OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
INF FAKULTÄT FÜR INFORMATIK

Industrie-Ransom 2.0:
Evaluation der Anwendung von YARA zur Erkennung netzwerkbasierter Ransomware

TAITS: DR3
17.01.2024

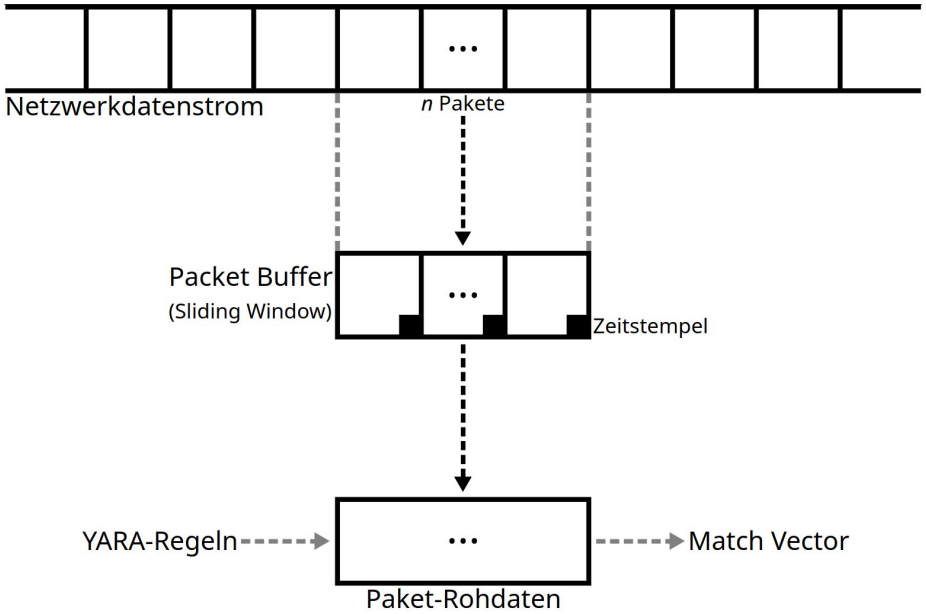
Bernhard Birnbaum

Inhalt

1. Konzept
2. YARA-Regeln
3. Evaluation
4. Zusammenfassung & Aussicht
5. Anhang: UNCOVER-Regeln

1. Konzept

- Konzept erweitert durch Zeitstempel hinter Paketdaten
- YARA-Regeln werden immer bei Eingang eines neuen Pakets evaluiert
- Aufbereitung des Netzwerkdatenstroms für YARA bestimmt maßgeblich, was möglich ist

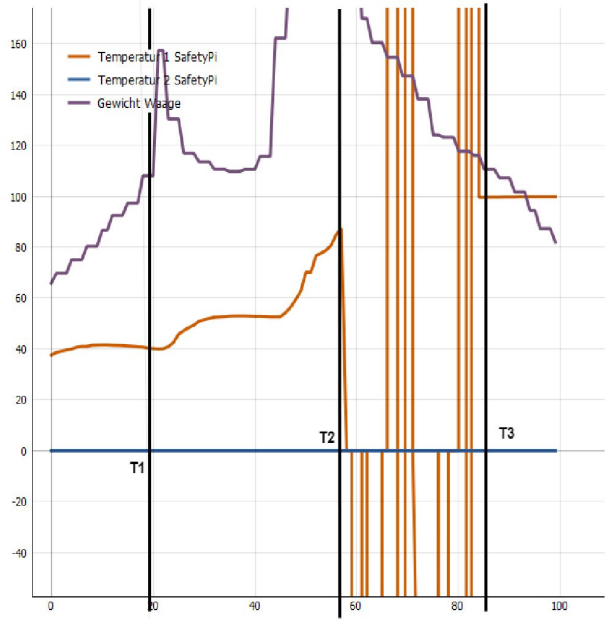


2. YARA-Regeln: Modbus-Query-Flooding II

```
import "numeric"
rule modbus_queryflooding_timing : main {
        strings:
            $modbus_query_request = { 00 80 F4 09 51 3B 00 0C 29 E6 14 0D 08 00 45 00 \
        00 34 [2] 40 00 40 06 [2] AC 1B E0 32 AC 1B E0 FA [2] 01 F6 [4] [4] 50 18 72 10 \
        [2] 00 00 [2] 00 00 00 06 01 06 00 06 00 00 FF [8] FE }
        condition:
            #modbus_query_request >= 2 and (numeric.int64(@modbus_query_request[1] + 66)
        - numeric.int64(@modbus_query_request[2] + 66)) < 100000
}
PBS=3
```

2. YARA-Regeln: Verschlüsselung (Werte)

```
rule opcua_writevalue : main {
        strings:
            $opcua_writerequest = { 4D 53 47 46 58 00 00 \
          00 [4] 01 00 00 00 [8] 01 00 A1 02 [46] FF FF \
          FF FF 03 0A [4] 00 00 00 00 }
        condition:
            #opcua_writerequest > 0 and \
          (int8(@opcua_writerequest[1] + 80) != 0)
}
PBS=1
```



Quelle: Emirkan Toplu, BA

2. YARA-Regeln: Verschlüsselung (SCID)

```
rule opcua_securechannelid : main {
        strings:
                $opcua_msgf = { 4D 53 47 46 [8] 01 00 00 00 }
        condition:
                #opcua_msgf >= 2 and (uint32(@opcua_msgf[1] + 8) - \
            uint32(@opcua_msgf[2] + 8)) != 0
}
PBS=3
```

2. YARA-Regeln: MITM (ARP-Spoofing) I

```
rule arp_request {
        strings:
                $arp_req = { 08 06 00 01 08 00 06 04 00 01 [20] 00 00 00 00 00 00 00 00 00 00 \
            00 00 00 00 00 00 00 00 }
        condition:
                any of them
}
rule arp_reply {
        strings:
                $arp_rep = { 08 06 00 01 08 00 06 04 00 02 [20] 00 00 00 00 00 00 00 00 00 00 \
            00 00 00 00 00 00 00 00 }
        condition:
                any of them
}
```

2. YARA-Regeln: MITM (ARP-Spoofing) II

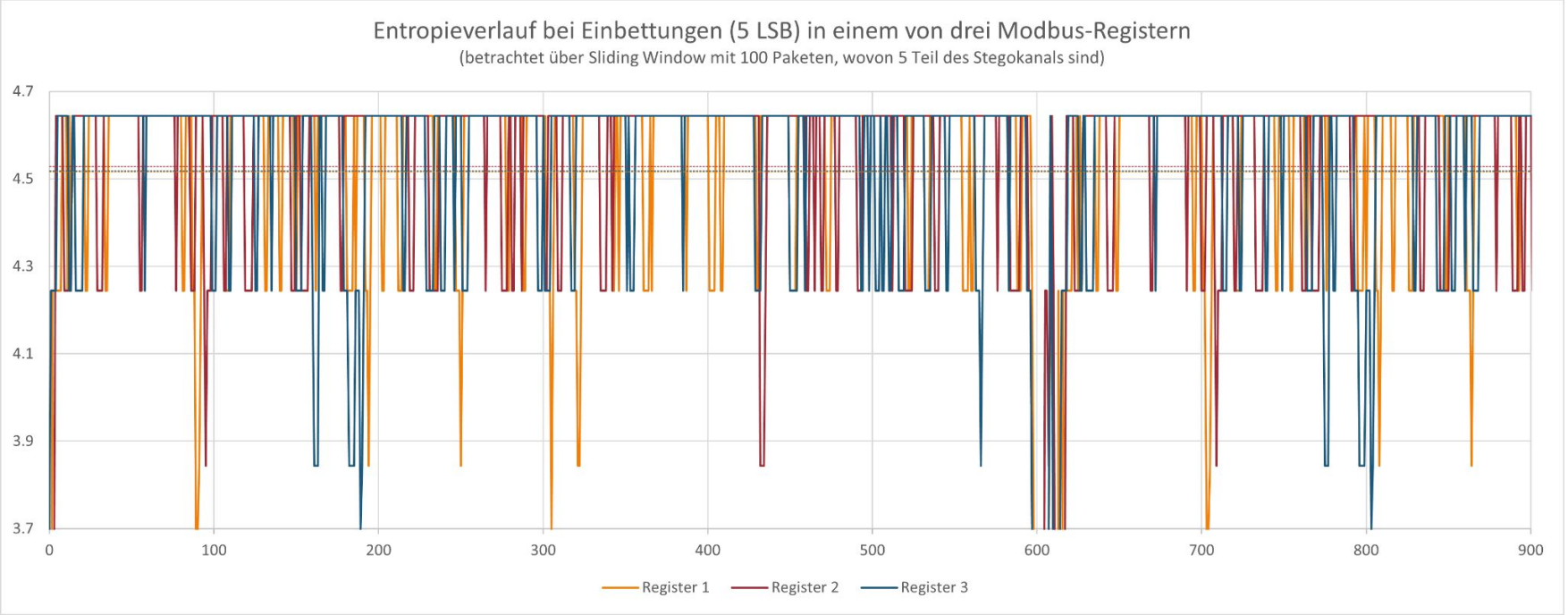
```
rule arp_mitm : main {
        strings:
                $mac0 = { FF FF FF FF FF FF } //broadcast
                $mac1 = { 48 5B 39 64 40 79 } //asustek
                $mac2 = { 00 80 F4 09 51 3B } //telemec
                $mac3 = { 00 0C 29 9D 9E 9E } //vmware
        condition:
                (arp_request and (#mac0 + #mac1 + #mac2 + #mac3) < 3) or \
            (arp_reply and (#mac0 + #mac1 + #mac2 + #mac3) < 4)
}
PBS=1
```

3. Evaluation: Modbus-LSB-Covert Channel

- Covert Channel:
  - 3 Register pro Paket (Read-Holding-Registers)
  - in einem der 3 Register: LSB-Einbettung in den 5 LSB
- Entropieberechnung über die 5 letzten RHR-Pakete (über YARA-Modul möglich)

→ **ABER**: Regel basierend auf fixen Schwellwerten zur automatisierten Detektion des Covert-Channels nicht möglich

3. Evaluation: Modbus-LSB-Covert Channel



3. Evaluation: Grenzen

- YARA-Patterns können nicht auf verschlüsselte Daten angewandt werden
- Abwesenheit von Paketen können nicht detektiert werden
  - YARA kann natürlich auf Abwesenheit von Patterns prüfen
  - Ansatz evaluiert Regeln aber immer nur dann, wenn neue Pakete eintreffen

  → Spielraum für Verbesserung
- YARA-Module können beliebig komplexe Regeln implementieren

  → Module sollten vorrangig zur Detektion fehlende Metriken implementieren; die Regel vollständig in Module auszulagern widerspricht der Idee von YARA

4. Zusammenfassung

Sachverhalt	Protokoll	Kurzbeschreibung	Ergebnis
Query-Flooding I	Modbus	Detektion durch mindestens 3 Vorkommen von Modbus-Query-Requests im Packet Buffer	möglich
Query-Flooding II	Modbus	Detektion durch Schwellwertunterschreitung der Zeitdifferenz zwischen 2 Modbus-Query-Requests im Packet Buffer	möglich
LSB-Covert-Channel	Modbus	Detektion eines 5-LSB Covert Channels in einem von 3 Modbus-Registern m.H. der Entropie	nicht möglich
Werteabgleich I	OPCUA	Abgleich von konkreten (Temperatur-)Werten in einzelnen OPCUA-Write-Requests	möglich
Werteabgleich II	OPCUA	Abgleich von konkreten (Temperatur-)Differenzen in 2 aufeinanderfolgenden OPCUA-Write-Requests	möglich
Verschlüsselung von Werten	OPCUA	Detektion von verschlüsselten Werten in OPCUA-Write-Requests durch Abgleich des ersten (d.h. des most-significant) Bytes	möglich
Verschlüsselung der SCID	OPCUA	Detektion von Verschlüsselung der SCID in OPCUA-Messages durch spontane Änderung der SCID	möglich
OPCUA-Fehler	OPCUA	Detektion von OPCUA-Fehlern als mögliche Folge eines Ransomware-Angriffs	möglich
Sign&Encrypt	OPCUA	Detektion von schädlichen Paketen, die im OPCUA Sign&Encrypt-Modus übertragen werden	nicht möglich
MITM-Angriff	ARP	Detektion eines MITM-Angriffs m.H. von ARP-Spoofing durch Zählen der bekannten MAC-Adressen	möglich
Abwesenheit		Detektion von Abwesenheit erwarteter Pakete	nicht möglich

4. Aussicht

- Bericht schreiben
- Ausblick für zukünftige Arbeiten
  - weitere Angriffsvektoren untersuchen
  - Aufbereitung des Netzwerkdatenstroms (Konzept) verbessern; beispielsweise zeitdiskrete Anwendung von YARA-Regeln, um Abwesenheit von erwarteten Paketen sicher zu detektieren

5. Anhang: UNCOVER-Regeln: f5

```
rule magic_number {
  strings:
    $jpeg_magic_number = { FF D8 }
  condition:
    #jpeg_magic_number > 0 and @jpeg_magic_number[1] == 0
}
rule suspicious_quantization_table {
  strings:
    $quantization_table = { FF DB [82] 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 \
    28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 FF }
  condition:
    #quantization_table == 1 and @quantization_table[1] == 20
}
rule f5 : main {
  condition:
    magic_number and suspicious_quantization_table
}
```

5. Anhang: UNCOVER-Regeln: jsteg

```
rule magic_number {
   strings:
      $jpeg_magic_number = { FF D8 }
   condition:
      #jpeg_magic_number > 0 and @jpeg_magic_number[1] == 0
}
rule segments_missing {
   strings:
      $expected_segments = { FF D8 [-] FF E0 [-] FF DA [-] FF D9 }
   condition:
      #expected_segments == 0
}
rule jsteg : main {
   condition:
      magic_number and segments_missing
}
```

5. Anhang: UNCOVER-Regeln: stegonaut

```
rule stegonaut : main {
   strings:
      $mpeg_sync_word = { FF (F? | E?) }
   condition:
      #mpeg_sync_word > 0 and (uint8(@mpeg_sync_word[1] + 2) & 0x0000000000000001) \
      == 1 and (uint8(@mpeg_sync_word[1] + 3) & 0x000000000000000F) == 15
}
```

Byte 1								Byte 2								Byte 3								Byte 4							
1	1	1	1	1	1	1	1	1	1	1													1					1	1	1	1
Sync											ID		Layer		Pr	Bitrate				Freq		Pa	Pv	Kanal		ModEx		Cp	Or	Emph	

5. Anhang: UNCOVER-Regeln: mp3stegz

```
rule mp3stegz : main {
   strings:
      $mpeg_sync_word = { FF (F? | E?) [34] 58 58 58 58 }
   condition:
      #mpeg_sync_word >= 3
}
```

5. Anhang: UNCOVER-Regeln: mp3stego

```
import "mp3"

rule final_frame_broken {
   strings:
      $mpeg_sync_word = { FF (F? | E?) }
   condition:
      #mpeg_sync_word > 0 and mp3.frame_data_end(@mpeg_sync_word[1]) > filesize
}
rule constant_bitrate {
   strings:
      $mpeg_sync_word = { FF (F? | E?) }
   condition:
      #mpeg_sync_word > 1 and mp3.bitrate_min(@mpeg_sync_word[1]) == mp3.bitrate_max(@mpeg_sync_word[1])
}
rule mp3stego : main {
   condition:
      final_frame_broken and constant_bitrate
}
```

5. Anhang: UNCOVER-Regeln: Auszug mp3-Modul

```
uint64_t calculate_frame_length(int64_t syncword_position) {
   uint8_t* frame_bytes = (block_data + syncword_position);

   uint8_t byte_version   = (*(frame_bytes + 1) & 0x18) >> 3; //00011000
   uint8_t byte_layer     = (*(frame_bytes + 1) & 0x06) >> 1; //00000110
   uint8_t byte_bitrate   = (*(frame_bytes + 2) & 0xF0) >> 4; //11110000
   uint8_t byte_frequency  = (*(frame_bytes + 2) & 0x0C) >> 2; //00001100
   uint8_t byte_padding   = (*(frame_bytes + 2) & 0x02) >> 1; //00000010

   uint64_t bitrate_value = match_bitrate(byte_version, byte_layer, byte_bitrate);
   uint64_t frequency_value = match_frequency(byte_version, byte_frequency);
   uint64_t samples_value = match_samples(byte_version, byte_layer);

   uint64_t frame_size = 0;
   if (bitrate_value > 0 && frequency_value > 0 && samples_value > 0) {
      frame_size = (samples_value * bitrate_value) / (8 * frequency_value);
      if (byte_padding == 1)
         frame_size++;
   }
   return frame_size;
}
```

Danke für Ihre Aufmerksamkeit!