A Quantitative Evaluation of the Target Selection of Havex ICS Malware Plugin

Julian Rrushi
Department of Computer Science
Western Washington University
516 High Street
Bellingham, WA, 98225, United States
julian.rrushi@wwu.edu

Hassan Farhangi, Clay Howey,
Kelly Carmichael, Joey Dabell
Department of Applied Research
British Columbia Institute of Technology
3700 Willingdon Avenue
British Columbia, V5G 3H2, Canada
{Hassan_Farhangi, Clay_Howey,
Kelly_Carmichael,
Joey_Dabell}@bcit.ca

ABSTRACT

While most of the current research focus is rightfully put on finding and mitigating vulnerabilities in industrial control systems (ICS), the opposite angle, namely researching operational weaknesses or unintelligent decisions of ICS malware that make them susceptible to detection, defensive entrapment, and forensics at large, is lesser explored. In this paper we perform a quantitative evaluation of the ability of Havex ICS malware plugin to correctly discover and query its target industrial control systems. We discuss the reverse engineering and analysis of various blocks of machine code of the Havex ICS malware plugin that pertain to its target selection process. We then quantify mathematically several performance measures of its target selection process. We find that despite its notoriety in the media as a nation state sponsored attack code, the Havex ICS malware plugin uses a plain and unsophisticated target selection process. That weakness in the malware opens the way to targeted defensive mechanisms to accurately neutralize the Havex malware and alike.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Security

Keywords

ICS malware; target selection; forensics

1. INTRODUCTION

The Havex malware gained notoriety in the press after the discovery that it targets industrial control systems (ICS) over the network. FireEye, F-Secure, and Microsoft identify the Havex malware as Backdoor:W32/Havex.A. The Havex malware uses conventional ways to infect its targets, namely direct exploitation of network services and fishing e-mails or malicious links to exploit web browsers. A third way, perhaps not so much explored by other malware, consists in the execution of trojanized installers. The attackers behind the Havex malware exploited vulnerabilities in the web sites of ICS software providers, and thus replaced legitimate ICS software packages with trojanized versions. When pulling software updates from those compromised web sites, various ICS operators received the trojan code, which they ran on their machines under the assumption that the needed software updates were being installed [4, 3].

The trojan code drops and executes a dynamic-link library (DLL) called mbcheck.dll. That DLL connects to command and control servers over the network, awaiting instructions. Those command and control servers can request mbcheck.dll to download and run malware plugins on the compromised machine. One of the Havex malware plugins targets industrial control systems. More specifically, it searches for Object Linking and Embedding (OLE) for Process Control (OPC) servers [5, 6], also referred to as Open Platform Communications (OPC) servers, that are reachable by an infected machine. The reverse engineering work discussed in [4, 3] led to the discovery of a number of command and control servers in Internet, which have been observed to communicate with Havex ICS malware plugins running on compromised machines.

In this paper we look at the Havex ICS malware plugin from a different perspective. While typically it is the Havex ICS malware plugin to target machines in the electrical power grid, in this work we analyze the Havex ICS malware plugin for operational weaknesses or unintelligent decisions that could help the defender detect and disable the attack. We focus our work on the efficacy of the target selection process of the Havex ICS malware plugin. The contribution of this paper consists of a quantitative evaluation of the performance of that process in empirical terms. We draw on mathematical techniques of detection theory, and also discuss the inner workings of the target selection

ACSAC Industrial Control System Security (ICSS) Workshop, 8 December 2015, Los Angeles, California, USA.

process of the Havex ICS malware plugin in support of the quantitative evaluation.

The remaining of this paper is structured as follows. In Section 2 we discuss some of the related work. The target selection process of the Havex ICS malware plugin is described in Section 3. The empirical mathematical analysis of the performance of the target selection process of the Havex ICS malware plugin is done in Section 4. In Section 5 we summarize this work, and conclude the paper.

2. RELATED WORK

Several aspects of the Havex malware and its ICS plugin have been reverse engineered and analyzed by malware researchers with FireEye and F-Secure [4, 3]. Here we focus on its target selection. Malware is known to perform target selection based on various forms of validations. One of those validations pertains to verifying that a compromised machine is not a trap aiming at aiding a forensics analyst to uncover the operations of the malware [12]. Most dynamic code analysis projects are implemented through the use of a debugger tool, and often on a virtual machine. Malware codes actively use anti-debug mechanisms to detect the presence of a debugger in a running system. Once a debugger is detected, the malware takes action to hinder code analysis. Some of those actions aim at creating frustration for the analyst and/or introducing a considerable time delay into the overall code analysis process.

Other more deceptive actions include skipping execution of blocks of code that are otherwise critical to understanding what the malware does, executing those blocks of code with data that conceal the actual behavior of the malware, or executing dummy blocks of code that are there only to perform dummy computations that appear as if the malware is doing something useful. All those actions can deceive the analyst. Similarly, malware can utilize a variety of techniques to discover whether or not the machine it is running on is virtual rather than physical [12]. Other validations that are typically performed by malware pertain to detection of central processing unit (CPU) emulators [13]. The target selection process of the malware is directly affected by those findings. If the malware discovers that its execution environment is a decoy, it does not proceed with pursuing its targets.

Stuxnet is known to perform some validation when searching for its WinCC targets. The attack code actively searches for Step 7 project files, but excludes those found under paths that match `*\Step7\Examples\*`. The latter are example project files, and typically serve as a model for developers to follow. Example project files are unlikely to be used in production, and thus are not viable targets for infection. The attack code also searches for `.mcp` files, which trigger Step 7 project infection and WinCC database infection. The attack code also monitors the PLC blocks that are being written to or read from a PLC so that to infect specific types of Simatic programmable logic controllers [14].

To the best of our knowledge, this paper is the first to quantitatively evaluate the performance of the target selection process of ICS malware, although our defensive deception interventions are different than those mentioned earlier in this section. While methods from detection theory are applied to psychology and medical sciences, in this work we find a correct application of those methods to malware performance evaluations.

3. TARGET SELECTION IN HAVEX

We reverse engineered with IDA Pro [2] the machine code of the main Havex malware DLL and its ICS malware plugin to see for ourselves the inner workings of how that malware discovers and selects OPC servers to spy upon. The Havex ICS malware plugin relies on Windows networking (WNet) [1] to discover all the servers, including OPC servers, that are reachable by the compromised machine over the network. The network search for servers is indiscriminate rather than specific. The Havex ICS malware plugin imports MPR.dll, i.e., the multiple provider router (MPR) DLL, so that its Wnet calls are forwarded to provider DLLs. A network provider DLL is responsible for handling network connections and data transfers between the machine as a client and remote servers. Communications with OPC servers are conducted by a provider DLL, which provides a set of functions to the MPR DLL to invoke.

The OPC servers are discovered if the MPR DLL has already loaded a registered network provider DLL for OPC at the moment the Havex ICS malware plugin issues the WNet calls. The Havex ICS malware plugin starts an enumeration of the network resources on the compromised machine. The Havex ICS malware plugin exhibits general interest in all network resources, given that the scope of the enumeration is RESOURCE_GLOBALNET. Furthermore, the interest in all network resources is indicated by the resource types and usage in the Wnet call that starts the enumeration. The NETRESOURCE structure in the Wnet call in question, which indicates the container to enumerate, references the root of the tree of network resources on the compromised machine. In other words, the Havex ICS malware plugin starts with the topmost container in the tree of network resources, and then explores the paths down to the leaves.

The enumeration of the tree of network resources on the compromised machine yields a list of all servers that are accessible over the network by the compromised machine. At this point, the Havex ICS malware plugin checks each one of those servers over the network for Microsoft Component Object Model (COM) interfaces [7]. More specifically, the Havex ICS malware plugin attempts to create over the network an instance of the OPC Server Browser class on each of the servers in question.

Upon creating each OPC Server Browser object with class identifier CLSID_OPCServerList, the Havex ICS malware plugin receives an array of MULTI_QI structures. One of those structures includes a pointer to the interface identifier IID_IOPCServerList or IID_IOPCServerList2, along with a pointer that can be used to invoke functions on the OPC Server Browser object through that interface. Issuing a call through the interface with identifier IID_IOPCServerList or IID_IOPCServerList2 on the OPC Server Browser object of each remote machine (referred to as server previously) that supports COM, returns a list of OPC servers, namely the class identifiers of OPC server objects on the remote machine. Knowledge of the class identifier of an OPC server object enables the Havex ICS malware plugin to issue calls through its standard interfaces.

An example is the IID_IOPCBrowse interface of an OPC server object, which allows for reading the attributes associated with an ITEMID of interest. The Havex ICS malware plugin also reads the data tags of each OPC server. A tag is a string that references a memory location in an industrial controller, and typically stores a physical parameter related

to the physical process monitored and/or controlled by the industrial controller.

The access to OPC tags over the network may be leveraged to identify the physical process behind the industrial controllers on the field. The OPC tag names are often chosen under the same principles as variables in a computer program, so that they are self-explanatory and representative of the physical parameters that they are mapped to in the memory of an industrial controller. In those cases, identifying the physical process becomes straightforward. The identification of the physical process is a required milestone that an attacker would have to attain for being able to program the malware to damage that physical process. The Havex ICS malware plugin does not take the step of initiating physical damage, as pointed out in [3, 4]. Instead, it limits itself to encrypting all those data, and sending the ciphertext to a command and control server over the network.

With these findings at hand, in this paper we focus our forensics analysis on the first three of the following components of the target selection process of the Havex ICS malware plugin:

- Ability to discover true servers over the network from the compromised machine, and ignore or discard nonexistent or absent servers on the network.
- Ability to determine whether or not a network server hosts COM objects and interfaces.
- Ability to find true OPC server objects on a remote machine, and dismiss COM objects that are not OPC.
- Ability to differentiate between real and hence valid OPC tags and honeytoken OPC tags or OPC tags that are no longer mapped to a location in the memory of an industrial controller.

Analyzing the last component in the above list requires deploying and configuring an industrial controller to monitor and control a real physical process, such as the passage of electrical energy from one circuit to another circuit through a power transformer. As the sensors provide measurements to an industrial controller such as an intelligent electronic device (IED) that is attached to the power transformer, the corresponding data tags in an OPC server are updated. Those tags then would represent true targets for malware such as the Havex ICS malware plugin to validate. This is something that we did not implement in this specific work, mainly due to safety reasons and lack of a physical lab space that was suitable to contain possible disruptions caused by the interaction of the malware with the power transformer.

Otherwise, the generation of invalid OPC tags that are to be targeted and possibly validated by the Havex ICS malware plugin is straightforward. Possible alternatives include randomly generated values, or values correlated according to known formulas under ideal conditions. Not only does the level of accuracy (or inaccuracy) of the target selection process in a piece of malware, such as the Havex ICS malware plugin, determine the magnitude of the true spread of the malware in the electrical power grid, but it also affects directly the amount of misinformation that can be injected into the attacking side. In the case of the Havex ICS malware plugin, for example, the malware's inability to identify true target industrial control systems enables the defender to provide large amounts of honeytokens, i.e., fake data, which the malware would encrypt and send to the attacker.

Those honeytokens could refer to large industrial honeynets operated by the defender. If at a later time the attacker decided to instruct the command and control servers to upload plugins into the Havex malware so that to initiate physical damage, the extended attack would drain in the industrial honeynets rather than disrupt true electric power equipment. We now quantify the performance of the target selection process of the Havex ICS malware plugin, and show that it is quite naive and hence vulnerable to the aforementioned cyber operation.

4. QUANTITATIVE ANALYSIS

Our quantitative analysis of the target selection process of the Havex ICS malware plugin involves the so-called signal trials and noise trials, as per detection theory [8]. The signal trials consist of true targets, which we expose to the Havex ICS malware plugin and then empirically observe whether or not the malware recognizes those targets as such. The noise trials consist of fake or nonexistent targets. As in the case of signal trials, we empirically observe whether or not the Havex ICS malware plugin pursues them. Thus, in both signal trials and noise trials the tasks are binary, namely the Havex ICS malware plugin implicitly takes a decision `yes` to pursue the target, and `no` to ignore it and take no further action on it. We are interested in measuring two factors as in traditional detection theory, namely response bias and sensitivity of the Havex ICS malware plugin.

The response bias of the Havex ICS malware plugin is a general tendency that it may have to deem a target as valid or invalid, i.e., signal or noise in detection theory terminology. The sensitivity of the Havex ICS malware plugin is the degree of overlap between its valid-target probability distribution and invalid-target probability distribution. For each value of the quantification of the internal factors that cause the Havex ICS malware plugin to pursue a target, the valid-target probability distribution shows the probability that the target is valid, while the invalid-target probability distribution shows the probability that the target is invalid. Both the response bias and sensitivity levels are affected by the hit rates and false-alarm rates that empirically characterize the target selection process of the Havex ICS malware plugin.

We conducted a large amount of signal trials, where a Windows machine was infected by Havex in a physically isolated network and thus had access to real servers on that network. In the majority of these signal trials, the Havex ICS malware plugin discovered the servers on the network correctly. Those few signal trials in which the Havex ICS malware plugin failed to discover the servers over the network involved reasons like the MPR DLL not finding a network provider DLL, or various forms of WNet errors, which interfered with the network server discovery. We also conducted a large amount of noise trials, where the Windows machine infected by Havex had no access at all to any servers over the networks. Instead, the Windows machine in question was running deceptive network drivers that emulated the operation of network interface cards on that machine. Those deceptive network drivers were paired with real network provider DLLs.

The setting was functionally similar to the testing environment discussed in [4], in which the authors utilized an

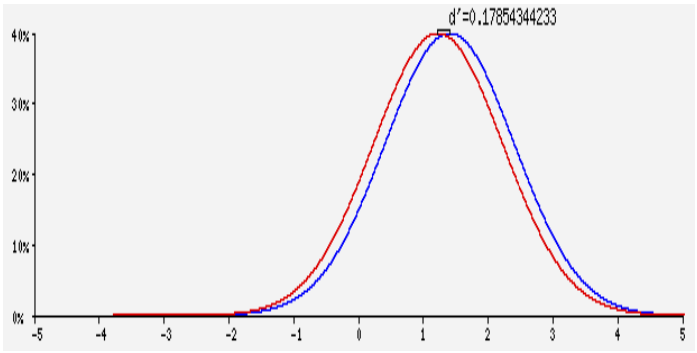

Figure 1: The valid-target probability distribution and invalid-target probability distribution as pertaining to the ability of the Havex ICS malware plugin to discover true servers over the network from the compromised machine.

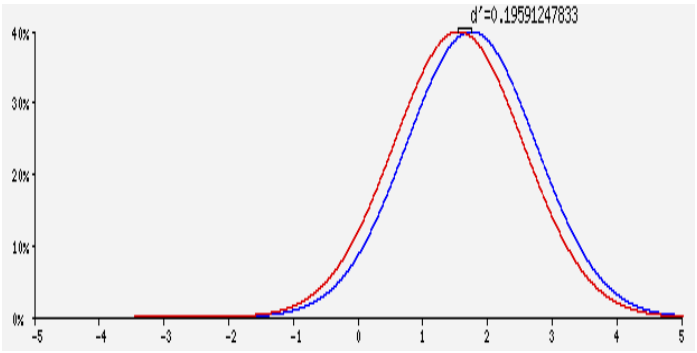

Figure 2: The two probability distributions in relation to the ability of the Havex ICS malware plugin to discover servers that host COM objects and interfaces.

Arduino Uno to run an OPC server not connected to any industrial controllers, and thus with no real OPC tags. The deceptive network drivers in conjunction with WNet caused the appearance of servers that were in fact nonexistent. In the majority of those noise trials, the Havex ICS malware plugin pursued the phantom servers as valid targets. The two probability distributions are depicted in Figure 1.

To quantify the sensitivity of the Havex ICS malware plugin, we calculate the d' measure, which measures the distance between the mean values in those probability distributions in standard deviation units. The two probability distributions in question are approximately normal distributions, and have standard deviations that are close to each other. We calculate the d' measure according to the formula given in [9], namely:

$$d' = \Phi^{-1}(H) - \Phi^{-1}(F) \tag{1}$$

Φ is a function that converts z scores into probabilities. Intuitively, the function Φ^{-1} converts probabilities into z scores. H is the hit rate measured empirically, while F is the false alarm rate, which is also measured empirically. The value of the d' measure that we obtain in the case of the Havex ICS malware plugin is $d' = 0.179$. The two curves are almost identical. A value of d' so close to zero indicates that the Havex ICS malware plugin is unable to distinguish over the network the real servers from nonexistent servers.

We also calculate A' as a nonparametric measure of the sensitivity of the Havex ICS malware plugin, using the formula given in [10], which is:

$$A' = \begin{cases} 0.5 + \frac{(H-F)(1+H-F)}{4H(1-F)}, & \text{if } H \geq F \\ 0.5 - \frac{(F-H)(1+F-H)}{4F(1-H)}, & \text{if } H < F \end{cases} \tag{2}$$

Recall from [10] that the values of the A' measure range from 0.5, which indicates that valid targets cannot be distinguished from invalid targets, to 1.0, which indicates full ability to distinguish valid targets from invalid targets. The value of the A' measure that we obtained empirically in the case of the Havex ICS malware plugin is $A' = 0.576$.

We quantified the response bias of the Havex ICS malware plugin by empirically calculating the β measure according to the formula from [11]:

$$\beta = \frac{e^{-0.5[\Phi^{-1}(H)]^2}}{\sqrt{2p}} \div \frac{e^{-0.5[\Phi^{-1}(F)]^2}}{\sqrt{2p}} \tag{3}$$

Recall that when $\beta = 1$, there is no bias. When $\beta < 1$, there is a bias towards accepting the target as being valid. When $\beta > 1$, there is a bias towards discarding the target as invalid. In the case of the Havex ICS malware plugin, we obtained a value of β equal to $\beta = 0.791$, which indicates that the Havex ICS malware plugin has the tendency to recognize as a valid server any software component that can respond to network queries.

We conducted a large amount of other signal trials in which existent servers hosted true COM objects and interfaces. Those servers accepted to create true COM objects based on the class identifier that the Havex ICS malware plugin passed them. The Havex ICS malware plugin was allowed to create over the network an instance of the OPC Server Browser class on each of those servers. Thus, those were all valid targets for the Havex ICS malware plugin. In the vast majority of these signal trials, the Havex ICS malware plugin pursued those servers as valid targets. It missed a few of them due to classes not registered in the registration database, or various errors with retrieving interfaces. A large amount of noise trials followed. This time, none of the servers hosted COM objects and interfaces. However, each of those servers generated a fake response when queried for COM objects and interfaces, acting like a honeypot.

The Havex ICS malware plugin accepted most of those servers as valid targets, and thus continued with the search for OPC server objects. Clearly all those searches failed since there weren't any OPC server objects on the machines in question. To quantify the findings, the two probability distributions are depicted in Figure 2.

We calculate a value of the d' measure equal to $d' = 0.196$. As it can be seen from Figure 2, the probability curves overlap to a large degree. The Havex ICS malware plugin is again unable to distinguish over the network the servers that host COM objects and interfaces from those that don't. The value of the A' measure that we obtained is $A' = 0.589$, which is very close to the floor value 0.5. We obtained a value of β equal to $\beta = 0.723$, which indicates that the Havex ICS malware plugin is biased towards accepting as a valid target any server that claims to host COM objects and

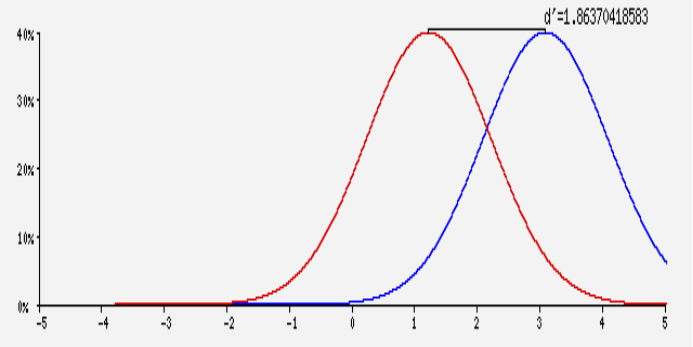

Figure 3: The two probability distributions in relation to the ability of the Havex ICS malware plugin to discover real OPC server objects.

interfaces. The final tests involved a large amount of other signal trials in which existent servers that support COM for real hosted OPC server objects. The Havex ICS malware plugin recognized all those servers as valid targets, and thus began retrieving attributes from the OPC server objects.

The final tests also involved a large amount of noise trials, in which existent servers that support COM did not host any OPC server objects. Those servers, however, acted like honeypots by returning lists of OPC server objects that did not exist. The Havex ICS malware plugin then followed up with attempting to read attributes from those nonexistent OPC server objects. The Havex ICS malware plugin accepted as valid targets several of those nonexistent OPC server objects. Others were not pursued due to difficulties with delivering a thorough OPC interaction with the Havex ICS malware plugin. To quantify these other findings, the two corresponding probability distributions are depicted in Figure 3. The value of the d' measure that we calculated is equal to $d' = 1.864$. Despite being higher than in the previous trials, the d' measure is still very low and hence an indicator that Havex ICS malware plugin in general is unable to differentiate real OPC server objects from nonexistent ones.

We calculate a value of the A' measure equal to $A' = 0.775$, which indicates a slightly above average success with causing the Havex ICS malware plugin to fail in the correct identification of real OPC server objects. The value of β that we obtained is equal to $\beta = 0.018$, which again indicates that the Havex ICS malware plugin is biased towards accepting any claim of OPC server object as valid.

5. CONCLUSIONS

Havex managed to spy on a large number of OPC servers not because it is a piece of intelligent malware, but simply because the defenses of its target machines, especially defensive deception mechanisms, may have been weak or entirely absent. After all, Havex was run as trusted code on the compromised machines. The various mathematical measures that we computed in this work show that the performance of the target selection process of the Havex ICS malware plugin is poor and hence unsupported by any validation. That leaves the Havex ICS malware plugin vulnerable to detection, forensics, and cyber operations based on defensive deception, which is positive from a security perspective. The findings of this work also bring a new perspective to securing industrial control systems, in that finding and leveraging weaknesses in ICS malware such as Havex helps create customized mechanisms for detection and disabling of the attack.

6. REFERENCES

[1] Microsoft, "Windows Networking", Available online at `https://msdn.microsoft.com/en-ca/library/windows/desktop/aa385406(v=vs.85).aspx`

[2] Interactive Disassembler, Available online at `https://www.hex-rays.com/products/ida/`

[3] Hentunen, D., and Tikkanen, A., "Havex Hunts for ICS/SCADA Systems", Available online at `https://www.f-secure.com/weblog/archives/00002718.html`

[4] Wilhoit, K., "Havex, It's Down With OPC", Available online at `https://www.fireeye.com/blog/threat-research/2014/07/havex-its-down-with-opc.html`

[5] OPC Foundation, "OPC Data Access Custom Interface Specification", March 2003.

[6] Mahnke, W., Leitner, S., and Damm, M., *OPC Unified Architecture*, Springer, March 2009.

[7] Rogerson, D. (1997) *Inside Com*, Microsoft Press.

[8] Schonhoff, T., and Giordano, A. (2006) *Detection and Estimation Theory*, 1st edition, Prentice Hall.

[9] Macmillan, N. A. (1993) *Signal Detection Theory as Data Analysis Method and Psychological Decision Model*, In G. Keren & C. Lewis (Editors), A handbook for data analysis in the behavioral sciences: Methodological issues (pp. 21-57). Hillsdale, NJ: Erlbaum.

[10] Snodgrass, J. G., and Corwin, J. (1988) *Pragmatics of Measuring Recognition Memory*, In Journal of Experimental Psychology: General, vol. 117, pp. 34–50.

[11] Brophy, A. L. (1986) *Alternatives to a Table of Criterion Values in Signal Detection Theory*, In Behavior Research Methods, Instruments, & Computers, vol. 18, pp. 285–286.

[12] Chen, X., Anderson, J., Mao, Z.M., Bailey, M., and Nazario, J. (2008) *Towards an Understanding of Anti-virtualization and Anti-debugging Behavior in Modern Malware*, Dependable Systems and Networks, pp. 177 –186, June 2008.

[13] Raffetseder, T., Kruegel, C., and Kirda, E. (2007) *Detecting System Emulators*, 10th International Security Conference, pp. 1–18, October 2007.

[14] Falliere, N., Murchu, L.O., and Chien, E. (2011) *W32.Stuxnet Dossier*, Symantec technical report, version 1.4, February 2011.