Network Attack Collaboration

Sharing the Shell

RAPHAEL MUDGE

Raphael Mudge is a Washington, DC, based code hacker. His current work is the Armitage GUI for Metasploit. His past projects include the After the Deadline proofreading software service and the Sleep scripting language. Raphael has worked as a security researcher, software engineer, penetration tester, and system administrator. Raphael holds a commission in the Air National Guard.
rsmudge@gmail.com

Working in a network attack team today is cumbersome. Penetration-testing tools such as Core Impact, Immunity Canvas, and Metasploit assume a single user. Team members have limited means to share access to compromised hosts, and good intentions are quickly mired in a disorganized free-for-all.

To address this problem I developed Armitage, a technology that allows a network attack team to communicate in real time, share data, and seamlessly share access to hosts compromised by the Metasploit exploitation framework. This article discusses the needs for network attack collaboration, the inner workings of the solutions in Armitage, and the lessons learned using this technology with the 2011 Northeast and Mid-Atlantic Collegiate Cyber Defense Competition red teams.

Metasploit

Metasploit [1] is a popular exploit development framework. H.D. Moore started the project in 2003. Metasploit makes it easy for security researchers to develop exploits for software flaws and use them in the context of a very feature-rich tool.

Metasploit features include multiple user interfaces, support for multiple platforms, and powerful post-exploitation tools. Metasploit users may choose which payload to execute when an exploit is successful. Metasploit payloads range from simple command shells to powerful post-exploitation tools like Meterpreter.

Through Meterpreter, users may transfer files, execute and interact with processes, dump the Windows SAM database, navigate the file system, and manage processes on the compromised host. When delivered with an exploit, Meterpreter is capable of running completely from RAM without ever touching disk. Metasploit provides a command-line interface for interacting with Meterpreter.

Meterpreter is not a Metasploit-only concept. Other exploitation tools, such as Immunity Canvas and Core Impact, have built-in post-exploitation agents too. The *Shellcoder's Handbook* [2] explains this practice. Exploitation tools that simply provide a shell lose the ability to transfer files, give up access to the Win32 API, and in some cases lose access to any privileged tokens the current thread might hold. Post-exploitation agents, such as Meterpreter, implement a protocol that allows users to carry out these and other actions. The session sharing ideas presented in this article should apply to these other post-exploitation agents.

Armitage

Armitage [3] is the graphical user interface I wrote to support teams using Metasploit. Armitage organizes Metasploit's features around the network attack process. There are features for host discovery, exploitation, post-exploitation, and maneuver.

Armitage exposes Metasploit's host management features. It's possible to import hosts and launch scans to populate Metasploit's database with target and service information. Armitage's user interface also displays the target database in a table or graph format.

Armitage's *find attacks* feature recommends remote exploits using known host and service information. Users may also launch browser exploits, generate malicious files, and create executable files to call back to Metasploit from Armitage.

Armitage provides several post-exploitation tools for Windows targets built on the capabilities of Metasploit's Meterpreter agent. Menus are available to escalate privileges, dump password hashes to a local credentials database, browse the file system, and open command shells. For Linux and Mac OS X targets, Armitage lets users interact with a command shell.

Finally, Armitage aids the process of setting up pivots, a Meterpreter capability that lets users exploit a compromised host to attack and scan other hosts. Armitage also exposes Metasploit's SOCKS proxy module, which turns Metasploit into a proxy server that routes outgoing connections through existing pivots. With these tools, users may further explore and move through a target network.

Figure 1 shows the Armitage user interface. Armitage's targets panel visualizes known hosts, active sessions, and existing pivots. A session is an active Meterpreter agent or a shell on a compromised host. The module browser in the top left lets users search for and launch Metasploit modules. These GUI components are always visible. Armitage uses tabs to organize open consoles, command shells, and browsers.

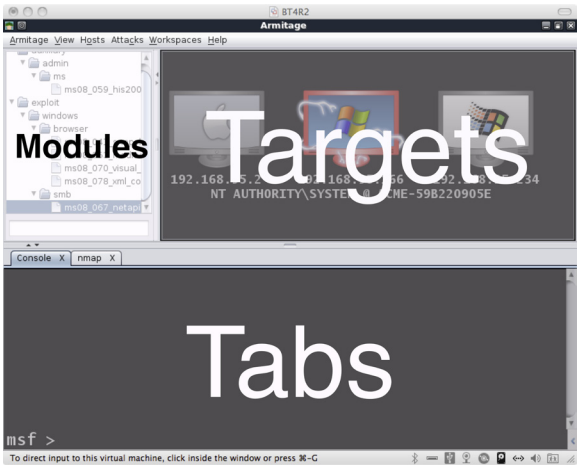

Figure 1: Armitage user interface

Teaming Architecture

Through Armitage, it's possible to manage and share a remote Metasploit instance via its RPC server. Metasploit's RPC server allows clients to send commands to Metasploit using an XML-based protocol.

Armitage extends Metasploit's RPC interface to provide real-time communication, data sharing, and session sharing through a deconfliction server. The deconfliction server offers Armitage clients additional functionality, helps with scalability, and manages multiple clients accessing Meterpreter and shell sessions.

The deconfliction server is part of Armitage. It's started using the `–server` command line option. The deconfliction server connects to Metasploit like any other client. When it connects, it sets a global variable in Metasploit to instruct Armitage clients to connect to it. Adding features through a separate server protects the teaming features from internal changes to the Metasploit framework.

Some Metasploit features require the client to modify or read a local file. Metasploit's RPC server does not offer an API for reading and writing local files. For these cases, the deconfliction server offers the missing functionality. The extra functions in the deconfliction server allow Armitage to offer real-time communication to team members, lock and unlock shell sessions, and transparently download screenshots taken through Meterpreter.

The deconfliction server also helps Armitage with team scalability. Armitage clients used to poll Metasploit to get the list of current hosts, sessions, and known services. Constant polling from multiple clients caused Metasploit to stop responding with more than five active clients. The deconfliction server temporarily caches the output of some commands to reduce load on the Metasploit RPC server. Armitage is now able to support a team of ten or more clients.

The deconfliction server's primary purpose is to act as a proxy between Armitage clients and Metasploit for session interaction. All session read and write commands go through the deconfliction server. The deconfliction server manages these operations using Meterpreter multiplexing to provide transparent session sharing for the user.

Figure 2 shows the relationship between the Armitage clients, the Metasploit RPC server, and the deconfliction server. The dashed lines show the communication path for Meterpreter commands. The solid lines show the path for most Metasploit commands.

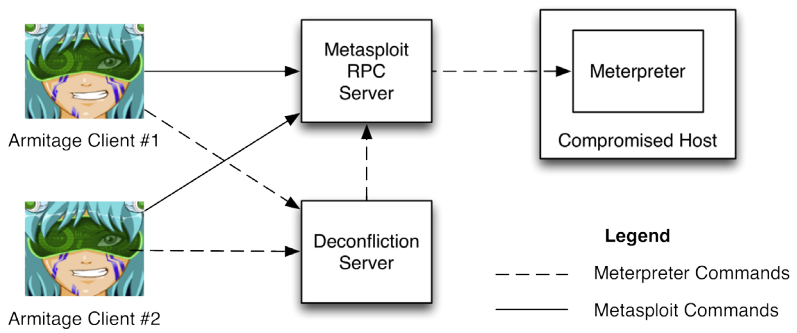

Figure 2: Teaming architecture

Real-Time Communication

In a collaboration situation, it helps to have real-time communication. Red teams often rely on Internet relay chat, instant messaging, a shared wiki, or even yelling across a room. Armitage's deconfliction server offers Armitage clients read and write access to a shared event log.

Armitage presents this shared event log as a new tab. Users may search it and type messages into it as if they're using a chat room. Armitage prefixes a user-provided nickname to each message. The Armitage client also reports events to this shared log. These events include scans, exploits, login attempts, changes to the pivot configuration, and clearing the database.

In practice, I haven't seen the Armitage event log overtake other real-time communication methods. However, the event log is useful for attributing damaging actions to team members. At one event, a team member launched a mass automated exploitation attack from a shared server meant for post-exploitation only. Another team member accidentally cleared all of the hosts in the Metasploit database. The event log helped us identify which team members to counsel.

Data Sharing

Network attack teams generate and capture a lot of data during an engagement. This data includes port scans, vulnerability scans, encrypted passwords, working credentials, and other captured artifacts. Making this data available so that the whole team can locate it and work with it is difficult. Teams often rely on a version control system, an ftp server, or a wiki to store this information. The Dradis Framework [4] is an example of a specialized project to help attack teams organize their data and make it available to the whole team.

Ryan Linn examined these sharing options and presented another alternative in his Multiplayer Metasploit [5] work. Mr. Linn observed that wikis and other non-attack-specific storage mediums suffer from arbitrary organization. Dradis is a good alternative but it's hard to take action on the data from Dradis. His alternative idea is to use Metasploit as a data repository. Metasploit has several database tables to store credentials, encrypted passwords, known services, and data taken from hosts by automated post-exploitation scripts. Mr. Linn modified the Metasploit Framework to expose this data through Metasploit's RPC interface.

Armitage builds on Ryan Linn's work by using Metasploit for data sharing. Armitage's targets view shows the current hosts and active sessions. Any team member may right-click a host and select Services to see the known services and any banner information associated with the service.

In practice, Armitage's data-sharing features provided shared situational awareness and easy access to data automatically stored by Metasploit. Shared access to the Metasploit credentials table proved valuable in many situations. Each team member had access to all successful credentials when attempting to log in to a service. This data sharing also allowed any team member to export stored password hashes and attempt to crack them.

The Access Sharing Problem

In a team situation, the person who gets access to a host is usually the one who handles post-exploitation. This happens because it is difficult to share access with

other team members. This limitation forces teams to organize themselves by target types. For example, team members who are Windows experts attack Windows systems. This is a limiting tactic, as it is still hard for task specialization to occur. The Windows expert can't delegate post-exploitation tasks to one or more team members.

One possible solution to the access sharing problem is session passing. The multi_meter_inject script in Metasploit generates a Meterpreter executable bound to a callback host and port, uploads it to a host, and executes it on the compromised host.

Session passing is a threat to the network attack team's stealth. The uploaded Meterpreter executable may trigger the local antivirus or other personal protection product. More connections may help the system administrator determine that the host is compromised. In some situations, session passing is not practical. Sometimes it is difficult to pass a session for an available host, because it would require pivoting connections through another compromised host.

Session Sharing

An ideal solution to the access sharing problem is session sharing. Network attack teams would benefit from putting all successful compromises into a shared pool for any other member to use. Session sharing has a stealth advantage. It does not require creating a new access by uploading and executing a new program. Session sharing also allows all actions to occur through one communication channel. A system administrator cannot know if one person or five people are working on their host. Session sharing allows team members to benefit from each other's work. Session sharing also allows specialization of tasks. Some team members may focus on getting access to hosts, others may focus on persistence, and the rest may focus on post-exploitation.

For Meterpreter sessions, Armitage implements session sharing using "meterpreter multiplexing" described in the next section. With meterpreter multiplexing, multiple team members are able to simultaneously use a session. This solution creates an illusion that the access is not shared.

For shell sessions, Armitage limits access to the session to one team member at a time. Opening the session sends a request to the deconfliction server to see who owns it. If the session is in use, Armitage notifies the team member that the session is in use. If the session is not in use, Armitage locks it and lets the user interact with it. When the user closes the tab, Armitage notifies the deconfliction server that the session is available again.

Session sharing is the most useful teaming capability in Armitage. At all events I participated in, session sharing allowed team roles to emerge organically. Team members gravitated toward tasks they were most comfortable with. This is different from my previous exercise experiences, where team members who couldn't compromise hosts had limited participation opportunities.

Meterpreter Multiplexing

Meterpreter multiplexing is the Armitage feature that allows multiple clients to share one session. Armitage adds every meterpreter command to a queue specific to that session. A separate thread executes these commands in a first-in first-out way. When Armitage executes a Meterpreter command, it reads output until the command is complete. This output is then sent to the command requestor using the identifier stored with the command.

Armitage uses a heuristic to decide when a command is complete. The simplest heuristic is to read from a session until a read returns an empty string. Some commands return an empty string before they're finished. For these commands, Armitage expects a set number of empty reads to consider the command completed. For all commands, Armitage has a 12-second timeout. This timeout prevents a failed command from making the session non-responsive.

Some Meterpreter scripts execute in the background and report their output later. These scripts create a problem for the command-multiplexing scheme. It's possible for the output of a script to mix in with the output of another command. Armitage mitigates this by reading from Meterpreter before executing a command. When used with a local Metasploit instance, Armitage displays stray output in any Meterpreter tab.

The deconfliction server drops stray output because it does not know which client to route the information to. This is a drawback, but in practice it's limited to a few post-exploitation scripts. Metasploit is moving away from post-exploitation scripts in favor of post-exploitation modules. These modules are configured and executed just like exploits. Eventually, this problem with Meterpreter scripts will not exist.

Armitage queues commands in the Armitage client and deconfliction server. In the local client, the command queue delivers Meterpreter output to the GUI component that requested it. A user may execute multiple actions and Armitage will not become confused.

The deconfliction server uses the stored command identifier to identify the client that requested the command. When the deconfliction server finishes executing a command, it routes the output to the right client.

This multiplexing scheme creates the illusion that a shared access is not shared. When a team member executes a command, the command is added to the local command queue. When the command is executed locally, it is added to the deconfliction server command queue. When the command completes, the deconfliction server sends the command to the right client. The local client receives this output and routes it to the local GUI component. Figure 3 illustrates this process.

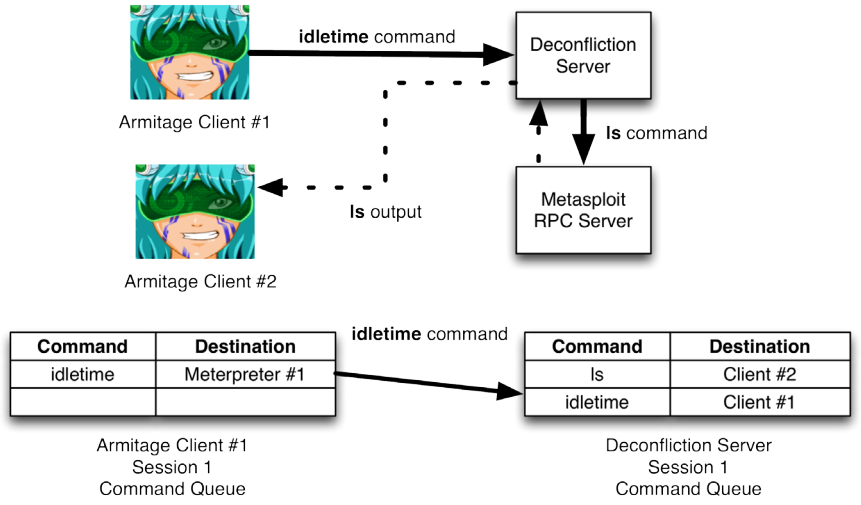

Figure 3: Meterpreter multiplexing in action

Windows command shells are a special case. Armitage interacts with the Windows command shell using Meterpreter channels. When Meterpreter executes a process, it creates a channel. Meterpreter provides commands to read from and write to these channels. When a client wants a command shell, it creates a new process through Meterpreter and it notes the channel associated with this process. Clients interact only with their own channels. Armitage uses the command queue to execute read and write commands to these channels. With this mechanism, multiple clients may interact with multiple command shells through one Meterpreter session.

Red Team Formations

This article has shown you how Armitage gives a network attack team real-time communication, data sharing, and session sharing built on the Metasploit framework. These features make it possible to experiment with different team organizations.

Excited about this shiny new technology, I used it to centralize the reconnaissance, exploitation, and post-exploitation activities in a collaborative capture-the-flag experiment. Under this organization, each team member used the shared Metasploit server to scan, attack, and carry out post-exploitation activities. I reckoned that this scheme would allow the team to move deeper into a target network, like an army marching deep into enemy territory. But I do not recommend this approach for attacking all hosts. A detected attack risks all sessions associated with the attack host. Detection is more likely when uncoordinated team members launch scans or attacks against the same host.

The most successful teaming option I've seen is to allow everyone to attack locally and handle post-exploitation through a shared Metasploit instance. Here, team members use their own tools to get access to a host. Once they're successful, they pass a session to the shared Metasploit server and kill their session. This decouples the attack host from the post-exploitation server. This worked well in practice, as everyone had access to existing sessions. This also reduced the normal red team chaos, as team members had no need to exploit already compromised hosts to get access. Team members with noisy attacks and scans risk detection of their local host only. Network defenders must find the attack source and the shared Metasploit server to keep the red team out.

Once a network foothold is available, it's safe to use a pivot set up on the shared Metasploit server for attack and reconnaissance of internal hosts. System administrators often focus on traffic entering and leaving their network, with little regard for what happens inside it. The risk of detection is low. Using a shared post-exploitation server also ensures that internal hosts get attention from the red team. Without session sharing, each team member needs a session on a host capable of reaching the desired internal targets.

A shared Armitage server also gives red teams the option to use Armitage as a dashboard for displaying the tactical situation. At the Northeast Collegiate Cyber Defense Competition we displayed Armitage using a projector in the red team room. The target area gave us situational awareness of what sessions we had at the time. The shared event log on the projector provided a timeline of recent sessions opening and closing.

Final Thoughts

Armitage helps network attack teams break away from the single-user assumption of Metasploit. In this article I described the communication, data sharing, and session sharing needs for network attack. I also described Armitage's features to meet these needs.

In practice, no feature completely replaced the old ways of collaboration. However, these features successfully augmented existing approaches. More importantly, session sharing allowed experimentation with different attack team organization and task delegation. In two cyber defense competitions, these features enabled collaboration on post-exploitation and shared situational awareness.

This article is the beginning of what's possible. I look forward to seeing what a mature red team does with this technology. It's now possible to experiment with and develop squad-level tactics for network attack.

References

[1] The Metasploit Project: http://www.metasploit.com.

[2] J. Koziol, D. Litchfield, D. Aitel, C. Anley, S. Eren, N. Mehta, and R. Hassell, *The Shellcoder's Handbook: Discovering and Exploiting Security Holes* (Wiley, 2004), pp. 147-148.

[3] Armitage homepage: http://www.fastandeasyhacking.com.

[4] Dradis Framework: http://dradisframework.org/.

[5] R. Linn, "Multiplayer Metasploit," DefCon 18 (2010): https://www.defcon.org/images/defcon-18/dc-18-presentations/Linn/DEFCON-18-Linn-Multiplayer-Metasploit.pdf.