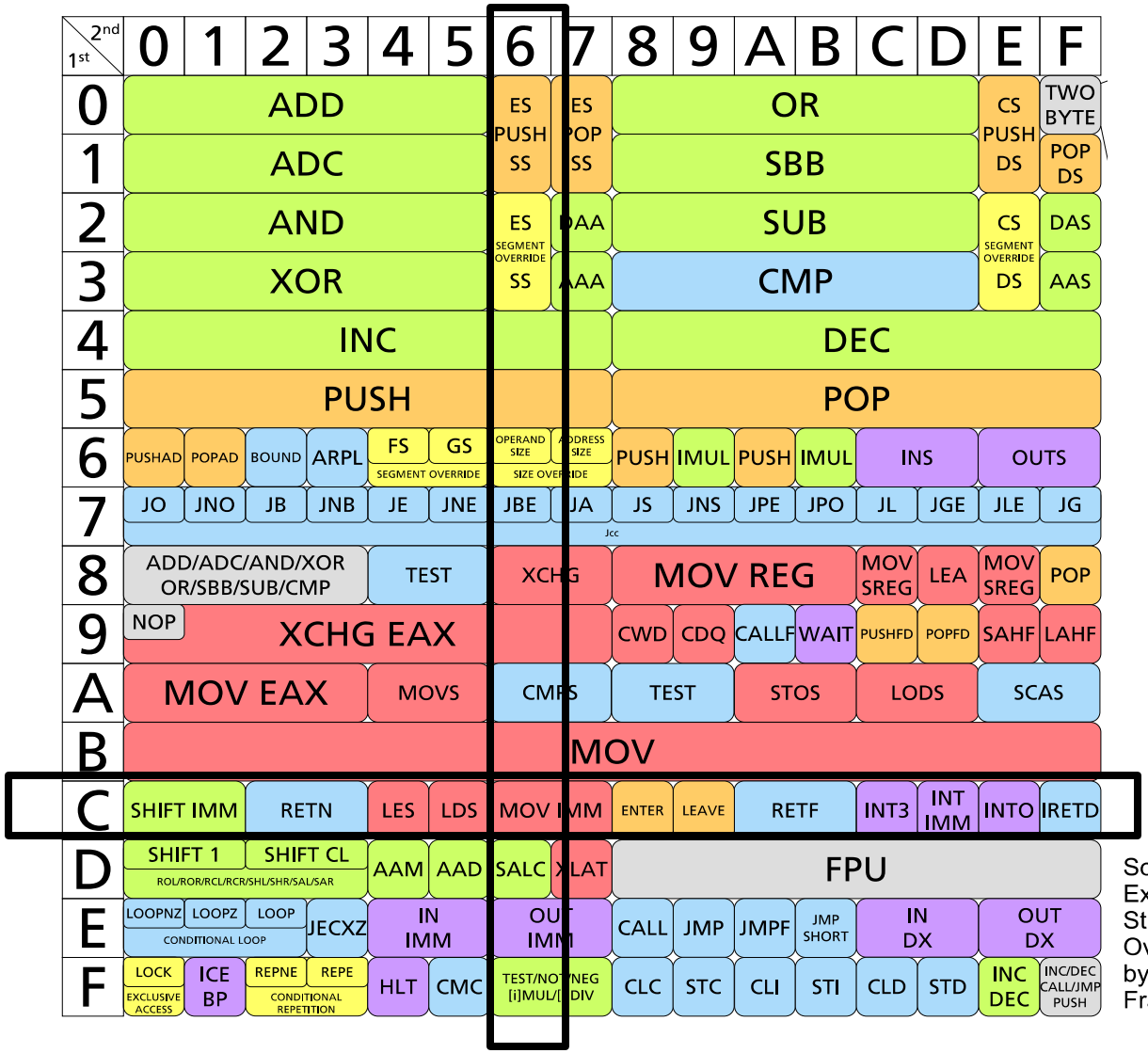
Find the opcode for 0xc6



Source:
Extract from „x86 Opcode Structure and Instruction Overview“
by Daniel Plohmann, Fraunhofer FKIE

Obfuscation: Move Single Byte

Find the opcode for 0xc6



Source:
Extract from „x86 Opcode Structure and Instruction Overview“
by Daniel Plohmann, Fraunhofer FKIE

Obfuscation: Move Single Byte

Read the manual page for MOV

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
REX.W + A3	MOV *r/m64*, RAX	D	Valid	N.E.	Move RAX to (*r/m64*)
B0+ *rb*	MOV *r8*, *imm8*	E	Valid	Valid	Move *imm8* to *r8*
REX + B0+ *rb*	MOV *r8****, *imm8*	E	Valid	N.E.	Move *imm8* to *r8*
B8+ *rw*	MOV *r16*, *imm16*	E	Valid	Valid	Move *imm16* to *r16*
B8+ *rd*	MOV *r32*, *imm32*	E	Valid	Valid	Move *imm32* to *r32*
REX.W + B8+ *rd*	MOV *r64*, *imm64*	E	Valid	N.E.	Move *imm64* to *r64*
C6 /0	MOV *r/m8*, *imm8*	F	Valid	Valid	Move *imm8* to *r/m8*
REX + C6 /0	MOV *r/m8****, *imm8*	F	Valid	N.E.	Move *imm8* to *r/m8*
C7 /0	MOV *r/m16*, *imm16*	F	Valid	Valid	Move *imm16* to *r/m16*
C7 /0	MOV *r/m32*, *imm32*	F	Valid	Valid	Move *imm32* to *r/m32*
REX.W + C7 /0	MOV *r/m64*, *imm32*	F	Valid	N.E.	Move *imm32* sign extended to 64-bits to *r/m64*

r/m8 -- A byte operand that is either the contents of a byte general-purpose register (AL, CL, DL, BL, AH, CH, DH, BH, BPL, SPL, DIL and SIL) or a byte from memory.

Obfuscation: Move Single Byte

- Can you spot the registry key name?

```
00415393  C6 45 CC 53 C6 45 CD 6F C6 45 CE 66 C6 45 CF 74  .E.S.E.o.E.f.E.t
004153A3  C6 45 D0 77 C6 45 D1 61 C6 45 D2 72 C6 45 D3 65  .E.w.E.a.E.r.E.e
004153B3  C6 45 D4 5C C6 45 D5 4D C6 45 D6 69 C6 45 D7 63  .E.\.E.M.E.i.E.c
004153C3  C6 45 D8 72 C6 45 D9 6F C6 45 DA 73 C6 45 DB 6F  .E.r.E.o.E.s.E.o
004153D3  C6 45 DC 66 C6 45 DD 74 C6 45 DE 5C C6 45 DF 57  .E.f.E.t.E.\.E.W
004153E3  C6 45 E0 69 C6 45 E1 6E C6 45 E2 64 C6 45 E3 6F  .E.i.E.n.E.d.E.o
004153F3  C6 45 E4 77 C6 45 E5 73 C6 45 E6 5C C6 45 E7 43  .E.w.E.s.E.\.E.C
00415403  C6 45 E8 75 C6 45 E9 72 C6 45 EA 72 C6 45 EB 65  .E.u.E.r.E.r.E.e
00415413  C6 45 EC 6E C6 45 ED 74 C6 45 EE 56 C6 45 EF 65  .E.n.E.t.E.V.E.e
00415423  C6 45 F0 72 C6 45 F1 73 C6 45 F2 69 C6 45 F3 6F  .E.r.E.s.E.i.E.o
00415433  C6 45 F4 6E C6 45 F5 5C C6 45 F6 52 C6 45 F7 75  .E.n.E.\.E.R.E.u
00415443  C6 45 F8 6E                                     .E.n
```

X86 opcode???

Obfuscation: Move Single Byte

Find the register and addressing mode for 0x45

Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) (In decimal) /digit (Opcode) (In binary) REG=			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--][--][1]		100	04	0C	14	1C	24	2C	34	3C
disp32[2]		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
[EAX]+disp8[3]	01	000	40	48	50	58	60	68	70	78
[ECX]+disp8		001	41	49	51	59	61	69	71	79
[EDX]+disp8		010	42	4A	52	5A	62	6A	72	7A
[EBX]+disp8		011	43	4B	53	5B	63	6B	73	7B
[--][--]+disp8		100	44	4C	54	5C	64	6C	74	7C
[EBP]+disp8		101	45	4D	55	5D	65	6D	75	7D
[ESI]+disp8		110	46	4E	56	5E	66	6E	76	7E
[EDI]+disp8		111	47	4F	57	5F	67	6F	77	7F
[EAX]+disp32	10	000	80	88	90	98	A0	A8	B0	B8

Obfuscation: Move Single Byte

Reveal the string

- Single byte MOVes are a common technique to obfuscate strings.

```
0000:00415393      mov     [ebp+SubKey],      'S'   ; C6 45 CC 53
0000:00415397      mov     [ebp+SubKey+1],    'o'   ; C6 45 CD 6F
0000:0041539B      mov     [ebp+SubKey+2],    'f'   ; C6 45 CE 66
0000:0041539F      mov     [ebp+SubKey+3],    't'   ; C6 45 CF 74
0000:004153A3      mov     [ebp+SubKey+4],    'w'   ; C6 45 D0 77
0000:004153A7      mov     [ebp+SubKey+5],    'a'   ; C6 45 D1 61
0000:004153AB      mov     [ebp+SubKey+6],    'r'   ; C6 45 D2 72
0000:004153AF      mov     [ebp+SubKey+7],    'e'   ; C6 45 D3 65
0000:004153B3      mov     [ebp+SubKey+8],    '\'   ; C6 45 D4 5C
0000:004153B7      mov     [ebp+SubKey+9],    'M'!  ; C6 45 D5 4D
0000:004153BB      mov     [ebp+SubKey+0Ah], 'i'!   ; C6 45 D6 69
0000:004153BF      mov     [ebp+SubKey+0Bh], 'c'!   ; C6 45 D7 63
0000:004153C3      mov     [ebp+SubKey+0Ch], 'r'!   ; C6 45 D8 72
0000:004153C7      mov     [ebp+SubKey+0Dh], 'o'!   ; C6 45 D9 6F
0000:004153CB      mov     [ebp+SubKey+0Eh], 's'!   ; C6 45 DA 73
0000:004153CF      mov     [ebp+SubKey+0Fh], 'o'!   ; C6 45 DB 6F
0000:004153D3      mov     [ebp+SubKey+10h], 'f'!   ; C6 45 DC 66
0000:004153D7      mov     [ebp+SubKey+11h], 't'!   ; C6 45 DD 74
```

Obfuscation: Move Single Byte

Reveal the string

- Single byte MOVes are a common technique to obfuscate strings.

```
0000:00415393      mov     [ebp+SubKey],     'S'   ; C6 45 CC 53
0000:00415397      mov     [ebp+SubKey+1],   'o'   ; C6 45 CD 6F
0000:0041539B      mov     [ebp+SubKey+2],   'f'   ; C6 45 CE 66
0000:0041539F      mov     [ebp+SubKey+3],   't'   ; C6 45 CF 74
0000:004153A3      mov     [ebp+SubKey+4],   'w'   ; C6 45 D0 77
0000:004153A7      mov     [ebp+SubKey+5],   'a'   ; C6 45 D1 61
0000:004153AB      mov     [ebp+SubKey+6],   'r'   ; C6 45 D2 72
0000:004153AF      mov     [ebp+SubKey+7],   'e'   ; C6 45 D3 65
0000:004153B3      mov     [ebp+SubKey+8],   '\'   ; C6 45 D4 5C
0000:004153B7      mov     [ebp+SubKey+9],   'M'!  ; C6 45 D5 4D
0000:004153BB      mov     [ebp+SubKey+0Ah], 'i'!  ; C6 45 D6 69
0000:004153BF      mov     [ebp+SubKey+0Bh], 'c'!  ; C6 45 D7 63
0000:004153C3      mov     [ebp+SubKey+0Ch], 'r'!  ; C6 45 D8 72
0000:004153C7      mov     [ebp+SubKey+0Dh], 'o'!  ; C6 45 D9 6F
0000:004153CB      mov     [ebp+SubKey+0Eh], 's'!  ; C6 45 DA 73
0000:004153CF      mov     [ebp+SubKey+0Fh], 'o'!  ; C6 45 DB 6F
0000:004153D3      mov     [ebp+SubKey+10h], 'f'!  ; C6 45 DC 66
0000:004153D7      mov     [ebp+SubKey+11h], 't'!  ; C6 45 DD 74
```

Obfuscation: Move Single Byte

Reveal the string

- Single byte MOVes are a common technique to obfuscate strings.

```
0000:00415393      mov     [ebp+SubKey],     'S'   ; C6 45 CC 53
0000:00415397      mov     [ebp+SubKey+1],   'o'   ; C6 45 CD 6F
0000:0041539B      mov     [ebp+SubKey+2],   'f'   ; C6 45 CE 66
0000:0041539F      mov     [ebp+SubKey+3],   't'   ; C6 45 CF 74
0000:004153A3      mov     [ebp+SubKey+4],   'w'   ; C6 45 D0 77
0000:004153A7      mov     [ebp+SubKey+5],   'a'   ; C6 45 D1 61
0000:004153AB      mov     [ebp+SubKey+6],   'r'   ; C6 45 D2 72
0000:004153AF      mov     [ebp+SubKey+7],   'e'   ; C6 45 D3 65
0000:004153B3      mov     [ebp+SubKey+8],   '\'   ; C6 45 D4 5C
0000:004153B7      mov     [ebp+SubKey+9],   'M'!  ; C6 45 D5 4D
0000:004153BB      mov     [ebp+SubKey+0Ah], 'i'!  ; C6 45 D6 69
0000:004153BF      mov     [ebp+SubKey+0Bh], 'c'!  ; C6 45 D7 63
0000:004153C3      mov     [ebp+SubKey+0Ch], 'r'!  ; C6 45 D8 72
0000:004153C7      mov     [ebp+SubKey+0Dh], 'o'!  ; C6 45 D9 6F
0000:004153CB      mov     [ebp+SubKey+0Eh], 's'!  ; C6 45 DA 73
0000:004153CF      mov     [ebp+SubKey+0Fh], 'o'!  ; C6 45 DB 6F
0000:004153D3      mov     [ebp+SubKey+10h], 'f'!  ; C6 45 DC 66
0000:004153D7      mov     [ebp+SubKey+11h], 't'!  ; C6 45 DD 74
```

Obfuscation: Move Single Byte

Reveal the string

- Single byte MOVes are a common technique to obfuscate strings.

```
0000:00415393      mov     [ebp+SubKey],     'S'   ; C6 45 CC 53
0000:00415397      mov     [ebp+SubKey+1],   'o'   ; C6 45 CD 6F
0000:0041539B      mov     [ebp+SubKey+2],   'f'   ; C6 45 CE 66
0000:0041539F      mov     [ebp+SubKey+3],   't'   ; C6 45 CF 74
0000:004153A3      mov     [ebp+SubKey+4],   'w'   ; C6 45 D0 77
0000:004153A7      mov     [ebp+SubKey+5],   'a'   ; C6 45 D1 61
0000:004153AB      mov     [ebp+SubKey+6],   'r'   ; C6 45 D2 72
0000:004153AF      mov     [ebp+SubKey+7],   'e'   ; C6 45 D3 65
0000:004153B3      mov     [ebp+SubKey+8],   '\'   ; C6 45 D4 5C
0000:004153B7      mov     [ebp+SubKey+9],   'M'!  ; C6 45 D5 4D
0000:004153BB      mov     [ebp+SubKey+0Ah], 'i'!  ; C6 45 D6 69
0000:004153BF      mov     [ebp+SubKey+0Bh], 'c'!  ; C6 45 D7 63
0000:004153C3      mov     [ebp+SubKey+0Ch], 'r'!  ; C6 45 D8 72
0000:004153C7      mov     [ebp+SubKey+0Dh], 'o'!  ; C6 45 D9 6F
0000:004153CB      mov     [ebp+SubKey+0Eh], 's'!  ; C6 45 DA 73
0000:004153CF      mov     [ebp+SubKey+0Fh], 'o'!  ; C6 45 DB 6F
0000:004153D3      mov     [ebp+SubKey+10h], 'f'!  ; C6 45 DC 66
0000:004153D7      mov     [ebp+SubKey+11h], 't'!  ; C6 45 DD 74
```

“Software\Microsoft”

Obfuscation: Move Single Byte
Develop a signature

- Signature:
  - 0xC6 0x45 is a constant (opcode and r/m8)
  - disp8 (index) is variable, but restricted to a single byte
  - the character (imm8) is variable, but also restricted to a single byte
- Pattern: C6 45 ?? ?? C6 45 ?? ?? C6 45 ...

Obfuscation: Move Single Byte

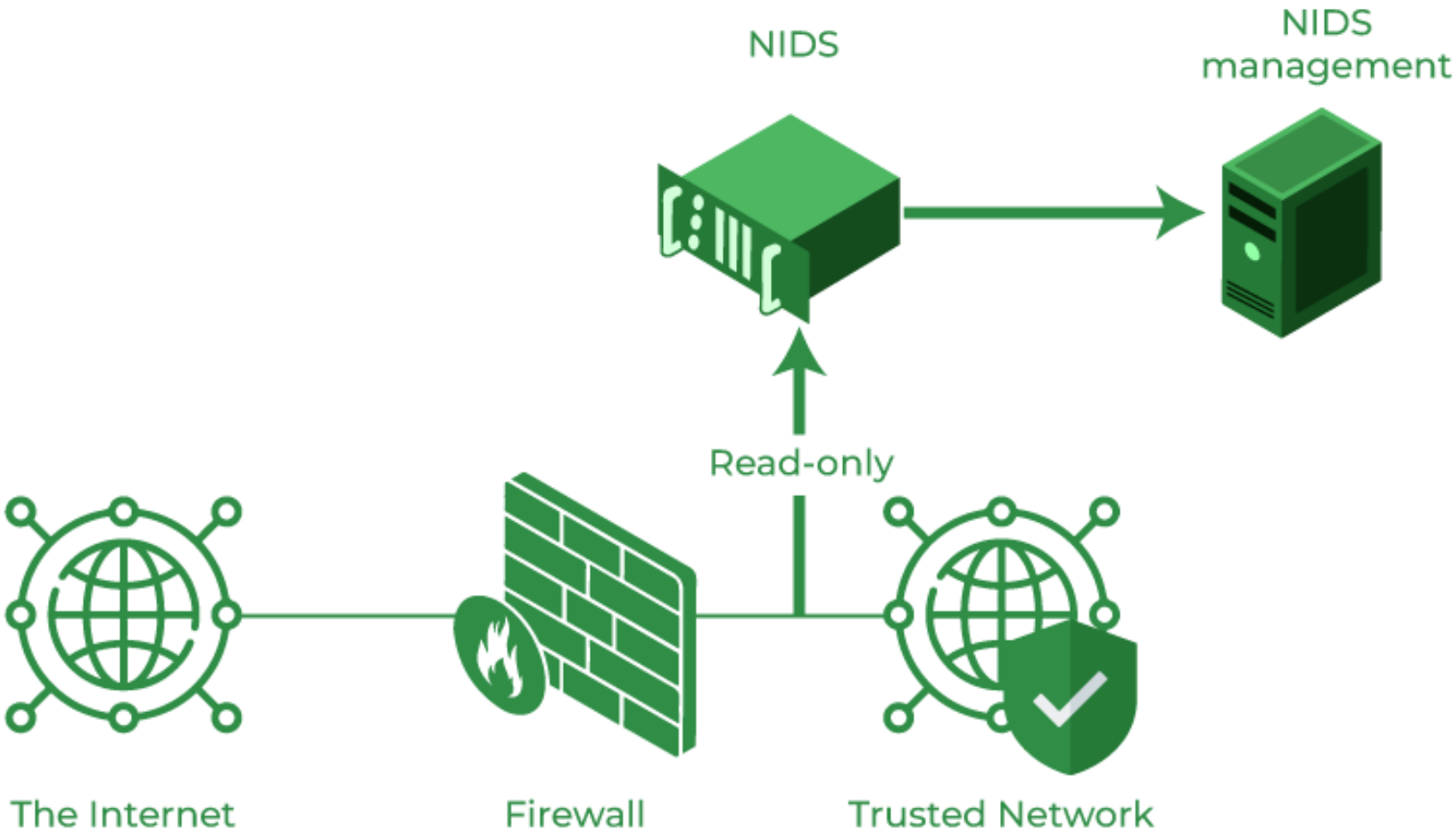
Create and test your signature

- This is how your rule file should look like:

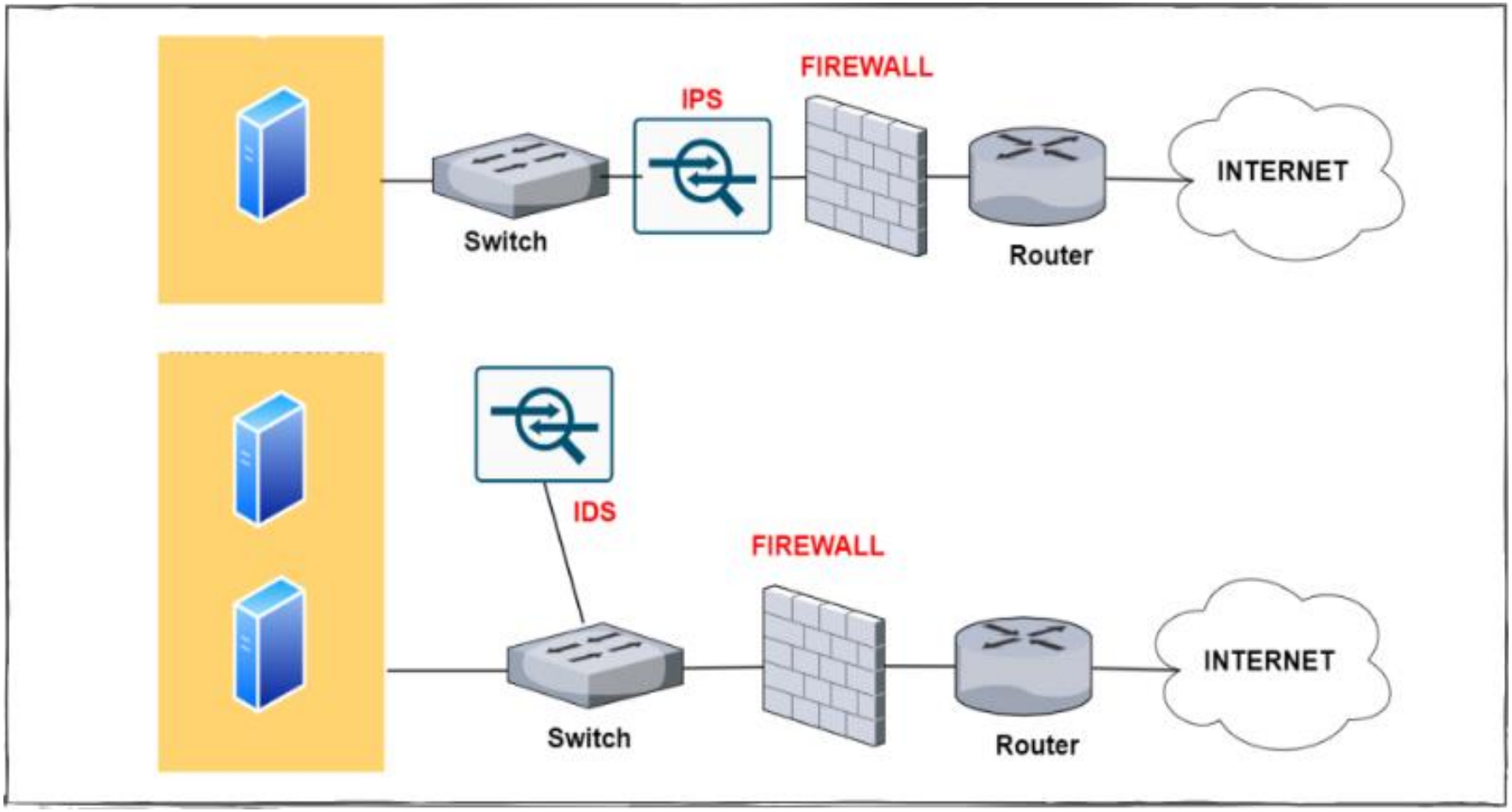
```
rule single_byte_mov
{
        strings:
                $a = { c6 45 ?? ?? c6 45 ?? ?? c6 45 }

        condition:
                $a
}
```

Intrusion Detection System: *Snort*

Network-based intrusion detection system



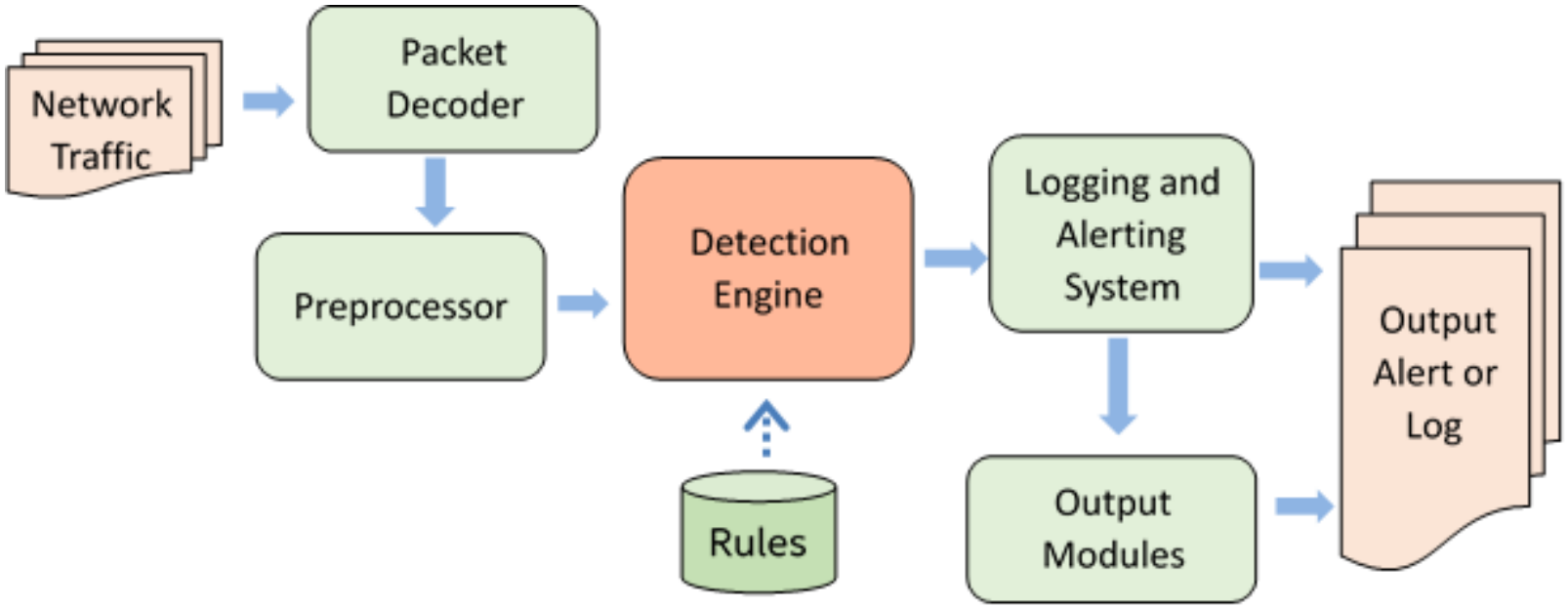
IDS vs. IPS


FIREWALL
IPS
Switch
Router
INTERNET
IDS
FIREWALL
Switch
Router
INTERNET

Snort IDS/IPS

- Snort IDS is an open-source intrusion detection system that monitors network traffic for malicious activity, such as malware, port scans, and denial-of-service attacks, by using a customizable, rule-based language.
  - Lightweight
  - Free
  - Configurable
  - No sophisticated training needed
- It functions in a "detect-only" mode to alert security teams to threats and can also be deployed as an Intrusion Prevention System (IPS) to actively block malicious traffic.

Snort architecture



Packet decoder

- **The packet decoder is the first stage in Snort's packet processing pipeline.**
- **Its job is to take raw packets from the network (or a capture file) and convert them into a format that Snort can analyze.**
- **What it does:**
  - Decode data-link layer (Layer 2) headers: Ethernet, VLAN, PPP, etc
  - Parse network layer (Layer 3) headers: IPv4, IPv6, ICMP, IGMP, etc.
  - Identify transport protocols (Layer 4): TCP/UDP
  - Normalize packet data (sometimes): Strip padding, reconstruct headers if needed
  - Flag malformed packets: If something looks suspicious or broken (e.g., bad checksums), the decoder can log a warning.

Detection engine

- The Detection Engine in Snort is the core component responsible for analyzing decoded packets and determining whether they match any attack signatures (rules).
- It maintains rules in a two-dimensional linked list of Chain Headers and Chain Options.
- First rule that matches a decoded packet triggers the specified action and returns.
- If the packet content or behavior matches a rule, Snort can generate an alert, log the packet, or take action (e.g. drop it, in IPS mode).

Rule Chain Structure

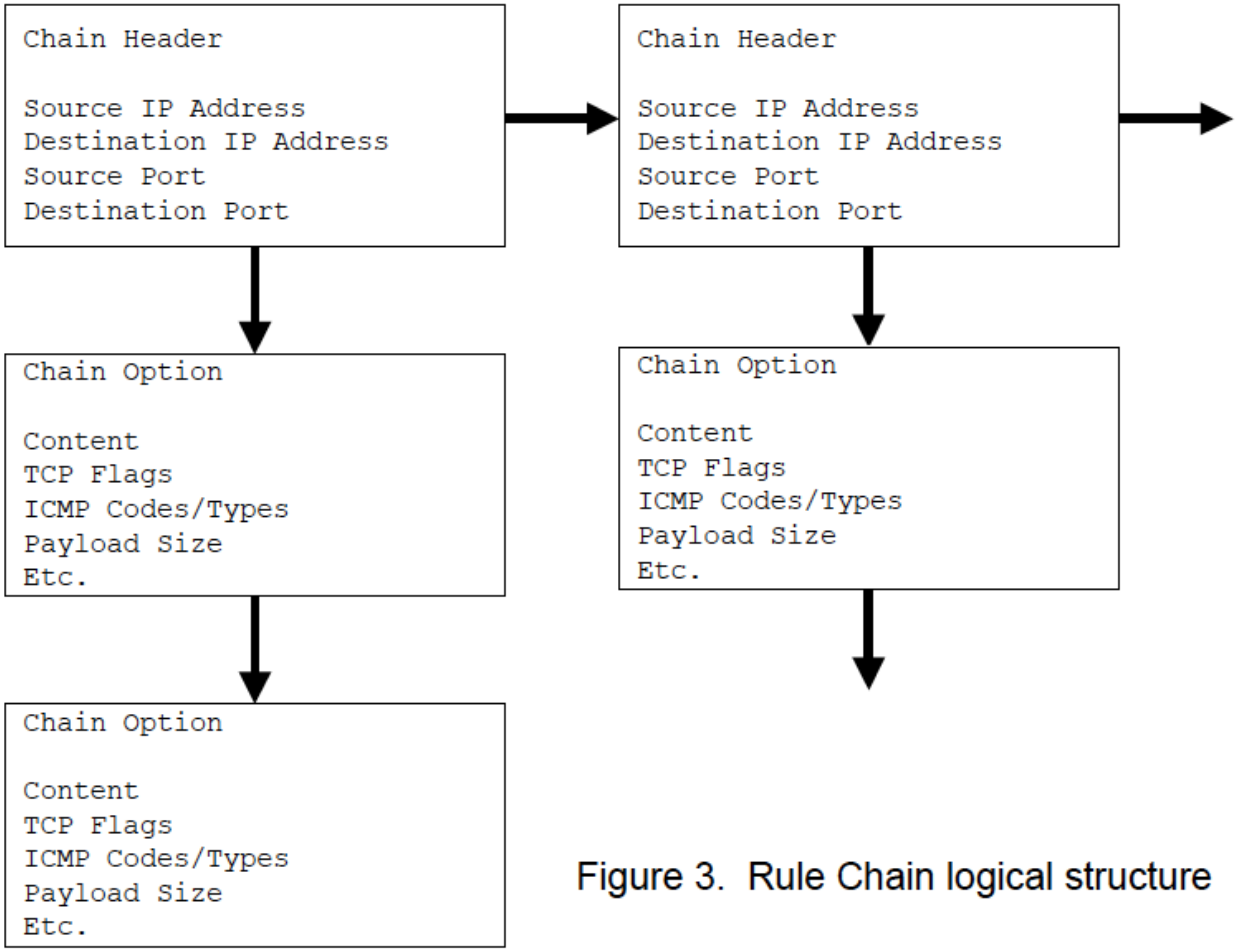
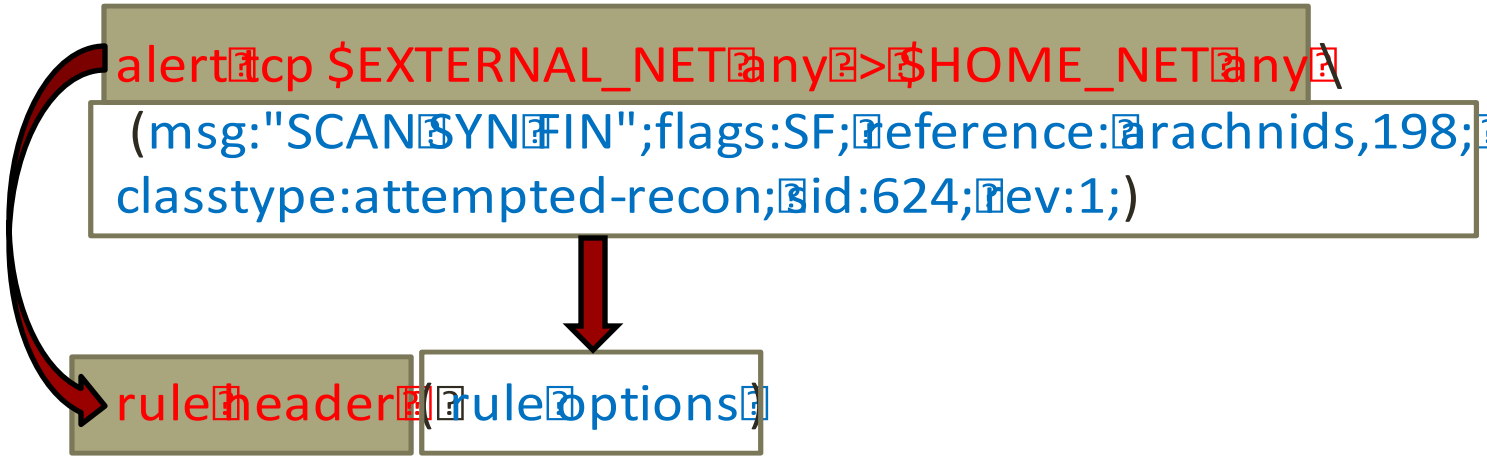


Figure 3. Rule Chain logical structure

Snort: Rules

- http://manual.snort.org/node1.html
- http://books.gigatux.nl/mirror/snortids/0596006616/snortids-CHP-7-SECT-3.html



Snort: Rule Header

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \
 (msg:"SCAN SYN FIN";flags:SF; reference: arachnids,198; \
classtype:attempted-recon; sid:624; rev:1;)
```

Defines "who" the rule applies to (coarsly).

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
```

Src IP

Src Port

Dst IP

Dst Port

action

protocol

Direction

Snort: Rule Header Actions

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \
 (msg:"SCAN SYN FIN";flags:SF; reference: arachnids,198; \
classtype:attempted-recon; sid:624; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any
```

Src IP

Src Port

Dst IP

Dst Port

action

protocol

Direction

1. **alert**: Alerts and logs the packet when triggered.
2. **log**: Only logs the packet when triggered.
3. **pass** : Ignores the packet
4. **activate** : Alerts then activates a dynamic rule or rules.
5. **dynamic** : Ignores, until started by the activate rule, at which time, acts as a log rule.
6. **drop** : block and log the packet
7. **reject** : block the packet, log it, and then send a TCP reset if the protocol is TCP or an ICMP port unreachable message if the protocol is UDP.
8. **sdrop** : block the packet but do not log it.

Snort: Rule Header Protocol

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \
 (msg:"SCAN SYN FIN";flags:SF; reference: arachnids,198; \
classtype:attempted-recon; sid:624; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any
```

Src IP

Src Port

Dst IP

Dst Port

action

protocol

Direction

Protocols: TCP, UDP, ICMP, and IP
Future may include: ARP, IGRP, GRE, OSPF, RIP, IPX, etc.

Snort: Rule Header IP

Src Port

Src IP Dst IP Dst Port

alert tcp $EXTERNAL_NET any -> $HOME_NET any

alert tcp 192.168.1.0/24 any -> 192.168.1.0/24 1:1024

alert tcp ![192.168.1.0/24,10.1.1.0/24] any -> 192.168.1.44

$EXTERNAL_NET is a config value set in snort.conf.

IP is specified also as dotted notation with CIDR masks. "any" is also valid.

! is the negation operator

Multiple IP specifications can be included using square brackets [] and comma-separating.
Do not add spaces!

CIDR (Classless Inter-Domain Routing) notation is a compact method for specifying IP address ranges and network masks. It is widely used in network configuration and management. Example: 192.168.129.23/17.

Snort: Rule Header Port

Src Port

Src IP Dst IP Dst Port

alert tcp $EXTERNAL_NET any -> $HOME_NET any

alert tcp 192.168.1.0/24 any -> 192.168.1.0/24 1:1024

alert tcp ![192.168.1.0/24,10.1.1.0/24] any -> 192.168.1.44

Port can be specified as:

any -- any port

1:1024 -- ports 1 to 1024 inclusive

55: -- ports 55 and higher

:55 -- ports 0 to 55 (inclusive)

negation still works:

!6000:6001 - matches any port except 6000 and 6001

Snort: Rule Header Direction

Src Port

Src IP Dst IP Dst Port

alert tcp $EXTERNAL_NET any -> $HOME_NET any

alert tcp 192.168.1.0/24 any -> 192.168.1.0/24 1:1024

alert tcp ![192.168.1.0/24,10.1.1.0/24] any -> 192.168.1.44

Direction can be specified as:

-> From IP/Port (source) to IP/Port (destination)

<> Any direction

Note: <- does not exist…so the snort rules always read consistently.

Snort: Rule Options

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \
 (msg:"SCAN SYN FIN";flags:SF; reference: arachnids,198; \
classtype:attempted-recon; sid:624; rev:1;)
```

name:value;

msg: <sample message>	Logs message into /var/snort/log
flags: <AFPRSU210>	Matches specific TCP flags
content: <text>	Matches specified text in packet
content: | <hexadecimal> |	Matches specified hex chars
sid: <snort ID>	Unique number to identify rules easily. Your rules should use SIDs > 1,000,000
rev: <revision #>	Rule revision number
reference: <ref>	Where to get more info about the rule
gid: <generator ID>	Identifies which part of Snort generated the alert. See /etc/snort/gen-msg.map for values

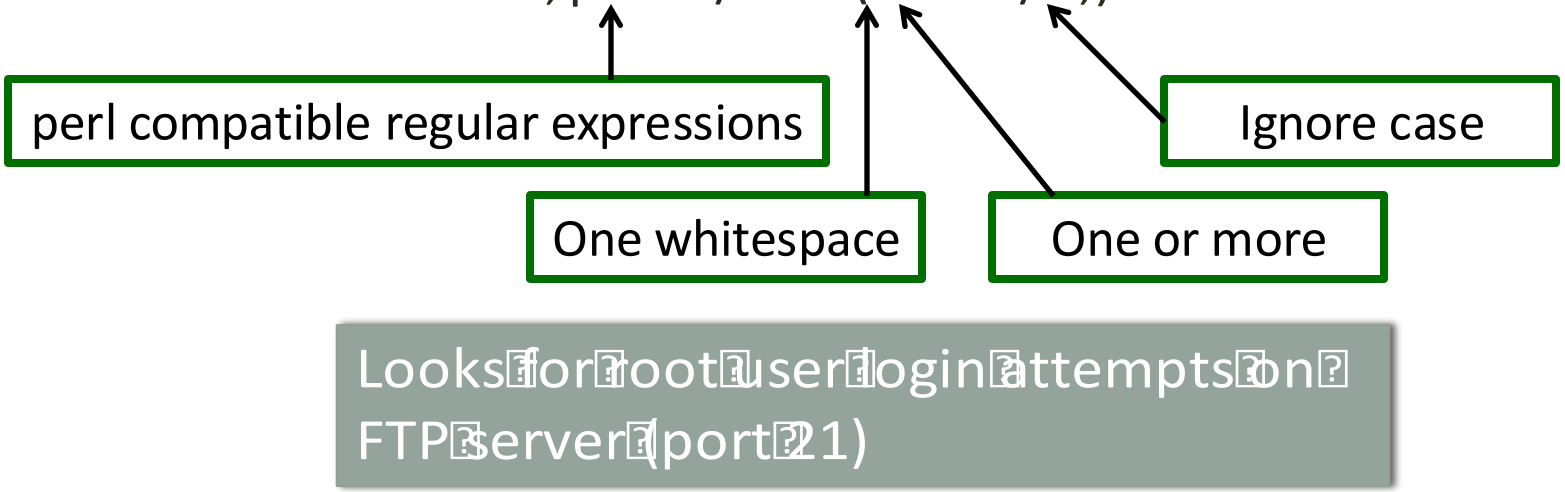
Snort: More Rule Options…

Read the docs.. there are MANY more options:

http://manual.snort.org/node1.html

Snort rule examples

1. alert tcp any any -> any 21 (flow:to_server,established; \
   content:"root"; pcre:"/user\s+root/i";)



The flow keyword, which is a non-payload detection option, is used in conjunction with TCP stream reassembly. It allows rules to only apply to certain directions of the traffic flow.

Snort rule examples

This is a real rule from malware-tools.rules

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS
(msg:"MALWARE-TOOLS HOIC http denial of service
attack"; flow:to_server,established; content:"User-
Agent|3A 20 20|Mozilla"; fast_pattern:only;
http_header; content:"Referer|3A 20 20|http";
http_header; content:!"Connection: keep-alive"; nocase;
detection_filter:track by_src, count 17, seconds 10;
metadata:policy balanced-ips drop, policy security-ips
drop, service http;
reference:url,blog.spiderlabs.com/2012/01/hoic-ddos-
analysis-and-detection.html; classtype:denial-of-
service; sid:21513; rev:6;)
```

Snort rule examples

This is a real rule from blacklist.rules

```
alert udp $HOME_NET any -> any 53 (msg:"BLACKLIST DNS
request for known malware domain guest-access.net -
Gauss "; flow:to_server; byte_test:1,!&,0xF8,2;
content:"|0C|guest-access|03|net|00|";
fast_pattern:only; metadata:impact_flag red, policy
balanced-ips drop, policy security-ips drop, service
dns; reference:url,gauss.crysys.hu/;
reference:url,www.securelist.com/en/blog/208193767/Gaus
s_Nation_state_cyber_surveillance_meets_banking_Trojan;
classtype:trojan-activity; sid:23799; rev:2;)
```

Snort rule examples

This is a real rule from os-windows.rules

```
alert tcp $EXTERNAL_NET $FILE_DATA_PORTS -> $HOME_NET
any (msg:"OS-WINDOWS Microsoft Windows Object Packager
ClickOnce object remote code execution attempt";
flow:to_client,established;
flowbits:isset,file.ppsx&file.zip; file_data;
content:"uuid:48fd9e68-0958-11dc-9770-9797abb443b9";
fast_pattern:only; content:"2007-05-23T15:06:10-03:00";
metadata:policy balanced-ips drop, policy security-ips
drop, service ftp-data, service http, service imap,
service pop3; reference:cve,2012-0013;
reference:url,technet.microsoft.com/en-
us/security/bulletin/ms12-005; classtype:attempted-
user; sid:26068; rev:3;)
```

End of Lecture 14