Back To The Basics: Security of Software Downloads for Smart Objects

Alberto Bartoli
University of Trieste
Trieste, Italy
bartoli.alberto@units.it

Eric Medvet
University of Trieste
Trieste, Italy
emedvet@units.it

Andrea De Lorenzo
University of Trieste
Trieste, Italy
andrea.delorenzo@units.it

Fabiano Tarlao
University of Trieste
Trieste, Italy
ftarlao@units.it

ABSTRACT

Smart objects will soon pervade our homes, cities, factories, plants, and hospitals and this fact will introduce widespread important risks for the society as a whole, due to unavoidable security vulnerabilities of those objects. The problem of updating the software of smart objects in order to fix vulnerabilities will thus become of crucial importance. In this work we investigate the security of current software download environments for smart objects. This investigation allows gaining important insights into the security awareness of organizations that distribute software across the web and, more broadly, on their readiness to take control of our everyday life.

CCS CONCEPTS

• **Security and privacy** → *Vulnerability management*; *Network security*; • **Social and professional topics** → *Malware / spyware crime*;

KEYWORDS

Internet of things, software updates, network attacks

ACM Reference Format:
Alberto Bartoli, Eric Medvet, Andrea De Lorenzo, and Fabiano Tarlao. 2018. Back To The Basics: Security of Software Downloads for Smart Objects. In *International Conference on Smart Objects and Technologies for Social Good (Goodtechs '18), November 28–30, 2018, Bologna, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3284869.3284885

Goodtechs '18, November 28–30, 2018, Bologna, Italy

ACM ISBN 978-1-4503-6581-9/18/11…$15.00
https://doi.org/10.1145/3284869.3284885

1 INTRODUCTION

Internet of things, smart objects, distributed sensing are enabling radically new applications and introducing unprecedented opportunities in a number of different application domains. Security researchers and practitioners are looking at these technologies from a different point of view, though, which can be expressed very simply and succintly with the so-called Hypponen's Law: *"Whenever an appliance is described as being smart, it is vulnerable."* [17]. There are already countless examples of smart objects that can be completely controlled by a remote attacker easily, because of design or implementation mistakes (i.e., *vulnerabilities*) in those objects: light bulbs [20], water heaters [15], dishwashers [10], sex toys [7], pacemakers [29], door locks [32], cars [6, 25] just to mention a few of them.

The security problems of smart objects will not disappear any time soon, due to a combination of technical, economic, and regulatory issues [3]. The potential risks for the society as a whole in a not too far future can thus be imagined easily. Once homes, cities, factories, plants, and hospitals will be full of network-connected objects capable of acting not only on data but also on the physical world, any vulnerability in those objects will constitute a potential entry point for attackers that can be located anywhere in the world, ranging from vandals, to terrorist groups, to state actors.

While coping with these problems is very difficult and we are only starting to understand their ramifications [28], there is one single technical fact on which virtually all security researchers and practitioners agree: the fundamental importance of *updating the software* [3]. A smart object whose software is not, or cannot be, updated is a very dangerous device: in the event a remotely exploitable vulnerability is found, that object will become a permanent entry point for an attacker. The experience has amply demonstrated that

virtually every device will fall into that category, sooner or later.

Updating the software is a difficult problem in general and may be even more difficult for smart objects, leaving aside that many such objects are not designed for allowing their software to be updated. Assuming that, in a hopefully not too distant future, the availability of software updates for smart objects will be the norm, the problem will become how to transfer such software securely across the web.

This problem is a very important one: an entire society pervaded by smart objects and that routinely downloads software from the web is a very attractive target for a terrorist group or a state actor capable of posing as a network attacker. Since the basic web protocol (HTTP) does not provide any guarantee of server authentication and message integrity, a network attacker may have an easy job of replacing the real executable with a malicious one. Indeed, network attacks on HTTP transactions can be executed so easily that many organizations are now promoting usage of HTTPS (HTTP over an encrypted channel providing guarantees of authentication, integrity, and secrecy) by default [8, 14, 27]. Although HTTPS does not neutralize network attackers completely, its usage make those attacks much more difficult and more costly, especially if attempted on a large scale.

In this work we investigate a very specific but fundamental question: is software actually distributed securely over HTTPS, or is it distributed insecurely over HTTP? We are not aware of any similar study. This investigation allows gaining important insights into the security awareness of organizations that distribute software across the web and, more broadly, on their readiness to take control of our homes, cities, plants, hospitals, and so on.

One of the key reasons that triggered our interest in the research presented in this work was the observation that several web sites, including high-profile ones, distribute software on HTTPS, but serve the web page for choosing the software of interest on HTTP. In other words, the web site is concerned with network attacks (because the download link is on HTTPS); yet the web site is concerned with network attacks *only* on the download link but *not* on the download page (because this page is on HTTP). This defensive approach is clearly very odd: a network attacker capable of attacking either authentication or integrity of the software download is obviously capable of executing such an attack on the download page as well [26]. It is thus trivial for the network attacker to force the browser, for example, to download an executable from an attacker-controlled URL different from the genuine executable URL. In other words, ensuring authentication and integrity of the executable but not of the page from which the link to the executable is obtained, makes very little sense and, unfortunately, many web sites take exactly this approach.

2 THREAT MODEL

We consider attackers whose goal is to alter the software downloaded by users from the web. To this end, we assume the attacker may passively observe network traffic between the browser and the web server, as well as modify the traffic sent by the server. In particular, the attacker may modify the links contained in a web page fetched through HTTP as well as the file transported by an HTTP response [26]. Such actions are sufficient for delivering an attacker-chosen software to the user platform.

We assume that the authenticity and integrity of the downloaded software is not verified. This assumption does not hold for several widely diffused systems, e.g., the Windows operating system, the Chrome browser, apps downloaded from stores of major operating systems for mobile platforms. On the other hand, such an assumption is realistic in many cases, in particular, for software meant to be executed on a small appliance or smart object. Of course, if a device has the ability to verify the authenticity and integrity of the downloaded software, and this ability is configured and used correctly, then the device will refuse to install the attacker-controlled software. We note, however, that there are many ways for an attacker to circumvent such defense [18].

We assume that attackers cannot corrupt or control the user platform[1] or the server infrastructure. We also assume that attackers cannot break HTTPS, thus HTTPS communication is guaranteed to have secrecy, integrity and server authentication guarantees. In the following we shall refer to attackers as *network attackers*.

Attacks in our threat model can be executed on a local scale easily, for example, with a fraudulent Wi-Fi access point or by taking control of an home router. There is plenty of evidence of such attacks executed on a global scale as well, though, for surveillance purposes [12] or for injecting scripts to be executed on browsers [2, 11]. Such attacks may be executed, for example, with vantage points on Internet providers or by injecting fake routing messages in the BGP routing system. Attacks of this sort are feasible by state-level actors but are believed to be feasible by criminal groups as well, as indicated by recent cryptocurrency frauds based on fake BGP routing messages [16, 24]. We remark that attacks of this sort are particularly dangerous as they cannot be detected with monitoring infrastructures aimed at assessing the server reputation [1, 13, 19] or at detecting anomalies in the content of the website [4, 5, 9].

The only analysis similar to ours that we are aware of, was concerned with the security of login pages in the most visited web sites according to the Alexa ranking [31]. The cited study considered network attackers even stronger than

[1] Of course, attackers could take control of the user platform *after* an attacker-chosen software is installed.

Table 1: Resource download configuration and corresponding security levels (see text).

Configuration	Security Level
HTTP only	Insecure
HTTPS only	Secure
HTTPS, HTTP redirected to HTTPS	Partly Secure
HTTP, HTTPS	Partly Secure
Executables not publicly available	Unknown

ours, i.e., with the ability to break HTTPS communication based on fake HTTPS certificates obtained from a certification authority trusted by user platforms. We observe that the potential risks involved in altering a downloaded software are arguably much deeper than those associated with credential stealing.

3 METHODOLOGY AND DATA SELECTION

We analyzed almost 200 *web-based software download environments*, i.e., of web sites that allow downloading an *executable resource*. An executable resource (an *executable*, in brief) models the software or a software upgrade for a smart object, i.e., a file that can be executed on a suitable platform: a driver, a firmware, a development environment, or an upgrade for any of them.

The crucial element for characterizing the security of a download environment is the protocol for downloading the download page and the protocol for downloading the executable. As it turned out from our analysis, there are several possibilities for each of the two cases as summarized in Table 1 (we use the generic term "resource" for indicating either the download page or the executable). The most secure configuration is HTTPS only and we chose to label this configuration as "Secure". Configuration HTTP only can be attacked by a network attacker easily. Configurations in which the resource can be accessed via HTTP, either directly or after an automatic redirection to HTTPS, are equivalent to HTTPS only when the user follows an HTTPS link, while they are equivalent to HTTP only when the user follows an HTTP link. Whether a user will follow an HTTP or an HTTPS link depends on how the user obtains the link, e.g., through the result page of a web search engine, or by typing directly the URL in a browser, or by following a link in a (potentially malicious) email, for example. Thus, we chose to label these configurations as "Partly Secure".

For several download environments we could not find any download page, which may correspond to very different security approaches. On the one extreme, the manufacturer could not have any plan for distributing software updates. While this choice makes the security of the software download protocol irrelevant, this is clearly the worst possible choice for the security of the corresponding smart object. On the other extreme, the manufacturer could distribute software updates through channels different from the web and arguably more secure, e.g., automatic device update (as done by NEST thermostats), updates executed in a tightly controlled environment (as in the case of pacemakers), updates distributed through memory supports sent via surface mail). Thus, we chose to label the security level of this configuration as "Unknown".

Clearly, the most secure architecture for a download environment is the one in which *both* the download page and the executable are configured as HTTPS only. Any other architecture provides little protection, if any, in the considered threat model: a network attacker capable of attacking either authentication or integrity of an HTTP transaction may easily redirect the browser toward an attacker-controlled URL, thereby delivering on the client platform an attacker-chosen executable—an event that may have catastrophic consequences. In particular, an architecture with HTTP download page and HTTPS executable provides a false sense of protection and very little security, as the network attacker may forge the content of the download page.

We executed our data collection in May 2018. We analyzed 194 download environments of 163 different brands, grouped in several macro-category of devices (Table 2). In order to select those download environments, we constructed a variety of *search queries*, inserted those queries in a web search engine, considered the first result pages, identified links to sites of official manufacturers, and, finally, navigated within those sites interactively for identifying the download environment and compiling its description. We constructed search queries attempting to emulate the behavior of a person looking for a software related to a specific device model or to a specific brand. We chose the device models and brands with several different procedures, each tailored to a device macro-category.

For *smart meters (energy and gas)*, we constructed several preliminary search queries by mixing "gas smart meter" or "energy smart meter" with the names of major gas and energy providers in Italy, while for *energy meter*, we constructed the search query "contatori energia elettrica certificati" (i.e., "certified electricity meter" expressed in Italian). We submitted all these preliminary queries to a web search engine. Then, we identified from the results lists of devices certified for residential use in Italy and collected the corresponding devices model names and company brands. Finally, we constructed several search queries composed by the brand of the product, followed by its model name, followed by several combinations of the terms "software", "firmware", "driver",

Table 2: Dataset description and resource download configuration. Column "Env" contains the number of download environments and the number of environments that do not distribute executable resources. Columns "Secure", "Partly Secure", "Insecure", "Unknown" are in the format download pages - executable resource. Column $H \rightarrow S$ counts configurations with a path from an HTTP download page to an HTTPS executable, while $S \rightarrow H$ counts configurations with a path from an HTTPS download page to an HTTP executable.

Category	Brands	Env.	Secure	Partly Secure	Insecure	Unknown	$H \rightarrow S$	$S \rightarrow H$
Smart meters (energy)	8	10 - 5	0 - 0	5 - 5	0 - 0	0 - 0	1	4
Smart meters (gas)	8	8 - 6	0 - 0	1 - 1	1 - 1	0 - 0	0	1
Webcam	11	17 - 0	0 - 1	12 - 12	4 - 3	1 - 1	5	7
IP cameras (outdoor)	6	9 - 0	0 - 0	7 - 4	2 - 5	0 - 0	2	7
IP cameras (consumer)	11	12 - 0	0 - 0	3 - 2	6 - 7	3 - 3	0	3
IP cameras (best)	15	16 - 7	0 - 0	6 - 5	2 - 3	1 - 1	1	10
Electronics	38	52 - 7	0 - 1	28 - 24	12 - 14	5 - 6	14	20
Telecoms	4	4 - 1	0 - 0	2 - 1	0 - 1	1 - 1	0	1
Network (consumer)	8	10 - 0	0 - 0	7 - 5	3 - 4	0 - 1	1	5
Network (enterprise)	13	14 - 0	0 - 0	12 - 10	2 - 4	0 - 0	3	8
Energy meter	9	9 - 5	0 - 0	3 - 3	1 - 1	0 - 0	2	4
Smart thermostat	7	7 - 3	0 - 0	0 - 0	2 - 0	2 - 2	0	0
Industrial thermostat	2	2 - 1	0 - 0	1 - 1	0 - 0	0 - 0	1	2
ICS firmware	23	24 - 0	1 - 0	17 - 13	6 - 10	0 - 1	6	9
TOTAL	163	194 - 35	1 - 2	104 - 86	41 - 55	13 - 16	36	81
TOTAL (percentage)			0.6% - 1.3%	65.4% - 54.1%	25.8% - 34.6%	8.2% - 10.1%	33.3%	63.3%

"upgrade" and of the terms "download", "install". For *Electronics/Telecommunications*, we considered the Wikipedia list of largest manufacturing companies by revenue and selected the top-ranked companies in sectors "electronics" and 'telecommunications", respectively. For each company brand in these categories, we executed the same procedure as above (except that we omitted the model name from search queries).

We executed the same procedure as above for several other device macro-categories, with the only difference that we executed search queries for collecting product names and company brands on Amazon rather than on a web search engine. We omit the terms used in the queries for space constraints.

We considered only download environments hosted on web sites of official manufacturers in order to assess the security awareness of the manufacturer itself. We compiled only one download environment for each manufacturer and associated that environment with the first search query procedure with which it was found. We compiled multiple download environments for a given manufacturer only when we found multiple official support sites for that manufacturer.

We compiled each download environment description manually. We considered the possibility of implementing an automated tool for extracting the required information from the web and compiling each description automatically (similarly to [31]), but rejected this option because of the extremely broad variety in structure and content of relevant sites and, in particular, because of the widely differing web technologies (often quite old) used by the manufacturers involved in our analysis.

For each download environment, we categorized the download page and the executable resources according to the security levels in Table 1. A download page usually links to several executable resources, sometimes with different security levels. We categorized the executable resources of a download environment with the lowest security level found in that environment. Several download environments distribute only technical or marketing documentation, i.e., no executable resources. Such documentation could be used by a network attacker for injecting malware, but the corresponding issues are very different from the focus of our study. Thus, we chose to not categorize these environments.

4 RESULTS

Table 2 summarizes the dataset composition and the corresponding download configurations observed. The results are not very encouraging: we could find only 1 download page that can be accessed only through HTTPS and 41 download pages that can be accessed only through HTTP, corresponding to 0.6% and 25.8% of the analyzed download environments. It follows that a network attacker may easily drive users to an attacker-controlled URL, thereby fooling users to

download a malicious executable, for 25.8% of the analyzed download environments. Concerning the remaining download pages (65.4%), the attacker may or may not succeed in his malicious intent depending on whether the user happens to follow an HTTP or an HTTPS link to the download page. The results for executable resources are even worse than those for download pages: there is a small increase in secure configurations (1.3% vs 0.6%) but also a much greater increase in the insecure ones (34.6% vs. 25.8%).

Column $H \rightarrow S$ contains a count of configurations with a path from an HTTP download page to an HTTPS executable resource. As pointed out in the introduction, such configurations provide very little protection and, most importantly, give users a false sense of security. It can be seen that 36 download environments, corresponding to 33.3% of those that allow downloading an executable resources with HTTPS, fall in this category. Similar considerations can be made for paths from an HTTPS download page to an HTTP executable resource (column $H \rightarrow S$, percentage with respect to We foundResults for The

Further important insights on the security of download pages that can be accessed through HTTPS, and perhaps most importantly on the security awareness of the corresponding manufacturers, can be obtained by looking at the *strict transport security (HSTS)* policy of the download page. Such a policy is an optional security enhancement specified by a web application through a dedicated HTTP response header. When a supported browser receives this header, that browser will prevent any communications to that web application from being sent over HTTP and will instead send all communications over HTTPS. A network attacker will thus be unable to attack HTTP transactions with the download page because the browser will only execute HTTPS transactions, even when following HTTP links (of course, provided the browser has contacted the genuine web application in the past and received the corresponding policy).

For each download page accessible through HTTPS, we have checked whether the corresponding download environment specifies an HSTS policy and summarized the results in Table 3. It can be seen that only 25% of the analyzed download environments specify an HSTS policy, thus the remaining 75% of them do not take advantage of such important defense. We remark that the set of download pages accessible through HTTPS is essentially the same as the set of Partly Secure download pages, hence for 75% of those pages an attacker may attack HTTP transactions.

Another important defense mechanism for download pages that can be accessed through HTTPS is the *"upgrade insecure requests"* directive, which is part of another optional security enhancement specified by a web application through a dedicated HTTP response header. When a supported browser receives this directive, the browser will behave as described above for the HSTS, that is, it will treat HTTP links to the web application as if those links were HTTPS. We have checked, for each download page accessible through HTTPS, whether the corresponding download environment specifies an "upgrade insecure requests" directive. Only 2 download environments exhibit this behavior, one in IP cameras (best) and the other in ICS firmware. Both environments specify also an HSTS policy and belong to the Partly Secure category.

Table 3: Strict Transport Security for HTTPS download pages

Category	HSTS	No HSTS
Smart meters (energy)	4	3
Smart meters (gas)	0	3
Webcam	7	4
IP cameras (outdoor)	0	7
IP cameras (consumer)	0	3
IP cameras (best)	4	9
Electronics	9	24
Telecoms	1	2
Network (consumer)	0	7
Network (enterprise)	3	9
Energy meter	0	5
Smart thermostat	0	3
Industrial thermostat	0	2
ICS firmware	4	13
TOTAL	32	94
TOTAL (percentage)	25%	75%

5 CONCLUDING REMARKS AND FUTURE WORK

The problem of distributing software updates for smart objects securely will soon become a crucial issue for the society as a whole. While large software companies are generally aware of the relevance of this problem and have developed robust infrastructures for distributing software updates automatically and on a large scale, companies whose core business is different from software will have to develop their own solutions. A fundamental step in this direction is acknowledging the relevance of the problem and acquiring the necessary security awareness. Our analysis has allowed gaining important insights in this respect and, we believe, it is fair to claim that the current situation is not very encouraging. We have analyzed many download environments for software to be executed on resource-constrained appliances and the software distributed by most of those environments may be altered by network attackers easily, with techniques available not only to state actors but also to criminal organizations.

We plan to broaden our analysis on a much larger scale, by attempting to uncover possible correlations between the security of a software download environment and the mentions of the corresponding organization on various forms of media, including financial news, security news and alike. We plan to investigate such possible links based on unsupervised analysis of unstructured news sources [21–23, 30]. We believe such analysis may allow gaining insights into the incentives shaping the security attitude of smart object manufacturers.

REFERENCES

[1] [n. d.]. Google Safe Browsing. https://safebrowsing.google.com/. Accessed: 2018-8-18.

[2] Nate Anderson. 2013. How a banner ad for H&R Block appeared on apple.com—without Apple's OK. https://arstechnica.com/tech-policy/2013/04/how-a-banner-ad-for-hs-ok/. Accessed: 2018-8-18.

[3] Ross Anderson. 2018. Making Security Sustainable. *Commun. ACM* 61, 3 (Feb. 2018), 24–26.

[4] Alberto Bartoli, Giorgio Davanzo, and Eric Medvet. 2009. The reaction time to web site defacements. *IEEE Internet Computing* 13, 4 (2009).

[5] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. 2015. Meerkat: Detecting Website Defacements through Image-based Object Recognition.. In *USENIX Security Symposium*. 595–610.

[6] Russell Brandom. 2015. New vulnerability lets attackers hijack Chrysler vehicles remotely. https://www.theverge.com/2015/7/21/9009213/chrysler-uconnect-vulnerability-car-hijack. Accessed: 2018-8-18.

[7] Matt Burgess. [n. d.]. Smart dildos and vibrators keep getting hacked – but Tor could be the answer to safer connected sex. https://www.wired.co.uk/article/sex-toy-bluetooth-hacks-security-fix. Accessed: 2018-8-18.

[8] Kate Conger. 2016. Apple will require HTTPS connections for iOS apps by the end of 2016. *TechCrunch* (June 2016).

[9] Giorgio Davanzo, Eric Medvet, and Alberto Bartoli. 2011. Anomaly detection techniques for a web defacement monitoring service. *Expert Systems with Applications* 38, 10 (2011), 12521–12530.

[10] Lorenzo Franceschi-Bicchierai. 2017. A Hackable Dishwasher Is Connecting Hospitals to the Internet of Shit. https://motherboard.vice.com/en_us/article/pg9qkv/a-hackable-dishwasher-is-connecting-hospitals-to-the-internet-of-shit. Accessed: 2018-8-19.

[11] Nakibly Gabi, Schcolnik Jaime, and Rubin Yossi. 2016. Website-Targeted False Content Injection by Network Operators. In *USENIX Security Symposium*.

[12] Ryan Gallagher and Glenn Greenwald. 2014. How the NSA Plans to Infect 'Millions' of Computers with Malware. https://theintercept.com/2014/03/12/nsa-plans-infect-millions-computers-malware/. Accessed: 2018-8-18.

[13] Sujata Garera, Niels Provos, Monica Chew, and Aviel D Rubin. 2007. A framework for detection and measurement of phishing attacks. In *Proceedings of the 2007 ACM workshop on Recurring malcode*. ACM, 1–8.

[14] Google. 2018. Protecting users with TLS by default in Android P. https://android-developers.googleblog.com/2018/04/protecting-users-with-tls-by-default-in.html. Accessed: 2018-4-12.

[15] Andy Greenberg, Emily Dreyfuss, Brian Barrett, Danny Gold, Issie Lapowsky, and Lily Hay Newman. 2018. How Hacked Water Heaters Could Trigger Mass Blackouts. *Wired* (Aug. 2018).

[16] Andy Greenberg, Lily Hay Newman, Emily Dreyfuss, Brian Barrett, Danny Gold, and Issie Lapowsky. 2014. Hacker Redirects Traffic From 19 Internet Providers to Steal Bitcoins. *Wired* (Aug. 2014).

[17] Mikko Hypponen and Linus Nyman. 2017. The Internet of (Vulnerable) Things: On Hypponen's Law, Security Engineering, and IoT Legislation. *Technology Innovation Management Review* (April 2017).

[18] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. 2017. Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1435–1448.

[19] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2011. Learning to detect malicious urls. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 3 (2011), 30.

[20] John Markoff. 2016. Why Light Bulbs May Be the Next Hacker Target. *The New York Times* (Nov. 2016).

[21] Eric Medvet, Alberto Bartoli, Giorgio Davanzo, and Andrea De Lorenzo. 2011. Automatic face annotation in news images by mining the web. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*. IEEE Computer Society, 47–54.

[22] Eric Medvet, Alberto Bartoli, and Giulio Piccinin. 2014. Publication venue recommendation based on paper abstract. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*. IEEE, 1004–1010.

[23] Youssef Meguebli, Mouna Kacimi, Bich-Liên Doan, and Fabrice Popineau. 2014. Unsupervised Approach for Identifying Users' Political Orientations. In *ECIR*.

[24] Shaun Nichols. 2018. AWS DNS network hijack turns MyEtherWallet into ThievesEtherWallet. https://www.theregister.co.uk/2018/04/24/myetherwallet_dns_hijack/. Accessed: 2018-8-18.

[25] Charlie Osborne. 2018. Over a dozen vulnerabilities uncovered in BMW vehicles | ZDNet. https://www.zdnet.com/article/over-a-dozen-vulnerabilities-uncovered-in-bmw-vehicles/. Accessed: 2018-8-18.

[26] Marco Prandini, Marco Ramilli, Walter Cerroni, and Franco Callegati. 2010. Splitting the HTTPS Stream to Attack Secure Web Connections. *IEEE Security and Privacy* 8, 6 (Nov. 2010), 80–84. https://doi.org/10.1109/MSP.2010.190

[27] Emily Schechter. 2018. A milestone for Chrome security: marking HTTP as "not secure". https://www.blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/. Accessed: 2018-8-18.

[28] Fred B Schneider. 2018. Impediments with Policy Interventions to Foster Cybersecurity. *Commun. ACM* 61, 3 (Feb. 2018), 36–38.

[29] Ms Smith. 2017. 465,000 Abbott pacemakers vulnerable to hacking, need a firmware fix. https://www.csoonline.com/article/3222068/hacking/465000-abbott-pacemakers-vulnerable-to-hacking-need-a-firmware-fix.html. Accessed: 2018-8-18.

[30] Monica Chiarini Tremblay, Carlos Parra, and Arturo Castellanos. 2015. Analyzing Corporate Social Responsibility Reports Using Unsupervised and Supervised Text Data Mining. In *International Conference on Design Science Research in Information Systems*. Springer, 439–446.

[31] Steven Van Acker, Daniel Hausknecht, and Andrei Sabelfeld. 2017. Measuring Login Webpage Security. In *Proceedings of the Symposium on Applied Computing (SAC '17)*. ACM, New York, NY, USA, 1753–1760.

[32] Josephine Wolff. 2017. The Ransomware Attack That Locked Hotel Guests Out of Their Rooms. http://www.slate.com/articles/technology/future_tense/2017/02/the_ransomware_attack_that_locked_hotel_guests_out_of_their_rooms.html. Accessed: 2018-8-18.