WHITE PAPER

分散制御システム向け Honeywell Experion® プラットフォームにおける重大な脆弱性の発見

ドブ・ミルグロム
バラク・ハダド
トム・ゴル

目次

はじめに

近年、オペレーショナル・テクノロジー（OT）を標的にした攻撃が顕著に増加しています。この傾向は、サイバー犯罪者と国家行為者の両方が、このようなシステムを標的にすることに大きな価値があることを認識するようになり、重要インフラシステムが遭遇するリスクが増大していることを浮き彫りにしています。Armis Research Labsのセキュリティ専門家は、農業や水管理から製薬や原子力プラントまで、さまざまな産業で広く使用されているハネウェルの分散制御システム（DCS）用Experion⊠プラットフォームに9件の脆弱性を発見しました。このうち7つの脆弱性は、リモートでのコード実行を可能にするクリティカルなものです。これらの脆弱性を悪用すれば、DoSや身代金要求から機器の破壊や周囲への危険まで、あらゆる事態を招く可能性があります。脆弱性は、ハネウェル社独自のCDAプロトコルと、同ベンダーが実装するCDAデータ・クライアント・ネームド・アクセス・プロトコルに見つかりました。脆弱性の発見以来、ArmisとHoneywellは、その影響を慎重に明らかにし、継続的な緩和策を提供するために協力してきました。

Armis について

Armis Research Labsのセキュリティ専門家は、テクノロジーとセキュリティの最前線に立つことにコミットしています。プロトコル解析やコード検査などの高度な調査技術を採用することで、レガシーな産業機械から最先端のクラウドコンピューティングインフラまで、さまざまなシステムを真摯に調査し、潜在的なセキュリティリスクを積極的に特定して軽減します。接続された家庭用電化製品であれ、シンプルなIoTデバイスであれ、重要な制御システムであれ、より安全なデジタル環境を追求するために手を抜かないことが私たちの使命です。

私たちのこれまでの研究には以下のようなものがあります:

TLStorm Learn More	APC Smart-UPSデバイスに3つの重大な脆弱性が発見され、攻撃者は数百万台のエンタープライズデバイスの電源をリモートで操作できる可能性がある。
PwnedPiper Learn More	病院で使用される重要なインフラであるSwisslog PneumaticTubes System (PTS)に9つの脆弱性。
Modipwn Learn More	認証バイパスにより、Schneider Electric Modicon PLC でリモートコードが実行される。
NAT Slipstreaming 2.0 Learn More	NATによるVoIPプロトコルのサポートを悪用したNATバイパス技術。
EtherOops Learn More	ファイアウォールや NAT をバイパスするためにイーサネットケーブル上のパケットインパケット攻撃を利用するエクスプロイト。

CDPwn Learn More	Cisco Discovery Protocol の様々な実装に 5 つの重大な脆弱性。
URGENT/11 Learn More	最も広く使用されているリアルタイムオペレーティングシステム（RTOS）であるVxWorksに影響を及ぼす11件のゼロデイ脆弱性
BLEEDINGBIT Learn More	エンタープライズグレードのアクセスポイントに組み込まれている Texas Instruments 社の BLE チップにチップレベルの脆弱性が 2 つある。
BlueBorne Learn More	53億台以上のデバイスで使用されているBluetoothスタックの RCE脆弱性を介して、デバイスを標的とする攻撃。

Crit.IX とは何か

Armis Research Labsの研究者は、ハネウェルが設計し、Honeywell Experion®サーバとC300コントローラ間の通信に使用されている独自のプロトコルであるCDAプロトコルの弱点を明らかにしました。このプロトコルには暗号化と適切な認証メカニズムがないため、ネットワークにアクセスできる人なら誰でもコントローラとサーバーの両方になりすますことができます。さらに、CDAプロトコルには設計上の欠陥があるため、データの境界を制御することが難しく、バッファオーバーフローにつながります。

また、HoneywellはExperion Server に CDA Data Client Named Access プロトコルを実装しており、ExperionR サーバーと ExperionR アプリケーション間の通信に使用されています。このプロトコルは、Honeywell Experion®サーバーとExperion®アプリケーション間の通信に使用され、これらのアプリケーションによるタグ名アクセスを可能にします。ハネウェルのこのプロトコルの実装には4つの脆弱性が含まれており、そのほとんどがExperion®サーバー上でリモートコード実行（RCE）を許すものであることが判明しました。

分散制御システム（DCS）

分散制御システム(DCS)は、ワークスペース全体で使用されるさまざまなデバイスの機能を合理化する自動制御システムです。これは、工場、機械、または制御エリア全体に地理的に分散した制御ループを使用するデジタル自動産業制御システムです。すべての機械を操作する集中制御システムとは異なり、DCSは機械の各部分に、操作を実行する専用のコントローラを持たせることができます。DCSは、コンピュータ、センサー、コントローラーなど、工場やプラント全体に分散配置された制御要素で構成され、それぞれがデータ収集、データ保存、プロセス制御など特定の目的を果たしています。DCSの中核的な機能は、プラント全体に配置された多数のコントローラのそれぞれを監督し、調整しながら、プラント全体のオペレーションを指揮することです。DCSのコンセプトは、制御機能をプロセスプラントの近くに局所化し、遠隔監視と監督を行うことにより、信頼性を高め、設置コストを削減します。

DCSは、化学プラント、石油・ガス、食品加工、原子力発電所、水管理システム、自動車産業など、さまざまな産業で使用されています。

DCSアーキテクチャ

分散制御システム（DCS）アーキテクチャでは、システムは通常いくつかのレベルに分割され、それぞれが特定の機能を果たし、さまざまなコンポーネントで構成されます。これらのレベルとコンポーネントが連携することで、工業プロセスの効果的な制御と監視が可能になります。DCSアーキテクチャのレベルは、具体的な実装によって異なる場合がありますが、以下のレベルが一般的です:

- **フィールド・レベル:**
  階層の最下層に位置するフィールド・レベルは、センサー、アクチュエーター、制御バルブなどの物理的なフィールド・デバイスで構成されます。これらのデバイスは、工業プロセスと直接相互作用し、データを収集し、電気信号またはデジタル信号に変換する。フィールドレベルには、温度センサー、圧力トランスミッター、流量計、電動バルブなどのデバイスが含まれます。

- **制御レベル:**
  制御レベルは、コントローラとI/Oモジュールが配置されている場所です。フィールド機器からデータを受信し、制御アルゴリズムを使用して処理します。コントローラは制御ストラテジーを実行し、産業プロセスを調整するための出力信号を生成します。I/Oモジュールは、コントローラとフィールド・デバイス間のインターフェースとして機能し、フィールド・デバイスからの信号をコントローラで処理できる形式に変換します。制御レベルには、DCSコントローラ、I/Oモジュール、コントロールバルブなどのデバイスが含まれます。

- **監督レベル:**
  監督レベルは、より高度な制御と監視を提供する。これには、オペレーター・ステーション、サーバー、およびエンジニアリング・ワークステーションが含まれます。このレベルのオペレータは、システム全体を監視し、リアルタイムのデータを表示し、システムのパフォーマンスに基づいて意思決定を行います。サーバーは、オペレーター・ステーションとコントローラ間のデータ移動を担当します。エンジニアリング・ワークステーションは、システムの設定、プログラミング、およびメンテナンスに使用されます。

- **アドバンス・コントロール・レベル:**
  基本的な制御機能の上に実装される、より高度な制御および最適化戦略を指します。工業プロセスの全体的な性能、効率、安定性を向上させるための高度なアルゴリズムや技術が含まれます。製造実行システム（MES）は通常このレベルで動作します。

- **エンタープライズ・レベル:**
  エンタープライズ・レベルはDCSアーキテクチャの最上位レベルであり、DCSをより広範なビジネス・プロセスと連携します。通常、エンタープライズ・リソース・プランニング（ERP）などのビジネス管理システムが含まれます。

分散制御システム（DCS）アーキテクチャは、集中監視制御、専用コントローラ、分散処理、スケーラビリティ、企業システムとの統合により、産業用制御システム（ICS）において際立っています。これらの特徴により、包括的な制御、正確な監視、効率的なデータ処理、柔軟な拡張が可能になります。

A Distributed Control System (DCS) Example

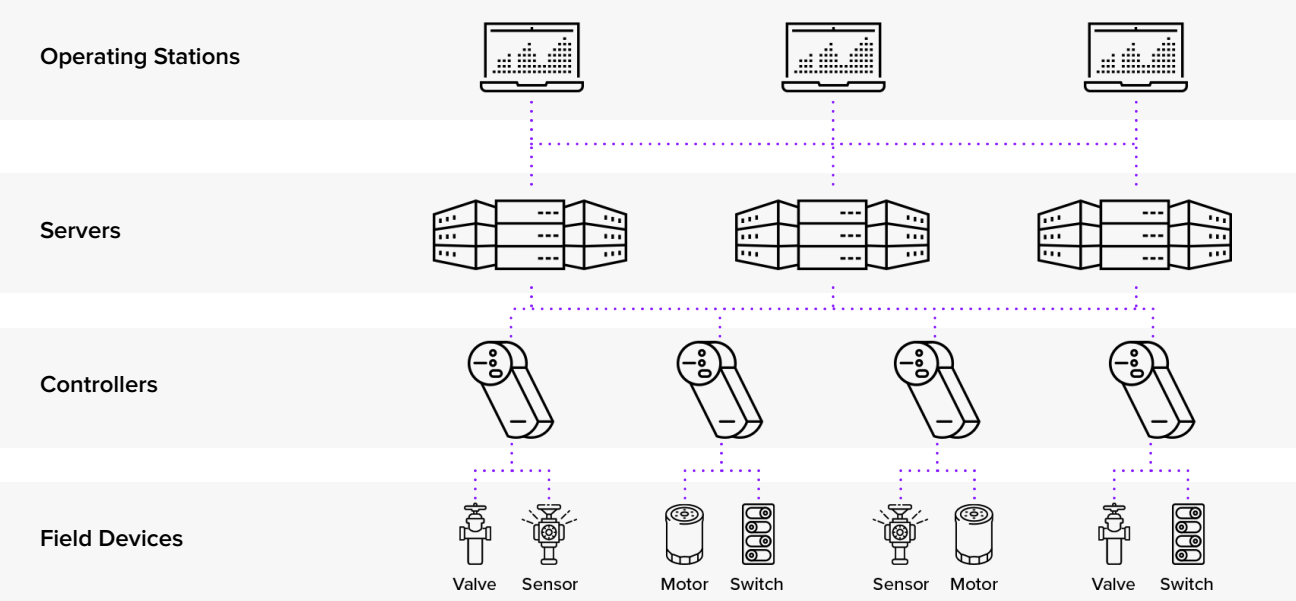

図1.分散制御システム（DCS）アーキテクチャ。

Honeywell Experion® Research

Armis Research Labsでは、様々なデバイスのプロトコルとセキュリティを理解するために研究を行っています。プロトコルを十分に理解した上で、関連するデバイスの使用状況を監視し、異常がないかを確認することで、攻撃者や誤動作を検出することができます。

私たちが分析したシステムの1つは、Experion®サーバーとC300 DCSコントローラーを含むハネウェルExperion®プラットフォームです。

ハネウェルは、さまざまな目的のためにいくつかのプロトコルを実装しています。この研究で興味深いものは以下の通りです:

- Generic Communication Layer（GCL）ネームサービス - ユニキャストサービスで、コントローラの存在と基本プロパティの一部をネットワークに通知する役割を持ちます。GCLネームサービスは、コントローラに接続するデバイスに、各プロトコル/サービスで使用するポートを通知する役割も果たします。
- CDA（Control Data Access）プロトコル - 管理サーバーがコントローラのプロパティを読み書きするために使用します。

CDAサーバーサービスは、CDAプロトコルを使用したコントローラとの通信を担当します。その目的は、C300コントローラとExperion®サーバーおよびコンソールステーションをリンクすることです。

つまり、私達は2つの調べたいCDAサーバーのインターフェースがありました。CDA Data Client Named Accessサーバー側とCDAクライアント側です。CDAプロトコルのクライアントとサーバーの動作は対称ではないので、サーバーとコントローラーの両方を調べる必要がありました。

C300 DCS コントローラの解析

C300コントローラはPowerPCプロセッサを搭載し、INTEGRITYオペレーティングシステムを実行しています。CDAプロトコルを利用する関連プロセスを検出した後、それを確認したところ、非直列化プロセスで興味深いものを発見しました。

コントローラプロセッサはビッグエンディアンであり、通常はリトルエンディアンのx86プロセッサと通信します。そのため、このプロトコルは、クライアントまたはサーバーのどちらかがデータのエンディアンを固定できるように設計されています。このプロトコルの接続リクエストには、クライアントとサーバーのどちらがエンディアンを固定するかを決定するフラグがあります。Honeywellは、クライアントとサーバーの両方がデータのエンディアンを処理し、固定するために使用する “ntoh “と呼ばれるライブラリを実装しました。

ntohライブラリーはすべてのメッセージを受け取り、その型をチェックします。型に従って、ライブラリーはメッセージの構造を解析し、各フィールドのバイトを個別に入れ替えます。このライブラリーは、メッセージの内容そのものは利用しませんが、エンディアンを固定するのに、型やサイズに関連するすべてのフィールドを処理しなければなりません。このため、このライブラリは解析の過程で非常に役に立ちます。メッセージの構造に関するすべての情報が一箇所にまとまっているので、たとえ各フィールドの意味がわからなくても、どんなパケットでもうまく解析するのに十分な情報が得られるのです。

このライブラリを調査していたところ、クライアント入力に対する境界チェックがほとんど行われていないことが判明しました。このため、リモートでコードを実行できるメモリの脆弱性があります。このライブラリはクライアントコントローラと管理用Windowsサーバで同じように使用されているため、これらの脆弱性はC300コントローラとExperion®サーバの両方に影響します。

ntoh “ライブラリーの脆弱性を発見した後、私たちはアタックサーフェスに面しているすべてのネットワークのマッピングを開始しました。

CDAサービスの分析

C300 コントローラは、Experion® サーバ上の CDA サービスと CDA データクライアントプロトコルを使用して通信し、これには認証が必要ありません。このプロトコルの目的は、エクスペリオン・サーバ/コンソール・ステーションとエクスペリオン・コントローラの間でデータを交換することです。このプロトコルには、コントローラのプロパティの読み取り、書き込み、変更のための複数のメソッドがあります。

サーバーは、アイテムの読み取り要求を取得すると、CDAプロトコルを介してそのプロパティを要求するリクエストをコントローラに送信し、CDAデータクライアントネームドアクセスクライアントに応答を返します。

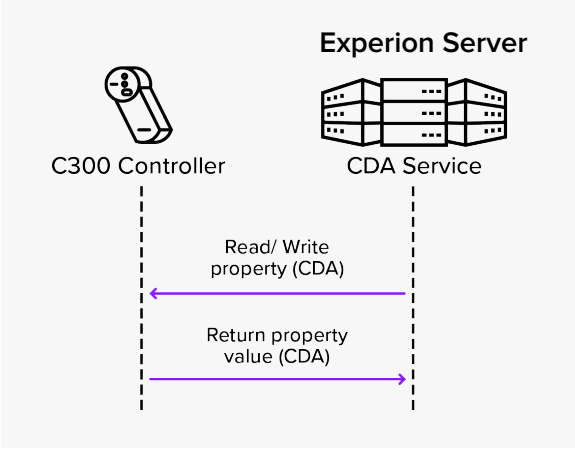

図2.ExperionサーバーはCDAを介してコントローラと通信する。

CDAデータクライアントネームドアクセスインターフェースには、プロパティの読み取りと書き込みを可能にするメソッド以外に、システム内のコンポーネントを制御するためのメソッドもいくつかあります:

- サーバー・データベースにコンポーネントを追加する
- 全コンポーネントのリストを取得
- 各コンポーネントのIPを取得する
- より多くの種類のコンポーネントをサーバーにロードする
- このクライアントのグループとハンドルをダンプするためのデバッグコマンド

少し調べてみると、このインターフェースには多くの弱点があることがわかりました。ほとんどの関数はセキュリティ・チェックを行っておらず、セキュリティ・チェックを行う関数は実装が不十分です。これらのセキュリティ問題は深刻で、サービス拒否や情報漏洩、さらにはリモート・コード実行につながる可能性があります。

ハネウェル・プロトコル

ハネウェルは一連のプロトコルを使用しており、それぞれが独自の目的をもっています:

- GCL - Generic Communication Layer（ジェネリック・コミュニケーション・レイヤー）。Honeywellプロトコルのラッピングレイヤー。
- CDA - Control Data Access（コントロール・データ・アクセス）。C300コントローラとExperion® Server上のCDAサービス間の通信に使用。

GCLプロトコル

すべてのHoneywellプロトコルは、GCL（Generic Communication Layer）ヘッダーのプレフィックスを持っています。GCLヘッダには、サービスとクライアントに関するメタデータが含まれます。コネクション・ハンドシェイクの一部として、クライアントはサービスの名前と識別子を含む “verify connection “リクエストをサーバーに送ります。

接続要求の確認

verify connection “リクエストには、LEまたはBE（リトルエンディアンまたはビッグエンディアン）フラグが含まれており、データのエンディアンを固定する必要があるのがクライアントかサーバーかを決定します。また、接続先のサービス名も含まれます。これは確認用であり、サービスの選択用ではありません。サービスの例:

- PsCdaCeeNtComm - コントロール データ アクセスサービス
- PsCdaSrvNamAcc - CDA データクライアントプロトコル
- EpicMo - コントローラ管理およびファームウェアアップグレードサービス

CDAプロトコル

CDAメッセージには2つのレイヤーがあります:

1. メッセージヘッダ - セカンドレイヤーのタイプ、メッセージのサイズ、リクエスト数 を含む。
2. リクエストリスト - リクエストのリスト。各リクエストは、1つのプロパティに対するアクションを表し、そのサイズ、タイプ、そして書き込みリクエストの場合 は、内容も表す。

このプロトコルは、コントローラのパラメータの取得と設定に使用されます。このプロトコルでは、GCLヘッダの後に、ヘッダサイズ、メッセージタイプなどを指定するメッセージヘッダが続きます。

リクエスト構造

各リクエストは以下のような構造になっています:

- ID - メッセージ固有のもので、リクエストとレスポンスをリンクするために使用される。
- Size - リクエストヘッダとデータの合計サイズ。
- Req or Resp - リクエストかレスポンスかを示す。
- Type - データ型（int、string、doubleなど）。このフィールドは、基本型の実際のデータサイズも決定する。サイズと型の間に矛盾があると、望ましくない結果につながることがある。
- Device Address - 問い合わせるデバイスのアドレス。
- Parameter ID - リクエストのパラメータのID。
- 書き込み要求の場合、データはリクエストヘッダーの後に続く。

CDAデータクライアントネームドアクセスサービスの分析

CDAデータクライアントネームドアクセスプロトコルは、名前付きアクセスを必要とするExperionアプリケーションに、アイテムの読み取り、書き込み、削除、変更を許可します。

まず、クライアントは一意のグループIDで識別される新しいグループを作成します。 各グループには、それぞれ特定のタイプ、サイズ、および一意の識別子としてのアイテムの配列が含まれています。各グループについて、ユーザーはアイテムを識別するために名前を使用するかどうかを決定することができます。これは、内部DBからこのハンドルに関するデータをロードすることによって行われます。ユーザはハンドル単体で記述することもできます。各メソッド（READ、WRITEなど）に2つのオプションがあるのはこのためです。例え

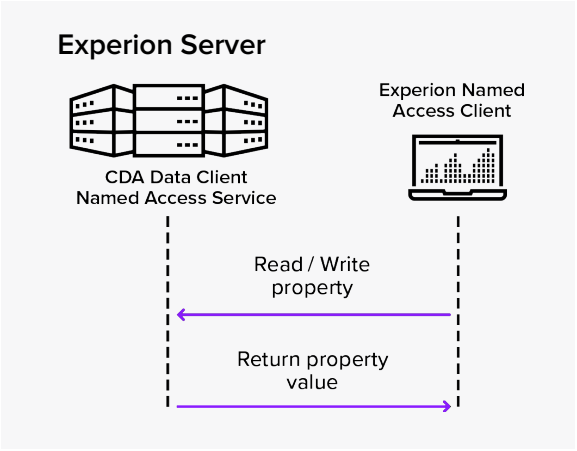

図3.CDAデータクライアントネームドアクセスクライアントはExperionサーバーと通信する。

ば、ReadNamesとReadHandlesです。

CDAデータクライアント名前付きアクセスプロトコルでは、基本的なデータオブジェクトはアイテムです。各データオブジェクトには、それを記述するハンドルがあります。ハンドルには、このプロパティのID、データのタイプとサイズ、このプロパティが格納されているコンポーネントのタイプとそのコンポーネントのアドレスが含まれています。例えば

Image VersionのC300コントローラーハンドル:

- ID - 0x13d
- タイプ - 文字列
- サイズ - 最大0x20
- コンポーネント・タイプ - PCM
- コンポーネントアドレス - 0x3e8

CDAデータクライアントの名前付きアクセスクライアントがサーバーに接続し、アイテムの値を読み取りまたは更新したい場合、関連するすべてのアイテムの新しいグループを定義し、グループ全体をまとめて読み取りまたは更新します。サーバーに項目を指定するために、クライアントには2つのオプションがあります:

- アイテム名を送信すると、サーバーは内部DBからアイテムのハンドルを取得する。
- アイテムの適切なメタデータとともにハンドルそのものを送信する。

グループの例:

グループID: ***0x000A0001***

アイテム

- Image Version、ID - 0x13d、タイプ - 文字列
- 初期化完了、ID - 0x13e、タイプ - 論理値

各CDAデータクライアントネームドアクセスプロトコルメッセージは40バイトのヘッダーから始まり、コマンド・タイプ、グループIDなどが記述されています。

CDAデータクライアント命名アクセスメッセージ構造

CDA Data Client Named Accessのメッセージタイプはそれぞれ異なる構造を持っています。このホワイトペーパーでは、2つのコマンドの構造について説明します: WRITE_HANDLESとGet_db_path_for_driverです。

WRITE_HANDLES コマンド

このメソッドは、名前ではなくハンドルを明示的に指定して、プロパティをデバイスに書き込むために使用されます。このメッセージには2つのリストが含まれます:

- ハンドルリスト - 各ハンドルはアイテムを表す
- データ要素リスト - アイテムに書き込む各データ要素

データ要素の構造

データ要素記述子は28バイトです。整数やバイトのような基本的なデータ型の場合、データそのものが記述子の一部となります。データが8バイトより長い場合は、記述子の後にデータのサイズを指定する別のuintがあり、その後にデータそのものがあります。

タイプ	サイズ	データ (8バイトまでの基本型用)	長さ (文字列と構造体の場合)	データ (文字列と構造体の場合)

ここに見られるように、データ型はメッセージの中で2回指定され、サイズは3回指定されています。このような冗長性は、最終的に型の混乱を招きます。

Get_db_path_for_driver コマンド

このメソッドは、クライアントから受け取ったパラメータから接続文字列を生成します。この接続文字列は、後でデバイスをシステムに追加する際に使用できます。

ネットワークタイプ	パスのサイズ	パス	名称

パスのサイズ - パスの要素数、各要素は4バイト長。

脆弱なプロダクト

ハネウェル社が実施したCrit.IXの脆弱性の調査および同社の各種DCSプラットフォームとの比較により、以下の製品が脆弱性の影響を受けることが判明しました:

CVE ID	重大性	CVSS	製品	プラットフォーム	バージョン
CVE-2023-23585 CVE-2023-22435 CVE-2023-24474 CVE-2023-25078	クリティカル	**9.8**	エクスペリオン・サーバー エクスペリオン・ステーション	**EPKS**	**501.1-520.2**
CVE-2023-25178	高い	**7.5**	エンジニアリング・ステーション ダイレクト・ステーション	**LX**	**510.5-520.2**
			エンジニアリング・ステーション ダイレクト・ステーション	**PlantCruise**	**510.5-520.2**
CVE-2023-24480 CVE-2023-25948 CVE-2023-25770	クリティカル	**9.8**	**C300**	**EPKS**	**501.1-520.2**
CVE-2023-26597	高い	**7.5**		**LX**	**510.5-520.2**
				プラントクルーズ	**510.5-520.2**

発見された脆弱性

Experion® CDAサーバーの脆弱性

エンディアンの修正 - CVE-2023-24474

発見された脆弱性はバッファオーバーフローで、Experion®サーバー上でリモートコード実行が可能です。CDAプロトコルでは、メッセージのエンディアンをクライアントとサーバーのどちら側で固定するかを柔軟に決めることができます。このオプションは、セッションの開始時に送信される接続要求のフラグによって決定されます。ハネウェルはLEからBEへの変換を処理するライブラリを実装しました。

ntoh “ライブラリは、まずメッセージのヘッダーと各リクエストのヘッダーを 正しい順序に変換し、次にデータサイズとデータ型に応じてデータフィールドを固定します。実装を単純化するために、変換が単純にバイトの逆変換であるすべての型(short、 int、int64、double)に対して、ライブラリは逆memcpyを呼び出し、逆変換が必要なバイト数を知るためにリクエストヘッダのsizeパラメータを使用します。sizeパラメータのチェックがないため、攻撃者は最大65535バイトのバイト配列を逆変換することができ、実際には最大0x2000（CDAメッセージバッファの最大サイズ）のデータを書き込むことができます。メッセージ・バッファはobject vtablesとグローバル・ポインタの両方を含む静的メモリの一部であるため、これはリモート・コードの実行につながります。

CDAデータクライアントの名前付きアクセス脆弱性

CDAデータクライアントネームドアクセスプロトコルは、名前付きアクセスを必要とするExperionアプリケーションに、アイテムの読み取り、書き込み、削除、変更を許可します。まず、クライアントは一意のグループIDで識別される新しいグループを作成します。各グループにはアイテムの配列が含まれ、各アイテムは特定のタイプ、サイズ、一意の識別子を持っています。

各グループについて、ユーザーはアイテムを識別するために名前を使うかどうかを決めることができます。これは、内部DBからこのハンドルに関するデータをロードすることによって行われます。ユーザーはハンドルを明示的に記述することもできます。各メソッド（READ、WRITEなど）に2つのオプションがあるのはこのためです。例えば、ReadNamesとReadHandlesです。

クライアントに公開されるCDA Data Client Named Accessサービスメソッドのリスト:

- CANCEL_REQUEST
- READ_NAMES
- READ_HANLES
- WRITE_NAMES
- WRITE_HANDLES
- SUBSCRIBE_NAMES
- SUBSCRIBE_HANDLES
- SUBSCRIBE_ADD_NAMES
- SUBSCRIBE_ADD_HANDLES
- SUBSCRIBE_DELETE
- SUBSCRIBE_TRIGGER
- MESSAGE
- REGISTER_NAME
- DEBUG_CMD
- Get_db_path_for_driver
- Get_driver_list
- Set_Comm_Path_For_Pcm
- Get_Ip_Address
- LOG_INFORMATION

各メッセージには40バイトのヘッダーがあり、メッセージのサイズ、アイテムの数、グループID、コマンドなどが含まれています。ヘッダーの後には、アイテムのシリアライズされた配列があります。

クライアントがアイテムの記述子として名前を使用する場合、アイテムはname文字列のサイズとunicode文字列を記述する32ビットとなります。クライアントがハンドルを使用する場合、アイテムのサイズとタイプを記述する24バイトのヘッダとなります。

Writeリクエストの場合、メッセージはデータ自体で構成される。データには、データ・サイズとタイプを記述する28バイトのヘッダもあります。データ型とサイズを記述するためのこのヘッダーの冗長性は不要であり、型の混同を引き起こし最終的には攻撃者がプロセスメモリでread権限を得る可能性があります。

WRITE_HANDLES におけるデータの混乱 - CVE-2023-25178

WRITE_HANDLESリクエストでは、最初にデータタイプとサイズを記述したハンドルの リストを送ります。次に、データのサイズとタイプを含むヘッダーとともに実際のデータを送ります。

データのヘッダーはそのまま_data_elementと呼ばれるオブジェクトの配列にコピーされこのオブジェクトは2つの方法でバッファを使用します：データのサイズが8バイト以下の場合、バッファにコピーされます。そうでない場合、最初の4バイトは新しく割り当てられたバッファへのポインタとして使われます。オブジェクトはそのままコピーされるため、攻撃者はポインタ・フィールドに任意の値を書き込み、型をint32_tに設定することができます。それはわずか4バイトであり、外部バッファを必要としません。その後、プログラムはデータを取り出し、CDAデータクライアントネームドアクセスリクエストで言及されたPCMに送信します。その際、_data_elementヘッダーを使用する代わりに、攻撃者が先にデータの型について提供したハンドルを使用します。その結果、攻撃者は型をStringに設定することができ、プログラムはデータの最初の4バイトをポインタとして扱い、ヌル終端にぶつかるまで、それが指すデータをデバイスに送信します。PCMとサーバー間の接続をリスニングすることで、そのデータをキャッチすることができます。

```
char *__cdecl copy_list_of_data_elements_into_Carray(clsPsCommBuf *msg, OPC_data_element *OPC_data_element_array, unsigned int num_of_elem
{
  char *buffer; // eax
  unsigned int i; // [esp+0h] [ebp-10h]
  char *_buffer; // [esp+Ch] [ebp-4h]

  buffer = get_buffer(msg);
  _buffer = buffer;
  for ( i = 0; i < num_of_elemnts; ++i )        // num_of_elemnts not checked against msg bounds
  {
    reset(&OPC_data_element_array[i]);
    memcpy(&OPC_data_element_array[i], &_buffer[*offset], sizeof(OPC_data_element));// the data is copied into the array
                                                // straight from the msg buffer without any checks.
    *offset += 0x1C;
    *offset += copy_len_of_bytes(&OPC_data_element_array[i], &_buffer[*offset]);//
                                                // copy bytes eve if there is null inside it
                                                // len is the first 4 bytes, and not checked.
    buffer = (char *)(i + 1);
  }
  return buffer;
}
```

図4.data_elementをリクエスト・バッファから_Data_Element配列にそのままコピーする

```
UINT __thiscall copy_len_of_bytes(OPC_data_element *this, void *Src)
{
  OPC_data_element *v3; // [esp+4h] [ebp-B4h]
  WCHAR v4[82]; // [esp+8h] [ebp-B0h]
  DWORD v5; // [esp+B0h] [ebp-8h]
  UINT len; // [esp+B4h] [ebp-4h]

  v3 = this;
  switch ( this->type )
  {
    case 2:
    case 6:
    case 7:
    case 8:
    case 9:
    case 0x14:
    case 0x15:
    case 0x1A:
    case 0x1D:
    case 0x1E:
    case 0x1F:
    case 0x22:
    case 0x23:
    case 0x24:
      memcpy(&len, Src, sizeof(len));           // read primitive because buffer is controlable
      v3->buffer = SysAllocStringByteLen((LPCSTR)Src + 4, len);
      if ( !v3->buffer )
      {
        reset(v3);
        v5 = GetLastError();
        wsprintfW(v4, L"Error return code from GlobalAlloc = %li.", v5);
      }
      len += 4;
      break;
    default:
      len = 0;
      break;
  }
  return len;
}
```

図5.タイプ1を使えば、this->bufferは上書きされない

```
v6 = (OPCItem *)get_ptr_data((void **)&v38);
item_handle_type = get_item_handle(v6)->type;// use item handle type instead of OPC_data_element type
v7 = (OPCItem *)get_ptr_data((void **)&v38);
v8 = get_item_handle(v7);
LOWORD(data_element_size) = get_item_size_from_item_type(&v35[v37 - 1], v8);
data_element = alloc_data_element(data_element_size, item_handle_type);// by default alloc_data_element
                                        // allocate 0x208 bytes for its buffer
if ( data_element )
{
  v9 = (OPCItem *)get_ptr_data((void **)&v38);
  item_handle = get_item_handle(v9);
  if ( !copy_from_OPC_to_data_element(&v35[v37 - 1], data_element, item_handle) )
  {
    if ( data_element::get_field_4(data_element) == 0xDB90 )
      sloggerclient 4(
```

図6.データ要素は_data_elementではなく、item_handleに従って割り当てられる

```
  case 2:
  case 0x22:
  case 0x23:
    v6 = 2 * wcslen((const wchar_t *)this->buffer) + 2;
    if ( v6 <= get_avail_size(data_element) )
    {
      v7 = wcslen((const wchar_t *)this->buffer);
      copy_wstring_into_data_element(data_element, (unsigned __int16 *)this->buffer, 2 * v7);
    }
    else
    {
      data_element::set_field_4(data_element, 0xDB90);
      v24 = 0;
    }
    break;
```

図7.型が2（文字列）の場合、プログラムは我々がコントロールするthis->bufferからデータをコピーする

Get_db_path_for_driver スタックオーバーフロー - CVE-2023-22435

このメソッドはいくつかのパラメータを受け取り、それを使って DB のパス文字列を構築するように設計されています。パラメータのひとつは PathParm で、これは整数の配列です。クライアントは PathParm の整数の数を送信し、次に整数のリストを送信します。サーバは 0x104 個の整数である 0x410 バイトの整数の配列をスタック上に確保します。サーバは整数を配列にコピーする際に、配列のサイズが0x104整数以下であることをチェックしますが、これは符号付きチェックであるため、クライアントが負の整数を送信した場合はチェックをパスします。このため、クライアントは大きなバイト数でスタックを破壊します。しかし、クライアントが送信するのはバイト数ではなく整数数なので、サーバーはその数を4倍、言い換えれば2ビット左にシフトする必要があります。

しかし、チェックは乗算の結果ではなく、元の数値に対して行われます。これは典型的な “使用時チェック “シナリオであり、攻撃者はMSBビットだけがオンになっている数値を送信することができ、その数値はチェック時には負の数値とみなされるが、2シフト後には整数オーバーフローにより正の数値となります。例えば、0x80000100は、4倍すると0x00000400に等しくなります。これにより攻撃者は、任意のバイト数だけスタックをオーバーフローさせ、任意のタイプのデータでスタックを埋めることができます。

```
int __cdecl FormPathString_parse_input(int a1, int *network_type, int a3, int *path_size, char *path, int *a6, int *offset)
{
  void *v7; // eax
  int result; // eax
  size_t Size; // [esp+4h] [ebp-Ch]
  int v10; // [esp+8h] [ebp-8h]
  char *v11; // [esp+Ch] [ebp-4h]

  v11 = get_buffer((clsPsCommBuf *)a1);
  *network_type = *(_DWORD *)&v11[*offset];
  *offset += 4;
  sloggerclient_4(215, 40, off_4DCEE8[0], "497", "Network type %d ", *network_type);
  *path_size = *(_DWORD *)&v11[*offset];
  *offset += 4;
  Size = 4 * *path_size;                        // path_size is checked instead of size
  sloggerclient_4(215, 40, off_4DCEE8[0], "504", " Path size  %d", *path_size);
  if ( *path_size > 0x104 )                     // signed check
                                                // can overflow the integer
                                                //
  {
    sloggerclient_4(215, 10, off_4DCEE8[0], "515", "Invalid input: Path size is bigger than expected %d ", *path_size);
    sloggerclient_5(215, 10, off_4DCEE8[0], "516", v11, Size, "Print the PathParm buffer");
  }
  else
  {
    memcpy(path, &v11[*offset], Size);          // can put any number aligned to 4 and overflow the stack
                                                // path is allocated on stack with size of 0x40c
    sloggerclient_5(215, 40, off_4DCEE8[0], "511", path, Size, "Print the PathParm buffer");
  }
  *offset += Size;
  if ( *path_size > 0x104 )
    *path_size = 0;
  *a6 = *(_DWORD *)&v11[*offset];
```

図8.Get_db_path_for_driverコマンドの解析

Set_Comm_Path_For_Pcm バッファオーバフロー - CVE-2023-23585/25078

このメソッドでは、クライアントはPCMに関する多くの値を表す文字列を送信します。文字列は以下のようなキーー値 形式です: 「<キー>=<値>:<キー_2>=<値_2>...” 。サーバーは文字列を解析し、各値を特定のバッファにコピーします。クライアントはUnicodeバッファを送信するので、サーバはバッファのエンコーディングを変更するために “UnicodeToMultiByte “と呼ばれる関数を実装しています。この関数は2つの引数、Unicodeバッファ・ポインタとmultiByte文字列のアウト・ポインタを取ります。アウト・バッファがマルチバイト文字列に対して十分な大きさであることを保証するために、この関数はIsBadWritePtrを呼び出し、コピーする必要のあるmultiByte文字列のサイズを引数として送ります。さて、この関数は時代遅れなので使うべきではないものですが、これは最初の引数が指すメモリ領域に書き込み権限があるかどうかをチェックします。例えばスタックへのポインタを送った場合、スタック全体が書き込み可能なプログラムであるため、この関数はスタック全体のサイズまでtrueを返します。

```
int __cdecl convert_from_utf16_to_multibyte(const char **a1, LPVOID lp)
{
  const WCHAR *v2; // eax
  const WCHAR *v3; // eax
  int ucb; // [esp+0h] [ebp-8h]
  unsigned __int8 v6; // [esp+7h] [ebp-1h]

  v6 = 0;
  v2 = (const WCHAR *)ATL::CSimpleStringT<wchar_t,1>::operator wchar_t const *(a1);
  ucb = WideCharToMultiByte(0, 0, v2, -1, 0, 0, 0, 0);
  if ( !lp || IsBadWritePtr(lp, ucb) )          // IsBadWritePtr not really check the size of lp, but only if the ptr has write.
                                                // if ptr is allocated on the stack, that means the all stack, which lead to a stack overflow
    return v6;
  v3 = (const WCHAR *)ATL::CSimpleStringT<wchar_t,1>::operator wchar_t const *(a1);
  WideCharToMultiByte(0, 0, v3, -1, (LPSTR)lp, ucb, 0, 0);
  v6 = 1;
  return v6;
}
```

図9.この関数は、古い関数IsBadWritePtrを使用してUnicodeからmultiByteに変換します

この時点から、” UnicodeToMultiByte “関数は安全ではなく、この関数が使用されているすべての場所で、攻撃者がRCEを取得するために悪用できる可能性があります。 set_Comm_Path_For_Pcmはこの関数を6回使用します。スタック上に割り当てられたバッファ上で3回、ヒープ上で3回です。このため、攻撃者はスタック・オーバーフローの機会を3回、ヒープ・オーバーフローの機会を3回与えることになります。

Experion® C300 コントローラの脆弱性

エンディアンの修正 - CVE-2023-25770

ハネウェル社は、C300コントローラでも「ntoh」ライブラリを使用しているため、サーバで発見された脆弱性と同じ脆弱性がコントローラにも適用されます。

CDA プロトコル - CVE-2023-24480/26597

コントローラがサーバからのCDAプロトコルメッセージを解析するとき、何らかの理由でエラー(境界チェック、無効なサイズなど)を検出すると、コントローラはエラーコードパラメータを指定してSendErrorResponseを呼び出します。この関数は、すべてのメッセージ部分を繰り返し処理し、同じ数のリクエストでエラー応答を構築し、それぞれにエラーコードを設定します。その過程で、この関数はリクエストデータをスタック上のバッファにコピーします。問題は、メッセージのサイズフィールドが有効でない場合にもこのメッセージが呼び出されることです。つまり、エラー応答を送信する際に、既に知られている問題のある入力が再び使われることになります。これにより、攻撃者は2つの異なるシナリオでスタックをオーバーフローさせ、最終的にRCEに導くことができます。

さらに、この関数はレスポンス・バッファに書き込まれるパーツの数をチェックしません。すべてのリクエストに対するレスポンスは8バイトなので、十分な数のリクエストを送ると、レスポンス・バッファは簡単にオーバーフローし、システムはクラッシュします。

署名なしファームウェア設計の欠陥 - CVE-2023-25948

コントローラのファームウェアは署名されていないため、攻撃者は偽造ファームウェアを作成し、ユーザーの操作なしにネットワーク経由でインストールすることができる。

緩和策

ハネウェルは、以下のセキュリティパッチを公開しましたので、影響を受けるすべての組織に対し、速やかに適用することを強くお勧めします。

ハネウェルのお客様は、 https://process.honeywell.com/ にログインし、Technical Publications セクションを検索することで、パッチにアクセスし、適用することができます。ハネウェルの脆弱性開示プロセスに関する詳細については、https://www.honeywell.com/us/en/product-security. をご覧下さい。

報告された脆弱性に効果的に対処するためには、これらのパッチやファームウェアのアップデートを優先的に実施することが不可欠です。さらに、組織は、定期的なセキュリティ評価、侵入テスト、開発チームに対する包括的なセキュリティ・トレーニングなど、強固なセキュリティ対策に粘り強く取り組むべきです。

Experion PKS では、C300 の以前のハードウェア・バージョン（C300 PCNT01、C300 PCNT02）は、デバイスでセキュア・ブートや IPSec をサポートするハードウェア機能を備えていません。C300 のセキュリティ・ロック機能は、ファームウェアのロードを含む特定の機能に対して物理的なアクセスを要求するために使用できます。Experion PKSには、セキュアブートとIPSec通信をサポートする新しいハードウェアバージョンのC300（C300 PCNT05）があります。

アルミスによるサポート

OT 環境のコントローラやエンジニアリング・ワークステーションに存在する脆弱性を解決するためのパッチの開発と導入は、アタックサーフェス領域を減らすために不可欠です。ビジネスクリティカルであり、運用プロセスへの影響が大きいため、これらの資産に対するパッチのリリースとインストールには、非常に綿密な QA プロセスと、ほとんどの場合システムシャットダウンが必要であり、その調整と最終的な完了には長期間を要する可能性があります。影響を受ける資産は、長期間脆弱なままであると考えるのが妥当です。この間、これらの重要インフラ資産に対する攻撃を検知・防止するための緩和策を実施することができます。

Armisのお客様は、アセットインテリジェンスセキュリティプラットフォームを活用して、以下の方法でネットワークを保護することができます:

1. 包括的な資産の可視化を実現ハードウェアからファームウェア、ソフトウェアバージョンまで、あらゆる側面を網羅する正確なインベントリを取得することで、企業は環境内の脆弱なサーバやコントローラを効果的に特定することができます。
2. リスクに応じて優先順位をつける脆弱性管理プログラムを導入することで、組織は効果的にウィークポイントを最小化し、利用可能なパッチがないデバイスを標的とするエクスプロイトのリスクを低減することができます。さら

に、セキュリティパッチがリリースされたら速やかに適用することで、これらのデバイスの脆弱性を大幅に減らすことができます。

3. 発見された脆弱性は、脆弱なデバイスへのネットワーク・アクセスのみを必要とするため、ネットワーク・セグメンテーションは、これらの脆弱性の悪用を防ぐ上で大いに役立ちます。セキュリティ・レベルやデバイスの種類に基づいてネットワークを明確なセグメントに分離することで、組織は攻撃者の探索の動きを制限し、潜在的な脅威を効果的に封じ込め、脆弱なデバイスへの影響を軽減することができます。OT 環境のセグメンテーション作業は、パデュー・モデルのような業界参照モデルを使用して達成することができますが、これはOT 環境の論理的または機能的なビューを表し、OT 資産が期待される動作からの逸脱、特に IT ネットワーク内の資産を含む上位レベルの資産からレベル 0 およびレベル 1 に通信する資産を特定するために使用できるものです。セグメンテーションで、これらの資産のふるまいを理解することによって期待されるものだけをホワイトリストに登録することができます。

4. 十分に保護されたネットワークでさえ侵害の影響を受けやすいものです。そのため、ネットワーク全体にまたがり、IT、OT、IoTを含むすべてのデバイスを網羅するエクスプロイトの試みを特定できる堅牢な脅威検出システムの実装が不可欠となります。シグネチャベースの分析、異常検知、IOC（indicator of compromise：侵害の指標）を含む検知技術のブレンドを採用することで、セキュリティのレイヤーが追加され、攻撃発生時の全体的な防御態勢が強化されます。

まとめ

ここ数年、重要インフラを標的にしたサイバー攻撃の頻度と影響が顕著に増加しています。このような攻撃の魅力は、セキュリティが不十分なレガシー・システムに起因しており、大きな損害と経済的損失をもたらす可能性があります。さらに、以前は孤立していたネットワークが相互接続されるようになると、攻撃対象が拡大します。その結果、Crit.IXのような脆弱性は、サイバー戦争とサイバー犯罪の双方にとって有利なものとなっています。

セキュリティを念頭に置いて開発されたわけではないレガシー・テクノロジーが、現代に適応する過程でどのように脆弱性を生み出すのか、私たちは何度も目にしてきました。この例では、暗号化も認証もないシリアル接続のために作られたプロトコルやコード・ライブラリが、現代のネットワーク環境で再利用され、攻撃者にとって影響が大きく、最小の労力で重大な脆弱性をもたらします。

最後に、包括的な脆弱性の開示には、影響を受ける機器のベンダーからの積極的な協力が必要です。今回の開示では、脆弱性の徹底的な理解、影響を受ける機器の範囲、効果的な緩和策を確保するために、ハネウェル社と協力的な取り組みを行いました。このケーススタディは、セキュリティ研究者とベンダーの重要な協力関係を例証するものであり、より安全なデジタル・エコシステムの育成における責任ある情報公開の重要性を強調しています。

About Armis

Armis, the leading asset visibility and security company, provides the industry's first unified asset intelligence platform designed to address the new extended attack surface that connected assets create. Fortune 100 companies trust our real-time and continuous protection to see with full context all managed, unmanaged assets across IT, cloud, IoT devices, medical devices (IoMT), operational technology (OT), industrial control systems (ICS), and 5G. Armis provides passive cyber asset management, risk management, and automated enforcement. Armis is a privately held company and headquartered in California.

armis.com

info@armis.com

20221107-1