Examples of models for security flaws and their countermeasures

Sevil Guler, Rao Siddharth Prakash
University of Tartu

Abstract

The goal of this essay is to analyze a set of security flaws in the real world software systems, and discuss what counter measures have been taken to overcome. While doing so, we will provide an informal definition of a threat/security model, constraints to build such models and adhere to the given definition while discussing the countermeasures. In this paper, we will discuss about the misuse of security libraries(such as SSL) in non-browser software with detailed study of vulnerabilities in Amazon and PayPal systems and the countermeasures taken. After which we will discuss about the vulnerabilities with StuxNet malware and the security models that have been built to combat against Stuxnet-type malwares. In this, we will discuss a cryptographic approach in the industrial control systems.

Introduction

Increase in malicious users and virus in the cyberspace has dramatically raised the expectation from secure software development. Since the software systems are continuously growing in complexity, size and connectivity, including the security measures in early stages of software development and building a security model is a challenging task for the security experts. Since many of the softwares have interface facing the outer world, developing one such model is complex process as it demands in depth knowledge of threats , vulnerabilities and security risks.

The threat modeling can be defined in broader meaning as the process of identifying the threats involved (or could be involved), which gives a deep insight to design or identify vulnerability countermeasures. The security patterns from one such model could be the well accepted security solutions which can be amalgamated to any general threats and hence allowing the solutions to be used in different situation.

Examples of models

2.1 Misuse of security libraries(SSL) in non-browser software

SSL is de-facto standard for secure end-to-end communication. It provides confidentiality, integrity and authenticity. In today's internet X.509 [1] is standard for implementation for public key infrastructure (PKI) where user has a preloaded list of root certificate authorities (CAs) and any certificate issued by those authorities is accepted by the user. In modern browsers, it is implemented by having root CAs hardcoded in the web browser. Some browsers, such as Google Chrome, also use certificate pinning [2] for the same. However, doing the same in non-browser software is tricky because of the limitations.

According to [3], due to the badly designed APIs and developers' misusage, the SSL certificate validation is totally broken in some security-critical applications and libraries, which make them vulnerable to man-in-the-middle (MITM) [4] attack.In this project, we examined some latest software in the list on the paper [3] to see whether they were still vulnerable to MITM attack or not. Our examination was done by source code analysis on open source software.

2.1.1 Vulnerable Software Examination

In [3], a list of software that were totally vulnerable to MITM attack were discovered. In this project, we have examined some software in that list to check whether they were still vulnerable or not. This section will present how we prepared and examined the vulnerable software and the result which was acquired from the examination.

One method were used to examine the software: analyzing the software's source code.Two open source payment software were selected for source code analysis, including *Amazon flexible payment service (PHP), Paypal payments standard* . The software's source code was downloaded and the implementation problems explained below were examined.

2.1.1.1 Merchant payment SDKs

SSL is also implemented not only web browsers but also in a non-browser software whenever secure internet connections are needed. Remotely administering cloud-based virtual infrastructure and transmitting customers' payment details from e-commerce servers to payment processors, such as in Paypal and Amazon, are some examples.

2.1.1.1.1 Amazon Flexible Payment Service

Amazon Flexible Payment Service provides SDKs that provides customers' detail to the merchants. However, Amazon Flexible Payment Service has fatal mistake because of misuse of SSL libraries that makes all of the systems vulnerable to MITM that is using FPS in their system[3].

According to [3] the PHP version of FPS SDK uses libcurl library to establish an SSL connection to the gateway. In [3] it is indicated that cURL's options for certificate validation are set in src\Amazon\FOPS\Client.php as follows:

```
curl_setopt($curlHandle, CURLOPT_SSL_VERIFYPEER, true);
curl_setopt($curlHandle, CURLOPT_SSL_VERIFYHOST, true);
$response = curl_exec($curlHandle);
```

Even cURL's default value of CURLOPT_SSL_VERIFYHOST is correctly set to 2 , calling curl_setopt($curlHandle,CURLOPT_SSL_VERIFYHOST, true), "true" silently turns into 1 and overrides the default and instructs cURL to check the existence of any common name in the certificate , which may or may not match the name requested [3].Therefore PHP code using Amazon-provided SDK to establish an SSL connection to the Amazon Flexible Payment Service gateway is vulnerable to MITM.

2.1.1.1.2 PayPal Payments Standard

Like Amazon Flexible Service, Paypal Payments Standard also provides SDKs that provides customers' detail to the merchants and also has fatal mistake because of the same reason with Amazon FPS[3].

2.1.2 Result

A positive result was acquired by our examination. The payment software, which are Amazon Flexible Payment Service, PayPal Payments Standard, demonstrated that their implementation problems have been fixed.We also checked the source codes of both and we examined that they fixed their overriding the default and instructing cURL by changing the code block which has fatal error and indicated above as follows :

```
curl_setopt($curlHandle, CURLOPT_SSL_VERIFYPEER, 2);
curl_setopt($curlHandle, CURLOPT_SSL_VERIFYHOST, 2);
```

$response = curl_exec($curlHandle);

2.2 Security modeling for an unpredictable case - StuxNet

StuxNet is a computer virus running windows operating system, which exploits vulnerabilities in Windows operating system and Siemens STEP 7 software [9]. It was created by an unknown source targeting the Iranian nuclear centrifuge. This stands as a new kind of weapon in the international warfare. Since this is a very informal and unpredictable security flaw, constructing a security countermeasure model is different from the one which we have discussed above. This is more challenging because, it does not have a clear threat, known risk or that can be predicted.

2.2.1 Vulnerabilities and early detection

The Stuxnet was designed to destroy industrial control systems (ICSs). It was designed to penetrate via removable drives, then through Local area networks; and then via programmable logic controllers (PLCs) to process control network (PCN) and the control system network (CSN). The major loophole was that the control systems lacked proper security measures for checking source authentication and message integrity. Industrial control systems (ICSs) are ubiquitous computing systems which are dependent on remote instructions for operating via LAN or Internet which is enabled with softwares which are of limited software security measures. This makes it a hot topic for cyber attackers as it can result in physical damage. In the case of Stuxnet, it modifies control messages in order to increase the frequency of nuclear centrifuges to unsafe levels, leading to equipment failure.

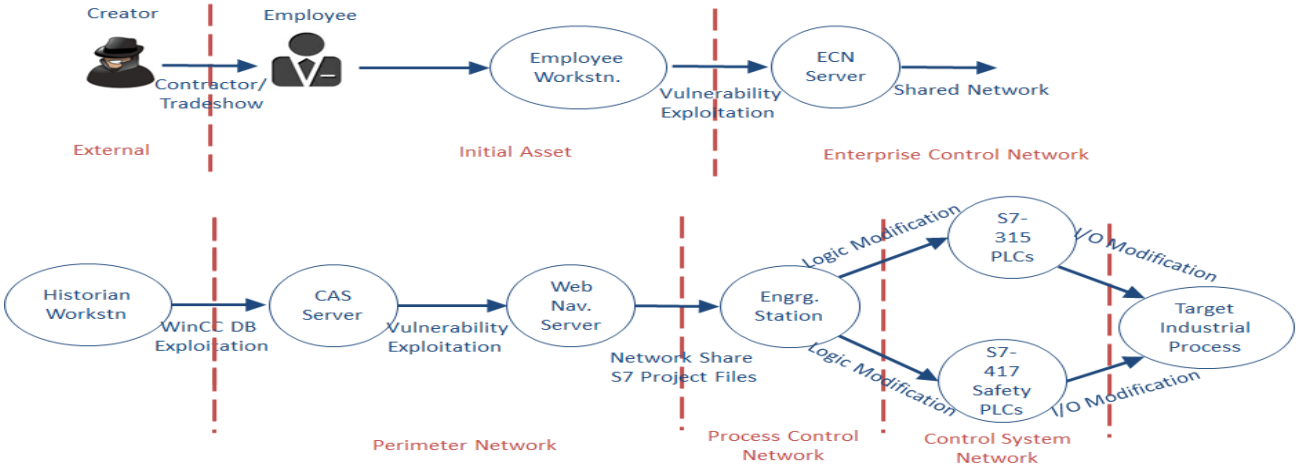

Figure 1 : Penetration process of Stuxnet malware

Figure 1 demonstrates the penetration process of Stuxnet malware. The propagation fo Stuxnet started through infected removable drives(e.g. flash drives and portable hard disks), and then it traverse though Local Area Networks(e.g. shared network drives and print spooler services) to the Programmable Logical Controllers(PLC). In the PLC, the Stuxnet infects Siemens project files, including both WinCC and STEP 7 files.

With the effort of security experts and open source community, the virus was profoundly detected at an early stage and hence it pulled a trigger among the white hackers community to build a counter measure for that. The vulnerabilities includes *LNK (MS10-046), Print Spooler (MS10-061), Server Service (MS08-067), Privilege escalation via Keyboard layout file* and *Privilege escalation via Task Scheduler.*

By calling different routines to read and write to memory on PLC, a library file manages the connection between PCN and CSN (Referring to Figure 2). Stuxnet was successful in replacing that particular library file. Hence it tampered the commands from PCN without being detected my PLCs or the system operator. This was mainly because there were no integrity checks used to verify source of any message [8].

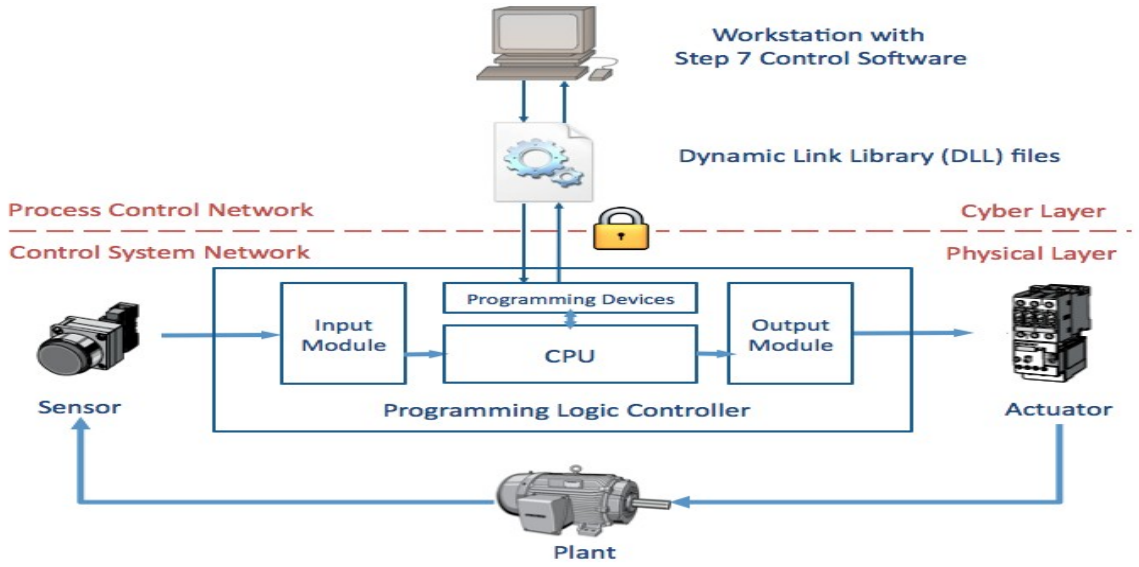

Figure 2 : Interaction between Process Control Network and control system network

2.2.2 Security model for Stuxnet type malware

Microsoft released a patch to close the issues which were detected. But what about the virus that has spread across already. Security experts from F-Secure, Norton and many open source communities are trying to come up with a model which will restrict the virus from reaching the target. This is quite similar to inducing security measures in the middle rather than starting from an early stage of secure software development. So have a proactive defense model for Stuxnet type malware is very much essential.

Here we also discuss a security model which is proactive defense system framework, in which commands from the system operator to the PLC are authenticated using a randomized set of cryptographic keys. The instructions from system operator to programmable logic controllers are authenticated with the help of a set of cryptographic keys which are randomized. The goal of such a defense system is to achieve as low as possible success probability for the attacker for a large number of keys.

The main idea of a proactive defense system against Stuxnet type malware is securing the communication between operators and the instructions sent by operator to programmable logical controllers. It can be achieved either using same cryptographic keys for all messages or different keys for each messages of that communication channel. These strategies imposes more workload for the adversary for tampering with the messages of such communication channel. It can be achieved in two possible ways in theory. First one is by increasing the length of the cryptographic keys which are being used, so that the messages are difficult to tamper because of increased complexity. But this is quite impractical approach as key lengths are set by common security standards and most of the cryptographic algorithms are optimized for specific set of key lengths. The second approach is, given a set of keys, choosing the subset of messages authenticated by each key so that the expected impact on the physical system of compromising any one key is minimized.

2.2.2.1 System model and adversary model

An operator sends messages to PLCs through an insecure communication channel which could be subject to attacks as described in the section 2.2.1 . The message management system for the security model [7] differentiates messages into different categories. Major two categories are, one being the standard command and control messages which are authentic and other contain dummy messages that are used for deception, which results in no PLC responses. The operator send commands to PLC from message space ***M.*** The set ***M*** is divided into **n** classes, denoted as $\boldsymbol{M_1, M_2 \ldots\ldots\ldots\ldots\ldots M_n}$, where $\boldsymbol{M_i} \cap \boldsymbol{M_j} = \boldsymbol{\Phi}$ for $i \neq j$. The receiving end executes the messages from the operator if and only if they are authentic. Message Authentication Code (MAC) is used to prevent an attacker from injecting or altering messages. The PLC authenticates the (message, MAC) pair (x, y) by consulting a predefined look-up table known to both the operator and the PLC, which identifies the set Mi with $x \in \boldsymbol{M_i}$. The PLC then checks if $h(x, K_i) = y$. Messages that fail this authentication check are discarded and ignored. Also, if $x \in Mi$ and $h(x, K_j) = y$ with $K_j = K_i$, then the message is identified as a possible forgery attempt and a warning is triggered by an alarm system as shown in Figure 3.

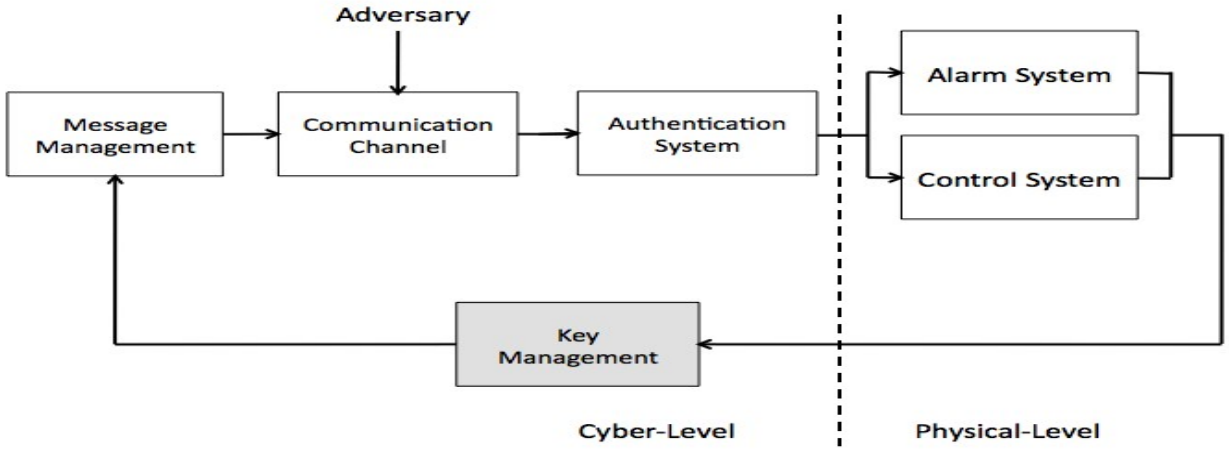

Figure 3 : Cryptographic security model for Stuxnet type Malware

In the adversary model, an adversary who is capable enough to bypass the communication between operator and PLC is considered. The adversary is also capable of sending (message, MAC) pairs (x, y) to the PLC, and performing computations using probabilistic polynomial-time algorithms. If Alg_0 denote the set of feasible algorithms for determining the keys, the goal of the adversary is to insert a (message, MAC) pair (x, y) such that $h(x, K_i) = y$ and $x \in M_i$.

The adversary model is based on two assumptions as follows. First assumption is that, the adversary knows the number of message categories **n**, but given a valid (message, MAC) pair(x, y), an adversary who does not know any of the keys $K_1 , \ldots K_n$ cannot determine whether $x \in M_i$ for any $i = 1, \ldots, n$. The security system model gives very fewer possibilities for the adversary to eavesdrop the messages, as the interation between the operator and PLCs takes place in a closed environment. This justifies the fact that the adversary does not know the mapping between messages and keys.

Second assumption is that the adversary has a probabilistic polynomial-time algorithm $Adv_0 \in Alg_0$ that takes as input a set of **q** (message, MAC) pairs $(x_1 , y_1), \ldots , (x_q , y_q)$ signed by a key $K \in ќ$. The algorithm outputs a key $\tilde{\mathbf{K}}$ such that $f(p, q) = Pr(\tilde{\mathbf{K}} = \mathbf{K})$. Furthermore, if there exist i, j, r, and s with $i \neq j$ and $r \neq s$, such that $y_i = h(x_i , K_r)$ and $y_j = h(x_j , K_s)$, with $x_i \in M_r$, and $x_j \in M_s$, then the $Pr(\tilde{\mathbf{K}} \in \{\mathbf{K_1, \ldots\ldots\ldots K_n}\}) = \mathbf{0}$. This implies that, if the adversary inputs two(message,

MAC) pairs with distinct keys into a cryptanalytic algorithm, then the algorithm will fail to return a correct key for the adversary.

Conclusion

For SSL usage from the results indicated Section 2.1.2 , we conclude that the developers of the examined softwares have made their softwares more secure by implementing better SSL certificate validation techniques. However, it is not always good to leave the decision for certificate validation on the developer as developer can misuse the libraries and hence users would fall prey to the attacker.

As discussed in the the problem such as Stuxnet-type malware,using proactive mechanisms which the system randomizes between different cryptographic keys for authentication. Based on security metrics such as the adversary's probability of success and the impact of a successful attack, the security model selects the number of messages signed using each key. The interactions between the network and the adversary can be viewed as a zero-sum game. Worst-case bounds shows that the proposed scheme can achieve arbitrarily low adversary success probability for a sufficiently large number of keys which makes it quite feasible security countermeasure.

References

[1] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," Internet Request for Comments, vol. RFC 5280 (Proposed Standard), May 2008. Available at: http://www.rfc-editor.org/rfc/rfc5280.txt

[2] "Certificate and Public Key Pinning," [Online]. Available at: https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning.

[3] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software," *CCS'12,* 2012. Available at: http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf

[4] W. Stallings, "Man-in-the-middle Attack," in Network Security Essentials: Applications and Standards, Prentice Hall, 4th edition, 2011, pp. 88-89.

[5] Kushner, David. "The Real Story of Stuxnet" from IEEE Spectrum. Retrieved 25 March 2014

[6] "Stuxnet Questions and Answers - F-Secure Weblog" F-Secure (Finland)

[7] Clark, A., Zhu, Q., Poovendran, R., Başar, T.: An impact-aware defense against Stuxnet. In: Proc. 2013 American Control Conference (ACC 2013), Washington, DC, June 17-19, pp. 4146–4153 (2013)

[8] N. Falliere, L. Murchu, and E. Chien, "W32. stuxnet dossier," White paper, Symantec Corp., Security Response, 2011.

[9] E. Byre, A. Ginter, and J. Langill, "How Stuxnet spreads – a study of infection paths in best practice systems," Tofino Security White Papers, February 2011.