They're Coming For Your Tools: Exploiting Design Flaws for Active Intrusion Prevention

By John Ventura

The response capabilities for Intrusion Prevention Systems (IPSes) have room to improve. Although vendors offer some countermeasures, like logging and the termination of TCP connections, this is an area that has been underexplored. Specifically, countermeasures that target popular attack tools and methodologies are possible, and this whitepaper will discuss specifics around two such strategies.

People break into computers for different reasons. Some people are paid to do it, some people are hobbyists, and some people violate security for reasons that defy our understanding or imagination. However, all of these groups face similar challenges, including command and control of compromised computers as well as problems related to spreading one's control throughout someone else's environment and avoiding detection. Developers have solved many of these problems by releasing popular software packages. Although some intruders may write all of their own exploits and assistance tools, the "artisanal" approach is inefficient. Intruders might not want to spend several days writing a new packet sniffer when several are already available for download.

Because these popular tools and methodologies are software based, they follow predictable patterns and are often easily detectable. During the research for The Seek Locate Destroy Toolkit (The "Salad" Project), we have created proof-of-concept tools to demonstrate how administrators can enhance intrusion prevention by exploiting the flaws exhibited by popular software packages. The counter-attack tools discussed here are currently publicly available and open source, and administrators can use them immediately for proactive responses. These tools use relatively little CPU time or RAM, so their deployment should not be expensive in terms of money or other resources.

The countermeasures presented here fall into two categories:

- Man-in-the-middle counter-attacks against popular command and control software
- Countermeasures against brute-force password recovery

Man-in-the-middle Countermeasures

Mass-market command and control systems are difficult to implement. Any communication system that requires two or more computers to communicate in a secure manner that resists eavesdropping and data tampering will be a surprisingly challenging project. TLS/SSL (and OpenSSL specifically) gives us a real world example of just how hard this can be. These efforts have been ongoing for multiple decades and some of the leading minds in cryptography and Internet security have been involved. Even with the advantages of time, experience, and general knowledge significant problems still emerge from time to time. During the past decade, researchers have uncovered DROWN, BREACH, POODLE, BEAST, CRIME,

FREAK, Logjam, and Heartbleed attacks (and these are just the ones with catchy names). Anyone who wishes to create a new communication protocol should not expect to avoid such issues given this history, and popular command and control software packages are no exception.

Countermeasures Targeting Metasploit Framework's Meterpereter

The Metasploit Framework is a powerful exploitation/post-exploitation software package. It would be difficult to overstate its popularity with the professional penetration tester community, due to its broad coverage for the specific set of problems faced by anyone who wishes to penetrate other people's security. This framework includes "Meterpreter", which is a command and control software package that allows attackers, penetration testers, and others the ability to control compromised computers. It works on a variety of platforms, so users can control servers running Linux, Microsoft Windows, BSD, MacOS, as well as other platforms.

Its contributors have designed Meterpreter in such a way as to allow some flexibility in its deployment. Users can create stand-alone executables to run on compromised hosts, deploy it within pre-packaged exploits, or even integrate it into newly discovered attacks. Under some circumstances, users can deploy it relatively securely using TLS and certificate pinning. However, this approach is not necessarily intuitive and requires its users to go through multiple steps that involve third party software. Additionally, this secure approach may not be compatible with an exploit driven deployment and may present other problems when used in a restricted environment.

The man-in-the-middle attack engineered for Meterpreter assumes that TLS/SSL is not being utilized, which is often the case for online tutorials and may even be considered the "default" position for many attackers, as it is much more intuitive and easier to use.

Staging of Meterpreter involves a TCP connection between a user's Metasploit console software and the compromised host. The penetration tester or attacker will run a staging binary or initiate staging via "shellcode" included within an exploit. Depending on whether or not this process involves a "reverse" connection, the Metasploit console or the compromised machine may initiate the TCP connection. Either way, the communication protocol remains the same.

After the TCP connection is established, the Metasploit console sends a 32bit "payload length" message within a single TCP segment to the compromised host. Following this transmission, there is a slight pause whose length varies. This pause gives the compromised host time to allocate memory equal to or greater than the "payload length." The Metasploit console then sends its payload, using multiple TCP segments. This payload includes the bytecode instructions for the Meterpreter or other software, including a VNC server, depending on how the penetration tester or attacker chooses to perform staging. Once the compromised host receives this payload, it copies it into the memory buffer that it allocated and switches execution to that portion of memory. Meterpreter uses the previously established TCP connection to communicate with the console for command and control.

The Salad Project's man-in-the-middle counter attack for this software utilizes a custom written packet sniffer that examines each TCP segment looking for known payload length segments. The payload length for any given instance of Meterpreter will be static. For example, the TCP segment that sent as the payload length for 64bit Microsoft Windows Meterpreter

sessions contains the four bytes 0x2f, 0x26, 0x12, and 0x00 followed by two null bytes (0x00 and 0x00).

When the sniffer observes these patterns, it responds by using packet spoofing to inject data into the TCP transmission. It copies the source and destination addresses for the payload length packet and continues the sequence numbers that it observes. Due to the slight pause between the transmission of the payload length and the actual payload, the sniffer's data will almost always get to the compromised host first. The host relies on TCP sequence numbers for stream reassembly and will take the sniffer's data as valid. When the actual payload arrives, the compromised host will treat that data as a retransmission and ignore it. Consequently, when the compromised host attempts to copy the payload data into its memory buffer, it copies the sniffer's data instead, and these instructions get executed instead of the malicious payload. The sniffer configuration included with the proof-of-concept code contains "shellcode" or "position independent code" that triggers a "reverse shell" that provides access to a listener of the user's choosing. When successful, the person running the sniffer will have a remote command shell for the compromised host and the attacker will have a broken TCP connection instead of a working command and control channel. In this way, the proof of concept software redirects control over the compromised host to a host it's user's choosing and away from the Meterpreter user.

Powershell Empire

Powershell Empire is also a popular command and control channel that allows people who break into computers a great deal of remote administrative functionality. It is remarkable for its relative ease of use, elaborate set of features, scalability, and overall design. However, the version 1.6 and 2.0 branches of this software are both vulnerable to man-in-the-middle attacks.

Both branches of Powershell Empire make attempts at obfuscating or encrypting data in transit during the staging process. There are subsequent cryptographic features deployed after the first stage, including the generation of asymmetric key pairs. However, the initial staging process is vulnerable, and proof-of-concept code associated with this project demonstrates how man-in-the-middle attacks are possible.

Users of Powershell Empire must find a way to execute a long command line from their compromised machines for the initial phase of staging. This command line starts by invoking Powershell with the command "powershell.exe". Its arguments also include a lengthy base64 encoded Powershell script that instructs the compromised machine to communicate with the Powershell Empire staging server via HTTP. The staging server responds to this initial request with an encrypted or obfuscated Powershell (or possibly Python) script that handles the more elaborate portions of the staging process. The security of this script data is dependent on a symmetric key shared by the staging server and the compromised machine. Any data received during this initial request will be considered valid if it either decrypts or de-obfuscates to working code. The proof-of-concept counter attack for Powershell Empire exploits the assumption that decryption or deobfuscation includes validation.

Just like with Meterpreter, this proof-of-concept code is also a packet sniffer. It first derives a working payload that performs actions desirable for its user and then packages it in a way that will decrypt or deobfuscate with the symmetric key used by the Powershell Empire user. Once it constructs this payload, the sniffer uses packet spoofing to overwrite HTTP response data coming from the staging server. The compromised host will receive this modified data in place of the actual script that the Powershell Empire user hopes the staging server will

return, and Powershell Empire's staging script will subsequently execute these instructions after decryption or deobfuscation.

This sniffer uses two different methodologies for this repackaging, because versions 1.6 and 2.0 use different encoding methods. Version 1.6 obfuscates its data by using XOR to encode it with the symmetric key. Version 2.0 is more advanced, and uses the symmetric key for encryption and decryption with Rivest Cipher 4 (RC4). Neither of these methods are secure, and the sniffer software handles both of them.

The shared key for Powershell Empire version 1.6 is an ASCII representation of a hexadecimal number. There are only 16 possible values for each offset within this key (0 through 9 and 'a' through 'f'), and the data being obfuscated is known to be a Powershell Script generated by the application. A review of this output showed that, over time, the following characters in this string were the most common (from most common to least common):

"pDO\\m\"'-[]yP()lcdRnCasN=Io;AirTt SE.e$

Given the small set of possible values for each offset within the key and the constraints on the plaintext data, eavesdroppers can recover this key by performing frequency analysis based on a single obfuscated payload. The sniffer grabs a payload from the staging server and then uses XOR to guess the value for each offset within the key. The "winning" or correct value is always the one that produces the common characters most often. This technique is incorporated in the software and is remarkably effective, even if it may seem counter-intuitive.

The packaging for Powershell Empire 2.0 is more complicated. This version uses RC4 for encryption, and key recovery was not achieved during this research. However, RC4 is insecure if the plaintext is known, and the plaintext for this portion of staging is either a Powershell or Python script, both of which are mostly static.

RC4 works by using a function that performs like a Pseudo-Random Number Generator (PRNG). The seed for the PRNG is the symmetric key used on both sides of the transmission, although Powershell Empire pre-pends the first four bytes of the transmission as a security measure. The plaintext is encrypted by taking the output of the PRNG and using XOR on each byte to produce the cipher text. Because the PRNG produces the same set of numbers for both sides of communication, this algorithm produces a nearly endless key stream with will be identical for both ends. Attackers who wish to perform man-in-the-middle attacks need to use XOR with the known plaintext and the cipher text to recover the keystream.

The proof-of-concept software downloads a python staging script from the server and uses XOR to recover the keystream in this manner. It then takes the keystream and uses XOR to repackage its own payload. All subsequent HTTP requests to the staging server are then overwritten with this data. This counter attack disrupts staging after the initial powershell.exe command and causes compromised machines to run the code chosen by the person controlling the sniffer.

This counter attack requires the use of a valid "session cookie." Administrators can intercept this data by packet sniffing, reviewing proxy log data, or by interrogating compromised computers. This session cookie contains instructions to the staging server that allow the user to choose Python or Powershell. Because the Python code returned by Powershell Empire is

much more static, the sniffer uses the same RC4 counter-attack to repackage the "session cookie" to insure that it downloads the right payload for repackaging.

Other Command and Control Software Packages

Although this project focused on Meterpreter and Powershell Empire, other popular command and control software packages were also considered. However, we did not produce a proof-of-concept tool to demonstrate insecurities in DoublePulsar. This plugin is part of Fuzzbunch, which was revealed to the world by "The Shadowbrokers". The omission of DoublePulsar was partially motivated by the excellent analysis provided by Countercept (https://github.com/countercept/doublepulsar-detection-script/) as well as the tool's glaring insecurity. Its users' payloads are "encrypted" with a XOR key that it also reveals in plaintext during the same transmission. Additionally, it is also easily detectable due to the fact that it uses static values that occur at known offsets that do not otherwise occur in its host protocols (SMB and MSRDP). Countercept produced snort signatures, which demonstrate the ease of detection.

If attackers see this activity on a network, they can use the leaked software to give themselves access, as DoublePulsar installs a back door on the compromised host without providing adequate authentication. Penetration testers and malicious attackers alike would therefore be well advised to avoid using this software in real world activities.

Cobalt Strike is also a popular software package. Its designers have taken care to make man-in-the-middle attacks more difficult. For example, it includes message authentication codes in its protocol and it has several "stageless" options that also complicate these attacks. Its developers expressed these concerns explicitly in a blog post from 2016 (https://blog.cobaltstrike.com/2016/06/22/talk-to-your-children-about-payload-staging/), which is worth a read. Anyone wishing to target Cobalt Strike would most likely need to impersonate the staging server/console with equivalent software before its actual use or hope that its user chooses an insecure staging/stageless option from Cobalt Strike's relatively vast set of features.

Password Recovery Countermeasures

In addition to tailored man-in-the-middle attacks, administrators can protect their networks in other ways, including countermeasures against attacks that rely on the collection of password hashes. Specifically, this project targeted NBNS/LLMNR (NetBIOS Name Services/Link Local Multicast Name Resolution) attacks as well as common methodologies for stealing pre-shared keys from WPA2 wireless networks.

Password "cracking" or offline attempts at recovering password data through brute-force guessing can be computationally intensive. The software developed to demonstrate strategic countermeasures against these techniques degrades the attacker's capacity by flooding him or her with bogus password hashes. Additionally, administrators have the option of using "canary accounts" or more involved methodologies for intrusion detection.

NBNS/LLMNR Spoofing Countermeasures

NetBIOS Name Services (NBNS) and Link-Local Multicast Name Resolution (LLMNR) spoofing is remarkably effective and widely popular. Laurent Gaffie's "Responder" software has contributed to this technique's widespread adoption with its ease of use and support for several

protocols. Many professional penetration testers deploy this attack in the early stages of an assessment and report a high degree of success.

NBNS/LLMNR spoofing relies on the insecurity of these protocols as well as their ubiquity. Misconfigured Microsoft Windows desktops and servers use these protocols for backup name resolution when DNS fails. These protocols require the hosts to send broadcast UDP packets to their network peers querying them as to whether or not they are the host that corresponds to that given name. Hosts that recognize the names in the queries as their own respond with their own IP addresses, and the misconfigured Microsoft Windows desktops or servers then begin communicating with the responding hosts over a given protocol, like HTTP or SMB.

This attack exploits this default behavior. "Responder" and similar software will respond to all of these queries by claiming to be the host in question. After the victim host receives this response, it initiates communication, and the attacker host attempts to collect a plaintext password or use NTLM authentication to gather a password hash for offline analysis.

The proof-of-concept software developed for this project takes advantage of these responses. Just like the misconfigured Microsoft Windows hosts, it sends NBNS and LLMNR queries to its peers on the LAN. These queries are for hosts with fabricated names that it generates consisting of between 16 and 20 random letters and numbers.

Any host that responds to these queries is assumed to be hostile in this scenario. The software will then begin to log the detection. Administrators can choose to have the software send set of presumably false credentials from a list. Attackers who collect these hashes will, hopefully, waste their time and computing power with offline cryptanalysis attacks, or they will find themselves frustrated by the futility of their newly gained credentials.

Administrators can choose a resource exhaustion attack where the hostile server is flooded with hashes, if they wish to be more aggressive. With this feature in use, attackers who fail to continuously monitor their consoles will return to find their disk drives full and their efforts similarly frustrated.

WPA2 PSK Sniffing

Pre-shared keys (PSKs) are often the primary authentication mechanism for WPA2 wireless networks. Attackers who acquire these keys will be able to put their own devices onto these networks and they will typically have unfiltered network access to their peers on these LANs. On a properly configured network, the PSK is never shared in plaintext. Instead, hosts attempting to authenticate into these networks must engage in a four-way "handshake" between themselves and the Access Points (APs) that regulate access. Each party generates a nonce and which both parties use together to generate a Message Identification Code (MIC), which is a hashed form of the pre-shared key.

Attackers can sniff these MICs along with the nonces and use this data for an offline brute-force password recovery attack. Automated password cracking software takes each password from a list of passwords along with the two intercepted nonces and creates a series of hashes using the same algorithm used to create the intercepted MIC. If any of the newly generated MICs matches the MIC sniffed from the victim's WPA2 network, then the attacker knows that the password used to generate that MIC matches the PSK used for that network.

In order to perform this attack, the attacker's software must sort through a stream of packets, many of which are not from the target's network. The software looks for these handshakes and then alerts the attacker once it has collected them. Typically, it uses the BSSID (analogous to an Ethernet MAC address) for the AP to sort out which handshakes are associated with any given network.

The proof-of-concept tool developed to mitigate this threat collects the BSSID from an access point by analyzing "probe responses" and uses it to "spoof" a handshake. It mimics portions of a legitimate handshake using the BSSIDs of the legitimate AP and a randomly generated BSSID assigned to an imaginary device attempting to authenticate. It generates a MIC and includes it in these spoofed handshakes based on a pre-shared key that is presumably different than the one already in use. The malicious software sniffs these handshakes and associates them with the victim network, because the BSSID matches the legitimate one. These spoofed handshakes are close enough to legitimate ones that many sniffers are fooled.

Once the attacker has this spoofed handshake, he or she can subject it to offline brute force analysis. Presumably, the spoofed handshake may be broadcast along with real handshakes. In order to acquire the real PSK for the target network, the attacker will have to either determine a way to filter out fake handshakes or crack all of them. Such an endeavor is likely to either complicate these attacks or consume enough of the attacker's resources to make these efforts less likely to be successful.

Conclusion

The proof-of-concept tools for "The Salad Project" demonstrate each of these counter-attacks. Anyone wishing to explore these techniques can download this software and integrate them into their own intrusion protection infrastructure. The contributors to this project wish to see network security improved more generally, and hope that active responses can promote security across multiple fronts.