Rogue7
Rogue Engineering Station Attacks on Simatic S7 PLCs

Eli Biham, Sara Bitan, Aviad Carmel, Alon Dankner, Uriel Malin, Avishai Wool
Technion – Israel Institute of Technology
Tel-Aviv University
Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs

Who Are We?

Dr. Sara Bitan
• Senior researcher at the Technion Hiroshi Fujiwara Cyber Security Research Center
• Founder and CEO of CyCloak - Secure System Design and Audit

Uriel Malin
• MSC Student at Tel Aviv University, advised by Prof. Avishai Wool
• Security Researcher at Medigate – Healthcare IoT Security
Talk Topics

- Uncovered design vulnerabilities in the S7 protocol
- An exploit that performs remote stealth programming of an S7-1500 PLC
Industrial Control System

• A distributed computerized system
• Operates and monitors physical devices
• Controls critical infrastructure
  • Power plants
  • Water facilities
  • Transportation systems
  • Chemical plants

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs
Programmable Logic Controller

• PLC - The core of the ICS
• Connected to sensors and active devices
• Runs a control program that periodically samples the sensors and triggers the devices accordingly
• A bridge between the virtual and the kinetic worlds
• The target of our attacks
PLC interfaces

The S7 Protocol

The Control Program

PLC Operating System

Web Server
SNMP Server

STEP 7
WinCC

HMI - SCADA

Engineering Workstation

TIA

Rogue 7: Rogue Engineering Station Attacks on Simatic S7 PLCs
Secure ICS Topology

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs

Source: NCCIC IR-18-214
• The most famous cyber-attack on ICS
• Targeted Siemens S7-300 PLC
• Infected both WinCC and Step 7 packages
TIA as a Soft Belly

• Typically attacks are exploiting vulnerabilities the engineering station:
  • **CVE-2012-3015**: untrusted search path vulnerability in Siemens SIMATIC STEP7 v5.5 – July-26-2012
  • **CVE-2019-10915**: authentication bypass in TIA v15.1 – July-11-19 by Tenable Security
Our Attack

- Exploits vulnerabilities in the PLC Operating System
  - S7 protocol
- Any vulnerable station/device in the network can serve as an attack machine

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs
The S7-1500 PLC

• One of two new members in the SIMATIC PLCs product line
  • S7-1500 is the high-end PLC
  • The other is S7-1200
S7-1500 Security Enhancements

- New version of the S7 protocol
  - Integrity protection of the messages
- Know-how and copy protection
- PLC access control
  - Using passwords
  - Mitigates the program download attacks
  - Not used by customers in practice

<table>
<thead>
<tr>
<th>Protection level</th>
<th>Access</th>
<th>Access permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection</td>
<td>HMI</td>
<td>Read</td>
</tr>
<tr>
<td>Full access (no protection)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Read access</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>HMI access</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>No access (complete protection)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The S7 Protocol

ISO transport over TCP

Session oriented. Session begin with a 4-ways handshake

Version P3

Client can create, modify and delete objects in the PLC's internal memory

Example: create a server session object
The P3 Handshake Protocol

KDK = Key Derivation Key
Enc_{PLC_PUB_KEY}(Keying Material)

SSL | S7 P3 Handshake
---|---
Server only authentication | Server (PLC) only authentication
Client generates keying material. Encrypts with server public key. | Client generates keying material. Encrypts with server public key.
Different servers use different public keys. | All PLCs from the same model and version use a single key.
P3 Handshake Design Vulnerabilities

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs

With Many Working Forged Copies
Attacking the P3 program
download exchange
Control Program Create Message

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs

- Create Object Request
- Create Program Cycle Object Block
- HMAC-SHA256 over packet with session key
- Object MAC
- Object Code
- Source Code
Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs
Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs

Program Download

Session_key=f(Challenge, KDK)

Req, Hello, RID, Seq=1

Res, Hello, SID, n, Model, Firmware version, Challenge, Seq=1

Req, SID, Encrypted Keying Material, Response, Seq=2

Res, OK, Seq=2

Download
Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs
The Rogue TIA

- No engineering station in the production network.
- The attacker brings it with him.
- Rogue TIA: The attack system consists of a modified legitimate TIA and a malicious proxy.
Rogue TIA Stealth Program Injection Setup Phase

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs

- Request: Hello, RID, Seq=1
- Response: Hello, SID, n_model, Firmware version, Challenge, Seq=1
- Request: SID, Encrypted Keying Material, Response, Seq=2
- Response: OK, Seq=2

Session_key = hash(Challenge, KDK)
Rogue TIA Stealth Program Injection Attack Phase

Download

Re-calc integrity

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs
But we cannot always bring our own TIA with us....

Introducing

The Rogue Engineering Station
Rogue Engineering Station

• A Python attack script that impersonates a TIA
• Inputs: PLC’s IP address; Yellow and blue programs download pcap files

Rogue Engineering Station

• Runs a live download session with the PLC
  • Four-way hand-shake
  • All S7 message fields are taken from the yellow pcap file
  • Except, the object code and MAC that are taken from blue pcap file
  • Fixes message SID, integrity protection and additional cookies
Rogue Engineering Workstation
Stealth Program Injection

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs

Download

Fix SID
Fix other cookies
Re-calc integrity

Session_key=f(Challenge, Keying material)
Step 7: Impersonation

Rogue 7: Rogue Engineering Station Attacks on Simatic S7 PLCs

- My Lab
- The Wall
- King’s Landing
P3 Handshake Details

Deep technical details not for the faint-hearted
P3 Session Key Establishment

• Recall that:

\[ \text{Session\_key} = f(\text{Challenge, KDK}) \]

• What is this \textit{EncryptedKeyingMaterial} Structure?
• Public-private key based asymmetric encryption
  • Elliptic curve: EC-ElGamal
• Symmetric encryption
  • AES
    • Electronic Code Book – ECB Mode
    • Counter mode
• Key Derivation Function: KDF

• Non-cryptographic checksum: Tabulation hash
1. Generate 20 bytes PreKey
   - Encrypt it using EC-ElGamal–like encryption with the plc public key and add it to Keying material

2. Calculate KDF on PreKey and get
   - Checksum Encryption Key (CEK)
   - Checksum Seed (CS)
   - Key Encryption Key (KEK)

3. Concatenate the KDK to the challenge, encrypt them using AES-CTR with the KEK, and add to Keying material

4. Initiate the Tabulation Hash with CS and calculate checksum over (3)

5. Encrypt (4) using AES-ECB with CEK and add to Keying material
The public keys are stored in compressed .key files at [TIA INSTALLATION]\Data\Hwcn\Custom.

Each key file contains:
- Metadata (version, key type, key family, etc.)
- Key data – PLC public key for the EC-ElGamal-like encryption
version: 1

orderNumber: s71500-connection

firmwareVersion:

keyType: connection

familyType: S7-1500

key data: 8456...
P3 Handshake Design Vulnerabilities

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs
1. Pre-calculate PreKey encryption and CEK, CS, KEK

2. Let Python do the symmetric encryptions and the checksum calculation to build keying material 2 & 3

3. The Python script wraps the session key derivation function from the relevant dll
   - We didn’t reverse the session key derivation function $f$, due to lack of time
REing the TIA Handshake

• TIA is huge – find your target files
  • OMSp_core_managed.dll – main S7 communication DLL
  • Mixed mode DLL
    • Managed CIL bytecode – C#
    • Native (unmanaged) x86/x64 opcodes – C++

• Choose your tools
  • Managed code (.NET) - Reflector/dnSpy
  • Native C++ code - IDA Pro

• Improve your reverse engineering starting point
REing the TIA Handshake

• TIA is huge – find your target files
  • OMSp_core_managed.dll – main S7 communication DLL
  • Mixed mode DLL
    • Managed CIL bytecode – C#
    • Native (unmanaged) x86/x64 opcodes – C++

• Choose your tools
  • Managed code (.NET) - Reflector/dnSpy
  • Native C++ code - IDA Pro

• Improve your reverse engineering starting point
Tip #1 – Identify Native Code Entry Points

- `omsp_core_managed.dll` managed part opened in dnSpy

```csharp
1659  public unsafe static Error SetServerPublicKeyGlobal(uint keyType, byte[] key)
1660  {
1661      if (key != null && key.Length != 0)
1662          {
1663              ref byte byteRef = ref key[0];
1664              byte* ptr = ref byteRef;
1665              uint num = (uint)key.Length;
1666              Blob blob;
1667              <Module>.OMS.Blob.{ctor}(ref blob);
1668              Error result;
1669              try
1670                  {
1671                      long num2 = <Module>.OMS.StructValue.copy_data(ref blob, (void*)ptr, num, 0u, false);
1672                      if (((num2 & -9223372036854775808L) != 0L) ? 1 : 0) != 0)
1673                          {
1674                              Error error = new Error(ref num2);
1675                              result = error;
1676                          }
1677                      else
1678                          {
1679                              num2 = <Module>.OMS.ClientSession.set_server_public_key[keyType, ref blob];
1680                              Error error2 = new Error(ref num2);
1681                              result = error2;
1682                          }
1683          }
```

- Let’s click on `s_set_server_public_key`
Wait... Where is the function body???

Looks like it’s an unmanaged function...

How can we find its implementation?
Tip #1 – Identify Native Code Entry Points

- RVA for the native part

Image Base (0x180000000) + RVA (0x109780) = 0x180109780
Tip #2: C++ and RTTI

- Run-Time Type Information
- Allow C++ programmers to examine object types dynamically
- The information must be inside the binary
- We could use it
1. Find RTTI Objects
2. Locate the relevant virtual tables
1. Find RTTI Objects
2. Locate the relevant virtual tables
1. Find RTTI Objects
2. Locate the relevant virtual tables
3. Rebuild them
1. Find RTTI Objects
2. Locate the relevant virtual tables
3. Rebuild them
• Scan the .NET metadata and grab the native entry points
• Scan your .idb and grab the relevant RTTI data
• Use IDAPython to add this information to your .idb
Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs
• Much better, isn’t it?

Rogue7: Rogue Engineering Station Attacks on Simatic S7 PLCs
Rogue Engineering Station
Stealth Program Injection

Attack Demo
• Vulnerabilities in the S7 protocol – P3
  • TIA is not authenticated
  • “One Ring to Rule them All”

• A Python attack tool that impersonates TIA
  • Download a recorded program to any S7-1500 PLC
  • Stealth program injection attack
Thank You!