Zack Johnson

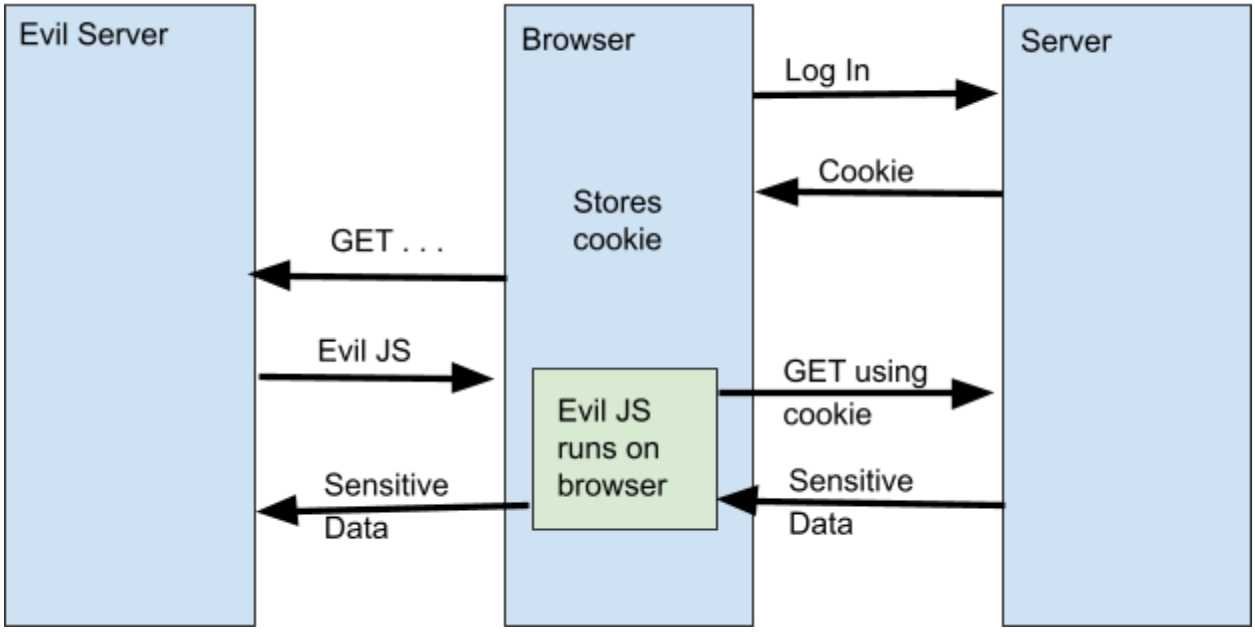
CS 338 Final

Pen Testing

a) I will be looking at the Postgres 8.3.0 - 8.3.7 vulnerability
b) Instructions on running the exploit, both payloads use nearly the same setup, to access the non-default (meterpreter/bind_tcp) simply “set” it in the Choose payload setp.
   - Start the metasploit framework
   - Use nmap to find the target (metasploitable) and scan its ports, we will be targeting postgreSQL on port 5432.
   - Begin preparing the exploit using the command: “use exploit/linux/postgres/posgres_payload”.
   - Set the target of the attack with “show targets” and “set TARGET <target>”
   - Choose the payload. The default is the reverse_tcp payload but there are several other options that can also be used with “show payloads” and “set PAYLOAD <id>”
   - Double check the options are correct: use “show options” and “set <option> <value>” to adjust options as needed, often the LHOST and RHOST need setting.
   - Initiate the exploit with the command: “exploit”
c) There is an issue with the default linux configuration of postgreSQL 8.3.0 - 8.3.7 that allows the service account (the one available to the outside world), to freely write to the /tmp directory. This is a problem because it is also possible to source user defined libraries from the /tmp directory. This allows a remote connection to execute arbitrary code.
   (source: https://www.rapid7.com/db/modules/exploit/linux/postgres/postgres_payload/, plus looking up a couple terms)
d) I used two similar payloads to get a shell program running on metasploitable. They were meterpreter/bind_tcp and meterpreter/reverse_tcp. They both work by establishing a tcp connection and using it to send the payload across the network. The main difference is the direction of the original traffic. In the bind_tcp case, kali (the attacker) initiates the tcp connection with metasploitable, (the victim). In the reverse_tcp case Kali sends a command that causes metasploitable to initiate a connection back to Kali to receive the files. I think the main reason to use the latter is that if one is making multiple attacks it looks the same as requests to a web server rather than a suspicious polling of many machines by one computer.
e) I found out that the particular payloads I am using install a shell called meterpreter (rather than a unix shell) I then found that there is a command called download that does exactly what I need. I called download targeting /etc/passwd and it automatically transferred into the Kali folder on Kali. This works with either of the payloads I used.
f) I was able to discover connections to Kali’s IP address with “netstat -antp” (they are noticeable because they have very large port numbers around 42,000)

Same-Origin Policy

a) Without the same origin policy, it is possible for multiple websites to access the same browser data. This allows a malicious website to potentially get access that it shouldn't into other web interactions that the user has.



In this example the user is logged into some important and legitimate website such as a bank. Since the bank and the user don't want to have to re-authorize the user every time they load a new page, the server sends the user a session cookie so they can remain logged in. Some time later, the user leaves the bank site, but does not log out (so the cookie is still valid and still saved in the user's browser). Then they visit a web page that turns out to be malicious. In loading the page, the browser receives some malicious javascript that tells it to access something from the original bank server using the cookie it already has. This allows the malicious server to access the bank server with all privileges the user has, allowing them to do something bad (steal data, make a fraudulent transaction, etc.).

b) The Same Origin Policy can prevent this sort of attack. If the browser in the above example was properly implementing the same origin policy, it would notice that the evil server and the bank server have different origins and would prevent the javascript from the evil server from requesting data from a different origin (the bank server). This means that the "GET with cookie" step would not be allowed by the browser because the script making the request has a different origin than the target of the request.

c) The same origin policy uses scheme (http vs https vs other protocol), host (URL or IP address) and port to determine the origin of something. In this case the port numbers would be different (80 vs 8888) so they would count as separate origins so the script from port 80 would be blocked from accessing data at port 8888.
In order to allow this architecture, a developer would need to include a system, on the database server, to maintain a list of allowed origins. Then it would need to read the

Origin header, check it against the list and reply with the correct Access-Control-Allow-Origin header (either the site url or an error).

Security Mindset

a) Preventing shoplifting at a retail store.
b) An attack looks like someone slipping an item into their bag, or potentially just walking out with it claiming to have checked out already.
c) Some mitigations include rearranging the layout of the store so that it is not very natural to leave without going through the checkout process.

Another mitigation that I think is surprising but was the policy at a store I worked at was to not do anything in particular about a suspected shoplifting, instead just engage with the customer in an upfront way like you would any other. The theory is that someone will not shoplift if they feel like they have been seen (even if they are not suspected).