Универзитет у Београду  
Електротехнички факултет



**Неисправна неутрализација  
специјалних елемената коришћених у  
команди – *Command Injection***

Семинарски рад – Развој Безбедног Софтвера

|  |  |
| --- | --- |
| Ментор: | Кандидат: |
| доц. др Жарко Станисављевић | Владимир Сивчев 3172/2019 |

Београд, мај 2020.

Овај документ представља семинарски рад на тему која објашњава неисправну неутрализацију специјалних елемената коришћених у командама, познатија као *Command Injection* рањивост. Рад је рађен као део предиспитне обавезе на предмету Развој Безбедног Софтвера на Електротехничком факултету у Београду.

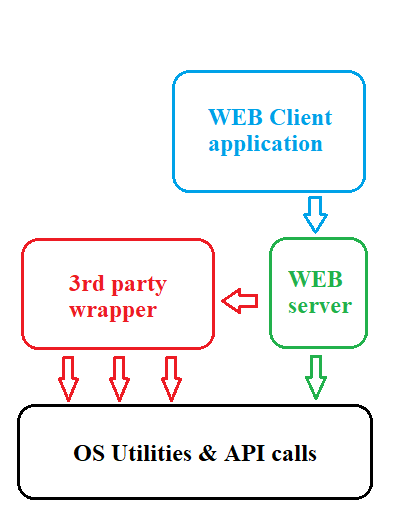
Формално, овај проблем се дешава када апликација директно или индиректно позива услужне програме/команде које нуди оперативни систем кроз одређен *API*. Ово може бити било каква апликација, а кроз овај документ простираће се пример веб апликације која у себи садржи овакву рањивост.

Наиме, како би ова рањивост постојала, неопходно је да постоје 4 одређена услова, а она гласе:

1. Подаци који се уносе у апликацију долазе из извора који је небезбедан (*untrusted*)
2. Ови подаци се користе као део стринга који се извршава као команда од стране апликације
3. Или апликација, или посредник (*wrapper* апликација) извршава команду на такав начин да има привилегије да изврши било какву команду
4. У оквиру једног позива, могуће је конкатенирати (надовезати) одређеним сепаратором (раздвајачем) додатне произвољне команде за извршавање

Последице ове рањивости јесу да омогућавају нападачу да у потпуности добије контролу над системом, експлоатишући могућност да на њему извршава произвољну команду, па тиме на пример да добије приступ читавом фајл систему, где може да извршава било какве деструктивне команде, да шпијунира активности или чак да дохвати документе који су осетљиви и чувају тајне податке.

Првенствено, треба дискутовати о архитектури система где се ова рањивост јавља. Као што је већ поменуто, нека је то нека веб апликација којој се из безбедносних разлога не дају потпуне привилегије над системом (дакле, ова апликација самостално не може да извршава одређене команде над оперативним системом). Међутим, у оквиру апликације постоји потреба да се неке услуге оперативног система користе. Често постоји одређени *API* који омогућава позив одређених *third-party* апликација које су писане у другим језицима (попут *C++, C#, Java, …*), тако да оне представљају *wrapper*-е који имају веће или чак потпуне привилегије на оперативном систему. На слици 1 се може видети архитектура оваквог система.

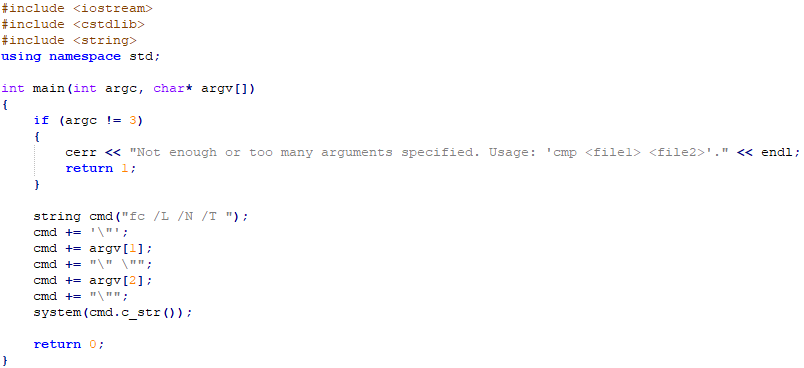


**Слика 1** – Архитектура примера веб апликације

На слици се може видети стандардна клијент/сервер веб апликација, где корисник види такозвани *front-end* апликације који такође и користи, назначена плавом бојом (клијентска апликација). Ова апликација користи ресурсе сервера, односно комуницира са серверском апликацијом. Ради безбедности, као што је већ поменуто, ова апликација нема потпуне привилегије па самим тим не може да извршава одређене команде на оперативном систему, и самим тим је и она безбедна (означена зеленом бојом). То не мора нужно да значи да нема приступ ни једној команди или услузи које оперативни систем нуди, али сви такви позиви су такође безбедни.

Међутим, у апликацији постоји потреба за извршавањем других команди за које веб апликација нема приступ. У том случају, често се прави друга апликација (*third party*) којој клијенти немају приступ, па тиме она сме да има веће привилегије него сама веб сервер апликација, а коју сама серверска апликација користи. Ови позиви су с намером означени црвеном бојом, јер потенцијално представљају рањиве позиве.

Разлог зашто се *third party* апликације зову омотачима (*wrapper*-има) јесте тај што само омотавају позиве услуга оперативног система, те једна таква написана у језику *C++* може изгледати као на слици 2.



**Слика 2** – пример омотавајуће апликације

Циљ ове апликације јесте да искористи команду *FC* (*file compare*) *Windows* оперативног система ради, на пример, поређења фајлова за контролу верзионисања. У овом случају, веб апликација путем одговарајућег *API*-ја позива овај услужни програм, а кроз аргументе командне линије прослеђује путање до два фајла која треба да пореди. Корисник кроз веб апликацију рецимо види само свој део партиције хард диска одвојен за његове фајлове, што се може сматрати безбедним, међутим ова претпоствка је погрешна.

У овом случају, корисник за путању до неког њему видљивог фајла, може да проследи и друге путање, насумично погођене или чак неким другим путем пронађене, или чак нешто још малициозније, а то је управо суштина ове рањивости.

Наиме, податке које сам корисник уноси у клијентској апликацији се само прослеђују кроз серверску до услужне, а затим она то прослеђује оперативном систему за извршавање. Услужна апликација има привилегије да се ови позиви изврше, а овим су већ прве три тачке горе наведених услова испуњене.

Претпоставићемо да корисник види 2 фајла у његовом кореном директоријуму, а то су *„file1.txt“* и *„file2.txt“*. Као други параметар, корисник уместо имена фајла може да унесе: „file2.txt\"&&rd /S /Q C:\\“.

Веома је битно постојање четвртог услова, а то је могућност коришћења раздвајача (енгл. *delimiter, separator*) за команде. У овом случају, то је двоструки амперсанд знак (&&). Пре коришћења наведеног раздвакача, тј. сепаратора, корисник је морао да заврши претходну команду тако што је употребио затварајуће наводнике. Након тога због постојања свих раније наведених услова, могуће је да наведе било какву произвољну команду коју жели да изврши и тиме чак и преузме контролу над читавим системом где се веб серверска апликација налази.

Постоји више начина одбране за ову рањивост. Наиме, првенствено треба неутрализовати све специјалне карактере које долазе од стране корисника како ово не би било могуће. Такође, треба радити такозвани *white-listing* могућих уноса, нарочито ако се одређени подаци очекују по одређеном формату. У другим случајевима, постоји шанса да је корисников унос малициозан и такав унос треба санкционисати.