24th International Conference on Information Technology, Computer and Telecommunication
بیست و چهارمین کنفرانس بین المللی فناوری اطلاعات، کامپیوتر و مخابرات | گرجستان
Event Place: Tbilisi,Georgia
www.itstconf.ir
24th International Conference on Information Technology,Computer and Telecommunication

بررسی حملات (Server-Side Request Forgery) و روش های پیشگیری

امیر محمد زنده دل

دانشگاه بیرجند

وحید وحیدی نژاد

دانشگاه بیرجند

چکیده–در وب امروز، برنامه‌های وب اغلب یک URL کامل را به‌عنوان ورودی از کاربران دریافت می‌کنند. به‌طور معمول، وقتی یک برنامه وب این URL را دریافت می‌کند، سرور به آن متصل می‌شود. اما اگر URL به یک سرویس داخلی اشاره کند و سرور همچنان به این اتصال ادامه دهد، سرور در معرض حملات جعل درخواست از سمت سرور (SSRF) قرار می‌گیرد. این نوع حملات می‌توانند در صورت هدف قرار دادن سرویس‌های داخلی بسیار مخرب باشند. وقتی سرور در یک محیط ابری میزبانی شده باشد، این حملات ساده‌تر و به همان اندازه مخرب هستند. بنابراین، با رشد استفاده از رایانش ابری، خطر حملات SSRF نیز افزایش یافته است. تحلیل ما از بیش از ۶۰ گزارش آسیب‌پذیری SSRF نشان می‌دهد که سطح آگاهی توسعه‌دهندگان از این نوع آسیب‌پذیری به‌طور کلی پایین است، و اغلب در روش‌های دفاعی آن‌ها نقص‌هایی وجود دارد. حتی در مواقعی که این دفاع‌ها بدون نقص هستند، محدودیت‌های قابل‌توجهی در امنیت و عملکرد دارند.

واژگان کلیدی : جعل درخواست سمت سرور (SSRF) ، امنیت برنامه های وب ، حملات سایبری ، امنیت رایانش ابری ، آسیب پذیری های امنیتی

مقدمه

برنامه‌های وب همواره یک هدف جذاب برای مهاجمان بوده‌اند. این برنامه‌ها معمولاً به اینترنت عمومی متصل هستند و به‌طور معمول برای انجام کارهای حیاتی و مدیریت داده‌های مهم استفاده می‌شوند. با وجود توجه زیادی که به امنیت برنامه‌های وب شده است، تحقیقات کمتری در مورد حملات به رابطه اعتماد بین این برنامه‌ها و خدمات حیاتی در داخل یک شبکه هدف انجام شده است. این حملات می‌توانند بسیار خطرناک باشند و تقریباً تمام مکانیسم‌های امنیتی معاصر را بی‌اثر کنند.

(Brian Krebs, 2019)

طبق گزارش های جمع اوری شده توسط تیمی از متخصصان امنیتی در سراسر جهان و داده های آن ها از تجزیه و تحلیل گزارش های به دست آمده از تعدادی سازمان ها، سال ۲۰۲۱ سندی در سایت OWASP منتشر شد که ده مورد از اسیب پذیری های برتر برنامه های کاربردی وب را معرفی می کند. علاوه بر معرفی شدن دسته بندی با نام و محدوده های (Scop) تازه، ۳ دسته بندی جدید هم اضافه شده اند که حملات Server-Side Request Forgery یا به اختصار (SSRF) هم در بین انها دیده می شود. (شهریار شریفی، ۱۴۰۰)

حملات SSRF می‌توانند به روش‌های مختلفی از خدمات داخلی سوءاستفاده کنند، از ارسال ایمیل‌های اسپم گرفته (Eugene Farfel,2016) (Corben Leo, 2018) تا اجرای دستورات سیستم‌عامل به‌صورت از راه دور بر روی سرورهایی که برنامه‌های وب را اجرا می‌کنند.(Peter Adkins, 2017) (Cheng-Da Tsai, 2017) به‌ویژه، اگر سرور آسیب‌پذیر (یعنی سروری که برنامه وب آسیب‌پذیر را اجرا می‌کند) در یک محیط ابری میزبانی شده باشد، این حملات برای مهاجمان راحت‌تر و معمولاً خطرناک‌تر هستند.(Brian Krebs, 2019) (Andre Baptista, 2018) یکی از این حملات، کمتر از یک سال پیش علیه شرکت Capital One رخ داد که در آن بیش از ۱۰۰ میلیون درخواست اعتبار مشتری به سرقت رفت. (Brian Krebs, 2019)

تحلیل ما دو چالش اصلی برای توسعه‌دهندگان برنامه‌ها را نشان می‌دهد. اولاً، SSRF یک مشکل کم‌بررسی‌شده است. آگاهی توسعه‌دهندگان از این آسیب‌پذیری محدود است و این کمبود دانش معمولاً منجر به ایجاد مکانیسم‌های دفاعی ناقص می‌شود که قابلیت‌های مهاجمان را دست‌کم می‌گیرد. ثانیاً، همیشه مشخص نیست که بار حمله کجا وارد می‌شود (به‌عنوان مثال، ممکن است در یک فایل بارگذاری شده قرار گیرد). همچنین، مطالعه ما نشان می‌دهد که حتی وقتی مکانیسم‌های دفاعی به‌درستی توسط توسعه‌دهندگان پیاده‌سازی می‌شوند، هنوز هم از محدودیت‌های مهم امنیتی و عملکردی رنج می‌برند.

انواع سناریو های حملات SSRF :

۱. URL redirection

URL redirection به خودی خود یک حمله SSRF نیست، اما اگر یک وب‌سایت به ورودی‌های کاربر اجازه دهد که URL ها را برای ریدایرکت تعیین کنند و این ورودی‌ها به درستی اعتبارسنجی نشوند، ممکن است نفوذگر از این قابلیت برای انجام حملات SSRF استفاده کند.

فرض کنید در یک قسمت از وب‌سایت example.com، دکمه ای وجود دارد که شما را به وب‌سایتexample2.com منتقل می‌کند. پس از اینکه کاربر روی دکمه کلیک می‌کند، با URL **[1]** به یک وب‌سایت دوم منتقل می شود

```
https://example.com/index.php?redirect_to=https://example2.com
```

[1]

در این مرحله، یک درخواست با متد GET از طریق مرورگر کاربر به وب‌سایت example.com ارسال می‌شود که پارامتر redirect_to در آن، مشخص می‌کند که مقصد مرورگر به سمت چه سامانه‌ای می‌باشد. بعد از این مرحله، یک پاسخ (Response) از سرور example.com برای مرورگر کاربر برمی‌گردد که کد وضعیت (status code) آن، یکی از کد های مربوط به ریدایرکشن می باشد. پاسخ های مربوط به ریدایرکشن، با کد های وضعیتی ۳۰۱، ۳۰۲، ۳۰۳، ۳۰۷ یا ۳۰۸ ارسال می‌شوند. در مرحله آخر، مرورگر با هدر Location در پاسخ دریافت شده، که مقدار آن برابر با پارامتر redirect_to می باشد، به وب‌سایت مذکور منتقل می‌شود.

حال چه اتفاقی می‌افتد اگر نفوذگر، URL گفته شده را مانند URL **[2]** تغییر بدهد

```
https://example.com/index.php?redirect_to=https://attacker.com
```

[2]

در این صورت، چنانچه وب‌سایت example.com، اعتبار سنجی درستی روی پارامتر redirect_to انجام نداده باشد یا سیاست های درستی را برای ریدایرکشن تعیین نکرده باشد، کاربر از طریق این لینک وارد سایت نفوذگر می‌شود که می‌تواند یک قدم از سناریو حملات دیگر مانند حملات فیشینگ باشد.

شاید این سوال برای‌تان پیش بیاید که، چگونه قربانی متوجه انتهای لینک نمی‌شود؟

- یکی از دلایل آن این است که نفوذگر قربانی را با پیشنهاداتی مانند خواندن یک مطلب فوری و جذاب، برنده شدن یک جایزه‌ی بزرگ و حقه هایی مانند آن، هیجان زده می‌کند و او را در اضطرار قرار می‌دهد.
- دلیل بعدی این است که در چنین شرایطی بیشترین دقت قربانی، به نام دامنه در لینک مدنظر است. ولی با این حساب این گونه لینک‌ها به صورت ساده فرستاده نمی‌شوند. نفوذگر برای اینکه توجه قربانی را سمت دامنه‌ی سایت آسیب پذیر جلب کند تا وی نتواند لینک سایت مخرب را ببیند، آدرس سایت مخرب را URL encode می‌کند.

دلایلی که باعث می شود از این تکنیک در سناریو حملات SSRF استفاده شود:

۱. قربانیان معمولاً به آدرس‌های اینترنتی شناخته شده اعتماد می‌کنند و کمتر احتمال دارد که به تغییر مسیر مشکوک شوند.

۲. با استفاده از تغییر مسیر، مهاجم می‌تواند ردپای خود را پنهان کند و تشخیص اینکه حمله از کجا آغاز شده است را دشوارتر کند زیرا قربانی پس از وارد شدن به سایت جعلی و اجرای فایل و کد های مخرب، به جای نفوذگر در خواست هایی را به سمت سرور ارسال کند. (مونسی، ۱۴۰۰)

۲. Blind SSRF

Blind SSRF نوعی حمله است که در آن نفوذگر می‌تواند از یک سرور آسیب‌پذیر برای ارسال درخواست به URL های مختلف استفاده کند، بدون اینکه به پاسخ این درخواست‌ها دسترسی داشته باشد. در ادامه این حمله را در چند گام توضیح خواهیم داد:

۱. **شناسایی نقاط آسیب‌پذیر**

هکر ابتدا باید نقاطی را شناسایی کند که به او اجازه می‌دهندURL هایی را به سرور ارسال کند. این نقاط می‌توانند شامل APIها، فرم‌های ورودی یا هر قسمت از برنامه وب باشند که به کاربر امکان وارد کردن URL را می‌دهد.

۲. **ارسال درخواست آزمایشی**

پس از شناسایی ورودی‌های آسیب‌پذیر، هکر یک درخواست اولیه به این ورودی ارسال می‌کند. این درخواست معمولاً به یک URLعمومی یا شناخته‌شده ارسال می‌شود تا بررسی کند که سرور چگونه به آن پاسخ می‌دهد. این مرحله به هکر کمک می‌کند تا درک بهتری از نحوه عملکرد سرور به دست آورد.

۳. **ارسال درخواست به URL های داخلی**

در مرحله بعد، هکر می‌تواند درخواست‌هایی به URL های داخلی یا منابع حساس ارسال کند.

٤. **بررسی کدهای وضعیت HTTP**

پس از ارسال درخواست، هکر به کدهای وضعیت HTTP پاسخ توجه می‌کند. این کدها به هکر اطلاعات مهمی درباره موفقیت یا عدم موفقیت درخواست می‌دهند:

- **کد ۲۰۰:** نشان‌دهنده موفقیت است؛ یعنی سرور به URL داخلی دسترسی داشته و اطلاعاتی برگشت داده است.
- **کد ۴۰۴:** نشان‌دهنده این است که URL وجود ندارد و به این معنی است که سرور به آن دسترسی نداشته است.
- **کد ۳۰۲:** نشان‌دهنده این است که سرور درخواست را به یک URL دیگر هدایت کرده است، که ممکن است به این معنا باشد که سرور به URL دسترسی داشته اما به دلایل امنیتی به آدرس دیگری منتقل شده است.

۵. **تحلیل نتایج**

با توجه به کدهای وضعیت، هکر می‌تواند بفهمد که آیا به منابع داخلی دسترسی داشته یا خیر. به عنوان مثال:

- اگر کد ۲۰۰ دریافت کند، احتمالاً اطلاعات حساسی در دسترس است.
- اگر کد ۴۰۴ دریافت کند، هکر متوجه می‌شود که دسترسی به URL ناکام بوده است.

۶. تکرار و گسترش تست

هکر می‌تواند با تغییرURL ها و ارسال درخواست‌های مختلف، به شناسایی دیگر منابع داخلی بپردازد و از کدهای وضعیت برای فهمیدن وضعیت دسترسی استفاده کند. این فرآیند ممکن است شامل آزمایش آدرس‌های مختلف متادیتا یا فایل‌های محلی باشد.(گروه لیان، ۱۴۰۰)

۳. Cloud Metadata SSRF

حمله Cloud Metadata یک نوع خطرناک از حمله ی SSRF است که به مهاجم اجازه می دهد تا با سوء استفاده از آسیب پذیری های موجود در برنامه های وب، به اطلاعات حساس در محیط های ابری مانند آمازون، مایکروسافت Azure و گوگل کلود دسترسی پیدا کند. مهاجمان با تزریق آدرس های خاص به برنامه های وب آسیب پذیر، می توانند به داده های متادیتا (Metadata) که حاوی اطلاعات مهمی مانند کلید های دسترسی، توکن های احراز هویت و پیکر بندی های سیستم هستند، دسترسی پیدا کنند.

این اطلاعات به مهاجم اجازه می دهد تا کنترل کاملی بر روی منابع ابری کسب کرده و عملیات مخرب انجام دهد.برای جلوگیری از این حملات، سازمان ها باید به شدت ورودی های کاربران را اعتبار سنجی کنند، دسترسی به اعتبار سنجی های متادیتا را محدود کرده و به روزرسانی های امنیتی را به طور مرتب انجام دهند.(Mallory Mooney, 2024)

۴. File Based SSRF

یک اسیب پذیری امنیتی است که به مهاجم اجازه می دهد تا به فایل های سیستم یک سرور دسترسی پیدا کند.این اتفاق زمانی می افتد که یک برنامه وب، بدون بررسی دقیق ورودی کاربر، در خواست های ارسالی برای دسترسی به فایل ها را به سرور ارسال می کند.

مهاجم با فریب دادن برنامه، به جای در خواست یک صفحه وب، درخواست دسترسی به یک فایل حساس (مانند فایل پیکر بندی یا پایگاه داده) را می دهد.این امر به مهاجم امکان می دهد تا به اطلاعات محرمانه دسترسی پیدا کند یا حتی کد های مخرب را اجرا کند.برای جلوگیری از این حمله، توسعه دهندگان باید ورودی کاربران را به دقت بررسی کنند و دسترسی به فایل های سیستم را محدود کنند.(Busra Demir, 2020)

پیامد های حملات SSRF

ردیف	پیامد	توضیحات
۱	دسترسی به منابع داخلی	حملات SSRF می‌توانند به مهاجم اجازه دهند تا به منابع داخلی دسترسی پیدا کند، مانند APIهای خصوصی و دیتابیس ها.
۲	دستکاری داده ها	مهاجم می‌تواند داده های درون شبکه داخلی را تغییر داده و باعث بروز مشکلات جدی شود.
۳	استفاده از وب سایت های مخرب	با استفاده از SSRF، مهاجم می‌تواند درخواست‌هایی به وب‌سایت‌های مخرب ارسال کند، که منجر به آلودگی سیستم می‌شود.
۴	افشای اطلاعات حساس	اطلاعات حساس مانند اطلاعات کاربری و رمزهای عبور می‌توانند از منابع داخلی استخراج شوند.
۵	محیط های Cloud	در سیستم‌های مبتنی بر ابر، حملات SSRF می‌توانند به مهاجمین دسترسی به خدمات و اطلاعات حساس را فراهم کنند.
۶	تجزیه و تحلیل ترافیک شبکه	مهاجم می‌تواند با استفاده از SSRF، ترافیک داخلی شبکه را تجزیه و تحلیل کند و اطلاعات ارزشمندی بدست آورد.
۷	اجبار به حملات دیگر	حملات SSRF می‌توانند مقدمه‌ای برای سایر حملات مانند DDoS یا سرقت هویت باشند.

نمونه مثال از حملات SSRF

۱. حمله به GitHub در سال ۲۰۲۰

در سال ۲۰۲۰، GitHub با یک آسیب‌پذیری SSRF مواجه شد که به مهاجمان این امکان را می‌داد تا به منابع داخلی این پلتفرم دسترسی پیدا کنند. این حمله به دلیل عدم اعمال محدودیت‌های کافی بر درخواست‌های خروجی سرور رخ داد. مهاجمان با بهره‌برداری از این آسیب‌پذیری، توانستند به اطلاعات حساس و داخلی GitHub دسترسی پیدا کنند. این موضوع نشان داد که حتی پلتفرم‌های بزرگ نیز در برابر حملات SSRF آسیب‌پذیر هستند، به ویژه زمانی که کنترل‌های دقیق بر درخواست‌های خارجی و داخلی اعمال نشود.

۲. حمله به AWS Metadata Service

یکی از شناخته‌شده‌ترین و خطرناک‌ترین حملاتSSRF ، حمله به سرویس Metadata در AWS بود. در این حمله، مهاجم توانست با ارسال درخواست‌های SSRF به سرور داخلیAWS ، به سرویس metadata آن دسترسی پیدا کند. این سرویس معمولاً شامل اطلاعاتی نظیر توکن‌های موقتی AWS است که برای دسترسی به منابع و سرویس‌های مختلف AWS استفاده می‌شوند. مهاجم پس از دریافت این توکن‌ها، توانست دسترسی غیرمجاز به منابع ابری را کسب کند. این نوع حمله نشان داد که حفاظت از منابع داخلی سرورهای ابری در برابر حملات SSRF چقدر حیاتی است.

۳. حمله به سرویس‌های Google Cloud

نمونه دیگری از حملات SSRF مربوط به Google Cloud است. در این مورد، مهاجم از آسیب‌پذیری SSRF برای ارسال درخواست‌های غیرمستقیم و دسترسی به سرویس‌های داخلی Google Cloud استفاده کرد. این حمله به مهاجم این امکان را داد تا به داده‌های مهم و حساس کاربران Google Cloud دسترسی پیدا کند. افشای چنین داده‌هایی نه تنها باعث نقض حریم خصوصی کاربران شد، بلکه امنیت زیرساخت‌های ابری Google را نیز تحت تأثیر قرار داد. این حادثه بار دیگر بر اهمیت پیاده‌سازی کنترل‌های امنیتی مناسب برای درخواست‌های خروجی تأکید کرد.

ایرادات رایج برنامه نویسان

در این بخش، به بررسی اشتباهات رایج برنامه‌نویسان در دفاع در برابر حملات «تقلید درخواست سمت سرور (SSRF) می‌پردازیم و نحوه‌ی سوءاستفاده‌ی مهاجمین از این نقص‌ها برای دور زدن دفاع‌ها را توضیح می‌دهیم.

۱. Incomplete Blacklisting

ساده‌ترین راه‌حلی که برنامه‌نویسان برای دفاع در برابر حملات SSRF استفاده می‌کنند، لیست سیاه است. در این روش، بخش میزبان URL ارائه‌شده توسط کاربر با لیست سیاه مطابقت داده می‌شود **[3]** (خط ۵ را در لیست شماره ۳ ببینید) تا اطمینان حاصل شود که به یک سرویس داخلی اشاره نمی‌کند. اگر تطابقی پیدا شود، درخواست توسط مکانیزم حفاظت رد می‌شود. این راه‌حل دفاعی به راحتی با استفاده از روش‌های مختلف کدگذاری (مثلاً نسخه‌ی کدگذاری شده‌ی ده‌دهی ۱۲۷.۰.۰.۱ که ۲۱۳۰۷۰۶۴۳۳ است) و IPv6 دور زده می‌شود، همان‌طور که در چندین گزارش آسیب‌پذیری به‌طور عمومی منتشر شده است. (Hackerone, 2019, report no 386292) (Hackerone, 2017, report no 215105)

```python
def is_allowed(url):
    # url = 'http://127.0.0.1:123/data'
    host = url.split('/')[2].split(':')[0]
    # host = '127.0.0.1'
    prefixes = ['192.168.', '172.', '10.', '127.', '0.', '169.254.']
    for prefix in prefixes:
        if host.startswith(prefix):
            return False
    return True
```

[3] قسمت میزبان با یک لیست سیاه مطابقت دارد

۲. DNS Pinning

برای جلوگیری از محدودیت های لیست سیاه، برخی از توسعه دهندگان به کتابخانه هایی تکیه می کنند تا URL را که یک ادرس IP خصوصی به عنوان مولفه میزبان آن دارد، ممنوع کنند.با این حال، این راه حل در برابر پین کردن DNS مقاوم نیست. همانطور که بسیاری از گزارش های اسیب پذیری نشان می دهند.(Corben Leo, 2018) (Ylujion, 2016) مهاجم می‌تواند از یک نام دامنه استفاده کند که به یک آدرس IP خصوصی اشاره می‌کند تا از این دفاع عبور کند.

مثال :

(http://localtest.me:123/data where localtest.me resolves to 127.0.0.1)

۳. HTTP Redirection

به عنوان پاسخی به DNS pinning ، توسعه‌دهندگان می‌توانند بررسی کنند که آیا بخش میزبان یک URL با انجام یک پرس‌وجوی DNS به آدرس IP خصوصی اشاره می‌کند یا خیر. با این حال، این راه‌حل از دور زدن ریدایرکت HTTP محافظت نمی‌کند.

در این روش دور زدن، مهاجم یک URL وارد می‌کند که به صفحه‌ای در سروری که تحت کنترل اوست اشاره دارد .از آنجا که این سرور دارای آدرس IP عمومی است، بررسی نشان‌داده‌شده در **[4]** موفقیت‌آمیز است و صفحه بازدید می‌شود. این صفحه به نوبه‌ی خود، سرور آسیب‌پذیر را به درخواست از یک سرویس داخلی هدایت می‌کند **[5]**. ما با این روش دور زدن در چندین گزارش آسیب‌پذیری مواجه شدیم. (Corben Leo, 2018) (Eugene Farfel, 2016)

```python
import socket
import ipaddress

def is_allowed(url):
    # url = 'http://127.0.0.1:123/data'
    host = url.split('/')[2].split(':')[0]
    # host = '127.0.0.1'
    ip = socket.gethostbyname(host)
    return not ipaddress.ip_address(ip).is_private
```

[4] بررسی پس از پرس و جو DNS انجام می شود.

```php
<?php
header('Location: http://127.0.0.1:123/data');
?>
```

[5] کد PHP در سرور کنترل شده توسط مهاجم.

٤. زمان بررسی تا زمان استفاده(TOCTOU)

برای جلوگیری از دور زدن ریدایرکت HTTP ، توسعه‌دهندگان ریدایرکت‌ها را در درخواست‌های ارسال‌شده از برنامه به اینترنت غیرفعال می‌کنند. با این حال، ممکن است برنامه هنوز تحت تأثیر آسیب‌پذیریTOCTOU قرار گیرد. به‌طور معمول، قبل از ارسال یک درخواست HTTP به یک URL هدف، دو درخواست DNS انجام می‌شود:

- برای بررسی اینکه آیا بخش میزبان URL به یک آدرس IP خصوصی اشاره می‌کند
- برای تبدیل نام دامنه URL به آدرس IP به‌منظور آغاز اتصال به سرور هدف.

مهاجمان از این شرایط سوءاستفاده کرده وURL ی را به سرور خود اشاره می‌دهند و سرور DNS خود را به‌گونه‌ای تنظیم می‌کنند که دو پاسخ مختلف DNS ارائه دهد:

- یک آدرس IP عمومی برای عبور از اولین بررسی
- یک آدرس IP خصوصی برای ایجاد تقلید درخواست به یک سرویس داخلی. حداقل یک گزارش آسیب‌پذیری در این رابطه مستند شده است.(Alex Chapman, 2019)

راهکار های دفاعی از SSRF

۱. پروکسی معکوس(Reverse Proxy)

پروکسی معکوس سروری است که در مقابل سرورهای وب قرار می‌گیرد و درخواست‌های مشتری (به عنوان مثال مرورگر وب) را به آن سرورهای وب ارسال می‌کند. پراکسی‌های معکوس معمولاً برای کمک به افزایش امنیت، عملکرد و قابلیت اطمینان پیاده‌سازی می‌شوند. (محمد امین دهقانی، ۱۴۰۱) یکی از بهترین روش‌ها برای کنترل و مدیریت درخواست‌های خارجی به سمت سرویس‌های داخلی، استفاده از **پروکسی معکوس** است. با استفاده از یک پروکسی معکوس، شما می‌توانید تمام درخواست‌های ورودی را پیش از رسیدن به سرور اصلی، فیلتر کرده و درخواست‌های مشکوک را مسدود کنید.

مزایا:

- امکان فیلتر کردن URL ها و پروتکل‌های غیرمجاز.
- جلوگیری از دسترسی به منابع داخلی.

- ایجاد لایه‌ی اضافه برای مانیتورینگ و لاگ‌گذاری درخواست‌ها.

مثال: اگر یک API داریم که درخواست‌هایی از سمت کاربران به یک سرور داخلی می‌فرستد، می‌توانیم از Nginx به‌عنوان پروکسی معکوس استفاده کنیم و درخواست‌ها را به دقت بررسی کنیم **[1]**

```
server {
    location /api {
        proxy_pass http://internal-service:8080;
        # جلوگیری از دسترسی به آدرس‌های مشکوک
        if ($host ~* "(localhost|127\.0\.0\.1|169\.254|192\.168|10\.|172\.[1-3][0-9])") {
            return 403;
        }
    }
}
```

[1]

۲. محدود کردن درخواست‌ها به دامنه‌ها وIP های مجاز

یکی از ساده‌ترین و موثرترین راه‌ها، محدود کردن دامنه‌هایی است که برنامه اجازه‌ی ارسال درخواست به آن‌ها را دارد. این موضوع را می‌توان با فهرست سفید (whitelist) کردن دامنه‌ها یا IP های مشخص و معتبر پیاده‌سازی کرد. فقط به درخواست‌هایی اجازه داده شود که به مقصدهای مورد اعتماد ارسال می‌شوند. (سجاد تیموری، ۱۴۰۳)

روش‌ها:

- Whitelisting: فقط به دامنه‌های مشخصی اجازه ارسال درخواست داده شود.
- DNS Resolution سختگیرانه: جلوگیری از استفاده از DNS مخرب یا دست‌کاری شده که به سرورهای مخرب اشاره می‌کنند.

پیاده‌سازی[2]

```
ALLOWED_DOMAINS = ['trusted-domain.com']

def is_allowed_url(url):
    parsed_url = urlparse(url)
    domain = parsed_url.hostname
    return domain in ALLOWED_DOMAINS
```

[2]

۳. اعتبارسنجی و پاکسازی ورودی‌ها

مهم‌ترین اصل در مقابل هر نوع حمله‌ای اعتبارسنجی و پاکسازی (Sanitization) صحیح ورودی‌ها است. برنامه‌ها باید بررسی کنند که ورودی‌های کاربران به درستی فیلتر شوند و از ارسال ورودی‌های مخرب به سرور جلوگیری شود.
(Admir Dizdar, 2022) (J.R. Hernandez,2024) (Hamdaan Ali, 2024)

اقدامات:

- استفاده از Regular Expressions برای محدود کردن ورودی‌ها.
- مسدود کردن درخواست‌ها به localhost، آدرس‌های داخلی و رزرو شده و آدرس‌های IP خصوصی.
- جلوگیری از درخواست‌های پروتکل‌های غیرمجاز.

مثال[3]

```
def validate_url(url):
    if url.startswith('http://') or url.startswith('https://'):
        return True
    return False
```

[3]

۴. پیکربندی صحیح فایروال و کنترل‌های شبکه

فایروال می‌تواند یک لایه حیاتی در دفاع در برابر حملات SSRF باشد. با پیکربندی مناسب فایروال می‌توان درخواست‌های خروجی به منابع غیرمجاز را مسدود کرد. فایروال‌ها در سطح شبکه و حتی در سطح اپلیکیشن می‌توانند درخواست‌های مخرب را مسدود کنند.(owasp)

راهکارها:

- فایروال خروجی: درخواست‌های خروجی سرور را فقط به دامنه‌ها وIP هایی که نیاز به دسترسی دارند، محدود کنید.
- Network Segmentation: سرویس‌های داخلی را در یک VLAN یا شبکه جداگانه قرار دهید تا از دسترسی‌های غیرمجاز جلوگیری شود.

مثال:

- در AWS می‌توان از Security Groups برای کنترل دسترسی به منابع داخلی استفاده کرد.

۵. تست و بررسی امنیتی(Penetration Testing)

تست‌های نفوذپذیری می‌تواند به شناسایی نقاط ضعف SSRF کمک کند. استفاده از ابزارهای امنیتی مانند Burp Suite، OWASP ZAP و اجرای تست‌های مخصوص SSRF در برنامه‌ها، یک بخش ضروری از فرآیند امنیتی است.

راهکارها:

- اجرای تست‌های خودکار برای کشف آسیب‌پذیری‌هایSSRF .
- استفاده از فریم‌ورک‌های تست امنیتی برای شناسایی درخواست‌های غیرمجاز.
- بررسی لاگ‌های سرور برای مشاهده تلاش‌های مشکوک برای دسترسی به منابع داخلی.

۶. محدودیت‌های سطح برنامه

پیاده‌سازی محدودیت‌های برنامه‌ای مانند نرخ محدودیت (Rate Limiting) و تایم‌اوت‌ها می‌تواند به کاهش اثرات حملات SSRF کمک کند. به‌ویژه در برنامه‌هایی که درخواست‌های خروجی زیادی دارند، تنظیم محدودیت نرخ درخواست‌ها می‌تواند از تلاش‌های متعدد مهاجم جلوگیری کند.

اقدامات:

- Rate Limiting: اعمال محدودیت تعداد درخواست‌ها از هر کاربر یاIP .
- Timeouts: تنظیم تایم‌اوت‌های مناسب برای درخواست‌های خروجی تا از حملات زمان‌بری مانند SSRF جلوگیری شود.

مثال:

در بسیاری از فریم‌ورک‌ها می‌توان به سادگی نرخ درخواست‌ها را محدود کرد[4]

```
from flask_limiter import Limiter
limiter = Limiter(app, key_func=get_remote_address)

@app.route("/api")
@limiter.limit("10 per minute")
def api():
    return "Hello World"
```

[4]

۷. استفاده از کانتینرها و Sandbox

جداسازی فرایندهای مختلف در سرور با استفاده از کانتینرها یا sandboxing یک روش مؤثر دیگر برای کاهش اثرات SSRFاست. به این ترتیب اگر یک درخواست SSRF موفق شود، دامنه آسیب‌پذیری به محیط جداگانه‌ای محدود می‌شود و مهاجم به سیستم‌های حساس دسترسی پیدا نمی‌کند.

اقدامات:

- استفاده از کانتینرهایی مانند Docker برای جداسازی سرویس‌های داخلی.
- اجرای برنامه‌ها در Sandbox که دسترسی به منابع سیستم را محدود می‌کند.

۸. استفاده از سرویس‌های DNS ایمن

در بسیاری از حملاتSSRF ، مهاجم از DNS Poisoning یا حملات مربوط به DNS استفاده می‌کند. استفاده از DNS Resolverهای ایمن و تنظیمات صحیح DNS می‌تواند از این حملات جلوگیری کند.(blog.securelayer7.net, 2023)

راهکارها:

- استفاده از DNS Resolver معتبر و غیرقابل تغییر.
- جلوگیری از استفاده ازDNS های محلی و غیرقابل اعتماد.

نتیجه‌گیری:

حملات Server-Side Request Forgery (SSRF) به‌عنوان یکی از تهدیدات جدی در امنیت سایبری، می‌توانند به نفوذ به زیرساخت‌های داخلی و دسترسی به داده‌های حساس منجر شوند. در این مقاله، با تحلیل روش‌های مختلف SSRF و آسیب‌پذیری‌های مرتبط، اهمیت آگاهی از این حملات و راهکارهای مؤثر برای مقابله با آن‌ها مشخص شد.

اجرای بهترین شیوه‌ها از جمله اعتبارسنجی ورودی، استفاده از فایروال‌های هوشمند و مانیتورینگ مستمر ترافیک، به‌طور چشمگیری می‌تواند ریسک این حملات را کاهش دهد. همچنین، ایجاد یک فرهنگ امنیتی و آموزش مداوم تیم‌ها در سازمان‌ها ضروری است.

در نهایت، با توجه به تغییرات سریع در فناوری و تهدیدات جدید، تلاش‌های مستمر برای تقویت امنیت سیستم‌ها، حفاظت از داده‌ها و ایجاد یک محیط دیجیتال امن‌تر، امری حیاتی است.

منابع

شریفی، شهریار، ۱۴۰۰، owasp top 10 چیست؟ https://www.irandnn.ir/mag/owasp-top-10/

مونسی، علی‌رضا، ۱۴۰۰، آسیب پذیری Open Redirect چیست؟https://www.irandnn.ir/mag/open-redirect-vulnerability/

گروه لیان، ۱۴۰۰، آسیب‌پذیری Blind SSRFhttps://liangroup.net/blog/what-is-ssrf/#:~:text=آسیب‌پذیری‌های Blind SSRFیا,SSRF اپلیکیشن به کاربر برگردانده نمی‌شود

سجاد تیموری، ۱۴۰۳ https://liangroup.net/blog/what-is-whitelisting/

Brian Krebs. 2019. Capital One Data Theft Impacts 106M People. https://krebsonsecurity.com/2019/08/what-we-can-learn-from-the-capital-one-hack/

Eugene Farfel. 2016. SSRF in https://imgur.com/vidgif/url.https://hackerone.com/reports/115748

Corben Leo. 2018. Sending Emails from DNSDumpster - Server-Side Request Forgery to Internal SMTP Access. https://hackerone.com/reports/392859

Peter Adkins. 2017. Pivoting from blind SSRF to RCE with HashiCorp Consul. https://www.kernelpicnic.net/2017/05/29/Pivoting-from-blind-SSRF-to-RCE-with-Hashicorp-Consul.html

Cheng-Da Tsai. 2017. How I Chained 4 vulnerabilities on GitHub Enter-prise, From SSRF Execution Chain to RCE! http://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html

Andre Baptista. 2018. SSRF in Exchange leads to ROOT access in all instances. https://hackerone.com/reports/341876

Mallory Mooney, 2024. Detect SSRF attacks in cloud applications and APIs https://www.datadoghq.com/blog/detect-ssrf-attacks/

Busra Demir, 2020. A Pentester's Guide to Server Side Request Forgery (SSRF) https://www.cobalt.io/blog/a-pentesters-guide-to-server-side-request-forgery-ssrf

Ed. 2017. SSRF vulnerability in gitlab.com via project import. https://hackerone.com/reports/215105

Elb. 2019. Bypass of the SSRF protection in Event Subscriptions parameter. https://hackerone.com/reports/386292

ylujion. 2016. Blind SSRF on synthetics.newrelic.com. https://hackerone.com/reports/141304

Alex Chapman. 2019. GitLab::UrlBlocker validation bypass leading to full Server Side Request Forgery. https://hackerone.com/reports/541169

Admir Dizdar, 2022. Server Side Request Forgery (SSRF) Attacks & How to Prevent Them https://brightsec.com/blog/ssrf-server-side-request-forgery/

Hamdaan Ali, 2024. How to Defend Against Server-Side Request Forgery https://www.freecodecamp.org/news/defending-against-ssrf-attacks/

J.R. Hernandez,2024. How to Prevent Server-Side Request Forgery https://www.evolvesecurity.com/blog-posts/how-to-prevent-server-side-request-forgery

Owsap. https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

blog.securelayer7.net, 2023.Server-side Request Forgery (SSRF) via DNS Rebinding Attack https://blog.securelayer7.net/server-side-request-forgery-dns-rebinding-attack/