PATERVA (PTY) LTD

Maltego Tungsten with Teeth / KingPhisher

Creating a collaborative attack platform with Maltego Tungsten

RT

2013-07-12

Contents

Background

One of the key features of Maltego Tungsten (to be released at BlackHat 2013) is collaboration. It allows analysts to share graphs in real time over XMPP. This allows a group of analysts (or attackers) to work together on the same goal. Combined with the already state of the art footprinting and personal profiling capabilities we have in Maltego (think machines in Radium), this provides a very capable attack platform. It combines human intelligence, pattern recognition and powerful automated attack tools with graphical information sharing software. When we designed and built this system our motivation was to create the ultimate attack platform that can be used by multiple analysts.

Design criteria

The following limitations where considered:

- **External attack** – meaning that the attack team will be conducting their attacks over the Internet. In other words, not internal to the network, not over wi-fi or from hosts compromised prior to the attack.
- **No 0day** – we assume that the attackers do not have access to 0day. The success of the attack should not hinge on the availability of 0day.
- **Zero knowledge (black box**) – we assume that at the start of the attack the attackers have no prior knowledge of the target systems in use or the people involved.
- **Attacking a large organization.** We will assume that the network or organization under attack is national or multi-national. In other words, the type of attack is most useful for a wide selection of targets – not a single specific host or person.

Other design criteria were that all transforms would run as local transforms (for the 'teeth' segment), that the code be open and in written in Python/PHP and that it would be very easy to modify and extend. The platform chosen to develop and deploy this is Kali Linux.

MaltegoTeeth

Since the first release of Maltego we've made sure that none of the bundled transforms were offensive. While the results of the transforms were very valuable for security analysts the transforms themselves did not do a lot to raise eyebrows. With our BlackHat 2013 talk we're going back to our roots – security. We've created a set of transforms that do not pretend to be friendly, that do not beg for forgiveness or ask for excuses. When using these transforms you'll be clearly attacking a target.

There are three sections of interest.

- **Infrastructure.** These are 'conventional' attacks against machines connected to the Internet – as opposed to people or personal devices.
- **People.** These are essentially attacks where people are involved in the success of the attack. Think social engineering & spear phishing.

These will be discussed in detail below.

Infrastructure - Teeth

Keeping the limitations in mind there are basically three steps we follow here:

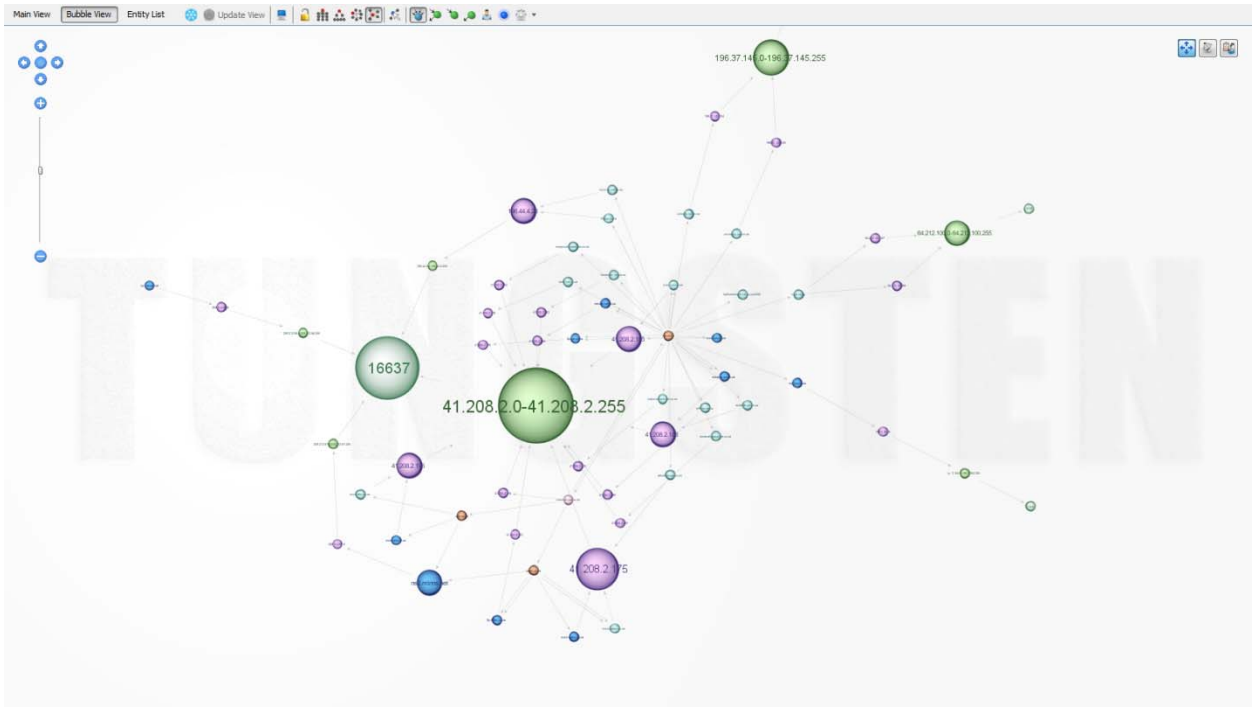
1. What do they have out there?
2. What can we get from what's already out there?
3. Can we get more by breaking controls?

Step 1: What's out there?

The first step is discovery of everything that the target exposes to the Internet. This is basic footprinting – something that's always been a key feature of Maltego. Using the pre-packaged footprinting machines introduced with Maltego Radium you can literally do a one-click footprint of a large organization.

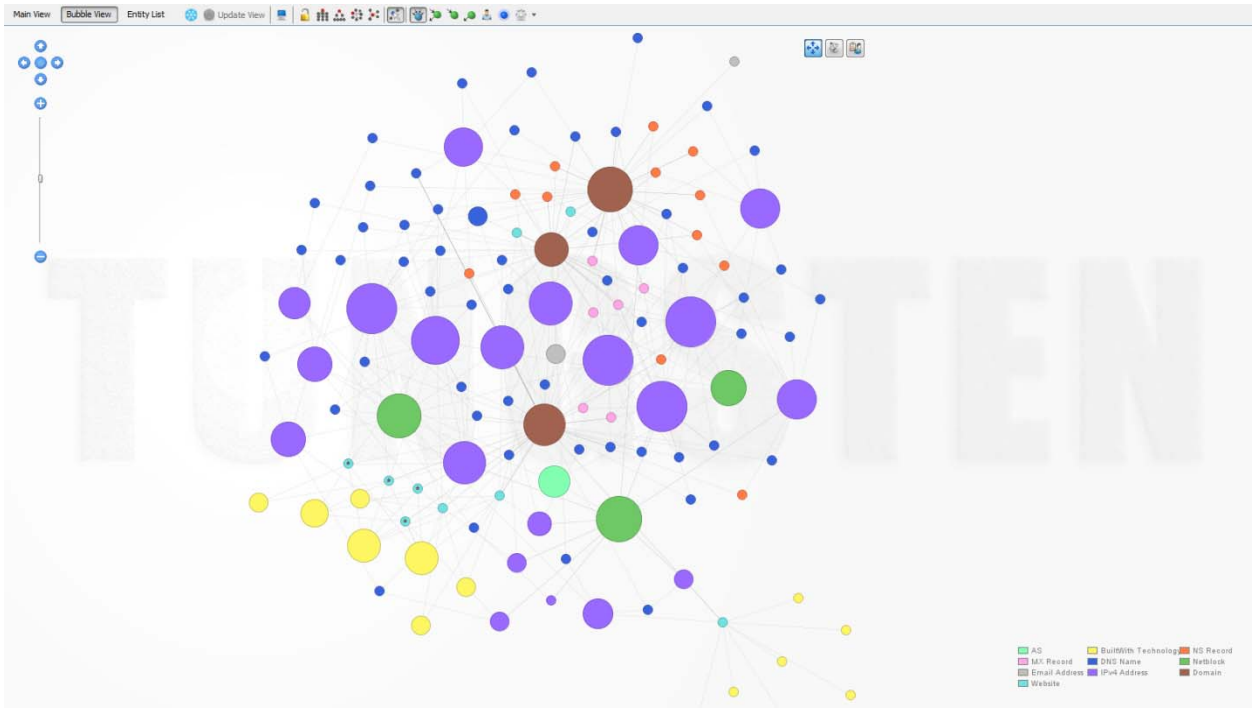
The methodology of this foot print has been discussed in depth in the past and is not repeated here.

Below is a graph generated by the L3 foot printing machine of Maltego on the Johannesburg stock exchange (JSE):

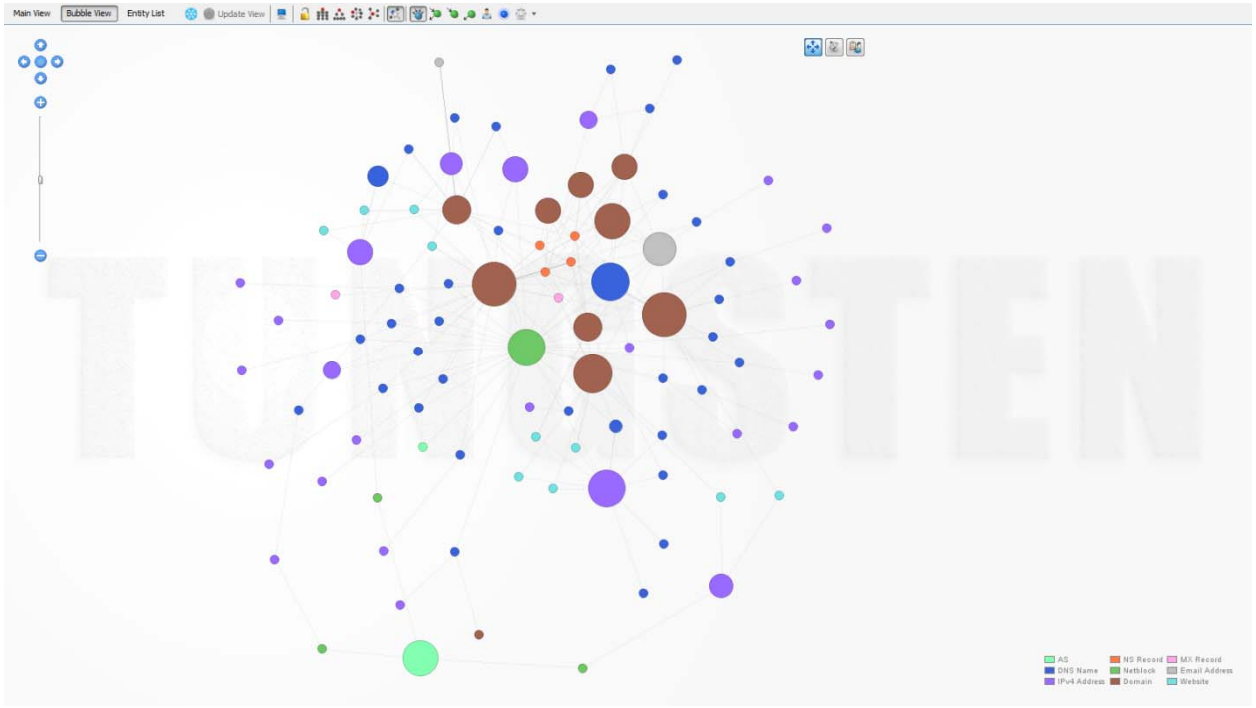


It's not a massive graph as the JSE does not have a large external footprint.

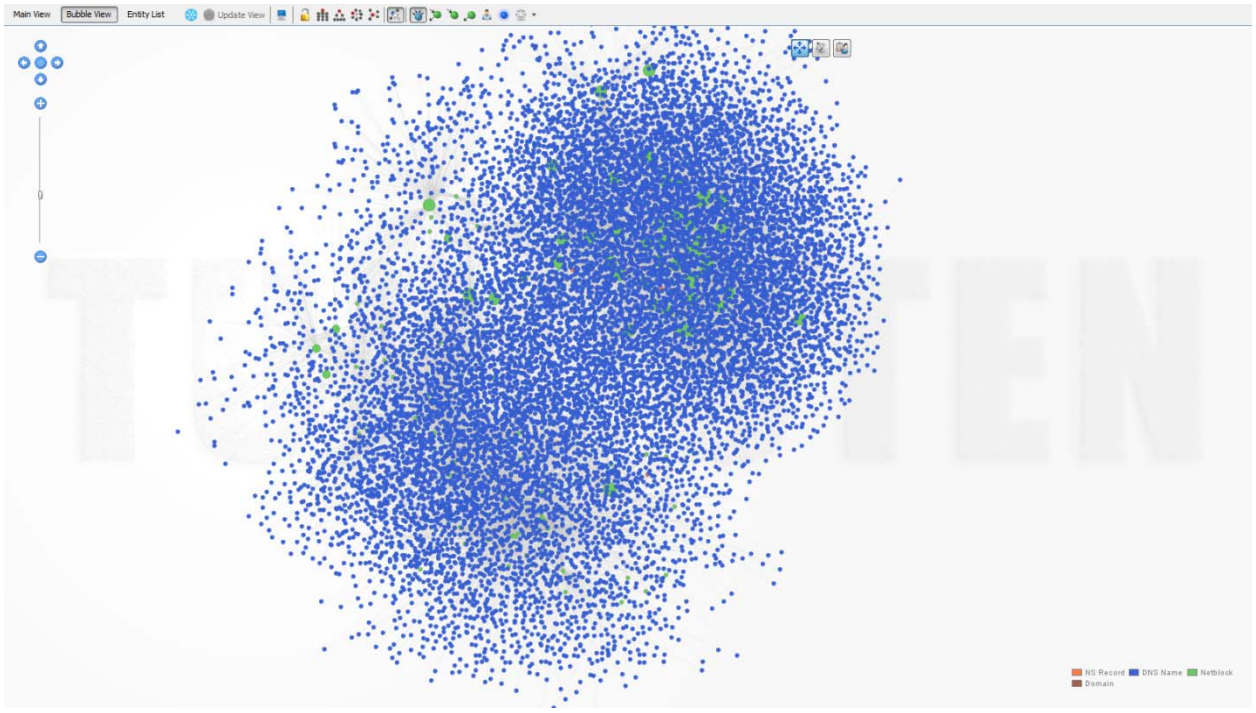
The footprint of Iran's AEOI:



Or the CIA:



In comparison, a really large network map of Yahoo (just DNS names and domains):



The result of a footprint provides us with:

- Domains and DNS names (Websites, MX records, NS records, reverse DNS names amongst others)
- IP addresses, netblocks and AS numbers

Step 2: What can we get to?

File and directory mining

In terms of exposed services you are most likely to see HTTP and HTTPs. The most basic form of info gathering would be to look for unlinked files and directories on these web servers – in other words file and directory mining.

There are many scripts and scanners out there that perform this function – however we've found that few very of them do this properly (e.g. not looking at HTTP status codes but rather comparing server responses). The right way to do this is as follows:

- Request a directory that you know does not exist and store the response in X
- Request another directory that you know does not exist and store the response in Y
- If X and Y are similar (we're using Levenshtein distance) then we know we can test properly. If they are not similar it's not possible to test other directories
- Now request the directory you're interested in and compare the response to Y (or X). If it's significantly different – then the directory is likely to be there.

In the same way we can test for file name – with two exceptions:

- Baseline testing (X/Y) needs to be performed per file extension type. The response for an unknown ASPX file might be different to that of a PHP file.
- Baseline testing (X/Y) needs to be performed per directory. The response from the system for */scripts/login.php* might be different to the response for */backup/login.php*

There are a few other things to keep in mind. To get the real response it's best to test the actual output of the file/directory request – meaning after following HTTP redirects. For this reason we're using the Mechanize library. Other than Selenium (which is too heavy weight for this and not as portable as we would like) this is the closest you are going to get to a real browser. The disadvantage of using Mechanize is that it does not interpret Javascript.

Testing for a list of directories or files in the root of a web server is only so exciting. In order to do this job properly we also need to check for files and directories in other paths of the server. Imagine that the entire web structure is located under */corp/* - then testing for */backup.zip* (while mandatory) is a lot less interesting than testing for */corp/backup.zip* or looking *for /corp/mediafile/archive/*.

We use two methods for getting valid web directories:

- performing a crawl/mirror of the site
- requesting and parsing for */sitemap.xml*.

Not all sites contain *sitemap.xml*, but for those that do it's an instant win – we'll get the entire site's structure. If that fails we can always revert back to crawling the site.

With the site's structure known (or partially known) we can now happily scan for files and directories in these known paths and we'll do this for all the websites we found in the footprint.

Checking for indexability

At this stage we should have a nice list of directories that we've found either by crawling, brute forcing or inspecting *sitemap.xml* and it would makes sense to see if any of them are indexable. Enough said.

Finding CMS backend

CMS (Content Management System(s)) have become very popular. Two popular CMSes are Wordpress and Joomla. Both of these provide the user with an administrative backend where they can log in to upload new content or modules. With Joomla and Wordpress it is possible to upload a malicious module/addon that will provide a web based command shell (like C99 shell). In most cases these interfaces are exposed to the Internet – this means access to the web content or upload a malicious payload it is merely protected by a single password. They are mostly located in a set location and it's trivial to test if they exist. In MaltegoTeeth we're looking at 3 different CMSes:

- Joomla at */administrator/index.php*
- Word Press at */wp-admin/*
- cPanel at *:2082/login/* (although not a real CMS still very popular)

In the next section we'll see how we brute force these CMSes.

Finding OWA (and other interesting) interfaces

OWA (Outlook Web Access) has almost become the de facto standard way to provide remote email for large corporate organizations. When looking at the Fortune 1000 companies we've found that about 60% of them have exposed OWA interfaces. In most cases you can simply browse to this interface and enter a valid username and password in order to gain access to the mail interface. The interface not only supplies the actual email but the organization's calendar and address book (names, phone numbers, departments) as well – this is a gold mine for other attacks (think social engineering – see next section).

There are a few OWA types used:

- OWA 2003
- OWA 2007
- OWA 2010
- OWA Office
- OWA Live (cloud)

These are protected either with a forms based login or with NTLM.

The way Teeth finds these are as follows:

1. Using Async DNS search for the following DNS names

   mail,webmail,owa,outlook,exchange,secure,gateway,vpn,activesync,connect

2. Check that it does not resolve to a wildcard entry
3. See if its open on port 443 (we assume nobody will run their webmail over HTTP)
4. If you find HTTP code 401 assume it's NTLM and mark as such
5. If you get HTTP code 200 follow redirects and store the response
6. Compare the response with a list of signatures and see which ones matches
7. If it does not compare to a response mark it as 'generic'.

It becomes clear from steps 5 and 6 that this method also works for other types of interfaces such as VPNs, gateways, webmail etc. providers. In Teeth we've stored 22 different responses – from CITRIX gateways, SecureID interfaces to Cisco Web VPN responses and adding additional types is trivial.

In the next section we'll show how we brute force OWA. Building a brute force attack script for other types is left as an exercise to the reader.

Finding Possible Injection Points (PIPs)

Using automated SQL injection tools such as SQLMap one can test for SQL injection with minimal effort. To be able to do so you need to point it to a URL, specify if it's a POST or a GET and which parameters to test for.

A side effect of a crawl/mirror is that we also have a good idea of what forms (or GETs with parameters) are surface visible on a website. Note that we say surface visible – e.g. not forms or GETs that are located behind web forms that require authentication or elaborate Javascript.

Armed with this info we can make a calculated guess which forms would be interesting to test for SQL injection. A form with a single submit button might not be interesting. A form with one parameter called 'print' might not be interesting. We might end up testing just a single parameter.

As an example consider the following form:

- POST **/search.php**
- Parameters:
    - **s**=
    - **print** = true
    - **redirect_url**=http://www.site.com/search_done.php
    - **sessionID**=12345

In this case the only parameter that is likely to yield interesting results when tested would be '***s***'. This does not mean that we can discard the other parameters when sending the request as it might cause the script to break. We thus get to a situation where we can filter which parameters to test for based on the names, actions or values of the parameter.

In the next section we'll look at how we can attack these PIPs.

Step 3: Breaking controls

While the methods used in the previous section could perhaps still be seen as information gathering techniques the methods discussed here is all out attack.

Brute forcing CMS

A Metasploit plugin for brute forcing Wordpress exists and we'll use that to brute force Wordpress. The password list is extended with three derivates of the domain name. Let's assume that the website is called *www.cookiesrus.co.za*. The password list will be extended to include *cookiesrus123*, *cookiesrusadmin* and *cookiesruspassword*. This can easily be extended to include other mutations.

Joomla brute forcing is performed using a custom Python script – this is because Joomla inserts a CSRF token in every login page which is checked when a POST is sent and most generic bruce forcing scripts does not support this. The same password extension happens for Joomla brute forcing.

Successful logins are graphically indicated.

Brute forcing OWA

In order to brute force OWA the attacker needs to select which email addresses she wants to use first. These are written via the Maltego GUI to a local file. When the user runs the brute force OWA transform an external dialog window is popped up to ask the user from which domain she wants to select email addresses.

The attacker can send email addresses with or without the domain (used in the email address, not to be confused with Microsoft-world domain). The thinking here is that OWA really uses a username and (Microsoft-world) domain. If the (MS) domain is not specified it will use the default domain – which in most cases is the best bet. The username *might* be the part of the email address in front of the @ sign and many large corporate organizations are now adopting this naming convention. The other option could be that the MS domain and the email address domain is really the same thing, in which case it would make sense to send both.

The OWA type is stored within the entity and based on the type either NTLM or web forms is used. NTLM brute forcing uses Hydra while form based login is done with Mechanize.

Another interesting side effect of OWA brute forcing is denial of service. Most OWA implementations lock the account after 3 incorrect password attempts. As such the password file for OWA only contains two passwords. If the file contains more than 2 passwords (and they are indeed incorrect) the account will be locked out. Since many OWA servers are directly connected to the main domain controller of the organization this might mean a system wide lockout – at least temporarily. Such a denial of service against multiple accounts would seem to be very costly for the target organization.

SQL injection – attacking PIPs

In the previous section we saw how we can identify possible injection points (PIPs). Using SQLMap we now test if these forms or GETs are indeed injectable. The transform simply runs SQLMap on the URL with the correct parameters. When an injection string is found to be successful it is shown graphically on the Maltego interface – the default behavior of SQLMap is to show the database and current table structures.

There are some limitations and considerations when using this attack:

1. The PIP scanner will only identify surface visible forms – this was discussed in more detail in the previous section.
2. Since we're not sure what database type is in use we can either scan using all types (SLOW) or specific the database type. In subsequent version of the framework this should be addressed – e.g. finding the most likely database type automatically.
3. SQLMap takes a long time to run. As such you need to carefully select which forms to attack.

It would be possible to extend this so that the list of database are shown as entities from the PIP, with transforms to enumerate tables from these databases. This functionality is left as an exercise to the reader.

Port scans / Service scans / Nmap with NSE scripts

After careful consideration (we looked at Nessus, Metasploit etc.) it was decided to go with Nmap with NSE scripts as makeshift 'vulnerability scanner'. We use this is quotes because it's not a full blown vuln scanner but rather a compilation of scripts and a good port scanner. We've found that in many cases the NSE scripts bundled with Kali Linux served our purpose very well. Other considerations were stability, speed of execution, extensibility and ease of integration with Maltego.

There are 8 different 'families' of NSE scripts. There are *auth, default, discovery, external, intrusive, malware, safe and vuln*. Two additional ones are 'all' and 'none' – a family name we've added when used with MaltegoTeeth. The attacker can select which of these families should be used, which ports should be scanned and which additional NMap parameters should be used.

The attacker can initiate scans against DNS names (includes web sites, MX/NS records) or IP addresses. Netblocks can be extended to several IP addresses in order to scan these.

Attacks against people - KingPhisher

Introduction

While infrastructure attacks are pretty well known to everyone they are not always very effective. Ask any pen tester what's the most certain way that they'll compromise a system and they are likely to tell you that it would involve people working at the target organization.

People are not machines. People make mistakes and are driven to act in certain ways either through fear, curiosity or greed. Additionally people intentionally open non conventional attack vectors to themselves. This includes email, IM, social networks or even real life devices such as phones or other mobile computing platforms.

In this section we'll look to see if it would be possible to deliver a well crafted email to a target – and doing so with little or no interaction from the attacker. In other words – is it possible to spear phish someone and let Maltego do the heavy lifting. In subsequent updates to the platform we'll also look at other vectors such as IM and social networks.

The goal

The goal of this phishing attack (in order of impact) is as follows:

1. Identify user characteristics from clicked links such as user agent and IP address.
2. Get the target to simply click on a link. We might choose to deliver some form of payload in resulting page.
3. Redirect the user to a fake social network / public webmail (Gmail etc) login page. Hope the target will enter credentials.
4. Redirect the user to a fake corporate web mail / VPN portal and hope to get credentials.

In 3 and 4 we're hoping that the user will re-use credentials and the attacker can use these on more sensitive infrastructure (think corporate VPN etc.) The keyword in scenario 2 is of course browser 0day but getting the external IP address of the user is also valuable.

Already in Maltego

The more information we know about a person the more enticing the email we can send to our target. Maltego already offers many transforms to profile a person. This includes getting the target's email address, social networks and friends from these social networks.

The starting point would likely be the domain of the organization we're attacking. From there we can enumerate email addresses and from that we can resolve social network memberships. In some instances we might be able to make educated guesses as to what the person's name and surname is or add more information about our target manually.

Templates

Maltego creates a graph of these relationships. By sending the graph to an external process we are able to extract these relationships and based on what information we find we can create a list of email templates. In a practical example – from a target domain of *target.com* we find an email address *john.doe@target.com*. The person's name is likely John Doe. We resolve social network membership and find a Flickr profile and a Facebook profile. From the Flickr and Facebook profiles we get interest and friends and connections. Our templates that we can choose from could now be:

1. Spoofed email from Facebook with a new friend request
2. Spoofed email from Facebook with photo tagged from person who is a FB friend of target
3. Spoofed email from Flickr with new photos from a friend
4. Spoofed email from a company specializing in interest X
5. Spoofed email from target.com with instructions

The attacker will be able to choose which template she wants to use, tweak a few configuration options, and the framework will figure out the rest.

Types of redirects / bounces

There are three types:

- Just bounce. Upon clicking on the link the user will just be redirected to another site.
- Get info, then bounce. The framework will collect the user's IP address, user agent and then redirect.
- Get info, phish, then bounce. The framework will collect the IP address and user agent then present the user with a copy of a popular social network's login page, collect the credentials and then redirect the user to the real site.

This give the attacker the flexibility to choose exactly what the target will see once he clicks on a link.

Challenges

The following mechanisms are used to automatically protect users against this type of email:

Sender Policy Framework (SPF)

This is implemented on the receiving SMTP server. The server will check if the remote host matches SPF records for the sender's domain. What this means is that you cannot send email as pete@companyX.com if your connection is not originating from the IP range(s) of company

DomainKeys Identified Mail (DKIM)

Senders of email can decide to implement DKIM, thereby cryptographically signing parts of the message – typically the sender and receiver's email addresses. The means you cannot send email as pete@companyX.com if you don't have their keys.

Filtering, trigger words

Email providers (like Gmail) or email clients (like Microsoft Outlook) have built-in spam and phishing detection and will mark email from our platform as so. They might choose to trigger on emails that are close to the templates of known providers.

These techniques are usually used in combination. The first two protection mechanisms means that you cannot effectively spoof email from major email sources (like Facebook, Twitter, LinkedIn etc). This is not a major problem as you can easily send the email from accounts that *looks* like these providers. In other words – you can send from 'notications@fcebook.com' or 'tweets@twitte.r.com'.

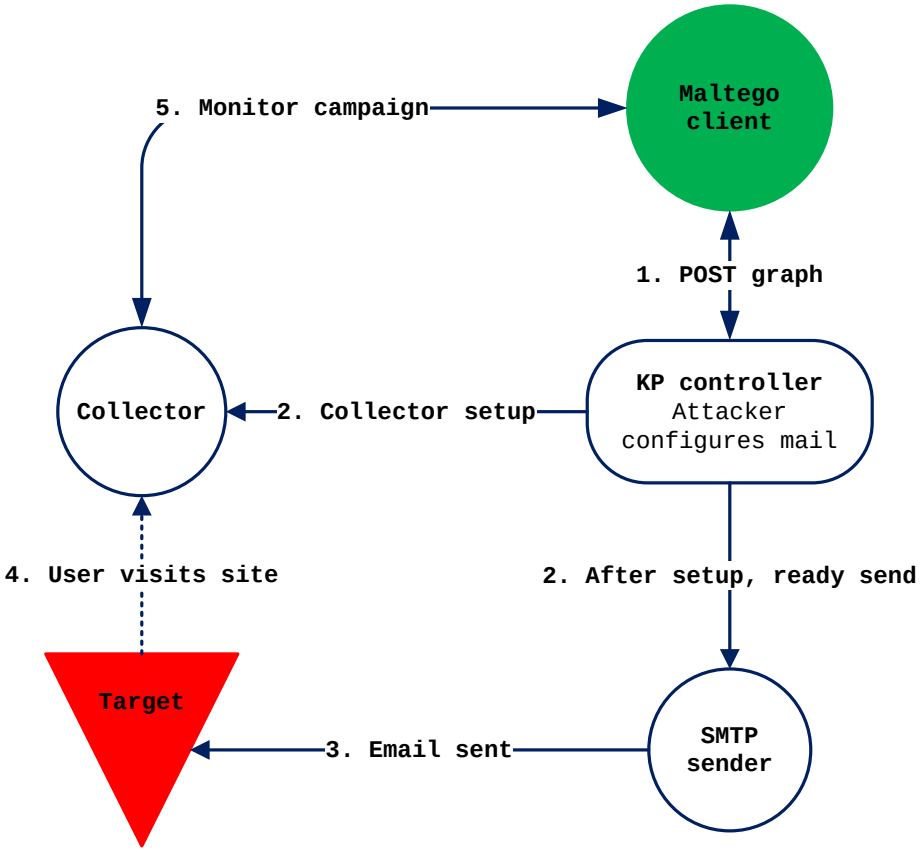
However when the techniques are used in tandem it becomes a little tougher. If you are sending email that looks a lot like that of a major email source but you are not that source it might be marked as spam. The good news is that there are many ways to create HTML that looks that same. The HTML source might not look the same but the rendered output will. This makes it possible for us to send email that looks exactly like the real deal, but on the wire looks a lot different. It's more work but it's worth it at the end.

Managing the campaign

When sending email to many different targets the attacker would want to have a way of managing the 'campaign' as it is easy to lose track of which emails where sent to which targets. With three simple transforms and perpetual machines this can be done graphically within Maltego. The transforms are as follows:

- From control center get the email addresses of targets
- From each target get the IP address and user agent (if it exists)
- From each target get the credentials mined (if any)

The entire system's flow diagram thus looks as follows:



Conclusion

There are several aspects of IT security that have not been touched on in this paper. The goal of this paper is not show new attack vectors or methods. It is to show that almost all methods of attack can be incorporated into a single platform. A platform where attackers can share information, visualize it and launch new attacks collaboratively.

We are painfully aware that 'for the man with hammer, everything looks like a nail'. In our case every security tool or attack technique looks like a nail and our hammer is Maltego Tungsten.