SAFEBREACH
LABS

IN PLAIN SIGHT: THE PERFECT EXFILTRATION

AMIT KLEIN – VP SECURITY RESEARCH, SAFEBREACH

ITZIK KOTLER – CTO AND CO-FOUNDER, SAFEBREACH

THIS RESEARCH WAS FIRST PRESENTED AT HACK-IN-THE-BOX AMSTERDAM ON MAY 27TH 2016

SafeBreach

INTRODUCTION

The last link in the cyber-attack chain is “exfiltration” – that is, transferring stolen sensitive enterprise data out of the enterprise network and into the hands of an attacker. This assumes, of course, that the enterprise was infiltrated (i.e. malware is running inside the internal enterprise network), and the sensitive data was successfully collected. The malware then faces the challenge of sending this data out of the enterprise network, while evading various anti-exfiltration/anti-leaking products/services (e.g. Data Leak Prevention – DLP, various gateways, network monitoring tools, IP reputation based tools, egress filtering, network anomaly detection products, etc.).

In this research we focus on developing **“Perfect Exfiltration”** techniques – techniques that can evade every network-based exfiltration detection/prevention mechanism, even when these are assumed to be perfect as well. To this end, we formulated 10 requirements that Perfect Exfiltration techniques should adhere to. We also focused on methods that do not necessarily yield high bandwidth exfiltration channels, since in strategic enterprises, it’s oftentimes enough to steal few hundreds or few thousands of bits, in order to compromise the enterprise business – think cryptographic private keys, or sensitive information about the company strategy and key personnel that can be compressed into very short messages.

Our research indeed came up with several such Perfect Exfiltration techniques, which we demonstrate and try to counter-attack.

PRIOR ART AND MOTIVATION

The concept of “covert channel” as a means of unobtrusively communicating information out of a restricted network has been around for decades. Extensive research and work has been devoted to this subject, for example:

- “Covert Channels in TCP/IP Protocol Stack (extended version)” by Aleksandra Mileva and Boris Panajotov
- “A survey of covert channels and countermeasures in computer network protocols” by Sebastian Zander, Grenville Armitage and Philip Branch
- “Covert timing channels using HTTP cache headers” by Denis Kolegov, Oleg Broslavsky and Nikita Oleksov

And yet, when we reviewed individual techniques, we were unsatisfied with the quality of the covertness they exhibited.

Example: The IP TOS (Type of Service) field (https://tools.ietf.org/html/rfc1349).
This field was suggested by some of the covert channel research we reviewed as a covert channel. In the covert channel scheme, this 8-bit field can carry up to 8 bits of information. The sending party (inside the enterprise) thus needs to craft an IP packet (with the TOS field populated with the “payload” data) and send it to the destination IP address, where it is read by the receiving party. Since regular traffic monitoring solutions don’t look at the TOS field, the data exfiltration will go unnoticed.

There are several drawbacks to this scheme:

1. Once the scheme is known, a security solution that does look at the TOS field may be implemented. In addition, the anomaly in the TOS field values will be obvious in certain environments; for example, Windows desktops send IP packets with zero TOS.

2. A firewall (or a security gateway) may override the TOS field with e.g. 0, thus eliminating the channel.

3. If the enterprise incorporates IP reputation information in its defense products, the (direct) traffic between an internal machine and an external IP address whose reputation is questionable (or at least, unknown) may trigger an anomaly detection alert.

4. Modification of the TOS field requires special software to communicate with the outside world, which may jeopardize the sending party e.g. if DLP solutions are used.

This led us to formulate criteria for the “Perfect Exfiltration”, presented in the following section.

THE 10 ASSUMPTIONS/REQUIREMENTS AND THEIR RATIONALE

1. A Perfect Exfiltration technique should adhere to Kerckhoffs’s principle (sometimes called Shannon’s maxim), which states that “A [crypto] system should be secure even if everything about the system, except the key, is public knowledge”. In other words - no “security by obscurity”.
Rationale: This requirement ensures that even when the method is known and understood by anti-exfiltration vendors, it is still effective. This also enables multiple channels with different keys, thus facilitating scalability.

2. Only web (browsing) and derived traffic is allowed
Rationale: A lot of enterprises simply block non-web traffic at the gateway or firewall.

3. Whenever there is any doubt, there is no doubt. Anything that may theoretically be perceived as passing information is forbidden. So no emails, no encrypted texts, no google docs, no forum posts, etc.
Rationale: We take a paranoid approach wherein every network packet needs to be positively accounted for.

4. Assume perfect network monitoring, for example, every packet is scrutinized at all protocol levels, every anomaly is detected (big data/machine learning, etc.), reputation and statistics for all IPs/hosts are available.
Rationale: We take a paranoid approach wherein every network packet is scrutinized in real time, both individually and as part of a larger context of network connection, application and user profile. We mustn't assume any shortcoming in any monitoring product. We should assume they're all perfect.

5. Assume TLS/SSL termination at the enterprise level.
Rationale: Covert channels that simply assume they're invisible to network monitoring scrutiny due to using TLS/SSL will get exposed if the enterprise terminates TLS/SSL and inspects the content of the application data.

6. The receiving party has no restrictions (for example, network surveillance)
Rationale: The attacker can choose the external endpoint in a friendly network.

7. No nation-state monitoring or 3rd party site monitoring. However, it's assumed the enterprise does have some basic IDS/IPS/security products, so the exfiltration technique should avoid flooding the 3rd party site with traffic or generate extremely suspicious traffic to it.
Rationale: The reference threat is industrial espionage. As such, it is less interesting for global / nation surveillance entities.

8. Assume that there is time synchronization (seconds-resolution) between the communicating parties
Rationale: It's 2017, most computers are properly synchronized with a reliable time source.

9. Get bonus points for methods that can be implemented manually (without software) at the sender side.
Rationale: There's the use case of an insider (not malware), who mustn't bring in any suspicious software and yet should be able to exfiltrate data (manually).

10. Active disruption by the enterprise is an option.
Rationale: Even if the enterprise can't detect the covert channel, it may still be able to eliminate the channel altogether or disrupt it.

EXAMPLES OF ADVANCED (THOUGH IMPERFECT) EXFILTRATION

In general, advanced methods typically demonstrate how to transmit a single bit. Obviously, longer messages can be transmitted by repeating the technique, varying its parameters as necessary.

An IP ID Covert Channel

The use of IP ID (16 bit field used for defragmentation of IP packets) as a covert channel was proposed by George Danezis in 2008 (**"Covert Communications Despite Traffic Data Retention"**).

The crux of the technique is similar to the "idle scan" concept (**Salvatore 'antirez' Sanfilippo, 1998**). The idea is that the two parties communicate through a 3rd party (non-cooperating) server which generates incrementing IP ID values (for its outgoing IP packets).

A single bit is communicated as follows: "1" is signaled by the sending party communicating with the 3rd party server (and causing it to send back data – for example, an HTTP response - and increment the global IP ID while doing so). "0" is signaled by doing nothing. The receiving party polls the IP ID value generated by the 3rd party server before and after the communication time (by sending the server a request and recording the IP ID value in the server response).

The receiving party can then determine if there's a gap in the IP ID values (indicating that the server communicated with another party in between, that is, the sending party contacted it – thus the bit signaled was "1") or not (In which case the server did not communicate with any client, particularly the sending party, and the bit signaled was "0").

This technique represents a vast improvement over many naïve covert channels, such as the IP TOS field, because it doesn't require direct IP connection between the communicating parties. Rather, a 3rd party popular site (like a news site) can be chosen, as long as the server exhibits the global incrementing IP ID counter property. Using a popular site can address requirement #4 (IP reputation).

This technique may have been relevant back in 2008, but nowadays all major operating systems take one or more of the following approaches for IP ID generation:

- Generate a cryptographic value for the IP ID
- Generate an IP ID value per a sender and receiver combination (different IP ID streams for different combinations)
- Send atomic packets, so per RFC 6864, the IP ID can take any value. Typically the IP ID is zeroed out.

Since each approach eliminates the covert channel, the technique no longer works in up to date operating systems. Additionally, this technique is prone to network noise – since the site is supposed to be popular, the IP ID counter will increment with every response the site sends out, which means it may increment thousands of times per second. This will make detection of the signaled bit a very difficult problem.

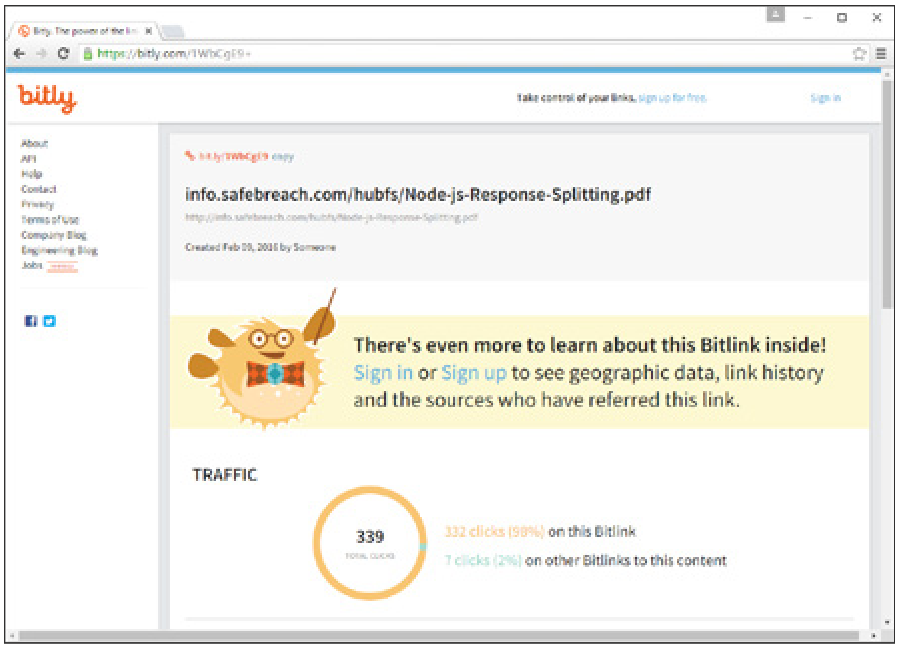
URL Shorteners – Web Counters

This technique is based on URL shorteners, and on the built-in web counter functionality many of them exhibit. We'll take bit.ly as our example, but obviously this can be applied to other URL shorteners with minimal changes.

To implement this technique, the sender and receiver agree on a shortened URL (ideally not an overly popular one). This can be a shortened URL created by a 3rd party (ideally) or a shortened URL created by the communicating parties at an earlier time.

At the designated time, the sender signals the bit "1" by requesting the shortened URL or the bit "0" by doing nothing. The receiver records the number of requests for the shortened URL before and after the communication time. This way, the receiver can determine whether the shortened URL was requested, and hence to extract the signaled bit. Getting the hit count on the shortened URL, in the bit.ly case, is done by appending "+" to the shortened URL.

In Figure 1 (a screenshot showing the response to such a request), the hit count can be seen at the bottom, in the yellow circle.



info.safebreach.com/hubfs/Node-js-Response-Splitting.pdf

http://info.safebreach.com/hubfs/Node-js-Response-Splitting.pdf

Created Feb 09, 2016 by Someone

There's even more to learn about this Bitlink inside! Sign in or Sign up to see geographic data, link history and the sources who have referred this link.

TRAFFIC

339 TOTAL CLICKS

332 clicks (98%) on this Bitlink

7 clicks (2%) on other Bitlinks to this content

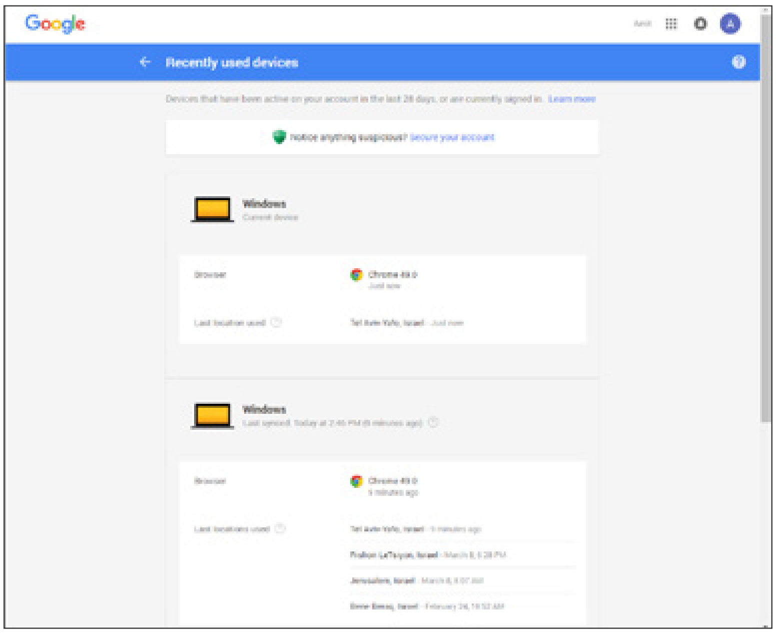
Figure 1

Arguably this method fulfills requirements 1-9. However, there is an effective disruption counter-attack against it, as follows. The organization can intercept all outgoing requests to the bit.ly shortening service at the gateway.

For each new shortened URL the gateway encounters, the gateway needs to send a request to the same URL with a “+” appended. The gateway then extracts the destination URL from the bit.ly response (see in the screenshot above, the destination URL appears in boldface at the top of the page). The gateway then caches the destination URL for the shortened URL, and requests this URL instead of the original shortened URL. Hence the gateway never requests the shortened URL from bit.ly, and therefore the bit.ly counter for the URL is not incremented. This eliminates the covert channel.
Thus this technique fails the 10th requirement.

Account Login Time
Many web mail systems and social network sites display the “last [i.e. previous] login time” upon successfully logging in. This can be used to define a covert channel, based on whether the sender is logged in at a predetermined time, or not. That is, the sender and receiver first share account credentials. The sender logs in at a predetermined time to signal the bit “1”, or stays logged out for the bit “0”. The receiver logs in shortly afterwards and observes if the last log in was in the predetermined time or not.

In general this approach is a bit problematic, because it may lack scale (there’s a limit to how many accounts a single person may own, without raising suspicion), and it may also violate requirement #3, since it may be difficult to explain why the sender keeps logging in to the account without doing anything (and doing something may again reduce the scalability). However, there’s a more serious drawback. As can be seen in Figure 2 and Figure 3, such services typically have security pages which show the recent log ins to the account (Figure 2 shows Gmail’s security page, Figure 3 shows Yahoo Mail’s security page):



Recently used devices

Devices that have been active on your account in the last 28 days, or are currently signed in. Learn more

Notice anything suspicious? Secure your account

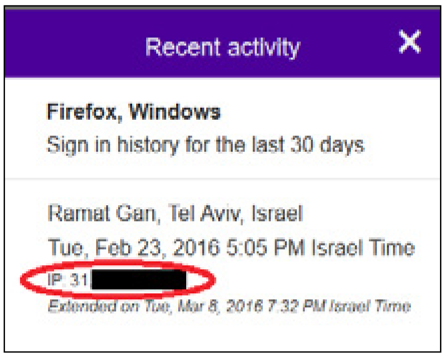
Windows
Current device

Browser — Chrome 49.0, Just now
Last location used — Tel Aviv-Yafo, Israel - Just now

Windows
Last synced: Today at 2:46 PM (9 minutes ago)

Browser — Chrome 49.0, 9 minutes ago
Last locations used — Tel Aviv-Yafo, Israel - 9 minutes ago
Rishon LeTsiyon, Israel - March 8, 6:28 PM
Jerusalem, Israel - March 8, 8:07 AM
Bnei Brak, Israel - February 26, 10:52 AM

Figure 2


Recent activity

Firefox, Windows
Sign in history for the last 30 days

Ramat Gan, Tel Aviv, Israel
Tue, Feb 23, 2016 5:05 PM Israel Time
IP 31[redacted]
Extended on Tue, Mar 8, 2016 7:32 PM Israel Time

Figure 3

Showing the recent log ins may expose the fact that there are log ins (of the receiving party) from other locations and browsers, and (in Yahoo Mail's case) even expose the IP used by the receiving party. Of course, the sending party should not request these pages, but since it is assumed that all the traffic is monitored, including TLS/SSL, we should assume that the enterprise has access to the account credentials, and thus the enterprise can log in and observe the recent activities at any given time.

Another drawback of this proposal is that some enterprises may block all social sites and web mail systems completely.

PERFECT EXFILTRATION

All the Perfect Exfiltration methods we have developed are based on regular (HTTP/HTTPS) web browsing. The main idea behind all the techniques is that the sender modifies the state of the application (or a higher protocol) – hence the techniques belong to the "storage channel" sub-category of covert channels. This state change is then observed remotely by the receiver.

Cache or Fresh

This method exploits the fact that caching mechanisms may "leak" the exact time an object was cached. For example, an HTTP cache in front of an application engine may respond with the following HTTP response headers when the web page is introduced into the cache (assuming the object's time-to-live in the cache server is 5 minutes):

```
HTTP/1.1 200 OK
Expires: Mon, 22 Feb 2016 14:25:24 GMT
Date: Mon, 22 Feb 2016 14:20:24 GMT
...
```

Observe that the Date header (which is approximately the current time) is precisely 5 minutes earlier than the Expires header.
On the other hand, if we request the same page at a later time (9 seconds later, in the below example), we will get the following response:

```
HTTP/1.1 200 OK
Expires: Mon, 22 Feb 2016 14:25:24 GMT
Date: Mon, 22 Feb 2016 14:20:33 GMT
...
```

Notice that the Date header indicates a time which is 9 seconds later than the Expires time minus the time-to-live (5 minutes). Thus we know that the page in question was cached 9 seconds before this response was sent.

In other words, by examining the HTTP response headers from the cache (and assuming we know the time-to-live used by the cache for its objects) we can tell when an object was cached. Particularly, we can tell if the object was loaded into the cache because the request was just sent, or whether it was already cached (and when).

We use this observation to construct a covert channel. The sender and receiver agree on a URL (a product page in an e-commerce site is ideal, so long as the product is not too popular) and a time. At the designated time, the sender either requests the URL (signaling the bit “1”) or does nothing (signaling the bit “0”).

Few seconds after the designated time, the receiver requests the same URL, and observes the HTTP response headers. If the expiration time matches 5 minutes (or whatever time-to-live the e-commerce cache enforces) after the designated time, the receiver infers that the sent bit is “1”, but if the expiration time matches 5 minutes after the time the sender sent its request, the inferred bit is “0”.

We tested this method on several prominent e-commerce sites depicted in Figure 4.

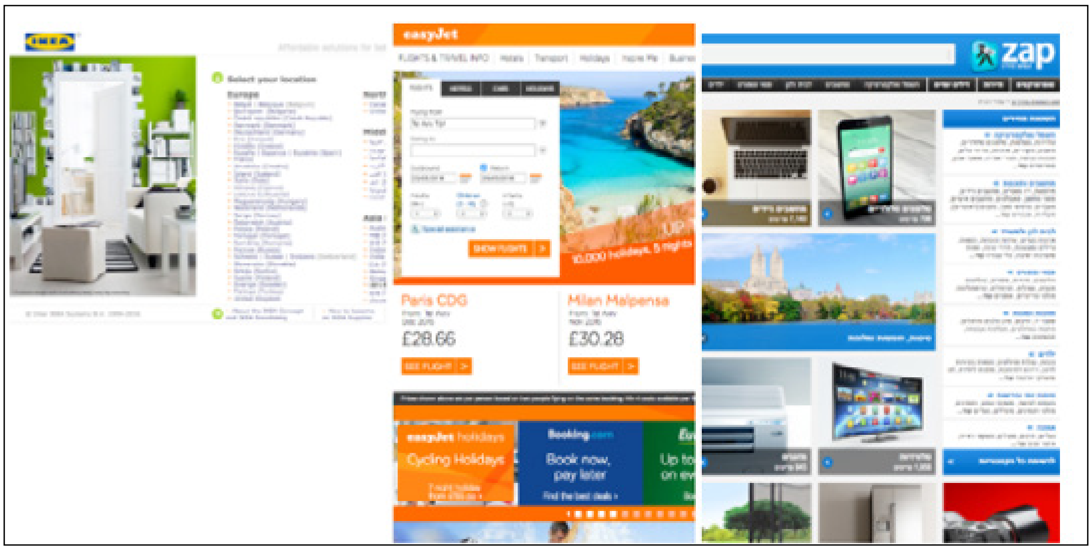


Figure 4

It should be stressed that this method does not expose or exploit any security vulnerability in the 3rd party (e-commerce) sites involved.

There are some fine points worth mentioning:

- **Protocol resilience:** in order to withstand network outages and other disruptions, it is recommended to implement an error correction layer on top of the raw data bits. This can be CRC or similar techniques. Another approach is to simply double the bits in a way that withstands simple network outages: send two physical bits “10” to signal a single logical bit “1”, and send two physical bits “01” to signal a single logical bit “0”.
- To simplify the discussion, the technique described involves hitting a single URL, but in order to make sure this traffic doesn’t stand out, the sender must include this URL as part of a browsing session in the e-commerce site starting from a realistic landing page, and getting to the desired URL in the same way a regular user would (for example, searching for the specific product or browsing product categories).
- Likewise, the sender must engage in several other legitimate web browsing sessions, to make sure the e-commerce session doesn’t stand out.
- This technique may face a challenge if multiple cache servers are used by the e-commerce site, and/or if geo-IP based content caching is employed.
- This technique is also sensitive to site changes, which is why it should be assumed that the URL list for communication can be updated, for example, by a downlink from the receiver to the sender.

- **Delay considerations:** the amount of time (delay) the receiver should wait after the designated communication time before it sends its own request is a matter of fine-tuning according to various considerations:

    - The longer the delay is, the better the technique can handle:

        - Time drift in both the sender and receiver

        - Request-response round trip for both the sender and the receiver (TCP, TLS, HTTP)

        - Possible retransmissions due to errors or outages

        - "Delay at the gateway" counter-attack (see below)

On the other hand, the shorter the delay is, the less it exposes the technique to errors due to someone else's possible request for the same URL. We expect real-life delay to be in the range of 5-30 seconds.

An (ineffective) counter-attack against this technique ("delay at the gateway") can go along the following lines. Let's assume the technique's delay is 10 seconds. The enterprise gateway delays each HTTP/HTTPS request by 11 seconds and also observes the response HTTP headers. This accomplishes both disruption and detection. Disruption is due to the fact that the sender's request arrives after the receiver's request, thus the receiver will always read "0". Detection is due to the gateway observing that the resource was very recently cached.

However, this counter-attack suffers from critical drawbacks:

- Introducing a long (11 seconds) delay to each and every HTTP request inflicts insufferable user experience pain to all the enterprise users.
- Detecting a page that was recently cached doesn't necessarily mean that this is part of a covert channel. In other words, this detection has an absurdly high false positive rate.

We implemented this technique as an open source tool called CacheTalk.
This is a proof of concept tool designed to illustrate the above method and facilitate further studying of it. The tool is implemented as a single Python script. Two instances of the script need to be run – one is a sender and the other is a receiver. They interact by sending HTTP requests (hitting URLs) in a 3rd party site. Further information can be obtained by reading the "README.md" file and invoking the script with the "-h" parameter (for "help").

SUMMARY AND ADDITIONAL RESEARCH

We have demonstrated that Perfect Exfiltration techniques exist and are practical to implement. During the research, we also came up with a list of requirements we deem necessary (and probably sufficient) for Perfect Exfiltrationtechniques. The techniques we developed facilitate very low bandwidth communications. As such, they give rise to two additional research questions:

1. Are there Perfect Exfiltration techniques that facilitate high bandwidth communication?
2. Are there counter-attacks against the techniques we developed and described in this paper?

These are challenges for us, as well as for other researchers, and we hope to see more research formulating a robust and concrete list of requirements for Perfect Exfiltration, and a more formal model for attacks against Perfect Exfiltration.