HW3
CSCI1660
SeungJi Lee
sl136

Problem 1:

1. The modification explained in the problem will not foil the attacks against WEP Cloaking. The technique 2 explained in the slides will successfully filter out the chaffs through sequence number and IV filtering. Moveover, the technique 3 explained in the slides will also identify chaffs since the chaffs are encrypted with wrong key and thus AP will not replay the chaffs.

2. This method should not help very much since this is essentially the same as the method presented in 1; the chaffs here are generated by the client instead of the chaffer. Moreover, one major disadvantage of this approach is that the client has to perform encryption of a packet k times instead of one, slowing down the performance of the connection.

3. a) This scheme will not protect against the active frame replays since the attacker can easily change the header and thus the identifier of the packet. The unique identifier can be easily determined simply by looking through the identifier used in the packets captured.

   b) This scheme should protect against the active frame replay since the attacker cannot change the encrypted payload of the packet. Thus, the same unique identifier is used when performing active frame replay. Thus, the AP will catch the replayed packets.

Problem 2:

One way to bypass the stackgaurd is to change the return address without changing the canary. One way to do this is to exploit the function calls to change the memory of a specific address. For example, if a stack-overflow vulnerable program(which now use stackgaurd) uses functions that changes the memory values such as strcpy(p, val, size), we can modify the p, pointer to point to the return address. This way, the attacker can modify the return address without changing the canary. However, this method is much more limited since this method requires p to be located after the buffer, and the program to use memory modifying function calls. We also need the p should not be reinitialized in between the memory modifying function call and the overflow exploit to change p. Although this method is more complicated and limited than regular stack-overflow attacks, this method is possible to use and bypass the stackgaurd.

Another way of bypassing the stackgaurd is to brute-force to get the canary value. Since the canary value is fixed at the compile time, the attacker can try to guess the canary value of the program by brute-force attack. Then, the attacker can simply overwrite the canary with the same value and perform the regular buffer overflow attack. Depending on the size of the canary, this may slow down the attack enormously. However, given enough time and resourses, it is plausible to exploit the program.

Problem 3:

1. One major drawback to using the explained approach is that it is expensive to keep track of the function calls and search the graph. Also, an edge needs to be searched every time the program calls a function. Since the program that is large enough uses enormous number of function calls, the running time of the program will increase enormously. For example, if we calls strcpy function in a for loop, then at each iteration, an edge in the call graph must be found.

2. This approach does not protect against all possible manipulations of the return address. Indeed, the attacker can simply change the return address to one of the address that is

mapped in the call graph.
In order to circumvent the protection, the attacker needs to either modify the call graph to allow the current function to call the injected code, or inject his code into one of the return addresses that is validated by the call graph.

3. This approach does not protect against the heap buffer overflow attack. The heap buffer overflow attack modifies the data that is stored in the heap. Namely, heap buffer overflow can change the program itself or the data that program stores on the heap to achieve its purpose. Thus, the call graph cannot protect againt this type of buffer overflow attack.

Problem 4:

1. One way that rootkit can hide itself is to modify the system call that lists the content of the directory; the rootkit can change the system call to list all the contents except itself. Another way to for the rootkit to hide itself is to modify the read system call. When reading each file in the directory, modified read() system call will only return the safe content of the malware or simply will return empty string as the content of the malware.

2. The attacker may modify rootkit to rewrite the hardrive device driver to hide the contents of the malware; it may simply output bitstream of other legitimate data when the contents of the malware is requested. One problem with this method is that OS uses the hardrive device driver to interact with the hardrive. Thus, rewriting the hardrive might cause the entire OS to fail and reveal the presence of the rootkit.

3. For the rootkit to be able to run/access the "important programs and files," that are "golden hashed," the rootkit must be included in the programs and files before the cryptographic hashes for these program is written to the database.
One way to achieve this is add any changes that rootkit must do as the program patches. When a program is patched, then the new cryptographis hash must be saved onto the database. Since the rootkit changes are already included, the database will have the hash of the corrupted program.
Another time when the rootkit can sneak in is when the OS resets. When the OS resets, the cryptographic hashes for the important files and programs must be rehashed and saved onto the database. The rootkit, when reseting the OS can interfere and modify OS at the reseting time. Then, the cryptographic hashes of the modified files and programs will be saved onto the database.

Problem 5:

1. Noxes will work for most common XSS attacks, which attempts to extract informations from the victim. The reason is that noxes controls the flow of web packages. For the attacker to gain information about the client, the XSS code must send the information to the attacker. However, Noxes control the flow, effectively blocking the package sent to the untrusted destination.

2. When the XSS code do not send any information to the attacker, but rather do something malicious on the machine itself, then Noxes will fail. For example, Noxes is vulnerable to denial-of-service XSS attacks. Also, the XSS attacks that exploits the server-side vulnerabilities cause failures in Noxes.

3. NoScript differs from Noxes in that NoScript controls the execution of JavaScripts while Noxes controls the flow of information. NoScript allow execution(usage) of JavaScripts only on the trusted websites (those on the whitelest), whereas Noxes will execute JavaScript, but will block illegitimate sending of information.

4. One of weaknesses with NoScript is that it keeps trust the sites that it trust. So an attacker can create a website that is benign at first to gain trust from the user, and

then modify the website to perform an attack.
Another weakness of NoScript is that it cannot give a solid protection against the persistent XSS attacks. The attacker can simply inject th malicious code into the trusted server, which will be executed by the client.

5. In stored XSS attacks, the malicious JavaScript code is permanently stored on the target server. On the other hand, the reflected XSS attacks, the malicious JavaScript code is resent("reflected") to the client as an error message or a search result.