RPYC

מבוא

שרץ לוקאלית python shellאנחנו רוצים להיות מסוגלים להריץ קוד פייתון על שרת מרוחק מתוך R ( RPyC –על המחשב שלנו. לשם כך עליכם לפתח תשתית שתתמוך בכך emote Python Call.) לאותו מחשב מרוחק ולהריץ עליו פייתון רגיל? כי אנחנו רוצים להיות sshלמה לא פשוט לעשות מסוגלים לקשר בין האובייקטים על המחשב שלנו לאובייקטים על השרת המרוחק (ואולי גם על כמה שרתים כאלה). בתור דוגמה, נניח שאנחנו רוצים להריץ חישוב מורכב על כמה מחשבים, ובסוף לבצע פעולה כלשהי על כל מחשב, לשמור את התוצאות בקבצים, לשלוח את ssh- על כל התוצאות. היינו יכולים להריץ ב הקבצים אלינו, לפרסר את התוצאות ואז לבצע את הפעולה. זה קצת מסורבל. הנוכחי שלנו ללא Python- נוכל לקבל את התוצאות בתור אובייקטים בתהליך ה RPyCבאמצעות המרות מסורבלות. דרישות

באופן כללי, ניתן יהיה לקרוא לפונקציה על השרת ע"י כתיבת הפקודה: c.modules.<MODULE NAME>.<METHOD NAME>(<ARGS>)

) אז int, long, float, bool, str, Noneאם ערך ההחזרה של הפונקציה יהיה טיפוס בסיסי בשפה ( הערך יוחזר כמות שהוא. אם יוחזר אובייקט מטיפוס אחר, השרת ישמור את האובייקט אצלו ויחזיר לאובייקט האמיתי, כלומר הלקוח יקבל ייצוג של האובייקט reference שמכיל רק proxyללקוח המורכב שעל השרת איתו ניתן לעבוד כמו עם האובייקט המקורי בצורה שקופה . באופן פורמלי עליכם לתמוך בתכולות הבאות: יכולת לקרוא מהלקוח לפונקציות (שמקבלות פרמטרים מטיפוסים בסיסיים) על השרת ולקבל • .) לאובייקט מורכב proxy- את התוצאה בחזרה ללקוח (כטיפוס בסיסי או כ

-----

.proxy, אשר גם הוא יכול להיות טיפוס בסיסי או proxy מסוים של attribute אפשרות לקבל • הוא פונקציה או כל אובייקט אחר שניתן proxy- על השרת (במקרה שהproxy אפשרות להריץ • להריץ) ולקבל את התוצאה בחזרה ללקוח. .c.modules.<MODULE NAME> יכולת להשתמש במודולים כלשהם בשרת ע"י • דוגמה בצד השרת מריצים מודול שממתין להתחברות מלקוח. כאשר הלקוח יתחבר, השרת יענה לבקשות שלו ויריץ את הקוד שהלקוח מבקש ממנו להריץ. (במסגרת הזמן, השרת נדרש לתמוך רק בלקוח אחד ובהתחברות אחת שלו.) : של השרת אליו הוא ינסה להתחברport- והIP- תוך ציון הRPyC- התחברות לשרת ה •

c = RemoteControlConnection('127.0.0.1', 6010) print(c.modules.os.getlogin()) # Will print uXXX

builtin הכללי (מודול namespace- שנמצאת ב open קריאה לפונקציה • ) על השרת, והחזרת s : של הקובץ שנפתחreference

f = c.modules.\_\_builtins\_\_.open('~/top\_secret.txt') content = f.read() f.close() # Will print the content of ~/top\_secret.txt # that is located on the server print(content)

צריך להיכתב כך שניתן להשתמש בו בדומה לאובייקט הלוקאלי. proxy- כפי שרואים בדוגמה, ה . הדבר string- ישלח בקשה לשרת לקרוא את הקובץ ויחזיר את התוצאה ללקוח כ f.read()למשל, מתנהג כמצופה מהאובייקט הלוקאלי.f לא משתנה, ו syntax-החשוב הוא שמבחינת הלקוח ה הרחבות

.proxy פונקציות שרצות על השרת יקבלו גם ארגומנטים שהם בעצמם • עם הודעה מתאימה ייזרק exception, אז exception אם הקוד שרץ על השרת יגרור זריקת • שמתקבל בלקוח יכיל את המידע מהשגיאה המקורית exception-אצל הלקוח ולא על השרת. ה של הלקוח, כבכל stacktrace- הרלוונטי (בנוסף, כמובן, ל stacktrace-על השרת, כולל ה exception.) proxy- הרחבת יכולות ה •

--------

o str(p), repr(p), dir(p) o p.x = y, del p.x o p[i], p[i] = y, del p[i], x in p, len(p) o if p: ... o for x in p: ...

השרת צריך להיות מאוד יציב. גם במקרי שימוש שגויים מצד הלקוח עליו להחזיק מעמד ולא • להתפוצץ.

שימו לב שמדובר בהרחבות ולכן אין סיבה שתחשבו עליהן לפני שיהיה בידכם מוצר בסיסי עובד. דגשים

על הקוד להיות מתועד למשעי על מנת להקל על המשתמש. • .)Unit Tests הקוד חייב להיות בדוק הן ברמת הלוגיקה ( • שימו לב להפרדה בין הלוגיקה לקוד של השרת (כך גם יהיה יותר קל לעשות טסטים). • .)webdocs- (יש תיעוד בflask מומלץ לממש את השרת בעזרת • , בו יודפסו verbose על מנת להקל את הדיבוג, מומלץ לאפשר להריץ את השרת והלקוח במצב • לוגים המתארים את מצב המערכת. טיפים

, החבילות • . עשויות להיות שימושיותimportlib- וjson requests כלומר כיצד תוכלו להגיע למימוש בסיסי המסוגל לבצע פעולה – MVP חשבו כיצד תוכלו להשיג • פשוטה, כך שבעתיד ניתן יהיה להרחיב אותו בקלות לדברים יותר מורכבים. בואו נתחיל לממש–יאללה

לא. בואו נתחיל לאפיין.

נעצב ונאפיין את מבנה המערכת והחבילה עצמה– כמובן שלפני שנתחיל לממש • חשבו היטב על כל האספקטים של המוצר בהתאם לדרישות • בהצלחה!