Home-Field Advantage
Using Indicators of Compromise to Hunt Down the Advanced Persistent Threat

GIAC (GSEC) Gold Certification

Author: Matthew Toussain, mtoussain@gmail.com
Advisor: Angel Alonso-Parrizas

Accepted: September 25, 2014

Abstract
Within the information security community it is almost universally agreed that the adversary has the edge when they attack our networks. This premise stems from the idea that the attacker only needs to succeed once in order to get access to an organization's sensitive information while the defender must succeed every time. This principle is a fallacy. An attacker must succeed in some way at each stage of the hacker's methodology in order to penetrate their targets. As defenders we only need to stop them at one point. Furthermore, defensive cyber operators know their environment better than any antagonist can ever hope to. There are ways for the defender to take the initiative and hunt down the adversary as attacks occur. Organizations should leverage their home-field advantage by seeding their network with traps, snares, and pitfalls that will generate alerts early in the intrusion kill chain. This paper will describe methods that a security team can use to posture their environment to detect advanced persistent threat activity based on adversary tactics, techniques, and procedures.

1. Introduction

Current cyber defense strategies focus on building a wall around the network and "digging in". Behind this cyber version of the Maginot Line, network defenders attempt to block adversary intrusions in any way possible. Unfortunately, as history has shown, the strongest defenses are inadequate when the foe can simply maneuver around those impediments. To combat these tactics a new approach is necessary, "Through intelligence-driven response, the defender can achieve an advantage over the aggressor for APT caliber adversaries" (Hutchins, Clopperty, & Amin, 2010). The Advanced Persistent Threat (APT) has grown accustomed to reactionary defensive tactics and has rarely seen the fight taken to them. While the defender is constrained by laws that may not hinder their adversary, he is not limited traditional defensive mechanisms. By leveraging their home-field advantage, network defenders can come to understand and counter their opposition in a way similar to the redlighting of the APT1 cyber attack group (MANDIANT, 2013).

Layering a computer network with traps and snares is not a completely novel idea. For instance, a honeypot can be thought of as a lure to attract attackers (Provos & Holz, 2008). Where honeypots are systems fully dedicated to monitoring and detecting attacks, a honeytoken is simple a file or object. It is possible to hide small pieces of cyber information along the path an attacker must take to properly exploit their target environment. Pairing this strategy with adversary Tactics, Techniques, and Procedures (TTP), threat intelligence, and innovative triggers is a recipe to accelerate a traditional cyber defense into a manuverable battle plan.

2. Home-Field Advantage

In the computer age no one stands alone. Whether you are a computer user, developer, network defender, or hacker at the end of the day you are standing on the shoulders of others. Hackers abuse these similarities to aid their offensive operations. For instance, an exploit developed for a specific Windows system can often be extended to affect many systems. Network defenders are further relegated to a limited pool of security products. This engenders standardized defense patterns that attackers can abuse. A sophisticated attacker will develop methods to remain undetected that based on the most popular vendor software.

Network defenders should tailor their network defense strategy to be as unique to their environment as possible. By managing the adversary's perception of your organization's cyberspace footprint it is possible to use their own Tactics, Techniques, and Procedures (TTP) against them. The network security team should consider implementing enclave specific triggers, traps, and defenses.

2.1. Managing the Adversary's Site Picture

Sun Tzu once said, "If you know your enemies and know yourself, you will not be imperiled in a hundred battles". It is unrealistic to "know" all of the adversaries our networks may face, but we can certainly adopt their mindset and use that understanding to empower our defensive strategies. This means knowing how our networks appear to attackers. Adversary attack analysis identifies three major questions: What is their avenue of approach? What are they looking for? How are they going to get it?

2.1.1. The Intrusion Kill Chain

What is their avenue of approach? Lockheed Martin's intrusion kill chain closely resembles a typical hacker's methodology. The seven phases are: Reconnaissance, Weaponization, Delivery, Exploitation, Installation, Command and Control, and Actions on Objectives. From a defensive standpoint this identifies a series of engagement points. The earlier in this cycle an adversary can be engaged the less opportunity there is for them to cause damage to the organization.

Phase	Actions	Token
Reconnaissance	Sensitive Information	Hidden Page Access
Weaponization	Attack Vehicle	
Delivery	Direct Exploitation Externally Accessible Services DMZ Client-Side Exploitation Email Web Exploits	Fake Email Usage

	Social Engineering	
Exploitation	Gain Access	Hidden System Access
Installation	Deploy Toolkit	
C2	Establish Persistence	
Actions on Objectives	Pilfer Data	Honeytoken

Table 1 – Intrusion Kill Chain

By identifying adversary avenues of approach an organization can posture their environment to detect attacks and mitigate damage. Analyzing the tactics an adversary deploys can further allow a security team to posture their network defense to evolve alongside attacks.

2.1.2. Honeytokens

What are your organization's adversaries looking for? Once you have identified this critical information you can follow Sun Tzu's advice, "Hold out baits to entice the enemy. Feign disorder, and crush him." This brings us to the discussion of Honeytokens. A honeytoken is a protected file or object that can only be accessed through illegitimate means. This means that any access or use of a honeytoken is by definition bad, and if it is detected that action should be flagged as malicious.

Many networks utilize honeypots to detect adversary activity and alert a computer response team to their presence. These machines are often made to resemble outdated or vulnerable systems. While this "low hanging fruit" principle is often enough to entice attackers into engaging the system, it has two major disadvantages over more localized detection mechanisms. Firstly, if an adversary is engaging an internal honeypot, he has begun to move laterally. This means that the attacker has already completed his first round of the intrusion kill chain and established persistence on more vital portions of the network before the defense is alerted to his presence. Secondly, because honeypots often appear to be vulnerable systems they often are vulnerable systems. This can be a dangerous configuration that may make a victim's network more vulnerable to exploitation then it was before. Furthermore, these systems can often become buried under configuration management processes making them permanent network soft spots.

Under the honeytoken concept the network security team seeds their network with enticing, but false, cyberspace observables. These tokens are then used to identify attackers. Section 2.2 will detail 20 of the top cyberspace observables to enable adversary detection based on TTP.

2.1.3. Engagement Systems

Identifying the adversary's avenue of approach, allows us to recognize where we can engage them. Identifying critical information shows us what we can engage them with. Identifying how we can engage them takes those strategic concepts and focuses them on a tactical objective: keeping control of your network enclave.

While traditional network sensors can be leveraged to detect adversary access to honeytokens, the most useful data can be derived from standard logging systems. For instance, if a honeytoken took the form of a file containing the password to some live network system, we could create a detection mechanism that triggered on that specific login, as it would obviously be malicious.

Phase	System
Reconnaissance	Service Logs Firewall Logs
Weaponization	IDS/IPS
Delivery	Proxy/Network AV
Exploitation	Gain Access
Installation	Deploy Toolkit
C2	Establish Persistence
Actions on Objectives	Pilfer Data

Table 2 – Kill Chain by System

2.2. Top 20 Adversary Cyberspace Observables

During the post compromise phase of the intrusion kill chain, Actions on Objectives, an adversary is focused on accessing sensitive data and leveraging it to further compromise their target environment. By deploying host-based sensors capable of detecting malicious file access, the network defense team can detect APT activity

Matthew Toussain, mtoussain@gmail.com

before they have had the opportunity to move laterally. Below is a list of 20 of the top APT targeted files: (Crenshaw, 2011)

Filename	Details
1. *.txt	Text files, common names include: passwords, accountinfo, keys, logins
2. *.pdf	PDF files, often targeted for organization specific or personally identifiable information.
3. *.doc OR *.dox	Microsoft Word files, often targeted for organization specific sensitive information.
4. *.xls OR *.xlsx	Microsoft Excel files, often targeted for financial or personal information to include credit cards, email addresses, and social security numbers.
5. Profiles\Interfaces[1]	Wireless password stored location (encrypted but crackable)
6. *.OST[2]	Microsoft Outlook file
7. Content.Outlook	Outlook attachment temporary file
8. Skype\<Skype ID>[3]	Skype Conversation Logs
9. IE History[4]	Location of Internet Explorer web history
10. IE Cookies[5]	Location of Internet Explorer stored cookies
11. cert8.db	Certificates information stored in Firefox Profile
12. key3.db	Master key for stored passwords in Firefox Profile
13. signons.sqlite	Passwords stored in Firefox Profile
14. secmod.db	PKCS#11 module configuration in Firefox Profile
15. *.rdp	Remote Desktop Protocol saved authentication profiles
16. ultravnc.ini	Ultra VNC password file
17. spool\PRINTERS[6]	Print spool job location
18. *.evt OR *.evtx	Windows Event Log files
19. Appdata\Credentials[7]	Windows SMB share credentials (path varies by OS version)
20. formhistory.sqlite[8]	Firefox form history

Table 3 – Top 20 Cyberspace Observables

[1] C:\ProgramData\Microsoft\Wlansvc\Profiles\Interfaces
[2] C:\Users\<Profile>\ AppData\Local\Microsoft\Outlook
[3] C:\Users\<Profile>\AppData\Roaming\Skype\<Skype ID>
[4] C:\Users\<Profile>\AppData\Local\Microsoft\Windows\History
[5] C:\Users\<Profile>\AppData\Roaming\Microsoft\Windows\Cookies
[6] C:\Windows\System32\spool\PRINTERS
[7] C:\Users\<Profile>\AppData\Roaming\Microsoft\Credentials\<Random ID>
[8] C:\Users\<Profile>\AppData\Roaming\Mozilla\Firefox\Profiles\<profile number>.default\formhistory.sqlite

2.2.1. Collating Cyberspace Intelligence with CybOX

The CybOX, STIX, TAXII stack is a cyber threat intelligence specification. It provides a standardized means to catagorize, define, and share information regarding cyber threats (Mimoso, 2014). CybOX, the Cyber Observable eXpression, can be used to define "objects". In this sense we can use a standardized XML based schema to define the files that may be the most enticing for an adversary to interact with. CybOX has the additional capacity to represent "Events" or behaviors (Brown, 2013). This enables the potential for full spectrum dictation of an adversary's activity within its framework.

Because of the versatile nature of CybOX we can not only define the observables we intend to seed our network with, but we can also transform basic intrusion data into multifaceted Indicators of Compromise. For instance, if we created a file called *passwords.txt* and monitored it for unauthorized access. We could not only identify that access, but also log information concerning what process on the system opened its file handle, what network ports it is currently utilizing, and what other system files it has accessed. CybOX supports the logging of all of this information, which makes it an ideal format to empower future incident handling and response actions once a compromise has been identified. This gives us a basic list of resources to investigate.

Resource	Definition
Infected Process	This is the process attempting to access the HoneyToken
Infected Resources	These are the resources loaded by the infected process. These resources include but are not limited to: dlls, files, PE executables
Network Sockets	Network sockets opened by the infected process. This data includes port, destination, and protocol
Infected Settings	Services, Registry entries, and startup files pointing to the infected process executable

Table 4 – CybOX Resources

By automating the collection of the information described in Table 4 the time from compromise to revitalization of the network can be significantly reduced. Moreover, by collecting this information at the time a malicious file access is detected the network defense team can maintain a higher fidelity in their mitigation actions, and can provide management a clearer picture of the intrusion as well as greater assurance of adversary removal. Collecting this information can further support direct integration into network sensors and security systems.

2.2.2. Building the CybOX

As we create honeytoken files to detect nefarious network activity it is important to properly define them in such a way that multiple sensors can trigger on the details associated with them. In this section we will be looking at a *passwords.txt* file containing three separate observables that can be detected at different network nodes.

Observable	Node
File Handle	Host System
File Hash	IDS/IPS
Token Password	Login System

Table 5 – Observable in a Passwords File

The python-cybox library, provided by MITRE, was created to enable developers to add CybOX support into their tools. This library also provides a useful API to develop CybOX xml schemas defining files or observables.

In order to begin working with python-cybox you must have installed lxml. On OSX this can be built through macports:

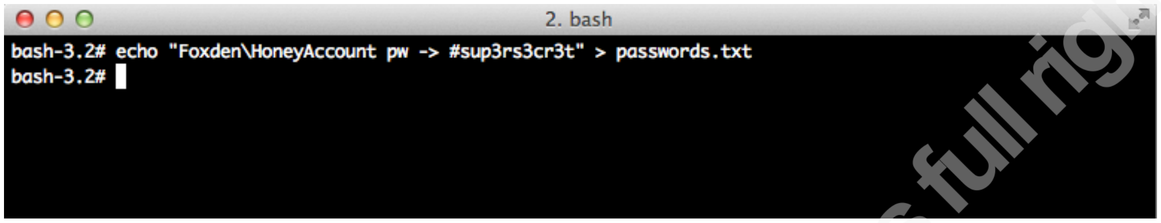
```
port install py27-lxml
```

Once lxml has been installed you can build and install CybOX:

```
git clone https:// github.com/CybOXProject/python-cybox.git
cd python-cybox
python setup.py build
python setup.py install
```

We will begin by creating a basic schema to represent a file called *passwords.txt.* As a text file the metadata associated with it is simple and therefore easy to define. Furthermore, as we attempt to define details associated our custom honeytokens it is important to be mindful of your organization's network configuration so that you can most adeptly leverage the enclave's security systems.

Create *passwords.txt* file:



```
bash-3.2# echo "Foxden\HoneyAccount pw -> #sup3rs3cr3t" > passwords.txt
bash-3.2#
```

Figure 1 – Password File

In defining this file the first step is to create a root element in the xml schema. All CybOX documents have "Observables" as their root element. The easiest way to build an observable is through the associated python code, so we will build it there prior to breaking down the resultant schema.

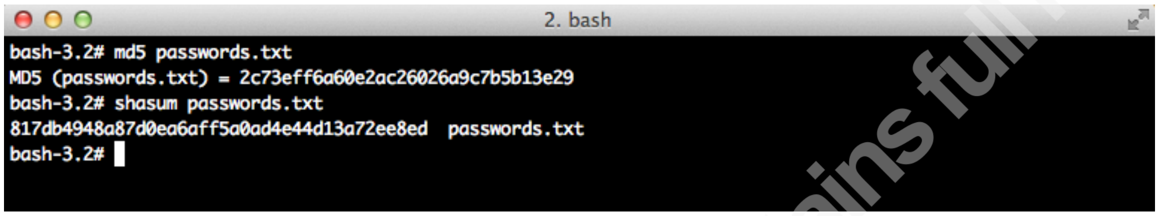
```
from cybox.core import Observables
from cybox.objects.win_file_object import WinFile
from cybox.utils import IDGenerator, set_id_method
set_id_method(IDGenerator.METHOD_INT)
f = WinFile()
f.file_name = "passwords.txt"
print Observables(f).to_xml(include_namespaces=False)
```

In the above code we import required CybOX modules and begin to put them to use. The WinFile() function allows us to record key information about a file. The result is a basic CybOX observable that looks like this:

```
<cybox:Observables cybox_major_version="2" cybox_minor_version="1"
cybox_update_version="0">
          <cybox:Observable id="example:Observable-1">
              <cybox:Object id="example:WinFile-2">
                  <cybox:Properties
xsi:type="WinFileObj:WindowsFileObjectType">

<FileObj:File_Name>passwords.txt</FileObj:File_Name>
                  </cybox:Properties>
              </cybox:Object>
          </cybox:Observable>
</cybox:Observables>
```

This allows us to store and share information about our honeytoken in a common format that can be understood by or translated to many different tools. Naturally, the fact that our honeytoken is named *password.txt* is a fairly trivial bit of data. Fortunately, CybOX is built to go deeper. The next step is to record the file hash. We will focus on the MD5 and SHA1 hashes.



```
bash-3.2# md5 passwords.txt
MD5 (passwords.txt) = 2c73eff6a60e2ac26026a9c7b5b13e29
bash-3.2# shasum passwords.txt
817db4948a87d0ea6aff5a0ad4e44d13a72ee8ed  passwords.txt
bash-3.2# 
```

Figure 2 – The Hash Observable

```
h = HashList.from_list([
       {'type' : 'MD5', 'simple_hash_value' :
       '2c73eff6a60e2ac26026a9c7b5b13e29'},
       {'type' : 'SHA1', 'simple_hash_value' :
       '817db4948a87d0ea6aff5a0ad4e44d13a72ee8ed'}
       ])
f.hashes = h
```

Within CybOX it is often possible to pass a dictionary or list to the observable function to create an embedded instance of a CybOX type. In this case that type is a file hash, which we can embed directly into our WinFile handle. The final step is to associate the Windows Domain account with the fake password in our honeytoken file. A Windows account is an entirely separate type of cyber observable. This means that we need to pair two observables within the same schema. We will define this one as p:

```
p = WinUser()
p.authentication
p.username = "HoneyAccount"
p.domain = "FoxDen"
p.locked_out = "True"
p.description = "PasswordToken: #sup3rs3cr3t"
```

Defining a Windows user is just as simple as defining a file and CybOX includes the depth to break each object into its constituent parts. Our final code to build this observable is can be found in Appendix B under Python – Cyberspace Observable.

Executing this code results in the schema that can be found Appendix B under XML – Cyberspace Observable.

As you can see, writing CybOX from scratch can be a fairly painful experience, but with the Python handles, it becomes feasible. The further programmatic application of cyber intelligence is well within the grasp of any aspiring security team

2.2.3. Translating Threat Intel into Sensor-Speak

CybOX includes a toolkit with python handles to enable correlation. CybOX-enabled detection of malicious activity is benefited by its ability to intake information from a diverse set of sensors, and output data into a common format. This further allows for pattern matching and analysis of adversary TTP to empower detection capability (Barnum & Saylor, 2011).

Another benefit of the CybOX, STIX, TAXII stack is its adaptability into languages used by sensors. Many tools exist to mold CybOX and STIX specifications into the formats of network security tools. Among the currently supported transformations are: OpenIOC, Open Vulnerability and Assessment Language (OVAL), SNORT rules, and YARA rules (MITRE). As there is no major shared repository for CybOX signatures the ability to transform CybOX into sensor speak is vital. For example, OpenIOC, an Indicator of Compromise specification championed by MANDIANT, can be converted by using the MITRE conversion script as easily as:

```
python openioc_to_cybox.py -i <OpenIOC XML file> -o <STIX XML file>
```

2.3. Rigging the Trap

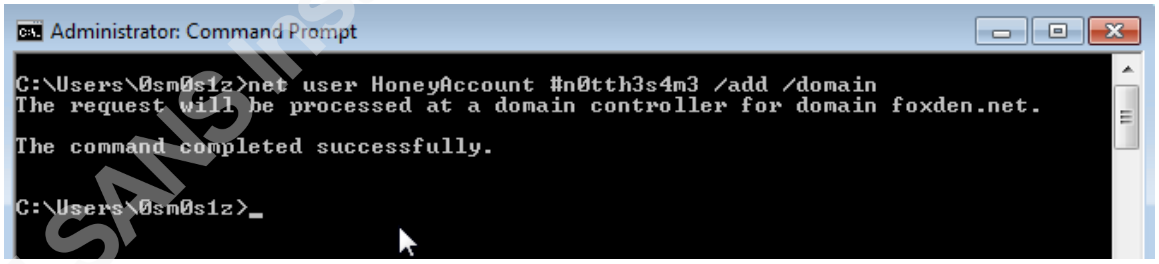
We have now discussed adversary engagement strategies, taken a look at what kinds of honeytokens might interest them, and delved into a universal format to define these cyber observables. Tactical application is the final step. First we will discuss detection based on honeytoken content by triggering an alert when there is an attempt to logon to a Windows domain using the honeytoken account hidden within our *passwords.txt* file. Next we will discuss how Data Loss Prevention (DLP) strategies can be used to alert the network security team if the honeytoken leaves the local network.

Finally, we will take a look at monitoring Windows file handles in order to detect when our honeytoken is first accessed.

2.3.1. Detecting APT with Account Lockout Policy and PowerShell

The premise of this detection mechanism is to create a fake user account in Active Directory with a lockout policy of one. The next step is to seed the network with files containing the associated username with an incorrect password. This file should be permissioned such that a nominal user cannot access the file to gain this authentication information. This means that when a logon attempt to this account does occur it must by its very nature be malicious. In order to detect these logons we simply need to monitor the associated account to see if it becomes locked and retrieve the associated event logs to obtain source information. In the following example we will be using the account information detailed within the CybOX specification in section 2.2.2.

First we begin by creating our honeytoken account under active directory. The name for this account should be something innocuous that appears to be typical of your organization's environment. For this demonstration our account name will be *HoneyAccount* with a fake password of: *#sup3rs3cr3t* and a real password of: *#n0tth3s4m3* as shown in figure 2.3.1.1.



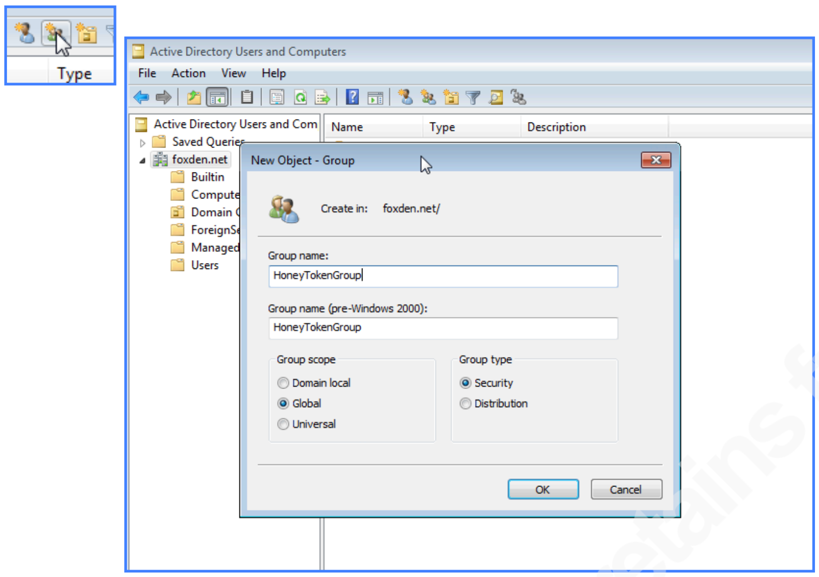
```
C:\Users\0sm0s1z>net user HoneyAccount #n0tth3s4m3 /add /domain
The request will be processed at a domain controller for domain foxden.net.

The command completed successfully.

C:\Users\0sm0s1z>_
```

Figure 3 – Creating a Honeytoken Account

Having created our fake user account we now create an associated group within the active directory through the ADUC administration tool:



Create in: foxden.net/

Group name: HoneyTokenGroup

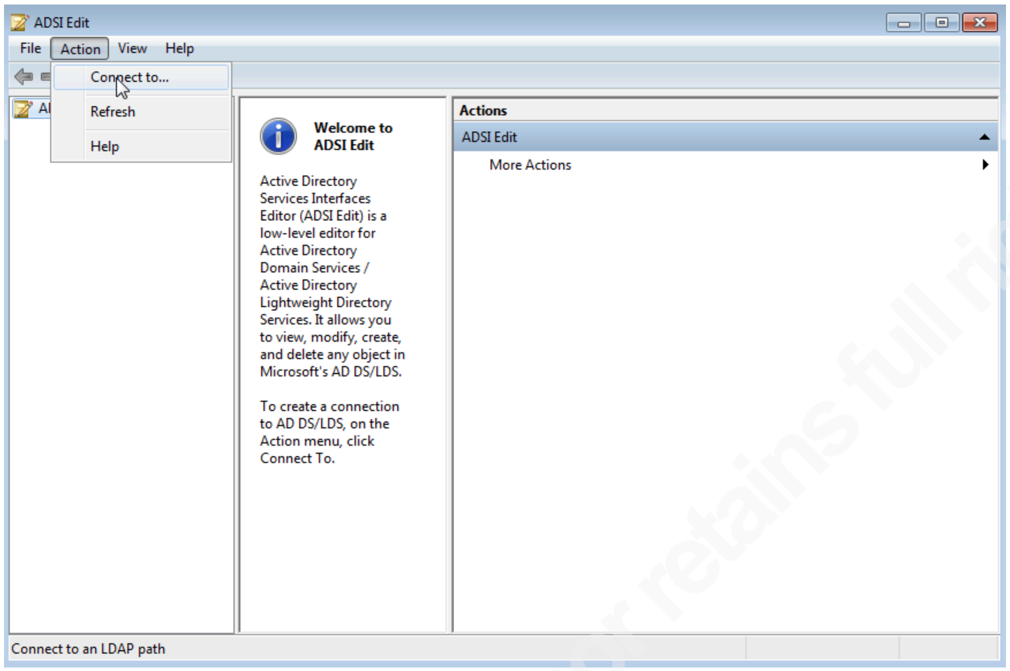
Group name (pre-Windows 2000): HoneyTokenGroup

Group scope: ( ) Domain local [✓ Global] ( ) Universal

Group type: [✓ Security] ( ) Distribution

Figure 4 – Using ADUC to Create the Honeytoken Group

From here, the account lockout policy is typically configured by setting up the Group Policy Object (GPO) located at: *\Computer Configuration\Windows Settings\Security Settings\Account Policies\Password Policy*. However, a limitation in Windows Active Directory is the inability to create group or user specific password policies without allocating them an entire domain. To get around this, Microsoft implemented AD DS Fine-Grained Password and Account Lockout Policy (Microsoft, 2012). These changes allow for the direct creation of a Password Settings Object (PSO) in order to explicitly assign account settings within a given domain. Note that these settings are not visible under *net accounts*. To create a PSO we use the Active Directory Services Interfaces (ADSI) Editor. The appropriate Microsoft Management Console (MMC) snap-in or a Windows Server can be used to access this toolkit.



Active Directory Services Interfaces Editor (ADSI Edit) is a low-level editor for Active Directory Domain Services / Active Directory Lightweight Directory Services. It allows you to view, modify, create, and delete any object in Microsoft's AD DS/LDS.

To create a connection to AD DS/LDS, on the Action menu, click Connect To.

Figure 5 – Connecting to the Domain with ADSI

From within the ADSI Editor connect to the appropriate domain as shown in Figure 5. Next, we navigate through *CN=DomainName -> CN=System* as shown below.

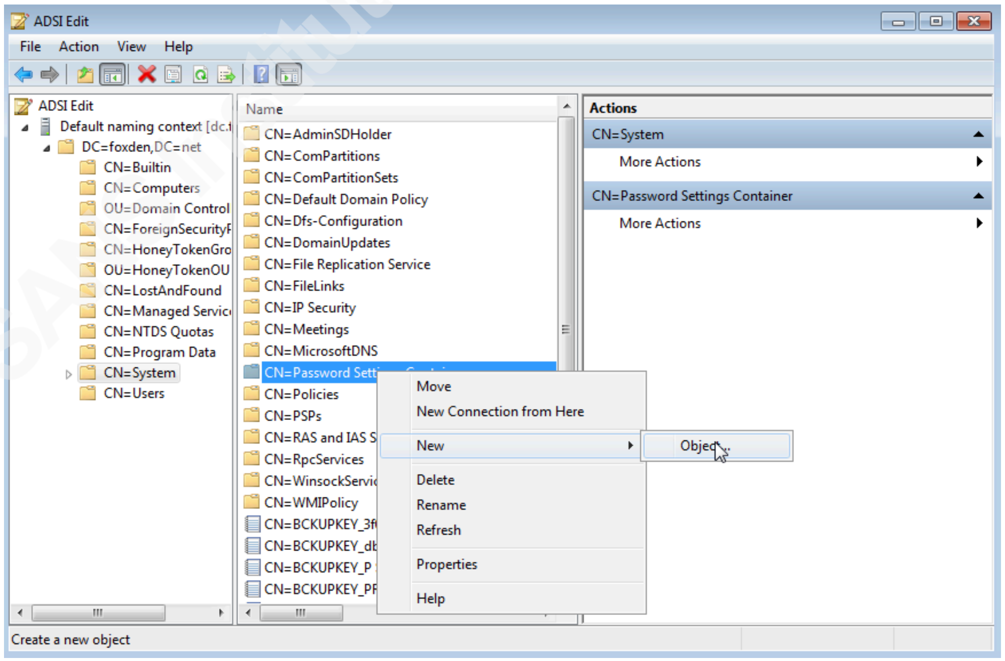


Figure 6 – The Password Settings Object

Now, we right click on *CN=Password Settings Container -> New -> Object*. And follow the wizard. We will be using Microsoft default settings with the exception of *Maximum Password Age* and *Lockout Threshold*. The settings configured are as follows:

Field Name	Setting
Common-Name	HoneyTokenUsers
Password Settings Precedence	10
Reversible Encryption	FALSE
History Length	24
Password Complexity	TRUE
Minimum Password Length	5
Minimum Password Age	1:00:00:00
Maximum Password Age	42:00:00:00
Lockout Threshold	**1**
Lockout Observation Window	0:00:30:00
Lockout Duration	**(never)**

Table 6 – Password Settings Wizard

The final step is to assign our newly created PSO to a specific user/group and monitor that group with PowerShell. To do this we need to enable advanced features under ADUC:

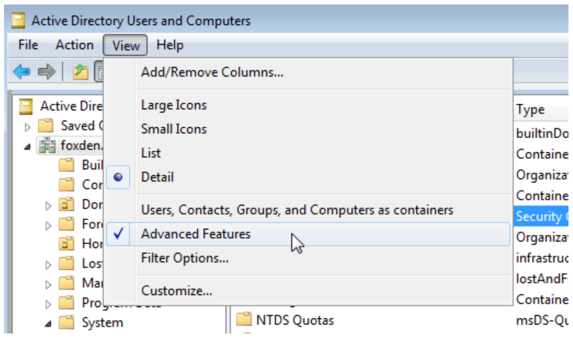
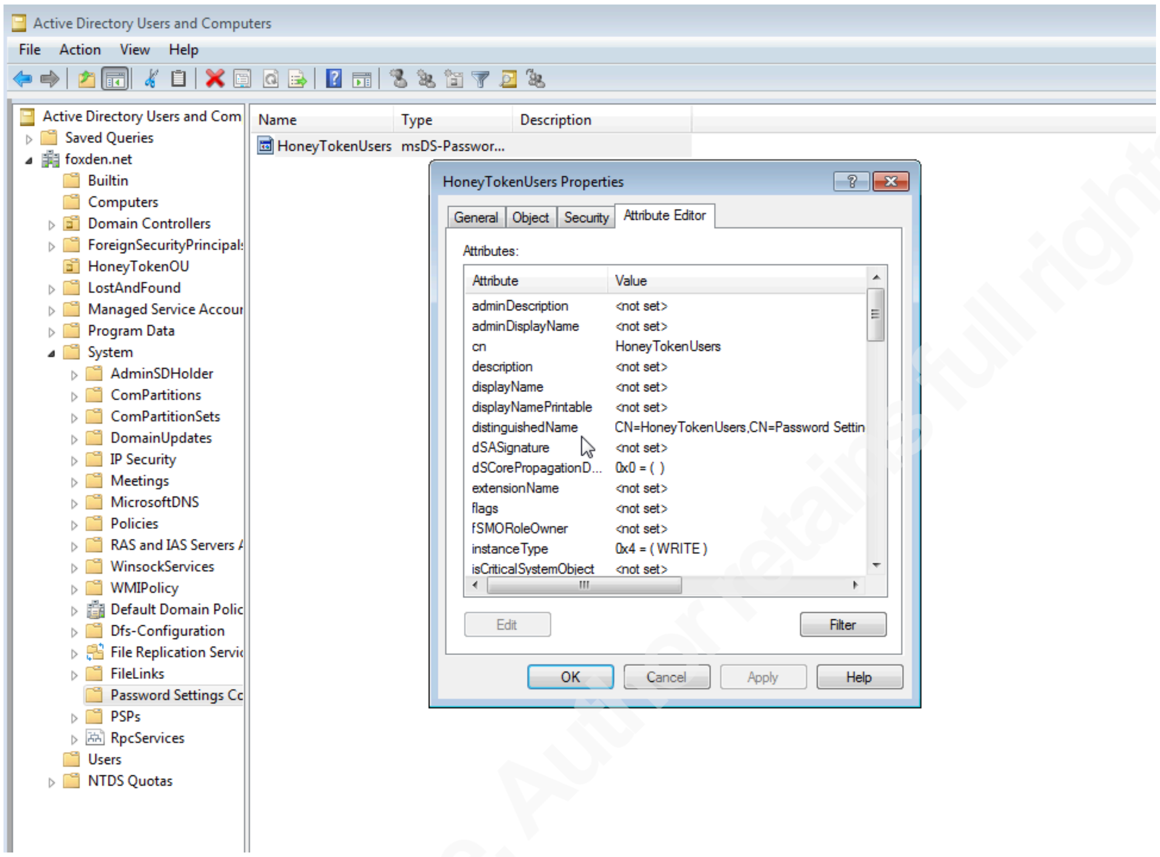


Figure 7 – Enabling ADUC Advanced Features

Next we have to navigate to our PSO. It can be found under *Domain -> System -> Password Settings Container*.



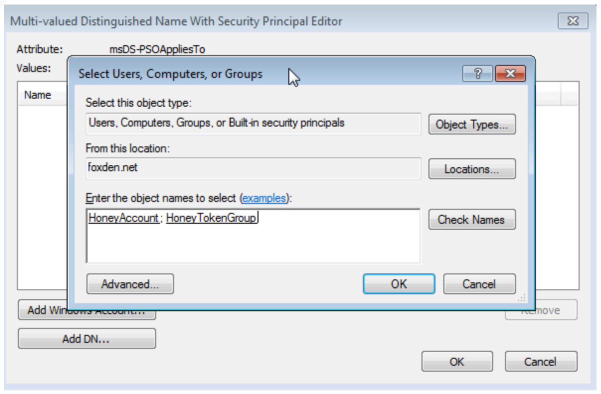
Name	Type	Description
HoneyTokenUsers	msDS-Passwor...	

Attributes:

Attribute	Value
adminDescription	<not set>
adminDisplayName	<not set>
cn	HoneyTokenUsers
description	<not set>
displayName	<not set>
displayNamePrintable	<not set>
distinguishedName	CN=HoneyTokenUsers,CN=Password Settin
dSASignature	<not set>
dSCorePropagationD...	0x0 = ( )
extensionName	<not set>
flags	<not set>
fSMORoleOwner	<not set>
instanceType	0x4 = (WRITE)
isCriticalSystemObject	<not set>

Figure 8 – The ADUC Password Settings Container

We now double click on the PSO and look under the Attribute Editor tab, then navigate to: *msDS-PSOAppliesTo* and set the variable to the honeytoken account/group.

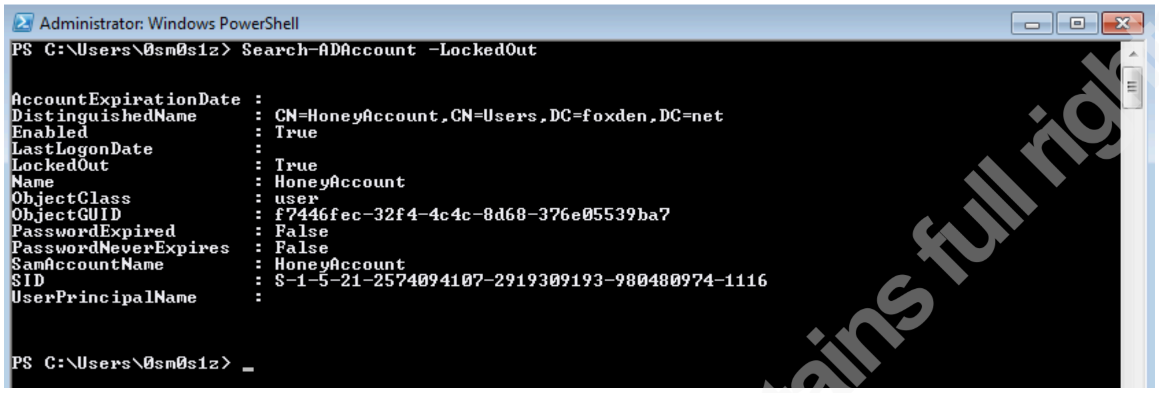


Attribute: msDS-PSOAppliesTo
Values:

Select this object type:
Users, Computers, Groups, or Built-in security principals
From this location:
foxden.net
Enter the object names to select (examples):
HoneyAccount; HoneyTokenGroup

Figure 9 – Set ADUC Group Participation

The account lockout policy has been set. The only objective remaining is to monitor the account's lockout status regularly to detect adversary activity. As is often the case with PowerShell, a built-in function takes what could have been an arduous task and makes it exceedingly easy:


Administrator: Windows PowerShell

```
PS C:\Users\0sm0s1z> Search-ADAccount -LockedOut


AccountExpirationDate  :
DistinguishedName      : CN=HoneyAccount,CN=Users,DC=foxden,DC=net
Enabled                : True
LastLogonDate          :
LockedOut              : True
Name                   : HoneyAccount
ObjectClass            : user
ObjectGUID             : f7446fec-32f4-4c4c-8d68-376e05539ba7
PasswordExpired        : False
PasswordNeverExpires   : False
SamAccountName         : HoneyAccount
SID                    : S-1-5-21-2574094107-2919309193-980480974-1116
UserPrincipalName      :


PS C:\Users\0sm0s1z> _
```

Figure 10 – Querying Account Lockouts with PowerShell

As you can see from the figure above *Search-ADAccount –LockedOut* grants network security personnel an easy means to determine when an adversary has attempted to gain access with a honey account.

2.3.2. Detecting Honeytoken Exfiltration with Data Loss Protection

Data Loss Prevention (DLP) is a relatively new technology in the computer security sphere. The primary purpose of DLP is to detect when sensitive information is stolen from your environment. Accomplishing this task can be monumentally difficult as it requires the appliance to break down the protocols associated with all network traffic. In the case of encrypted traffic like SSL it also requires the appropriate certificates or some method of man-in-the-middling (MITM) the connection. The effectiveness of these appliances is further reduced by nonstandard protocols. Since malware, naturally, does not utilize any standard means of communication, exfiltration through adversary remote administration tools is typically immune to DLP intervention. This makes honeytoken detection at the network level an extremely difficult affair.

Another confusing metric concerning DLP systems is their similarity and, perhaps, merger with the next-generation Intrusion Detection System (IDS) space. According to Gartner the DLP market is set to expand by 28.6% in 2014 alone (Pingree & Kota, 2014). Players in this market include: CA Technologies, Fidelis Cybersecurity Solutions, GTB Technologies, McAfee, RSA, Symantec, Verdasys, and Websense. Some opensource tools like Snort and Suricata also have limited DLP cababilities. Specifically, Suricata has the ability to breakdown HTTP data and examine files encapsulated within the protocol.

While DLP is an interesting consideration when investigating honeytoken usage, implementations of these systems are uncommon, and they are often cost prohibative. These downsides coupled with an inability to capture the requisite data make them a poor choice for most applications. However, when employed in particularly controlled environments DLP can be a powerful technology. That is to say if the appliance has visibility into all network traffic and a layer 7 firewall is used to enforce protocol usage network-wide. In this case a DLP solution would theoretically have the ability to accurately reconnoiter all network traffic.

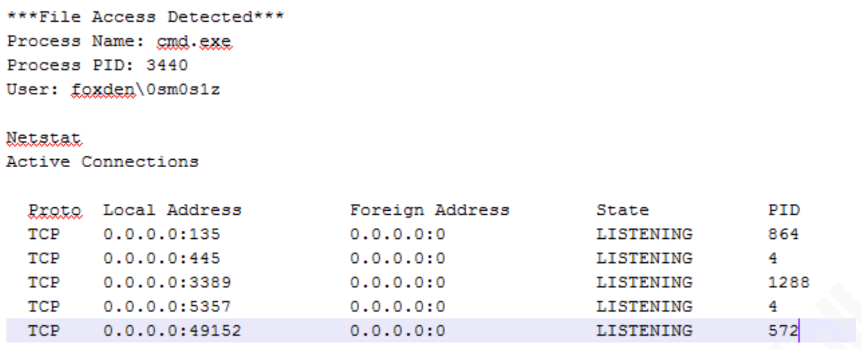
2.3.3. Monitoring File Handles with Ummidia

Ummidia is a trapdoor toolkit. Its primary focus is to monitor seemingly sensitive files. It accomplishes this task by listening to Windows process handles in order to detect file accesses. Once a malicious file access has been identified Ummidia will log source process name, process PID, process user, and current network connections. Ummidia can be downloaded from its Google code site here: https://code.google.com/p/ummidia/

The Ummidia configuration file is simple, consisting of one file to be monitored per line. For example:

```
C:\Users\0sm0s1z\Desktop\passwords.txt
C:\..\..\Mozilla\Firefox\Profiles\..\signons.sqlite
C:\Users\0sm0s1z\DC-login.rdp
```

Upon execution, Ummidia will immediately begin monitoring these files for unauthorized access. Once an access has been detected Ummidia will store the pertinent details in the *ummidia.log* file as seen below:



```
***File Access Detected***
Process Name: cmd.exe
Process PID: 3440
User: foxden\0sm0s1z

Netstat
Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       864
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:3389           0.0.0.0:0              LISTENING       1288
  TCP    0.0.0.0:5357           0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:49152          0.0.0.0:0              LISTENING       572
```

Figure 11 – Ummidia Output

In order to configure Ummidia to begin at startup, it should be setup as a Windows service. Note that in order to read Windows file handles Ummidia must always be executed with elevated privileges. To create a service under Windows we use the sc command.

```
sc create ummidia binPath= "cmd /K start C:\Ummidia\ummidia.exe" start=
auto error= ignore
```

A more legitimate-sounding service name should be utilized in live scenarios to reduce the chance of detection. Once Ummidia has been configured to monitor the honeytokens and run at startup the only remaining task is to monitor the log files. This can be accomplished through a UNC file path. Alternately, the Windows service could be configured to use a remote Ummidia executable. By running the program in this fashion it will log directly to the remote directory.

2.4. Additional Honeytokens to Consider

The most vital premise of honeytoken-based detection is its multifaceted nature. In this paper we have discussed 20 cyber observables to target and discussed three methods to focus on them; however, there are many other elements that bare consideration (Ullrich, 2009).

1. **Robots:** Fake admin pages referenced in the robots.txt file can be used as honeytokens. Legitimate web spiders will access these links and ignore the “Disallowed” pages. An attacker, however, will use this information to begin their attacks against a web application. Therefore, any web requests to these fake pages can be used as indicators to flag threats.

2. **Fake Cookies:** You can build attack detection honeytokens directly into web applications by using cookies. Attackers often look for poor authentication practices within cookies. By creating the appearance of this error, for instance creating an admin cookie and setting its value to FALSE, you can create a mechanism to detect when and if that cookie changes, signifying an attack.

3. **Hidden Form Fields:** Create <input type=”text”> fields with “display:none” set within your existing forms. Web app users will not see these fields; however, automated bots and vulnerability scanners will. This method will allow detection of an attack in its earliest stages, allowing the security crew to track, monitor, and block adversaries before any damage has been caused.

3. Conclusion

Active defense strategies typically focus on operators combing through packet capture and alert data, but by moving to a strategy that centers on tripping up the adversary and pouncing on their trail, the network defender can supplement their current efforts by interdicting the intrusion kill chain directly. In order to remain a step ahead of the Advanced Persistent Threat, it is important to remain conscious of the latest cyber defense strategies. Furthermore, it is the direct application of theory into the tactical environment that allows an organization to remain a step ahead of their threats. Static security without customization can be bypassed the same way each time, but when the network defense team pairs their specific environment with defensive mechanisms tailored in an information centric fashion it is the adversary who must tread lightly.

When implementing trap type security mechanisms, it is important to remember where it is that you are engaging the adversary. Awareness of the intrusion kill chain is vital when determining exactly what information your protections are guarding. For instance, if you place honeytokens amongst information with strategic value, you may catch the adversary when he is stealing your data; however, at this stage the attacker may have already completed their mission. Stopping them at this point could be too late. It is important to engage the adversary as early in their attack cycle as possible in order to best protect the target environment. Placing honeytokens on systems that are most prone to exploitation but do not themselves contain highly sensitive information, may be the best strategy to mitigate damage during an attack.

To best defend a network it is important to adeptly manage all information resources associated with adversary activity. Universal cyber intelligence standards like OpenIOC, OVAL, and CybOX can facilitate easy threat information sharing between organizations. Repositories for this type of intelligence exist and should be leveraged where possible. Cataloging and sharing threat intelligence is a vital step along the path of combating cyber-attacks. Collating this data into a universal format further allows for pattern analysis of this data. Most importantly, it yields analysts the ability to track trends, recommend additional defensive measures, and share information.

Honeytokens come in many forms. This paper discussed twenty of the top cyber observables to give the reader a starting point. It further demonstrated three methods to detect the access and usage of these tokens. Nevertheless, there is no end to the results creativity can produce. Honeytokens can be broken down into two pieces. The file or object itself and the data contained within this object. Both of these details are individual tokens, and each can be used in different manners.

Standard, static cyber-attack detection and prevention measures are insufficient to adequately protect networked environments. Conventional wisdom states that because security is mathematically unprovable attackers have the advantage over network defenders. By focusing on the intrusion kill chain and using offensively geared techniques, network defenders can leverage their home-field advantage to trip up and catch the APT. A new definition of active defense must be pioneered in order to detect advanced threats. Rather than focusing on signature-based detection methodologies, network defenders should investigate the adversary from a human factors standpoint. What is their avenue of approach? What are they looking for? How are they going to get it? By triggering on this activity when it occurs and by booby-trapping the information they are pursuing, the network defense team can take the fight to the APT's doorstep. While security is unprovable, an attacker must still succeed in some way at every stage of their intrusion. If Defense-in-Depth principles are utilized and the defense leverages flexible detection methods at each phase of their enclave, it is possible to take back the initiative and hunt down the adversary as attacks occur.

Matthew Toussain, mtoussain@gmail.com

4. References

Barnum, S. (2013). *Standardizing Cyber Threat Intelligence Information with the Structured Threat Information eXpression (STIX™).* MITRE.

Barnum, S., & Saylor, G. (2011, August). *Enabling Distributed Event Management: Interoperability for Automated Response and Prevention.* Retrieved from CybOX: https://cybox.mitre.org/documents/Enabling%20Distributed%20Event%20Management%20(GFIRST%202011).pdf

Barnum, S., Martin, R., Worrell, B., Kirillov, I., Chase, P., Beck, D., & Melachrinoudis, S. (2012). *The CybOX Language Defined Objects Specification.* MITRE.

Brown, D. (2013, November 6). *Examining Threat Information Representations, Part 2: CYBOX and Friends*. Retrieved July 25, 2014, from Bit9: https://blog.bit9.com/2013/11/06/examining-threat-information-representations-part-2-cybox-and-friends/

Crenshaw, A. (2011, September). Pilfering Local Data: Things an Attacker Would Want to Grab with Short Term Local Access. *InfoSec Nashville.* Nashville Technology Council.

Hutchins, E. M., Clopperty, M. J., & Amin, R. M. (2010). Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. (A. C. Ltd., Ed.) *Proc. 6th Int'l Conf. Information Warfare and Security (ICIW 11)*, 113-125.

MANDIANT. (2013). *APT1 - Exposing One of China's Cyber Espionage Units.* MANDIANT.

MANDIANT. (n.d.). *An Introduction to OpenIOC.* Retrieved May 18, 2014, from OpenIOC: http://openioc.org/resources/An_Introduction_to_OpenIOC.pdf

Microsoft. (2012, August 20). *AD DS Fine-Grained Password and Account Lockout Policy Step-by-Step Guide*. Retrieved August 21, 2014, from Microsoft Technet: http://technet.microsoft.com/en-us/library/cc770842(WS.10).aspx

Mimoso, M. (2014, June 23). *MICROSOFT TO PREVIEW INTERFLOW INFORMATION SHARING PLATFORM*. Retrieved August 10, 2014, from ThreatPost: http://threatpost.com/microsoft-to-preview-interflow-information-sharing-platform

MITRE. (n.d.). *STIX FAQ*. Retrieved August 15, 2014, from MITRE: http://stix.mitre.org/about/faqs.html

Pingree, L., & Kota, H. (2014, January 31). *Competitive Landscape: Content-Aware Data Loss Prevention Market, 2014*. Retrieved August 21, 2014, from Gartner: https://www.gartner.com/doc/2660219?srcId=1-2819006590&pcp=itg

Provos, N., & Holz, T. (2008). *Virtual Honeypots: From Botnet Tracking to Intrusion Detection.* Boston: Bearson Education Inc.

Robertson, C. (2013, February). Indicators of Compromise in Memory Forensics. *SANS Reading Room*, 24.

Ullrich, J. (2009, June 2009). *My Top 6 Honeytokens*. Retrieved August 24, 2014, from SANS Software Security: http://software-security.sans.org/blog/2009/06/04/my-top-6-honeytokens/

Matthew Toussain, mtoussain@gmail.com

Appendix A: List of Tables & Figures

5. Appendix B: Building the CybOX

Python – Cyberspace Observable

```
import cybox.bindings.cybox_core as core_binding
from cybox.core import Observables
from cybox.objects.win_file_object import WinFile
from cybox.objects.win_user_object import WinUser
from cybox.utils import IDGenerator, set_id_method
from cybox.common import HashList
set_id_method(IDGenerator.METHOD_INT)

f = WinFile()
f.file_name = "passwords.txt"

h = HashList.from_list([
       {'type' : 'MD5', 'simple_hash_value' :
       '2c73eff6a60e2ac26026a9c7b5b13e29'},
       {'type' : 'SHA1', 'simple_hash_value' :
       '817db4948a87d0ea6aff5a0ad4e44d13a72ee8ed'}
       ])
f.hashes = h

p = WinUser()
p.authentication
p.username = "HoneyAccount"
p.domain = "FoxDen"
p.locked_out = "True"
p.description = "PasswordToken: #sup3rs3cr3t"


print Observables([f, p]).to_xml(include_namespaces=False)
```

XML – Cyberspace Observable

```
<cybox:Observables cybox_major_version="2" cybox_minor_version="1"
cybox_update_version="0">
    <cybox:Observable id="example:Observable-1">
        <cybox:Object id="example:WinFile-2">
            <cybox:Properties
xsi:type="WinFileObj:WindowsFileObjectType">
                <FileObj:File_Name>passwords.txt</FileObj:File_Name>
                <FileObj:Hashes>
                    <cyboxCommon:Hash>
                        <cyboxCommon:Type
xsi:type="cyboxVocabs:HashNameVocab-1.0">MD5</cyboxCommon:Type>

<cyboxCommon:Simple_Hash_Value>2c73eff6a60e2ac26026a9c7b5b13e29</cyboxCo
mmon:Simple_Hash_Value>
                    </cyboxCommon:Hash>
                    <cyboxCommon:Hash>
```

```
                        <cyboxCommon:Type
xsi:type="cyboxVocabs:HashNameVocab-1.0">SHA1</cyboxCommon:Type>

<cyboxCommon:Simple_Hash_Value>817db4948a87d0ea6aff5a0ad4e44d13a72ee8ed<
/cyboxCommon:Simple_Hash_Value>
                    </cyboxCommon:Hash>
                </FileObj:Hashes>
            </cybox:Properties>
        </cybox:Object>
    </cybox:Observable>
    <cybox:Observable id="example:Observable-3">
        <cybox:Object id="example:WinUser-4">
            <cybox:Properties
xsi:type="WinUserAccountObj:WindowsUserAccountObjectType"
locked_out="true">
                <AccountObj:Description>PasswordToken:
#sup3rs3cr3t</AccountObj:Description>
                <AccountObj:Domain>FoxDen</AccountObj:Domain>

<UserAccountObj:Username>HoneyAccount</UserAccountObj:Username>
            </cybox:Properties>
        </cybox:Object>
    </cybox:Observable>
</cybox:Observables>
```

Matthew Toussain, mtoussain@gmail.com