SEC Consult #ITSECX

Entropie im Sinkflug: rand() und srand()

SEC Consult

an Eviden business

Entropie im Sinkflug: rand() und srand()

Präsentationsübersicht

Einführung

- Vorstellung
- Research Device
- Status
- Vorgehen

Schwachstellen, Exploits & Empfehlungen

- Improper Signature Verification of Firmware Upgrade Files
- Missing Protection Mechanism for Alternate Hardware Interface
- Predictable Session ID

CVEs / Responsible Disclosure

Q&A und Diskussion

Entropie im Sinkflug: rand() und srand()

Einführung - Vorstellung

- Seit 2017 bei SEC Consult

- Bachelor
  - “Unternehmens- und IT-Sicherheit”
    - University of Applied Sciences Offenburg
- Master
  - IT-Security
    - FH Campus Wien

Entropie im Sinkflug: rand() und srand()

Einführung – Research Device

- Rittal IoT Interface
- “Vernetzung von Rittal Kühllösungen oder Sensoren zur Überwachung von physikalischen Umgebungsbedingungen“
- Research Projekt 2024 bei SEC Consult

SEC Consult
an Eviden business

Entropie im Sinkflug: rand() und srand()

Einführung – Status

- Zugang zu Gerät
  - Physisch
  - Web Interface (Admin)
  - Kein Zugriff auf Dateisystem (IoT Interface)

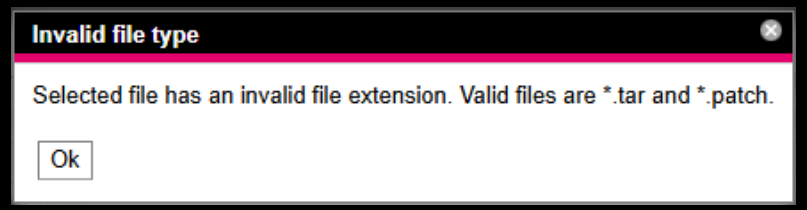
- Zugriff auf Firmware-Datei des Herstellers (Download)
  - V6.19.00

Entropie im Sinkflug: rand() und srand()

Einführung – Vorgehen

- Physische Überprüfung in unserem HW-Labor
  - Keine Auffälligkeiten
  - Debug Interface nicht offensichtlich

- Check des Web Interfaces
  - Check alter Schwachstellen
  - Session ID auffällig kurz
  - Firmware Upgrade verlangt nach .patch oder .tar Dateien



Entropie im Sinkflug: rand() und srand()

Einführung – Vorgehen

- Suche nach Dateien die **`.patch`** enthalten
- Code zur Verifizierung der Signatur
  - Ausschnitt aus **`patch_verify.sh`**

```
key="8ab4ce0f72bce9b3837281b792…"
filename=${filename_full%.*} # remove .patch

verify(){
        cd $patchhome
        for file in $(find . -type f \( ! -iname "rittal.sig" \) | sort ); do
                sha512sum $file >> rittal
        done
        echo $key >> rittal
        calcsig=$(sha512sum rittal)
        trustsig=$(cat rittal.sig)
        if [[ $calcsig == $trustsig ]]; then
                patchverify=OK;
       }
```

Entropie im Sinkflug: rand() und srand()

Einführung – Vorgehen – patch_verify.sh

- Speichere SHA512-Hash aller Dateien, außer **`rittal.sig`** in Datei **`"rittal"`**

```
key="8ab4ce0f72bce9b3837281b792…"

verify(){
        cd $patchhome
        for file in $(find . -type f \( ! -iname "rittal.sig" \) | sort ); do
                sha512sum $file >> rittal
        done
        echo $key >> rittal
        calcsig=$(sha512sum rittal)
        trustsig=$(cat rittal.sig)
        if [[ $calcsig == $trustsig ]]; then
                patchverify=OK;
       }
```

Entropie im Sinkflug: rand() und srand()

Einführung – Vorgehen – patch_verify.sh – Key

- Füge statischen **key** an Ende der Datei **"rittal"** **hinzu**

```
key="8ab4ce0f72bce9b3837281b792…"

verify(){
        cd $patchhome
        for file in $(find . -type f \( ! -iname "rittal.sig" \) | sort ); do
                sha512sum $file >> rittal
        done
        echo $key >> rittal
        calcsig=$(sha512sum rittal)
        trustsig=$(cat rittal.sig)
        if [[ $calcsig == $trustsig ]]; then
                patchverify=OK;
       }
```

Entropie im Sinkflug: rand() und srand()

Einführung – Vorgehen – patch_verify.sh

- Erzeuge SHA512-Hash der Datei **"rittal"**
- Vergleiche, ob dieser Hash mit Inhalt der **rittal.sig** übereinstimmt
- Wenn Ja, **patchverify=OK**

```
key="8ab4ce0f72bce9b3837281b792…"

verify(){
        cd $patchhome
        for file in $(find . -type f \( ! -iname "rittal.sig" \) | sort ); do
                sha512sum $file >> rittal
        done
        echo $key >> rittal
        calcsig=$(sha512sum rittal)
        trustsig=$(cat rittal.sig)
        if [[ $calcsig == $trustsig ]]; then
                patchverify=OK;
       }
```

Entropie im Sinkflug: rand() und srand()

Einführung – Vorgehen – patch_verify.sh

- Für Fake Signatur wird folgendes benötigt
  1. $key
  2. rittal.sig
  3. Datei/Script, das nach erfolgreicher Verifikation ausgeführt wird

Entropie im Sinkflug: rand() und srand()

Einführung – Vorgehen – patch_verify.sh – run.sh

- Nach erfolgreicher Prüfung der Signatur wird run.sh ausgeführt

```
runpatch(){
    echo "run patch $filename_full($patchname)"
    cd $patchhome

    if [ -f run.sh ]; then
        source run.sh
    else
        echo "Error: No start script run.sh found"
    fi
    cd $home
    rm -r $patchhome
    echo "finished patch $filename_full($patchname)"
}
```

Entropie im Sinkflug: rand() und srand()

Einführung – Vorgehen – patch_verify.sh

- Für Fake Signatur wird folgendes benötigt
  1. $key
  2. rittal.sig
  3. run.sh

- Möglich beliebige Dateien zu signieren
  - SHA512-Hashes der Dateien
  - “Secret” (key) ist bekannt
  - rittal.sig kann erstellt werden

Improper Signature Verification of Firmware Upgrade Files

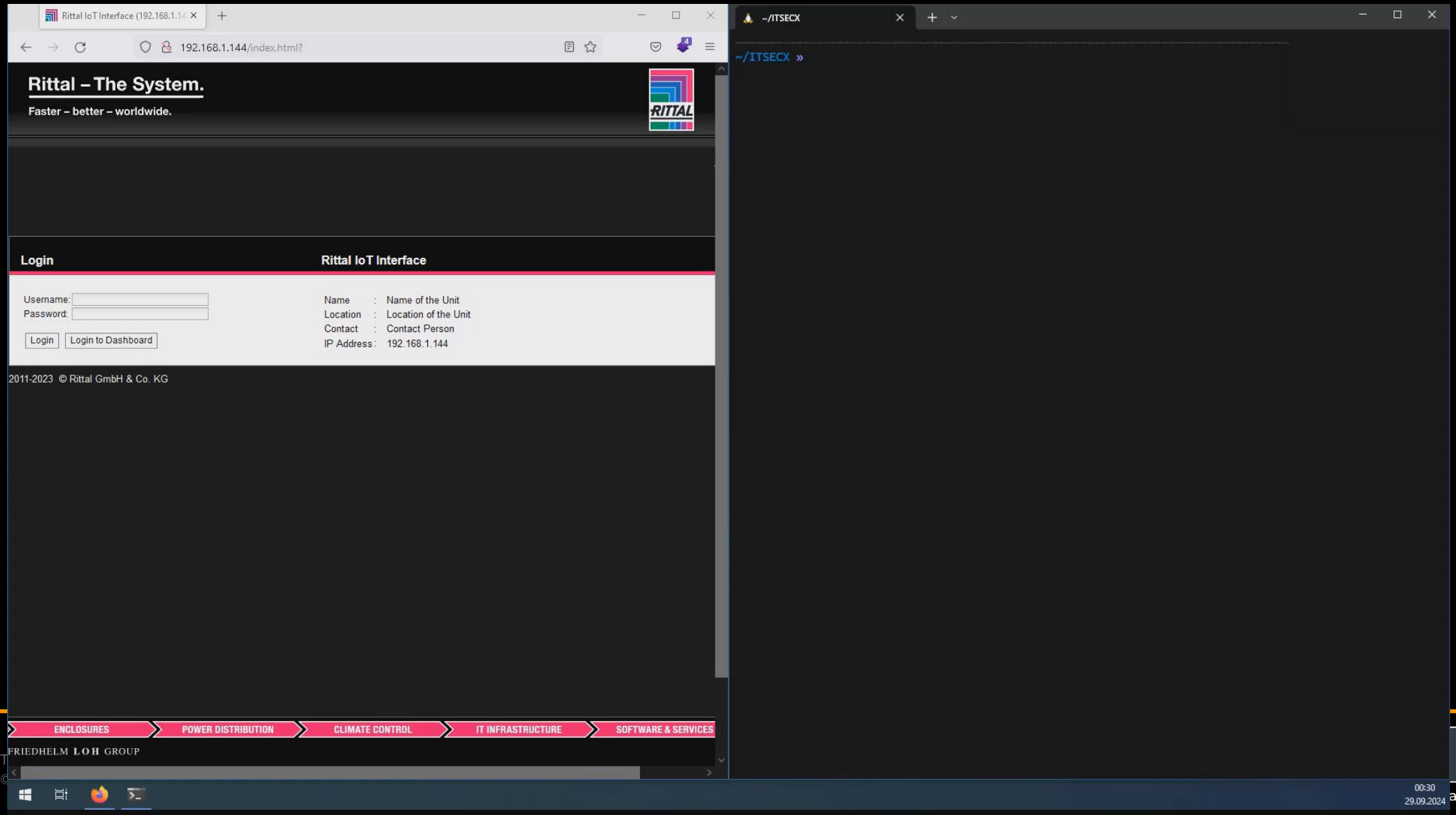
Entropie im Sinkflug: rand() und srand()

Schwachstellen – Improper Signature Verification of Firmware Upgrade Files – Exploit

1. SHA512sum aller Dateien (z.B. run.sh) für Fake Firmware in **`rittal.fake`**
2. Hardcoded Key an Ende von **`rittal.fake`** schreiben
3. SHA512sum von **`rittal.fake`** in **`rittal.sig`** schreiben
4. **`rittal.fake`** ggf. löschen
5. Tar der Fake Firmware Dateien und **`rittal.sig`** erstellen
6. In **`<name>.patch`** umbenennen
7. Firmware Upgrade durchführen (als Admin)
8. Gewinnen

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Improper Signature Verification of Firmware Upgrade Files – Demo



Entropie im Sinkflug: rand() und srand()

Schwachstellen – Improper Signature Verification of Firmware Upgrade Files - Lösungsempfehlung

- Asymmetrische Kryptographie
  - Firmware-Dateien werden mit privatem Schlüssel signiert
  - Signatur wird mitgeliefert
  - Private Key bleibt bei Hersteller
  - Public Key auf Gerät in Hardware Security Modul (HSM)
    - Wird verwendet, um Signatur zu prüfen
    - Mehrere Slots für Schlüssel
      - Falls der Private Key geleaked wird
      - Schlüsselrotation
- Secure Boot
  - Sicherstellen, dass nur authentifizierte Firmware geladen wird

Entropie im Sinkflug: rand() und srand()

Schwachstellen

Missing Protection Mechanism for Alternate Hardware Interface

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Missing Protection Mechanism for Alternate Hardware Interface – Beschreibung

- Script **`emmc_update_fromdir.sh`** sucht u.A. nach .patch Dateien auf USB-Stick oder SD-Card
- Führt ohne Authentifizierung **`patch_verify.sh`** aus

```
[…]
if [[ $UPDATE_FILE == *.patch ]];then
[…]
    bash /usr/sbin/patch_verify.sh "$WATCHDIR/${UPDATE_FILE}"
[…]

fi
continue
```

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Missing Protection Mechanism for Alternate Hardware Interface – Exploit

1. .patch Datei aus vorigem Kapitel auf USB-Stick in root-Verzeichnis kopieren
2. Anstecken
3. Gewinnen

Predictable Session ID

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – Exkurs random

`srand(unsigned int seed)`

- Initialisiert `rand()`
- Typisch `srand(time(0))`

`rand()`

- stdlib.h
- Max Value: 2^31-1 = 2,147,483,647
- Nimmt `int 1` als Seed, wenn nicht anders initialisiert
- NICHT kryptographisch sicher
- Deterministisch
  - Gibt (ohne `srand()`) immer 1804289383 zurück

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – Authentifizierung

- Standard Zugangsdaten admin:admin
- Session nach 1800 Sekunden (30 Minuten) ungültig

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – Authentifizierung Cont.

- HTTP Request bei Anmeldung gibt sehr kurze Session ID zurück
  - `/cgi-bin/json.cgi` ist für alles zuständig

Request

Pretty Raw Hex

```
POST /cgi-bin/json.cgi HTTP/1.1
Host: 192.168.1.144
Content-Length: 63
Sec-Ch-Ua: "Not;A=Brand";v="24", "Chromium";v="128"
Sec-Ch-Ua-Platform: "Windows"
X-Requested-With: XMLHttpRequest
Accept-Language: en-US,en;q=0.9
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.6613.120
 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin: https://192.168.1.144
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://192.168.1.144/index.html?
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive

createSession=
{"username":"admin","password":"admin","force":0}
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Content-Type: text/plain;charset=iso8859-1
Strict-Transport-Security: max-age=15768000;
Content-Length: 246
Date: Thu, 01 Jan 1970 00:58:46 GMT
Server: lighttpd/1.4.55

{"result":0,"sessionId":1792516098,"user":{"name":"admin","grou
p":"admins","groupid":1,"description":"Default
Group","puAdministration":1,"filetransfer":2,"http":1,"console"
:1,"isLdap":0,"autologout":1800,"language":0,"unitType":0,"prof
inet":0}}
```


SEC Consult
an Eviden business

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – Online Brute-Force #1

- Versuch, via HTTP Anfragen Session ID zu erraten
- Im Vergleich zu Offline Brute-Force extrem langsam
  - ca. 1000 Requests pro Minute
- Online Brute-Force gültiger Session (1800 Sekunden)
  - 2^31 req / 1800 s= 1,2 Mio Requests pro Sekunde
  - Für Rittal IoT Interface unmöglich

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – `json.cgi`

- Reverse Engineering der **`json.cgi`**
  - **`json.cgi`** wird für jeden Request neu aufgerufen, kein Fork
  - Ausschnitt des Codes zur Session ID Generierung

```
int FUN_00017384(undefined param_1,int param_2)

{
  int pid;
  int seed;

  pid = getpid();
  seed = rand();
  srand(seed + pid);
  return (rand() + 123 & 0xffff00f) + param_2 * 0x10;
}
```

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – `json.cgi` – param_2

- Vergleich von originalen Session IDs und mit nachgebautem Code generierte Session IDs
  - Überproportionale Häufung von Kollisionen bei Wert **`0`** bzw. **`1`** für **`param_2`**
  - **`param_2`** = Nutzer
    - Admin = **`0`**
    - User = **`1`**

```
int FUN_00017384(undefined pa[redacted]_1,int param_2)

{
  int pid;
  int seed;

  pid = getpid();
  seed = rand();
  srand(seed + pid);
  return (rand() + 123 & 0xffff00f) + param_2 * 0x10;
}
```

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – `json.cgi` – pid

- **`getpid()`** gibt Prozess ID (PID) des aufrufenden Prozesses zurück
  - Settings auf IoT Interface 1-32768
    - PID 1 = Init oder system Prozess

```
int FUN_00017384(undefined pa[redacted]_1,int param_2)

{
  int pid;
  int seed;

  pid = getpid();
  seed = rand();
  srand(seed + pid);
  return (rand() + 123 & 0xffff00f) + param_2 * 0x10;
}
```

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – `json.cgi` – seed

- srand wird mit **`rand()`** und PID initialisiert

```
int FUN_00017384(undefined pa[redacted]_1,int param_2)

{
  int pid;
  int seed;

  pid = getpid();
  seed = rand();
  srand(seed + pid);
  return (rand() + 123 & 0xffff00f) + param_2 * 0x10;
}
```

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – `json.cgi` – Berechnung

- Session ID wird mit **`rand()`** und weiteren Werten berechnet

```
int FUN_00017384(undefined pa[redacted]_1,int param_2)

{
  int pid;
  int seed;

  pid = getpid();
  seed = rand();
  srand(seed + pid);
  return (rand() + 123 & 0xffff00f) + param_2 * 0x10;
}
```

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – `json.cgi` – Problem

- Wo ist das Problem?

```
int FUN_00017384(undefined pa[redacted]_1,int param_2)

{
  int pid;
  int seed;

  pid = getpid();
  seed = rand();
  srand(seed + pid);
  return (rand() + 123 & 0xffff00f) + param_2 * 0x10;
}
```

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – `json.cgi` – Problem

- Wo ist das Problem?
  - **`rand()`** ist bei erstem Aufruf nicht initialisiert
  - Bei zweitem Aufruf Entropie nur durch PID

```
int FUN_00017384(undefined pa[redacted]_1,int param_2)

{
  int pid;
  int seed;

  pid = getpid();
  seed = rand();
  srand(seed + pid);
  return (rand() + 123 & 0xffff00f) + param_2 * 0x10;
}
```

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – `json.cgi` – Problem

- `rand()` immer 1804289383
- `param_2` = 0
- Entropie daher nur von PID (~32768 Werte)
- Bei `fork()` wäre `rand()` beim zweiten Aufruf bereits initialisiert

```
int FUN_00017384(undefined p[redacted]_1,int param_2)

{
  int pid;
  int seed;

  pid = getpid();
  seed = rand();
  srand(seed + pid);
  return (rand() + 123 & 0xffff00f) + param_2 * 0x10;
}
```

Entropie im Sinkflug: rand() und srand()

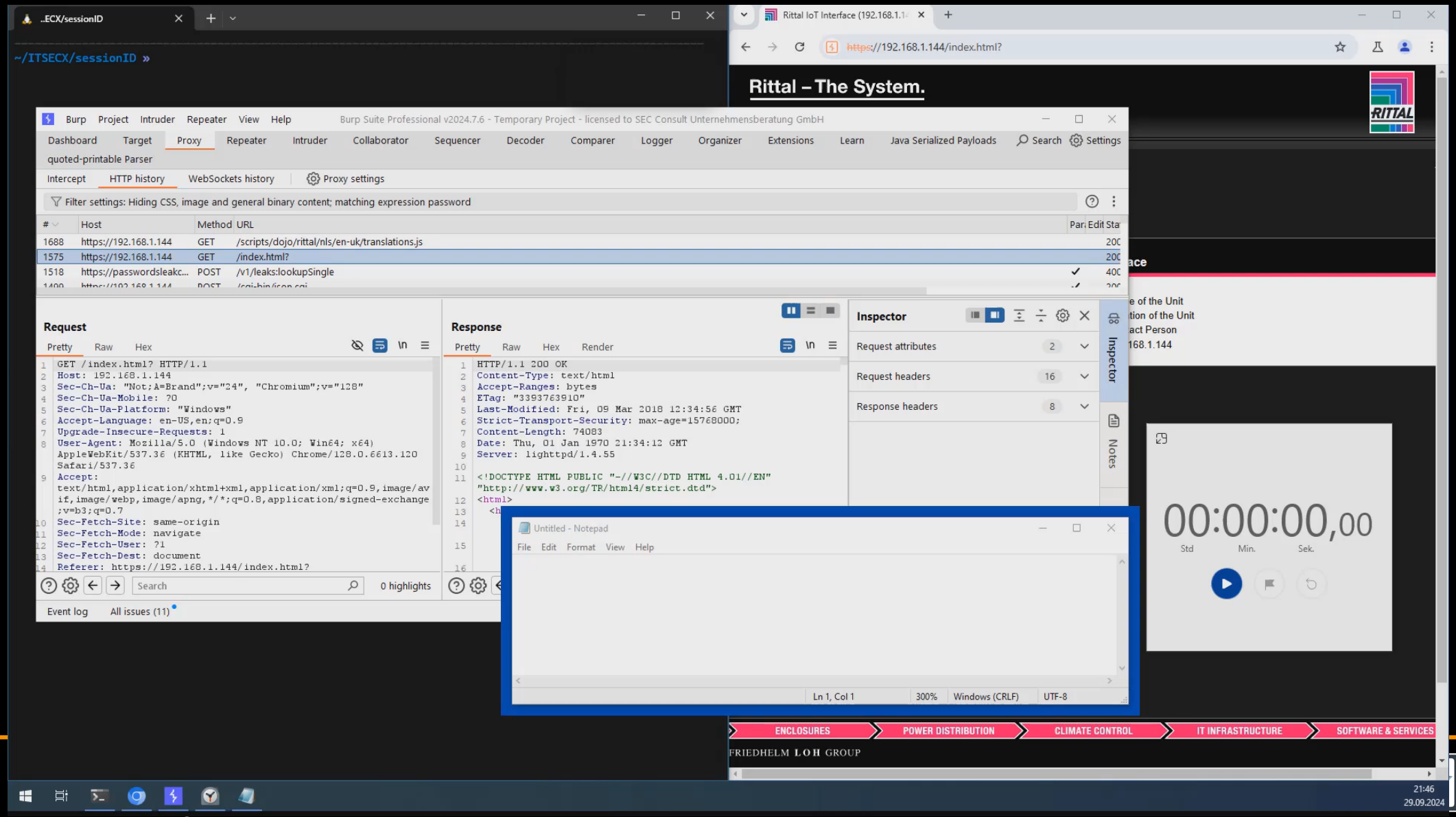
Schwachstellen – Predictable Session ID – Exploit

- Erzeugen aller möglichen Session IDs
  - Mit Werten von 1 – 32768 für Variable **`pid`**

- Online Brute-Force #2
  - Server mit HTTP Anfragen fluten
    - Zuvor nicht sinnvoll, da >2Mrd Aufrufe in 30 Minuten unrealistisch sind
    - 32768 Anfragen sind durchaus möglich
    - Script sendet ca. 1000 Anfragen/Minute

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – Demo



SEC Consult
an Eviden business

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Predictable Session ID – Lösungsempfehlung

- Verwendung kryptographisch sicherer Zufallszahlengeneratoren (CSPRNG)
  - z.B. getrandom() erhält Entropie von urandom
- Keine Vorhersagbarkeit der Session IDs
- Ausreichende Länge der Session IDs

- Best Case (falls möglich):
  - Verwendung von Frameworks für die Authentifizierung
    - OpenID Connect / OAuth 2.0
    - Spring Security (Java)
    - Django REST Framework (Python)
    - …

Entropie im Sinkflug: rand() und srand()

Schwachstellen – Wie könnte ein Angriff aussehen?

- Angreifer erlangt Zugriff auf internes Netzwerk
- Verschlüsselt Dateien / Zieht wertvolle Daten ab
- Für Persistenz
  - Brute-Force der Session ID
  - Einbauen einer Backdoor über Firmware Upgrade in Rittal IoT Interface
- Opfer setzt Active Directory neu auf
- Angreifer erhält erneut Zugriff über Backdoor des IoT Interfaces

Entropie im Sinkflug: rand() und srand()

CVEs / Responsible Disclosure

1) Improper signature verification of firmware upgrade files (CVE-2024-47943)

2) Missing Protection Mechanism for Alternate Hardware Interface (CVE-2024-47944)

3) Predictable Session ID (CVE-2024-47945)

- Kontaktaufnahme mit Hersteller gemäß Responsible Disclosure im Juni 2024
- Patch V6.21.00.2 verfügbar seit 2024-09-30
- Advisory-Release in KW42 geplant

Entropie im Sinkflug: rand() und srand()

Q&A und Diskussion

Fragen?