USB Devices Phoning Home

Roland Schilling*
Hamburg University of Technology
schilling@tuhh.de

Frieder Steinmetz*
Hamburg University of Technology
frieder.steinmetz@tuhh.de

February 10, 2016

Abstract

USB is a versatile standard defining various features to allow maximum flexibility for devices. This flexibility, by design, leads to complex device configurations, combining multiple functions into one, making it impossible for users to identify the function of a device by its looks. This can be exploited by crafting programmable USB devices, looking and behaving like an ordinary flash drive that also expose virtual network devices and other functionality to their host OS. This paper outlines such a device, exploiting several USB features to establish a rogue HTTP channel used to leak data stored on the device's disk to an internet back end. We describe the device itself and its architecture and our conclusions and methods for dealing with the issues presented in a user–friendly way.

Introduction

We present a USB flash drive capable of uploading all files stored on it to a remote host on the Internet. The device does not have its own Internet uplink and uses its victim's web browser to establish a rogue channel while she is browsing. We will present several ways of establishing such a channel hidden from the victim, including theoretical ones that we found not to work reliably or not to work at all on a modern operating system.

While presenting a very technical view on our implementation of the attack, we would like to emphasize the underlying problem, a connection standard for electronic devices in which peripheral devices announce their function to the operating system. Concepts such as *compound* devices in USB result in an ecosystem in which users can never be sure to which extent their expectations of a device are met. Instead of explicitly authorizing specific functions of a USB device, users implicitly authorize whatever functions a device advertises, simply by plugging it in. In consequence, there can be a mismatch between the user's expectations and the functions that have been authorized.

USB has become the de facto standard for almost all computer periphery. It is used for input devices, storage devices, printers, network devices, sound devices, web cams, and more. This flexibility comes at the cost of a complex architecture where operating systems have to be able to accept newly attached devices on the fly and the devices themselves are responsible for communicating their purpose and capabilities. These devices often are programmable which adds an additional layer of complexity. The USB enumeration process in which a device announces its features to the operating system is completely hidden from the user. The actual behavior of a device only becomes apparent once it is plugged in and fully enumerated, making it impossible to guess the function of a device from its looks. To an educated user it might seem plausible that a smart phone plugged into a PC has an Internet uplink of its own that can be used to redirect the network traffic or leak data of the system it is plugged into. If, however, the device plugged in is a simple flash drive, such functionality might not be expected. It is this discrepancy between user intent and device behavior that we focus on with the flash drive presented here. In the following sections we will present the setup in detail and show that it is a feasible albeit less reliable approach than a device with a dedicated uplink.

Related Work

The device we describe breaks with fundamental assumptions for storage devices. It demonstrates

*Authors listed alphabetically

that modern storage devices are general purpose computers of their own and have the potential to be used for things far beyond their original purpose. This potential of USB storage devices in particular was demonstrated by Mulliner et al. when they emulated a storage device on a gumstix board and used it to exploit a Time of Check to Time of Use (TOCTTOU) bug in a smart TV [5]. Travis Goodspeed presented deeper insight into the capabilities of custom USB storage devices in his talk at the 29c3 conference in 2012 [3]. Part of his findings were different operating systems implementing USB protocols distinctively enough to be uniquely identified. Andy Davis of NCC Group did an in–depth exploration of USB fingerprinting and found numerous ways to reliably identify a host OS or even installed applications [2]. A more offensive approach which could in turn make use of in–depth knowledge of the host configuration is emulating a keyboard and sending keystrokes to compromise the host system. This attack has been known since around 2010 [1] and has even found its way into commercial products [4].

One of the most prominent works in this field is BadUSB, presented by Nohl et al. in 2014 that summarizes several projects from other groups into a few concise scenarios [6]. Beside its core idea, the talk describes seemingly harmless USB devices that emulate keyboards and issue commands on behalf of their victims, or smart phones, connected to a PC that could use their own Internet uplink to redirect a host's network traffic hidden from its victim. Nohl presented two case studies using USB network device emulation; one consisting of a DNS server assigned by a USB device via DHCP, the other being a simple network device rerouting all traffic through a smart phone Internet uplink. His team used an android phone advertising itself as DNS server via DHCP and rerouted a list of domains. It resolved all DNS requests for domains not on that list by recursively asking another name server.

Implementation

Given the versatility of USB and the fact that most computers today are connected to the Internet already, we started looking for a way of exploiting the host uplink with a common USB virtual network device and established technology like DNS, DHCP, and routing. Our attack vector capitalizes on the fact that today's web sites often load content and scripts from several different locations, which is mostly invisible to the user. More to the point we focus on web tracking and analytics services which provide no benefit for the user and run in the background of a browser session. Removing or replacing such a script would not impact the browsing experience and yet offer a way of injecting arbitrary JavaScript into the victim's browser sessions. We found that many of these services use plain HTTP, which makes it easy to reroute and modify their traffic. In the following we will present two approaches for exploiting a victim's browser session to upload files stored on our flash drive using different hidden communication channels.

The setup of our attacks is simple. A USB flash drive will expose both a removable disk and a network device to its host OS. This is possible because the USB standard defines a *composite* device that uses one bus address to encapsulate the functionality of multiple virtual devices called *functions*. For more information on this, please refer to [7], or [8]. The network device runs a DHCP and – depending on the attack – a DNS server. The DHCP server is used to push a set of static routes to the host OS. These routes instruct the OS to use the flash drive as gateway for our target hosts. This setup is detailed in figure 1.

The DNS server can be used to spoof DNS records and redirect requests to arbitrary hosts. Because different operating systems handle newly registered DNS servers differently, this attack cannot be reliably mounted in all situations. Our tests with Linux and Mac OS X showed that DNS servers added after the network has been set up are added with the lowest priority so they are only used if all other resolvers fail. On Windows 8 and 8.1 we were surprised to see that a newly added DNS server becomes the immediate default, making these versions the easiest target operating systems for the attack.

In addition to the DNS and DHCP servers, the drive runs a web server that serves all files necessary to establish outside communication. Rerouting host requests to this web server allows us to inject our own JavaScript code into the victim's browser sessions as explained shortly.

All our attacks were implemented on a USB Armory by Inverse Path[1], a flash drive-sized ARM mini computer running Linux. The drive was configured to behave as a regular USB flash drive using the Linux

[1] `http://inversepath.com/usbarmory`

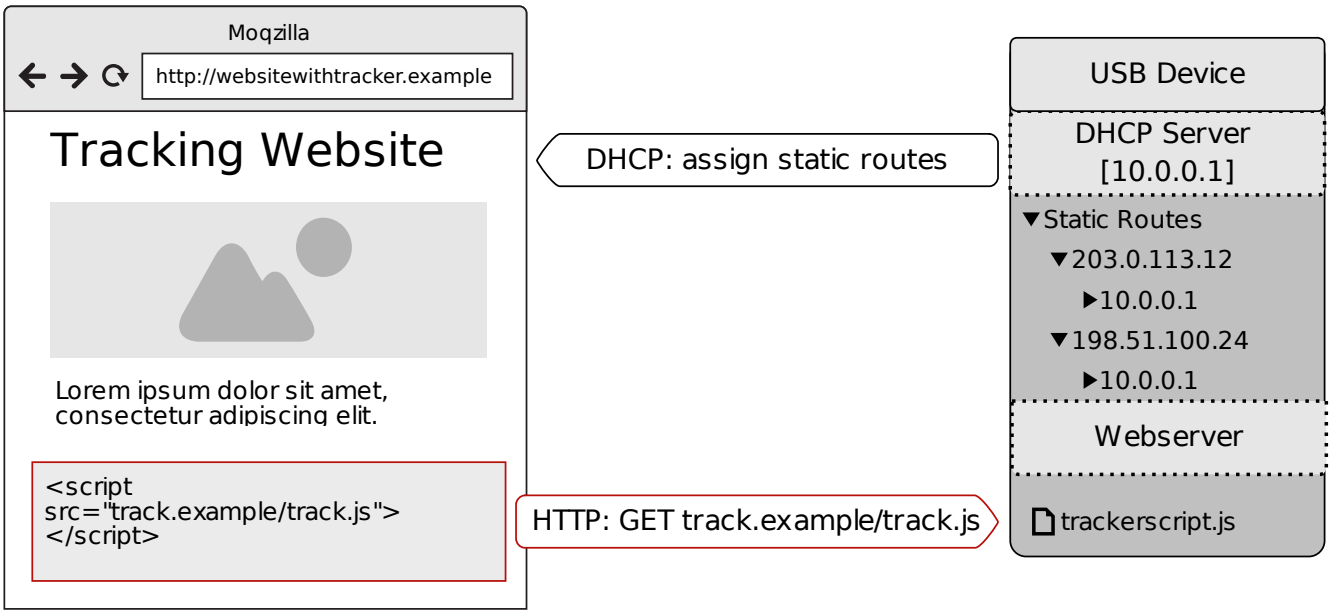

Figure 1: Injecting code in a browser session on a website using a popular web tracking service: static routes for the IP addresses of the tracker are used to redirect HTTP requests to the USB device. Those requests were originally meant to load the web tracker's JavaScript code. Since they are now routed to the USB device it can serve the C&C script instead.

USB Gadged API[2]. When plugged into a host machine, the drive will behave like a USB storage device and its user will find a removable disk she can read from and write to. At the same time, a new network device will be configured. This happens in the background on all tested operating systems. Once setup is done, storage data from the flash drive can be leaked to a host on the Internet using one of two approaches, depending on the host OS.

As stated, we target this attack on web tracking services. These services are included by websites for metering and advertising purposes. They do not provide a benefit for the user and their operation is completely hidden from her. We prepared a list of both domain names and IP addresses of popular tracking and advertising services. We concentrate on services that can be used without HTTPS to avoid running into validation issues when presenting possible fake certificates. This list is stored on the hidden operating system of the flash drive. The DHCP server on the drive will publish a route to each of these IPs via its own network device. The DNS server is configured to resolve each domain name on the list to the IP address of the web server running on the USB device. After setup is completed the target OS will try to resolve all prepared tracker domain names via the DNS resolver on the flash drive and try to establish a connection to these hosts via its published gateway. This is the base setup for this attack. All this is performed within the first seconds after the device is plugged in and the setup is ready once the removable disk is enumerated.

As soon as the victim visits a site that includes a tracker, the browser makes a request for the tracker's JavaScript file. If the tracker is part of the pre-compiled list and works without HTTPS, the request is redirected to the USB device as explained earlier. The web server on the USB device will respond with a prepared JS file that will be executed in the browser and establish a C&C channel between the drive and a back end control server using a simple REST API as depicted in figure 2. The JS code periodically polls both servers for data using AJAX requests and will download and forward data from and to each side respectively. A service on the USB device monitors files stored on its disk and prepares them for its web server to publish to the relay script. Files stored on the removable disk of the device can now be transferred to the back end server and modified data from the back end server can be placed back on the disk. The back end server can be a simple web server accepting POST requests with file data in the body, or a more complex setup

[2] http://www.linux-usb.org/gadget/

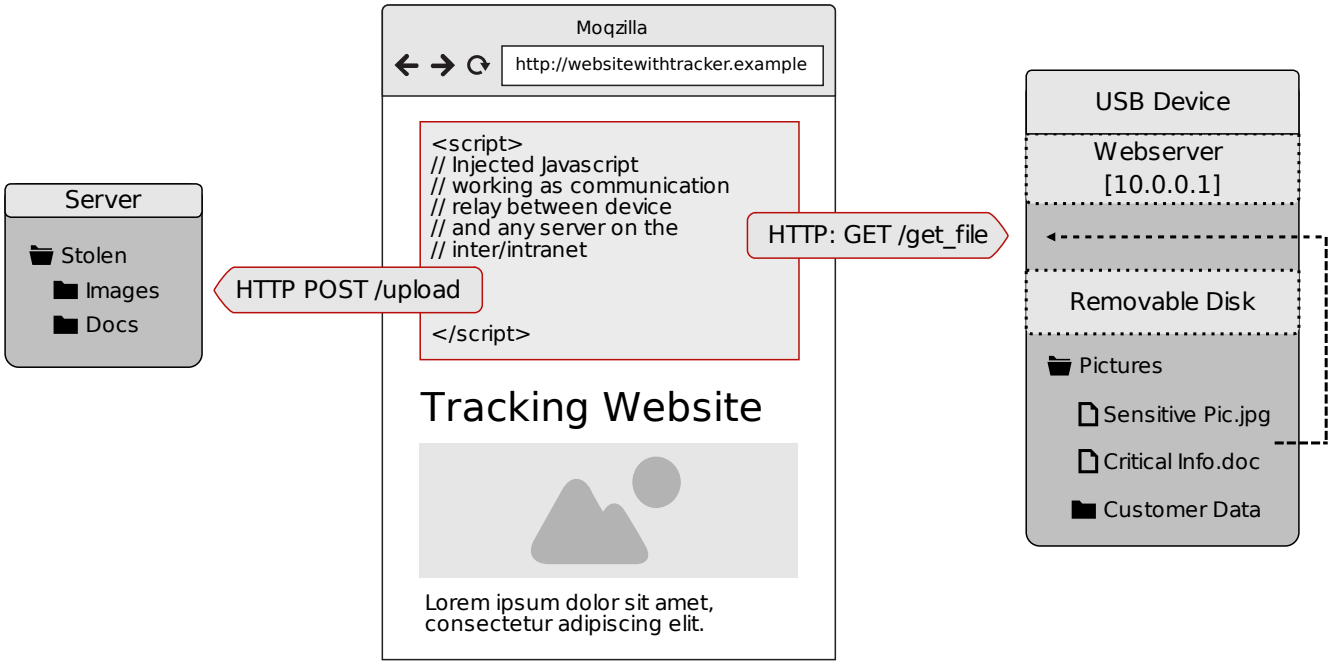

Figure 2: Transfer files from the device to the C&C-server: a script has already been injected in the currently open website. This script uses AJAX to get files from the device's web server and upload them to the back end server. The device monitors its removable disk for new files and enqueues them for transmission.

capable of infecting received files with malware and offering them to be transmitted back to the drive. The return channel can also be used to update the list of ad network IP addresses on the drive, adding a bit more flexibility to this static approach. This way the list of IP addresses can be kept reasonably up–to–date by resolving the whole list on the back end and having the USB device poll it in fixed intervals.

A more elegant solution than rerouting requests to a static list of IP addresses is to intercept DNS requests for their respective domain names and resolve them to the IP address of the web server running on the USB device. This can be achieved by assigning the drive as DNS server via DHCP. We wrote a custom DNS server based on GoDNS[3] for this purpose that resolves domain names from our list of trackers and ad networks to the IP address of the USB device and responds to other queries with a server error that will make the host OS automatically fall back to another DNS server from its list. This automatic fallback – while introducing a slight latency – works reliably on all tested platforms. However, in our tests with several flavors of Linux, Windows in versions 7–10, and MAC OS X 10.9, only Windows 8 and 8.1 accepted the new DNS server as their new default resolver. All other tested operating systems would add a new resolver to the end of their chain, rendering this approach ineffective. Only a few corner cases exist despite that, where the DNS method still works. If the target OS is configured to use a popular third party DNS service as offered by Google or OpenDNS, we can make this scenario work on other platforms as well by pushing static routes to these hosts via the USB device. On all other tested platforms without these corner cases, the static–route–approach with frequent updates to the list of IP addresses is the more reliable solution.

Both presented solutions have shown to work reliably from an attacker's point of view. Pushing routes to IP addresses from a static list via DHCP comes with obvious disadvantages but proves to have the least impact on the performance of the target OS. On the systems that accept the propagated DNS server as default we see a little latency whenever a domain name not on the list is resolved by the OS. This is due to our solution of responding with an error message and hence making the system fall back to another DNS resolver.
Overall both solutions had the same effect of a reliable rogue HTTP channel whenever a web tracker

[3] `https://github.com/kenshinx/godns`

from the list was included in a web session.The fact that this attack exploits features of USB and that hardware is generally trusted by a modern OS makes this possible without the user taking notice of any abnormal behavior.

Conclusion

We present a device that exploits the features of common USB Plug and Play architectures. No implementational bugs were necessary to create a device with capabilities far beyond its apparent purpose. It appears to be a common flash drive but uploads all files stored on it to a remote server, without having an uplink of its own. The setup happens mostly invisible to the user and works reliably on all tested platforms. Since it hijacks a browser session it relies on someone actively using the target system. This is however something that can be assumed for the most likely targets of such attacks being workstations.

The problem that allows our attack to succeed lies within the handling of attached USB devices. To guarantee seamless Plug and Play modern OS immediately load drivers according to the self–proclaimed description of the device. Thereby – depending on the driver – granting it access to certain aspects of the host system. The self–description may however not reflect the form factor – which is the only indication a user has of the nature of the device. At this point the host might have exposed its complete network stack to a device looking like a flash drive or keyboard. Finally the lack of any authentication in the DHCP protocol lets anyone on the same network alter important aspects of a systems network configuration. The combination of those intended features expands the possibilities for USB devices to a certainly unintended extend.

Preventing this by changing the USB Plug and Play behavior may be done by asking the user for confirmation before loading a driver as proposed by Steinmetz in [7]. A concise question whether the type of device that presented it self to the Operating System is what the user intended to connect should suffice. Since USB periphery has a clear set of device classes, even an uneducated user should be able to answer the question whether she wanted to plug in a keyboard, mouse, removable disk, network device, or something else. Finally, as users expect something to happen on their screen upon device plug–in, this dialog should not be perceived as much of a nuisance.

All of the code we wrote to build the proof of concept can be found in our GitHub repository[4]. Our version of this attack is based on the USB Armory, a €100 programmable drive by Inverse Path. However, most of the versatility of this drive is not needed to successfully mount the attack. Therefore it is reasonable to assume that this can be realized a lot cheaper on dedicated hardware; possibly even reprogrammed actual flash drives as demonstrated in the BadUSB attack [6].

References

[1] Adrian Crenshaw. Programmable HID USB Keyboard/Mouse Dongle for Pentesting. *DEF CON*, 18, 2010. https://www.defcon.org/images/defcon-18/dc-18-presentations/Crenshaw/DEFCON-18-Crenshaw-PHID-USB-Device.pdf.

[2] Andy Davis. Revealing Embedded Fingerprints: Deriving Intelligence from USB Stack Interactions. Technical report, NCC Group, August 2013. https://media.blackhat.com/us-13/US-13-Davis-Deriving-Intelligence-From-USB-Stack-Interactions-WP.pdf.

[3] Travis Goodspeed. Writing a Thumbdrive from Scratch, December 2012. https://events.ccc.de/congress/2012/Fahrplan/events/5327.en.html.

[4] HAK5. USB Rubber Ducky - The Original Keystroke Injection Tool, November 2014. https//www.usbrubberducky.com.

[5] Collin Mulliner and Benjamin Michéle. Read It Twice! A Mass-Storage-Based TOCTTOU Attack.

[6] Karsten Nohl and Jakob Lehl. BadUSB–on accessories that turn evil, 2014. https://srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf.

[7] Frieder Steinmetz. USB – An Attack Surface of Emerging Importance. Bachelor's Thesis, Hamburg University of Technology, March 2015.

[8] USB Implementers Forum Inc. Universal Serial Bus Revision 3.1 specification. In *USB Implementers Forum, Inc*, July 2013. https//www.usb.org/developers/docs/usb_31_121314.zip.

[4] http://www.github.com/willnix/usbpoc