Hacker Machine Interface

The State of SCADA HMI Vulnerabilities

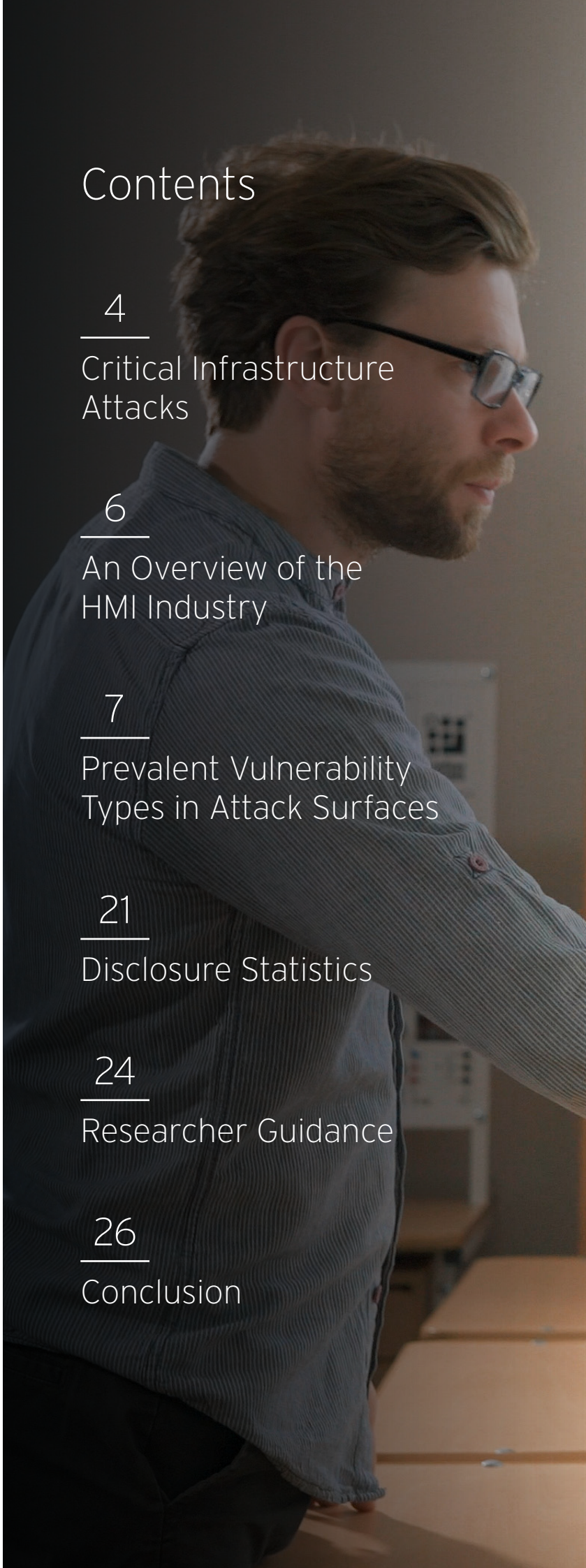
Trend Micro Zero Day Initiative Team

A **TrendLabs**SM Research Paper

WRITTEN BY:

Brian Gorenc and Fritz Sands
of the Trend Micro Zero Day Initiative Team

Contents



The impact of attacks targeting supervisory control and data acquisition (SCADA) systems depends on the threat actors' intent and the level of access and knowledge they have about the target. The Stuxnet and Ukranian power grid attacks give us clear ideas about how much damage a determined adversary can inflict not only on the business or operation concerned, but also on the general public.

Threat actors can use their access to SCADA systems to gather information such as a facility's layout, critical thresholds, or device settings for use in later attacks. Sabotage, including disrupting services or triggering dangerous and even lethal situations involving flammable or critical resources, represent an undesirable extreme.

Attackers infiltrate SCADA systems through various means, one of which is through the exploitation of software vulnerabilities prevalent in Human Machine Interfaces (HMIs). An HMI displays data from machines to a human and accepts commands from a human operator to machines. Through this interface, an operator monitors and responds to the information displayed on a system. A modern HMI provides a highly advanced and customizable visualization about the current state of a system. More often than not, the operator controls a SCADA system through this interface, which is often installed on a network-enabled location. As such, the HMI must be considered a primary target within a SCADA system, which should only be installed on an air-gapped or isolated on a trusted network. Experience shows this is not always the case.

Despite the obvious risks of obtaining unauthorized access to critical systems, the industry behind the development of SCADA systems, specifically HMI vendors, tend to focus more on equipment manufacture and less on securing the software designed to control them. The lack of global standards for HMI software further exacerbates software security problems within this field. We have also observed the same predictable software development oversights, which demonstrate that HMIs are not utilizing basic defense-in-depth measures.

This research examines the current state of SCADA HMI security by reviewing all publicly disclosed vulnerabilities in SCADA software that have been fixed from 2015 and 2016, including 250 vulnerabilities acquired through the Zero Day Initiative (ZDI) program. We found that most of these vulnerabilities are in the areas of memory corruption, poor credential management, lack of authentication/authorization and insecure defaults, and code injection bugs, all of which are preventable through secure development practices.

Finally, we observed the average time between disclosing a bug to a SCADA vendor to releasing a patch reaches up to 150 days, 30 more days than it would take highly deployed software such as those of Microsoft or Adobe, but significantly less than enterprise offerings from companies such as Hewlett Packard Enterprise (HPE) and IBM. This means that it takes an average of five months before SCADA vulnerabilities ever get patched. This, of course, differs among vendors. Some vendors may take as little as the same week while larger ones can take up to 200 days to do so.

Critical Infrastructure Attacks

Though you may not realize it, everyday modern life relies heavily on SCADA systems. These systems are responsible for running industrial processes involved in activities such as power generation, manufacturing, mechanized production, and refining. SCADA systems are at the core of water treatment plants, gas pipelines, electrical power distribution systems, wind farms, expansive communication systems, and even civil defense sirens. In other words, behind most modern conveniences, there exists a SCADA system somewhere that controls them. Therefore, attacks on SCADA systems have the potential to impact a wide range of systems and numerous pieces of critical infrastructure. Such was the case for two well-known attacks—the Iranian nuclear systems attack, better known as "Stuxnet," and the Ukrainian power grid attack.

Stuxnet Attack on an Iranian Nuclear Plant

Much has already been written about Stuxnet. In fact, there is so much information available about this piece of malware that it has its own book[1] and movie.[2] While Stuxnet's end goal was to sabotage Iran's nuclear program by damaging programmable logic controllers (PLCs), the likely state-sponsored worm did so by targeting the Siemens WinCC engineering software, which provides HMI-like functionality.[3] Reports state that as many as one-fifth of Iran's centrifuges were damaged by Stuxnet.[4] The emergence of malware that target the critical infrastructure of a foreign nation shows how critical finding and fixing problems within SCADA systems has become.

The Ukrainian Power Grid Attack

On 23 December 2015, unscheduled power outages began affecting a large number of customers serviced by Ukrainian power companies. There were also reports of malware infections affecting Ukrainian companies in a variety of critical infrastructure sectors. According to the Industrial Control Systems Cyber Emergency Response Team (ICS-CERT),[5] "During the cyber attacks, malicious remote operation of the breakers was conducted by multiple external humans using either existing remote administration tools (RATs) at the OS level or remote ICS client software via virtual private network (VPN) connections." In other words, attackers turned off the lights for more than 230,000 customers with a few mouse clicks. These

attacks did not target HMI vulnerabilities, but they did target the HMI solution. Because there was no network isolation, the attacker was able to connect via VPN and used remote access solutions to disable systems via the HMI.

While some Ukrainians were quick to place the blame on their neighbors to the east, the truly frightening piece of this story is the acknowledgment that the Ukrainian power system was found to be more secure than counterparts in the U.S.[6] Regardless of who was responsible or why, the attack showed what a determined actor could do to a nation's power supply given enough time and resources.

The Role of the U.S. ICS-CERT

A part of the U.S. Department of Homeland Security, ICS-CERT is tasked with reducing risks within and across all critical infrastructure sectors by partnering with law enforcement agencies and the intelligence community, and coordinating efforts among federal, state, local, and tribal governments and control system owners, operators, and vendors.[7] When the ZDI program purchases vulnerabilities that affect SCADA systems, a majority are reported to ICS-CERT for resolution. In 2015, ICS-CERT responded to 295 incidents and handled 486 vulnerability disclosures.[8]

Why Target the HMI

When going after SCADA systems, attackers tend to target the HMI for obvious reasons. The HMI represents the main hub for managing the critical infrastructure. If it can be compromised, just about anything can be done to the infrastructure itself, including causing physical damage to SCADA equipment.

Even if the intent is not to harm the SCADA system, by controlling the HMI an attacker could harvest critical architecture information for other purposes. Since the HMI acts as the main hub for managing the critical infrastructure, controlling it allows an attacker to harvest information about its architecture. An attacker could also disable alarms and notifications meant to alert operators to dangers to SCADA equipment.

An Overview of the HMI Industry

The marketplace for SCADA HMI is very active, but often not as secure as desired due to a variety of reasons. The largest HMI vendors in the industry include Siemens, Advantech, and GE, but there are also smaller players in many other countries. For instance, vendors in Asia specialize in systems for the region. The same can be said for vendors in Europe and North America.

Furthermore, since vendors of various sizes coexist, mergers and acquisitions happen frequently as well. In some cases, a small vendor gets bought before a patch comes out, making it hard to track the state of vulnerability throughout the disclosure process. Additionally, SCADA system vendors tend to focus on the actual industrial equipment and not on the software that manages them because they make the most profit selling the hardware. In fact, often, the HMI for a system is freely downloadable.

When it comes to the actual codes behind SCADA systems, a majority does not utilize basic defense-in-depth measures such as address space layout randomization (ASLR),[9] SafeSEH,[10] or stack cookies.[11] This may be related to the mistaken belief that these solutions will operate in a completely isolated environment. SCADA solution developers often have little experience with regard to user interface (UI) construction. This is coupled by the fact that developers do not know what the final operating environment will be like for the systems. This causes developers to make assumptions that are often incorrect. Without a mature development life cycle program to guide them, SCADA developers will likely continue to make the same mistakes that application and OS developers made a decade ago.

Prevalent Vulnerability Types in Attack Surfaces

In order to determine what vulnerabilities exist in HMIs, ZDI researchers reviewed the 2015 and 2016 ICS-CERT advisories to identify all of the solutions that had bugs fixed within the last two years. This data was then cross-referenced with the more than 250 zero days purchased by the ZDI program. This information was also compared with the Common Weakness Enumeration (CWE) to determine what existed within this space. We have categorized these SCADA vulnerability types from this time period into memory corruption, credential management, lack of authentication/authorization and insecure defaults, and code injection.

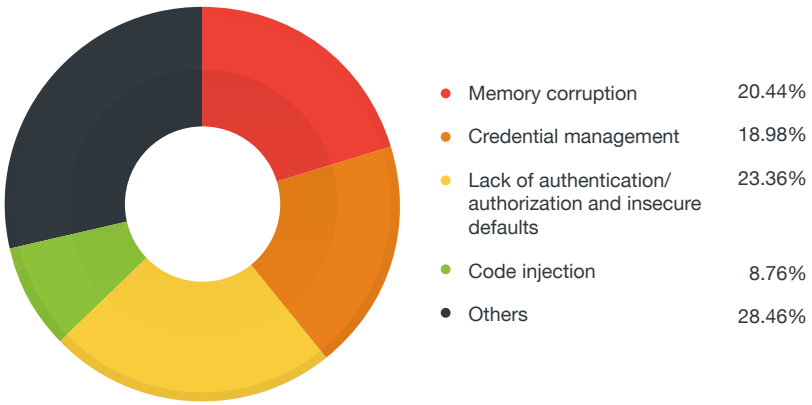


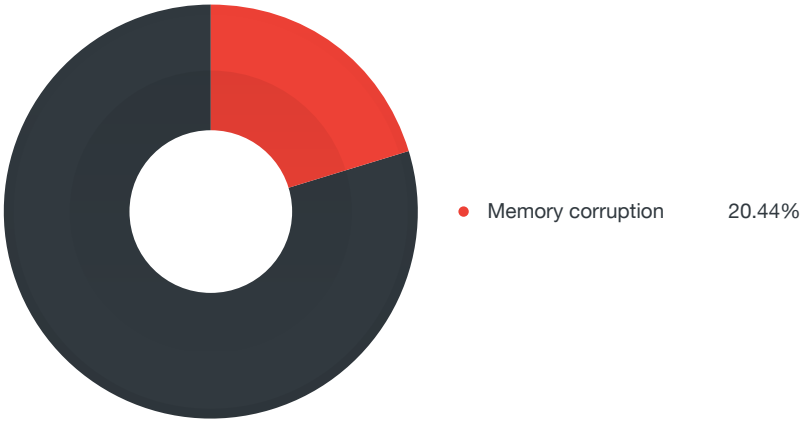
Figure 1: Vulnerability categories

While some cross-site scripting (XSS) and Cross-Site Request Forgery (CSRF) issues were expected, most of the HMIs are Windows- and not web-based applications. Some XSS and CSRF bugs do exist within the “Others” category, though they are not numerous. Instead, most of the vulnerabilities exist within one of the four areas, namely:

- Memory corruption
- Credential management
- Lack of authentication/authorization and insecure defaults
- Code injection

Memory Corruption Issues

Memory corruption issues represent 20% of the vulnerabilities identified. The weaknesses in this category represent classic code security issues such as stack- and heap-based buffer overflows and out-of-bounds read/write vulnerabilities. Memory corruption may occur in HMIs when the contents of a memory location are unintentionally modified due to errors somewhere in the code. This may also be referred to as violating memory safety. When the corrupted memory contents are used later in that program, the program either crashes or executes code that was not intended to run.



Case Study: Advantech WebAccess HMI Solution

At one point, the ZDI program received 100 separate reports involving the Advantech WebAccess HMI on a single day. The majority of these cases turned out to be buffer overflows, most of which are similar to the example below.

Figure 2: Advantech WebAccess dashboard

One interesting item to note about this product is that while it currently is a SCADA solution, it is also advertised as an Internet of Things (IoT) solution. The solution contains the service *webvrpcs.exe* that runs in the context of a local administrative user. The service listens on Transmission Control Protocol (TCP) port 4592 by default and may be accessed over a Remote Procedure Call (RPC)-based protocol. The service calls from the application are intended to resemble Microsoft® Windows® Device IoControl function calls. Each service call contains an input/output control (IOCTL) value that enables jump tables to be used to perform hundreds of service types. In this example, the parameter involved is the window name that is copied via the *_sprintf* function to a stack buffer that is 0x80 characters.

```
0000154 05 00 00 03 10 00 00 00  c0 00 00 00 05 00 00 00 ....... ....... 
0000164 a8 00 00 00 00 00 00 00  c0 3f 58 05 8b 38 01 00 ....... x..8.. 
0000174 8c 00 00 00 8c 00 00 00  7f 7f 7f 7f 7f 7f 7f 7f ....... ....... 
0000184 7f 7f 7f 7f 7f 7f 7f 7f  7f 7f 7f 7f 7f 7f 7f 7f ....... ....... 
0000194 7f 7f 7f 7f 7f 7f 7f 7f  7f 7f 7f 7f 7f 7f 7f 7f ....... ....... 
00001A4 7f 7f 7f 7f 7f 7f 7f 7f  7f 7f 7f 7f 7f 7f 7f 7f ....... ....... 
00001B4 7f 7f 7f 7f 7f 7f 7f 7f  7f 7f 7f 7f 7f 7f 7f 7f ....... ....... 
00001C4 7f 7f 7f 7f 7f 7f 7f 7f  7f 7f 7f 7f 7f 7f 7f 7f ....... ....... 
00001D4 7f 7f 7f 7f 7f 7f 7f 7f  7f 7f 7f 7f 7f 7f 7f 7f ....... ....... 
00001E4 7f 7f 7f 7f 7f 7f 7f 7f  7f 7f 7f 7f 7f 7f 7f 7f ....... ....... 
00001F4 7f 7f 7f 7f 7f 7f 7f 7f  7f 7f 7f 7f 7f 7f 7f 7f ....... ....... 
0000204 00 00 00 00 04 00 00 00  04 00 00 00 00 00 00 00 ....... ....... 
```

The area marked in yellow is the IOCTL code, followed by the buffer length (in and out)—both 0x8c bytes followed by a terminating null. Since the attack puts 0x8c bytes of data into a 0x80 byte length buffer, the predictable overflow results. As the process is not ASLR aware, the area ahead of the IOCTL is a binding handle and connection ID, which is set up by an earlier call to register as a client of the service.

Reviewing the vulnerable code reveals the classic *_sprintf* call, which allows the overflow condition.

```
.text:100015D6 loc_10015D6     ; CODE XREF: BwSvcFunction+44j
.text:100015D6                 ; DATA XREF: .text:off10001B08o
.text:100015d6 push offset aRpc_dllBwcmd_g ; jumptable 10001124
case 20011
.text:100015DB call    sub_10001BB0
.text:100015E0 move    ebx, [esp+0F8h+arg_8]
.text:100015E7 add     esp, 4
.text:100015EA lea     edx, [esp+0F4h+WindowName]
.text:100015EE push    ebx
.text:100015EF push    offset aS       ; "%s"
.text:100015F4 push    edx             ; char *
.text:100015F5 call    _sprintf
```

Inspecting the stack layout reveals that *WindowName* is set to -80 with 0 as the return address. Since it was not set during the compile[13] stage, there are no stack cookies set to aid in protection.

```
.text:100010E0 lParam          = dword ptr -0E4h
.text:100010E0 var_E0          = dword ptr -0E0h
.text:100010E0 var_DC          = dword ptr -0DCh
.text:100010E0 var_D8          = dword ptr -0D8h
.text:100010E0 wParam          = dword ptr -0D4h
.text:100010E0 var_D0          = dword ptr -0D0h
.text:100010E0 var_CC          = byte ptr -0CCh
.text:100010E0 var_C0          = byte ptr -0C0h
.text:100010E0 File            = byte ptr -0A0h
.text:100010E0 WindowName      = byte ptr -80h
.text:100010E0 arg_0           = dword ptr 4
.text:100010E0 arg_8           = dword ptr 0Ch
.text:100010E0 arg_C           = dword ptr 10h
.text:100010E0 arg_10          = dword ptr 14h
```

The lack of stack cookies as well as of other protections such as ASLR and SafeSEH is likely due to the original code being written prior to the existence of these coding practices. However, the use of banned application program interfaces (APIs) and the lack of defense-in-depth measures mean that an attacker merely needs to overwrite the return address to the beginning of the attacker-controlled return-oriented programming (ROP) chain. With no ASLR, no complexities are required to execute attacker-controlled code at an elevated privilege.

The patch analysis for this vulnerability also shows some interesting choices in cleaning up an aging code base. The original offending function *_sprintf* was included in the Microsoft-banned API[14] list originally released in 2007. ZDI researchers expected Advantech to implement a banned-API list and remove known bad functions from its code. Instead, the patch for this bug changed the *_sprintf* function to the *_snprintf* function.

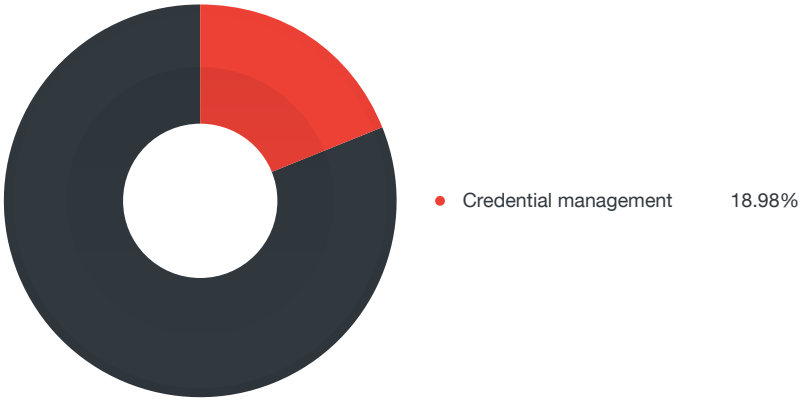
```
.text:10001600        lea     edx,    [esp+0F4h+WindowName]
.text:10001604        push    ebx
.text:10001605        push    offset aS               ; "%s"
.text:1000160A        push    7Fh                     ; size_t
.text:1000160C        push    edx                     ; char *
.text:1000160D        call    _snprintf
```

However, *_snprintf* is **also** in the banned-API list. While *_snprintf* provides more resiliency to overflows than *_sprintf,* it fails to null terminate when provided with too many characters. This means that when the stack is not cleaned out—a rare situation—it becomes possible for attackers to use string manipulation on this *WindowName* to trick the program into thinking the buffer is 0x80 characters long when in reality, the buffer is longer since it did not null terminate.

Out of the 75 patches that Advantech produced, all were point fixes. In other words, the vendor fixed specific issues but did not globally replace banned APIs or other problematic functions. Thousands of *_sprintf* and *_snprintf* functions remain in the code base. The probability that none of these remaining banned APIs can be reached by attacker-controlled data is next to zero. Advantech did not release patches for the remaining 25 issues reported by ZDI researchers. These issues were subsequently disclosed to the public in accordance with the ZDI program's policies.[15]

Credential Management Issues

Credential management issues represent 19% of the vulnerabilities identified. The vulnerabilities in the category represent cases such as using hard-coded passwords, storing passwords in a recoverable format (e.g., clear text), and insufficiently protecting credentials.



Case Study: GE MDS PulseNET Hidden Support Account

General Electric (GE) MDS PulseNET is used to monitor devices and industrial communication networks deployed in energy, water, and waste water sectors worldwide. The ZDI program received a vulnerability report that stated, "The affected products contain a hard-coded support account with full privileges." The full investigation resulted in the disclosure of CVE-2015-6456 with a Common Vulnerability Scoring System (CVSS) rating of 9.0.

A look at the user management panel indicates the existence of only two accounts in the system—operator and admin. However, as noted by Andrea Micalizzi (who also goes by the handle "rgod"), a hidden third account exists that has full admin privileges.

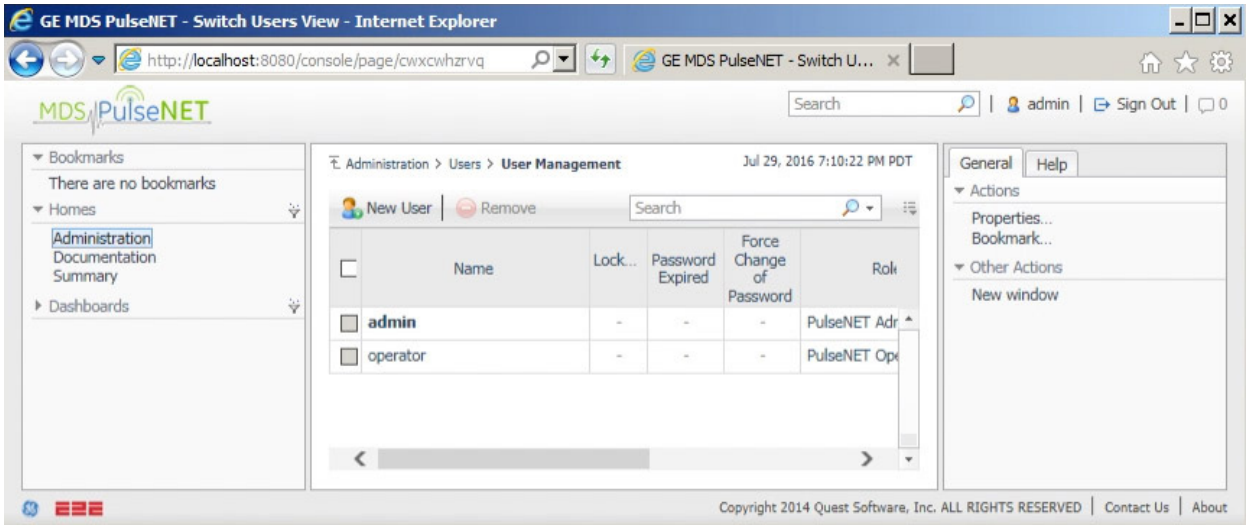


Figure 3: GE MDS PulseNET user management panel

Using HeidiSQL[16] to extract information from the database exposes the account “ge_support” with a password hash of *<![HDATA[MD5$8af7e0cd2c76d2faa98b71f8ca7923f9.* By cracking the MD5 hash, the password is exposed as “Pu1seNET.”

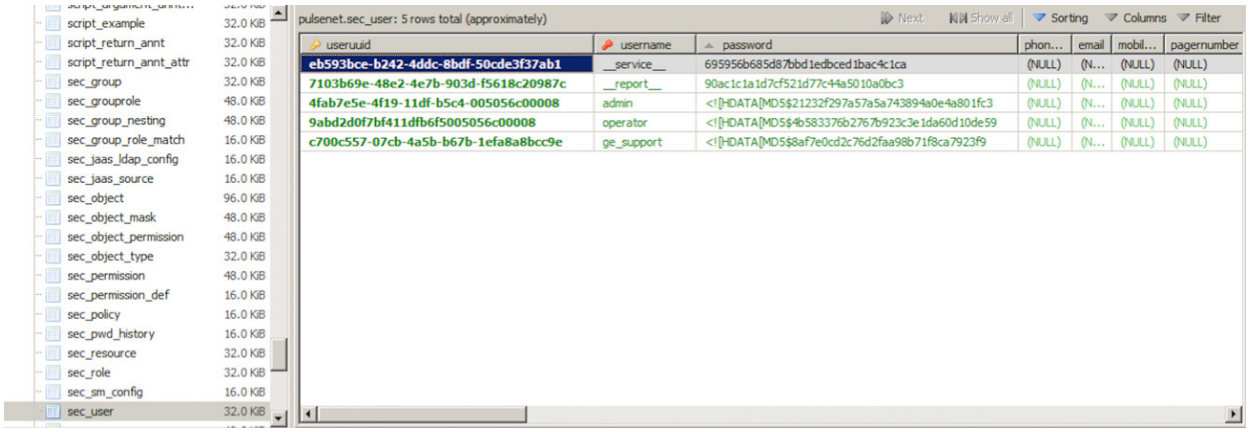
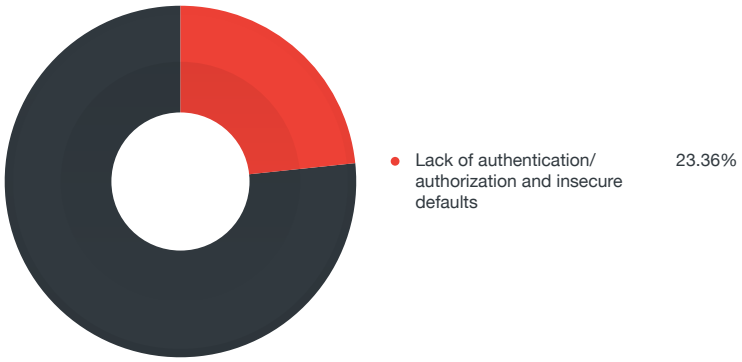


Figure 4: GE MDS PulseNET user management panel after logging in

Even after logging in using the “ge_support” account, the third user name does not appear in the user management panel.

Lack of Authentication/Authorization and Insecure Defaults

This category represents 23% of the SCADA vulnerabilities. It includes many insecure defaults, clear-text transmission of sensitive information, missing encryption, and unsafe ActiveX controls marked safe for scripting.



Case Study: Siemens SINEMA Server Insecure File Permissions Privilege Escalation Vulnerability

One misconfiguration often seen in HMI products occurs when companies decide to create their own Access Control Lists (ACLs) and top-level directory instead of using the default Windows Program Files directory. Instead of being properly protected, these top-level directories are by default world-writable, and the directory tree includes the solution's service binaries. This allows any local user to drop new binaries that will execute as a system service into the directory.

This issue manifested itself in the Siemens SINEMA Server and was addressed through CVE-2016-6486.[17] The product installed itself in a directory tree with a weak ACL. By default, Microsoft sets up a directory—Program Files—with secure ACLs.

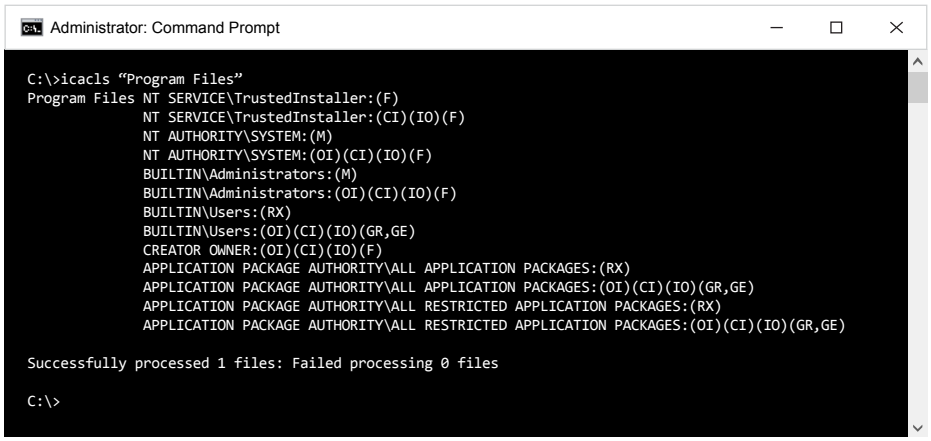


Figure 5: Command line showing the Program Files standard permissions

Note that only administrative and special accounts have the ability to write files into that tree. Users also have read and execute permission.

Instead of using Program Files or at least the same permissions found in Program Files, the Siemens installer creates its own top-level tree.

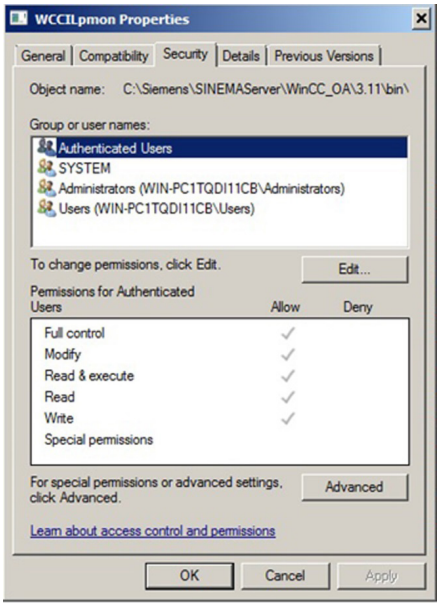


Figure 6: Security permissions properties for Siemens SINEMA Server

As seen above, anyone on the system may append and write data to the files contained in this directory. Users may also add files and subdirectories, which can lead to code execution because of localization directories. Even worse, if we examine the permissions where the programs reside, the results prove troubling.

```
C:\Siemens\SINEMAServer\WinCC_OA>icacls 3.11
3.11 NT AUTHORITY\Authenticated Users:(I)(OI)(CI)(F)
     NT AUTHORITY\SYSTEM:(I)(OI)(CI)(F)
     BUILTIN\Administrators:(I)(OI)(CI)(F)
     BUILTIN\Users:(I)(OI)(CI)(RX)
     BUILTIN\Users:(I)(CI)(AD)
     BUILTIN\Users:(I)(CI)(WD)
     CREATOR OWNER:(I)(OI)(CI)(IO)(F)
```

As noted by the highlighted string, Authenticated Users have full control. To join the Authenticated Users group, a user only needs to authenticate with the Windows domain. Essentially, that means almost everyone on the system will be in the Authenticated Users group. Examining a binary within this directory shows similar results.

```
C:\Siemens\SINEMAServer\WinCC_OA\3.11\bin>icacls WCCILpmon.exe
WCCILpmon.exe NT AUTHORITY\Authenticated Users:(I)(F)
              NT AUTHORITY\SYSTEM:(I)(F)
              BUILTIN\Administrators:(I)(F)
              BUILTIN\Users:(I)(RX)
```

It is important to note that this particular binary starts a service that runs at the Local System level.

```
C:\Windows\system32>sc qc sinema_server
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: sinema_server
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   :
C:\Siemens\SINEMAServer\WinCC_OA\3.11\Bin\WCCILpmon.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : SINEMA_SERVER
        DEPENDENCIES       : Sinema_Server_Db
        SERVICE_START_NAME : LocalSystem
```

The end result of this insecure default installation allows any Authenticated User to swap out the binary at will. Furthermore, that new attacker-controlled binary executes under the context of Local System after the next reboot.

Case Study: Advantech WebAccess

Advantech WebAccess provides a cross-platform, cross-browser data access experience, and a UI based on HTML5 technology.[18] The ZDI program received a bug report about a condition that allowed other passwords to be viewed when changing a password. The ICS-CERT report description of this bug states, “A properly authenticated administrator can view passwords for other administrators.” It should be noted that this does not mean the system administrator. Instead, this refers to the administrator of a given SCADA solution.

Within the solution, there exists the script *upAdmin.asp* that allows a SCADA administrator to update his username, password, or description. This Active Server Pages (ASP) script can be abused by someone with permission on the system.

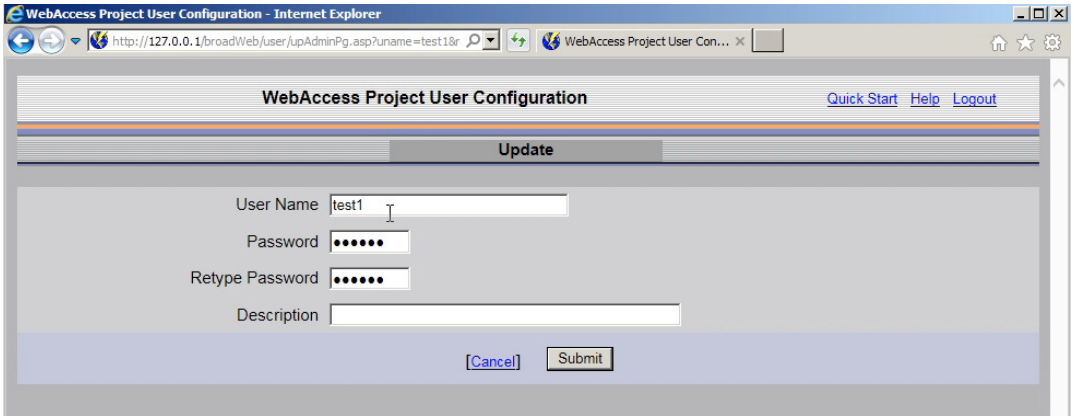


Figure 7: Advantech WebAccess User Configuration panel

For an attacker to take advantage of this bug, he must first log in to a known account and click "Project user property." This takes him to the URL *http://<ip>/broadWeb/user/upAdminPg.asp?uname=**known**&return=bwproj.*

In this example, <ip> is the IP address of the system and known is the known user account. The attacker then changes the name of the account in the URL to the targeted account as in *http://<ip>/broadWeb/user/upAdminPg.asp?uname=**victim**&return=bwproj.*

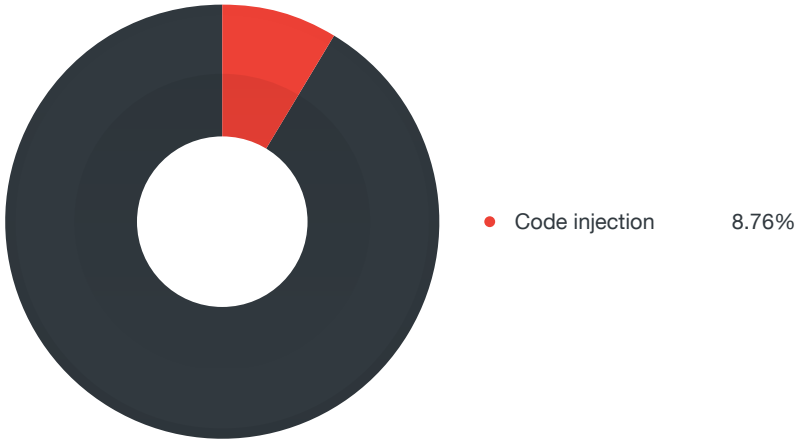
While the password for the victim account is initially obscured by asterisks, simply viewing the HTML source on the page will display the password of the target user.

```
<tr bgcolor="#C9C9CB">
                <td width="30%" align="right"><font class=e3>Password</font></td>
                <td width="70%" align="left" valign="middle">
                        <input type="password" name="Password" maxlength="8" size="8" value="secret">
        </td>
</tr>
```

This bug allows an authenticated user to get the password of any other user, including the overall administrator of the SCADA solution.

Code Injection Issues

Much like other online services, code injection vulnerabilities exist within the HMI realm. These issues represent 9% of the vulnerabilities identified. While common injection types—SQL, command, OS, code—still occur, there are domain-specific injections that also pose a risk to SCADA solutions. One of the domain-specific languages prone to injection is Gamma script, which is used by the Cogent DataHub system.



What is Gamma script?

Cogent DataHub products use a proprietary language called Gamma for its HMI solutions. According to the manufacturer, Gamma is a dynamically typed interpreted programming language specifically designed to allow rapid development of control and UI applications. The language relies on a syntax similar to C and C++ but with a range of built-in features that make it a far better language for developing sophisticated real-time systems. Gamma also comes with a fully documented API and is available to anyone on the Internet.

Case Study: Cogent DataHub

According to the manufacturer, Cogent DataHub is a memory-resident, real-time database that acts as a hub that provides fast and efficient concentration and distribution of data for OLE for Process Control (OPC) and other Windows applications. The ZDI program received a report on a bug that allows an attacker to turn on insecure processing mode in the web server. This provides a way for an attacker to send arbitrary scripts to the server and execute arbitrary code.

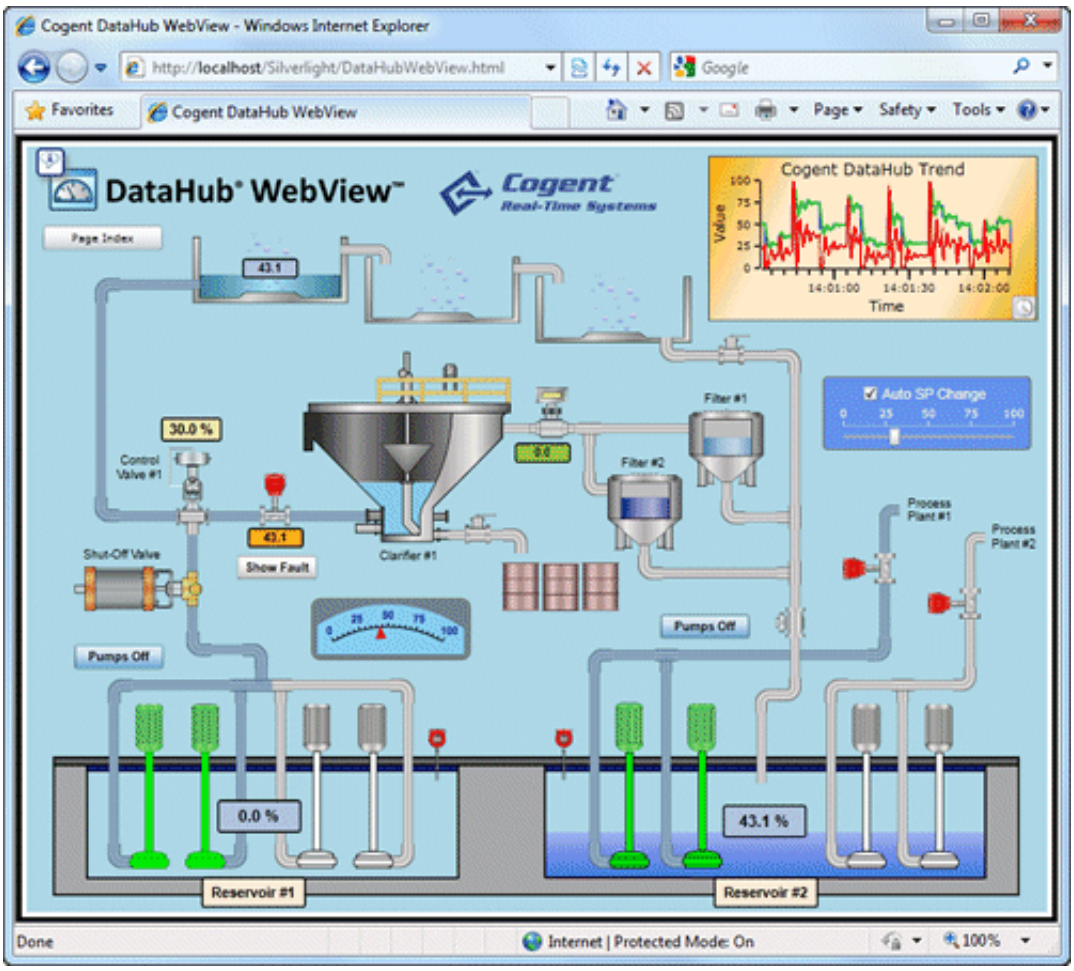


Figure 8: Cogent DataHub WebView

(Source: http://www.softwaretoolbox.com/images/101210_webview_1.gif?crc=3945836907)

As seen above, Cogent DataHub provides a real-time middleware solution that adds visualizations to complex SCADA systems.

In this case, the attacker leverages a flaw in the *EvalExpression* method within Gamma to execute attacker-controlled code in the target system. This method is remotely accessible through the Ajax facility listening on TCP port 80. Supplying the target a specifically formatted Gamma script allows the execution of arbitrary OS commands.

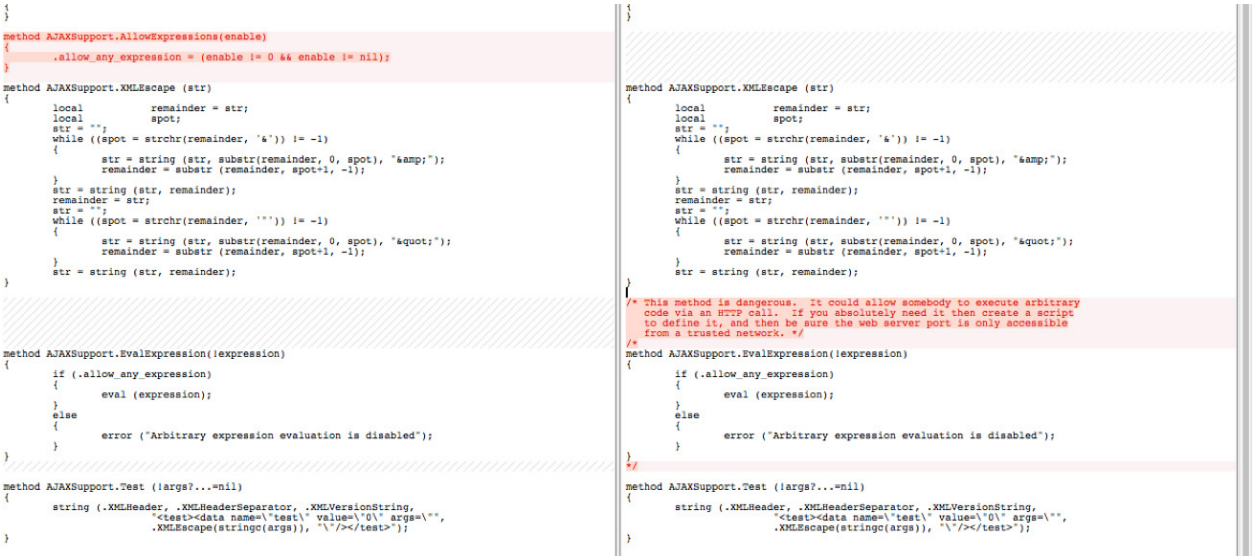
Since Gamma is a domain-specific language, it contains many built-in features and functions specific to the SCADA industry. However, the script also contains the ability to access and execute system commands. It is this ability that may be abused by the attacker, as seen in the vulnerable code below.

```
Method AJAXSupport.EvalExpression(!expression)
{
        if (.allow_any_expression)
        {
                eval (expression); << Bug here
        }
        else
        {
                error (“Arbitrary expressions evaluation is disabled”);
        }
}
```

The highlighted bug takes an expression and checks one flag to determine if the system is allowed to execute the expression. If this check returns true, the expression executes, regardless of what it contains. An attacker still must trick the system into loading the necessary libraries and change the value to ensure the system returns true to execute the expression. Fortunately for the attacker, the system allows for this as well.

To complete this exploit, an attacker must first send an HTTP request to any Gamma script that loads the necessary libraries. As mentioned, while developers assume these systems will operate on an isolated network, this is often not the case. The attacker uses the request to call *AJAXSupport.AllowExpressions* and set *allow_any_expression* to True. This allows the attacker to then call the *AJAXSupport.EvalExpression* method and pass in the script that he wants to execute. This attack method is very reliable and repeatable.

Reviewing the patch shows how Cogent decided to tackle the problem. The left-hand side of the image below shows the old code while that on the right-hand side shows the patched code. The changes are highlighted.



```
}
method AJAXSupport.AllowExpressions(enable)
{
        .allow_any_expression = (enable != 0 && enable != nil);
}
method AJAXSupport.XMLEscape (str)
{
        local           remainder = str;
        local           spot;
        str = "";
        while ((spot = strchr(remainder, '&')) != -1)
        {
                str = string (str, substr(remainder, 0, spot), "&amp;");
                remainder = substr (remainder, spot+1, -1);
        }
        str = string (str, remainder);
        remainder = str;
        str = "";
        while ((spot = strchr(remainder, '"')) != -1)
        {
                str = string (str, substr(remainder, 0, spot), "&quot;");
                remainder = substr (remainder, spot+1, -1);
        }
        str = string (str, remainder);
}
method AJAXSupport.EvalExpression(!expression)
{
        if (.allow_any_expression)
        {
                eval (expression);
        }
        else
        {
                error ("Arbitrary expression evaluation is disabled");
        }
}
method AJAXSupport.Test (!args?...=nil)
{
        string (.XMLHeader, .XMLHeaderSeparator, .XMLVersionString,
                        "<test><data name=\"test\" value=\"0\" args=\"",
                        .XMLEscape(stringc(args)), "\"/></test>");
}
```

```
}
method AJAXSupport.XMLEscape (str)
{
        local           remainder = str;
        local           spot;
        str = "";
        while ((spot = strchr(remainder, '&')) != -1)
        {
                str = string (str, substr(remainder, 0, spot), "&amp;");
                remainder = substr (remainder, spot+1, -1);
        }
        str = string (str, remainder);
        remainder = str;
        str = "";
        while ((spot = strchr(remainder, '"')) != -1)
        {
                str = string (str, substr(remainder, 0, spot), "&quot;");
                remainder = substr (remainder, spot+1, -1);
        }
        str = string (str, remainder);
}
/* This method is dangerous.  It could allow somebody to execute arbitrary
   code via an HTTP call.  If you absolutely need it then create a script
   to define it, and then be sure the web server port is only accessible
   from a trusted network. */
/*
method AJAXSupport.EvalExpression(!expression)
{
        if (.allow_any_expression)
        {
                eval (expression);
        }
        else
        {
                error ("Arbitrary expression evaluation is disabled");
        }
}
*/
method AJAXSupport.Test (!args?...=nil)
{
        string (.XMLHeader, .XMLHeaderSeparator, .XMLVersionString,
                        "<test><data name=\"test\" value=\"0\" args=\"",
                        .XMLEscape(stringc(args)), "\"/></test>");
}
```

Figure 9: Code differences before and after patching

The first portion of the patch removed *AllowExpressions* completely. This prevents an attacker from toggling that flag within the system. While the vendor did not remove *EvalExpressions* completely, it did add a comment noting that the use of the method represents a security risk.

```
/* This method is dangerous. It could allow somebody to execute arbitrary
code via an HTTP call. If you absolutely need it then create a script to
define it, and then be sure the web server port is only accessible from a
trusted network */
```

Cogent also commented out the offending code to prevent it from being called in by default. To enable the code, a developer must go in and manually uncomment the method. This method also makes it unlikely for Cogent to regress the bug at a later time.

Disclosure Statistics

Vulnerability Exposure Windows

When researchers do find vulnerabilities in SCADA products, the amount of time needed to fix the bug varies. This time is often referred to as the vulnerability exposure window. ZDI researchers reviewed all HMI vulnerabilities received through the program (more than 250) and measured how long they actually took to get fixed. As seen in the data over the last four years, the mean time to fix is not trending down. Since 2013, the average time between when ZDI researchers disclose a bug to the vendor and the time when a patch is released is right about 140 days.

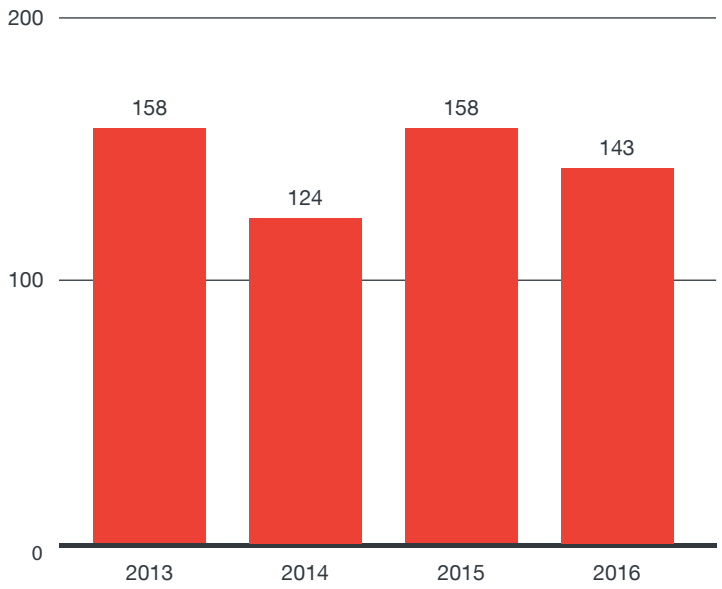


Figure 10: Mean time to patch vulnerabilities from the time they were disclosed by year

Patch quality also plays a role in this time line. A problematic update applied to a system could result in a self-induced denial of service (DoS) on a critical piece of infrastructure. Due to these and other concerns, there is a lag between the availability of a patch and the installation of that patch to a production system.

Not all SCADA vendors work under the same time lines. Certain vendors respond much more quickly than others. For example, ZDI has seen Cogent Real-Time Systems consistently respond quickly to bug reports. In one of the first disclosures to Cogent, the CEO actually emailed ZDI researchers to better understand the vulnerability and ensure it was quickly patched (six days). Larger vendors—ABB, GE, Indusoft, and PTC—take on average over 200 days to produce a patch.

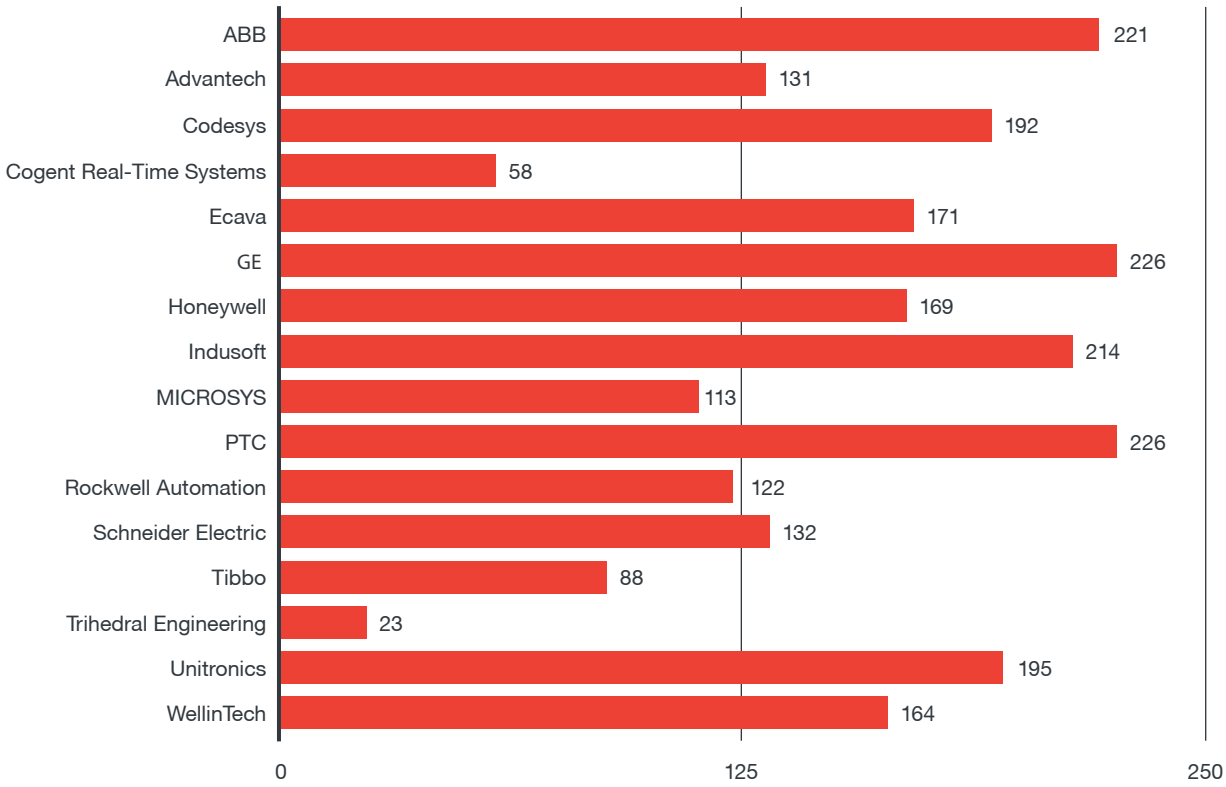


Figure 11: Mean time to patch vulnerabilities from the time they were disclosed by vendor

Industry-by-Industry Comparison

Comparing the response time of the SCADA industry to other industries reveals their response time is not unlike other industries. While the highly deployed software of large vendors such as Microsoft, Apple, Oracle, Adobe, and others have an average response time of under 120 days, SCADA and security software vendors' average around 150 days. The security software industry is slightly faster but not significantly. The business software category includes enterprise offerings from companies such as HPE and IBM. These vendors take significantly longer to address bug disclosures.

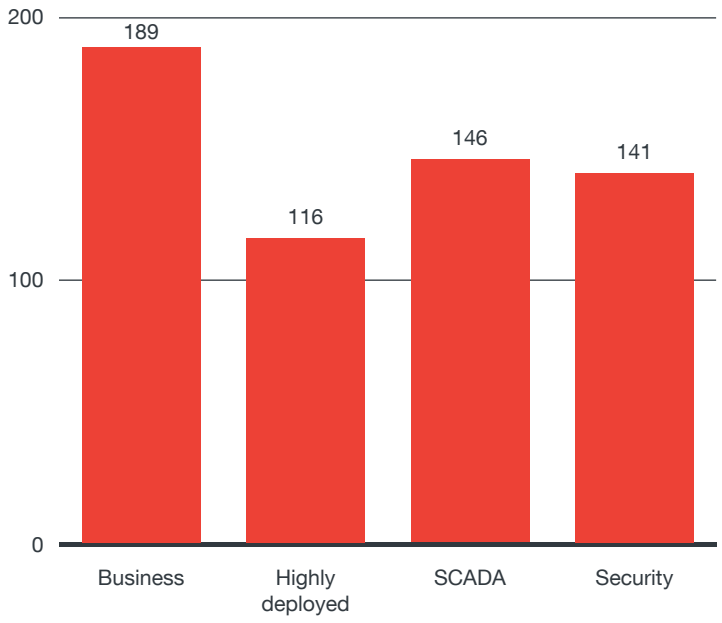


Figure 12: Mean time to patch vulnerabilities from the time they were disclosed by industry

Researcher Guidance

For researchers who are looking for vulnerabilities within HMI solutions, including vendors who are auditing their own solutions, there are steps you can take to help find bugs quickly and efficiently.

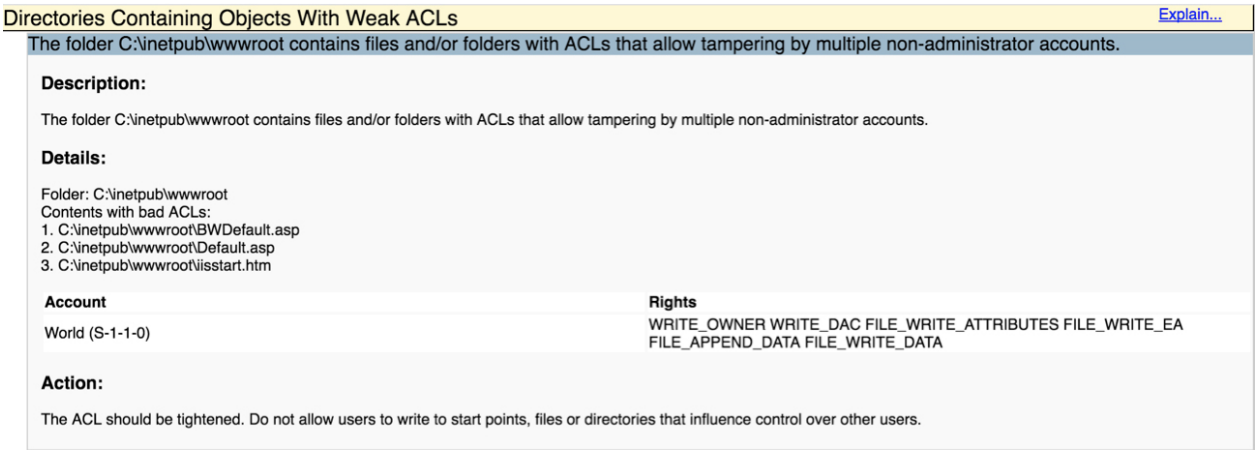
Basic Fuzzing

The first step to finding vulnerabilities within HMI solutions involves fuzzing. Even simple bit-flipping fuzzing produces highly effective results against HMI vulnerabilities. Researchers should look for new file associations during installation to aid in fuzzing, as many of the file formats are wide open. Also, make sure to enable page heap on the process that is being attacked to find heap corruption since then it breaks at the corruption point instead of later on when it is being used.

Attack Surface Analyzer

The Microsoft Attack Surface Analyzer (ASA)[19] assists developers in understanding changes in Windows systems' attack surface resulting from the installation of the applications the vendor develops. The tool creates system snapshots before and after installation. ASA also highlights security misconfigurations and increases in attack surfaces. These include such things as Component Object Model (COM) objects, ActiveX controls, file associations, and RPC endpoints.

For example, the output below shows the web root directory—a location where files are placed for execution in the high-privileged web server context—is currently set as world-writable.


Directories Containing Objects With Weak ACLs

Explain...

The folder C:\inetpub\wwwroot contains files and/or folders with ACLs that allow tampering by multiple non-administrator accounts.

Description:

The folder C:\inetpub\wwwroot contains files and/or folders with ACLs that allow tampering by multiple non-administrator accounts.

Details:

Folder: C:\inetpub\wwwroot
Contents with bad ACLs:
1. C:\inetpub\wwwroot\BWDefault.asp
2. C:\inetpub\wwwroot\Default.asp
3. C:\inetpub\wwwroot\iisstart.htm

Account	Rights
World (S-1-1-0)	WRITE_OWNER WRITE_DAC FILE_WRITE_ATTRIBUTES FILE_WRITE_EA FILE_APPEND_DATA FILE_WRITE_DATA

Action:

The ACL should be tightened. Do not allow users to write to start points, files or directories that influence control over other users.

Figure 13: Output showing the web root directory

Note that ASA currently does not work in Windows 10 but is still available for previous Windows versions.

Audit for Banned APIs

Since the introduction of the Security Development Lifecycle (SDL)[20] in 2006, Microsoft banned the use of problematic C library functions. Many APIs within the C runtime are known security issues and should be avoided. Our investigation found that these banned APIs and other functions are far too common in HMI code and have predictably negative effects. Use a disassembler such as IDA to trace the tainted data back to the source of the unsafe copy APIs *(sprintf, strcpy),* especially if an exploit can get to these functions from attacker-supplied data.

Conclusion

Within the various SCADA solutions, the HMI represents the clearest and most present target for attackers. The HMI acts as a centralized hub for managing critical infrastructure. If an attacker succeeds in compromising the HMI, nearly anything can be done to the infrastructure itself, including causing physical damage to SCADA equipment. Even if attackers decide not to disrupt operations, they can still exploit the HMI to gather information about a system or disable alarms and notifications meant to alert operators of danger to SCADA equipment.

In our research, we found that most HMI vulnerabilities fell into four categories—memory corruption, credential management, lack of authentication/authorization and insecure defaults, and code injection—all of which are preventable through secure development practices. We also observed that the average time between when ZDI researchers disclose a bug to a SCADA vendor and the time when a patch is released reaches up to 150 days, 30 more days than it would take vendors such as Microsoft or Adobe to do so, but 43 days less than enterprise offerings from companies such as HPE and IBM. Considering the impact of attacks against SCADA systems, where vulnerabilities are an effective entry point, our hope is that HMI vendors, SCADA owners, and administrators take notice and respond accordingly.

Researchers who are looking to find new vulnerabilities in HMIs should start with basic fuzzing techniques. Even simple bit-flipping fuzzing produces highly effective results against HMI vulnerabilities. Researchers should also look for new file associations during installation to aid in fuzzing, as many of the file formats are wide open.

Developers of HMI and SCADA solutions would be well advised to adopt the secure life cycle practices implemented by OS and application developers over the last decade. By taking simple steps such as auditing for the use of banned APIs, vendors can make their products more resilient to attacks. SCADA developers also need to expect their products to be used in manners that they did not intend. For example, even though it should be considered a poor security practice, developers must assume their products and solutions will be connected to a public network. By taking the mindset that assumes a worst-case scenario, developers can implement more defense-in-depth measures to add protection.

Malware specifically targeting ICS exists[21] and actively target HMIs. The ZDI program encourages researchers to find and report the bugs associated with HMI and other SCADA systems to our bounty program.[22] By working together, the researchers receive compensation for their work while the vendors receive data on where their products can be improved. Bugs in SCADA systems will likely be with us for many years to come. By working together, the security of these systems will continue to improve. While a completely secure system will likely never be created, implementing strong research and development tactics will be our best chance to keep the lights on as long as needed.

References

1. Kim Zetter. (2014). *Amazon*. “Countdown to Zero Day: Stuxnet and the Launch of the World’s First Digital Weapon.” Last accessed on 23 February 2017, https://www.amazon.com/Countdown-Zero-Day-Stuxnet-Digital/dp/077043617X.

2. Alex Gibney. (2016). *IMDB*. “Zero Days.” Last accessed on 23 February 2016, http://www.imdb.com/title/tt5446858/.

3. Jeremy Kirk. (8 November 2012). *ComputerWorld*. “Siemens Software Targeted by Stuxnet Still Full of Holes: Details from a Cancelled Defcon Presentation Were Revealed on Thursday in Seoul.” Last accessed on 23 February 2017, http://www.computerworld.com/article/2493358/security0/siemens-software-targeted-by-stuxnet-still-full-of-holes.html.

4. Michael B. Kelley. (20 November 2013). *Business Insider*. “The Stuxnet Attack on Iran’s Nuclear Plant Was ‘Far More Dangerous’ Than Previously Thought.” Last accessed on 23 February 2017, http://www.businessinsider.com/stuxnet-was-far-more-dangerous-than-previous-thought-2013-11.

5. US-CERT. (25 February 2016). *ICS-CERT*. “Alert (IR-ALERT-H-16-056-01): Cyber Attack Against Ukrainian Critical Infrastructure.” Last accessed on 23 February 2017, https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01.

6. Kim Zetter. (3 March 2016). *Wired*. “Inside the Cunning, Unprecedented Hack of Ukraine’s Power Grid.” Last accessed on 23 February 2017, https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/.

7. US-CERT. (2017). *ICS-CERT*. “The Industrial Control Systems Cyber Emergency Response Team (ICS-CERT).” Last accessed on 23 February 2017, https://ics-cert.us-cert.gov/.

8. Homeland Security. (2015). “NCCIC/ICS-CERT Year in Review: National Cybersecurity and Communications Integration Center/Industrial Control Systems Cyber Emergency Response Team, FY 2015.” Last accessed on 23 February 2017, https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/Year_in_Review_FY2015_Final_S508C.pdf.

9. TechTarget. (2000–2017). *Search Security*. “DEFINITION: Address Space Layout Randomization (ASLR).” Last accessed on 23 February 2017, http://searchsecurity.techtarget.com/definition/address-space-layout-randomization-ASLR.

10. Microsoft. (2017). *Microsoft Developer Network*. “/SAFESEH (Image Has Safe Exception Handlers).” Last accessed on 23 February 2017, https://msdn.microsoft.com/en-us/library/9a89h429.aspx.

11. Microsoft. (2017). *Microsoft Developer Network*. “/GS (Buffer Security Check).” Last accessed on 23 February 2017, https://msdn.microsoft.com/en-us/library/8dbf701c.aspx.

12. Advantech Co. Ltd. (1983–2017). *Advantech*. “WebAccess HMI/SCADA Software.” Last accessed on 23 February 2017, http://www.advantech.com/products/webaccess-hmi-scada-software/sub_gf-1m94v.

13. Brandon Bray. (February 2002). *Microsoft Developer Network*. “Compiler Security Checks in Depth.” Last accessed on 23 February 2017, https://msdn.microsoft.com/library/aa290051.aspx.

14. Michael Howard. (June 2011). *Microsoft Developer Network*. “Security Development Lifecycle (SDL) Banned Function Calls.” Last accessed on 23 February 2017, https://msdn.microsoft.com/en-us/library/bb288454.aspx.

15. Trend Micro Inc. (2017). *TippingPoint Zero Day Initiative*. “Disclosure Policy.” Last accessed on 23 February 2017, http://www.zerodayinitiative.com/advisories/disclosure_policy/.

16. *HeidiSQL*. (2017). “What’s This?” Last accessed on 23 February 2017, https://www.heidisql.com/.

17. US-CERT. (13 October 2016). *ICS-CERT*. “Advisory (ICSA-16-215-02A): Siemens SINEMA Server Privilege Escalation Vulnerability (Update A).” Last accessed on 23 February 2017, https://ics-cert.us-cert.gov/advisories/ICSA-16-215-02.

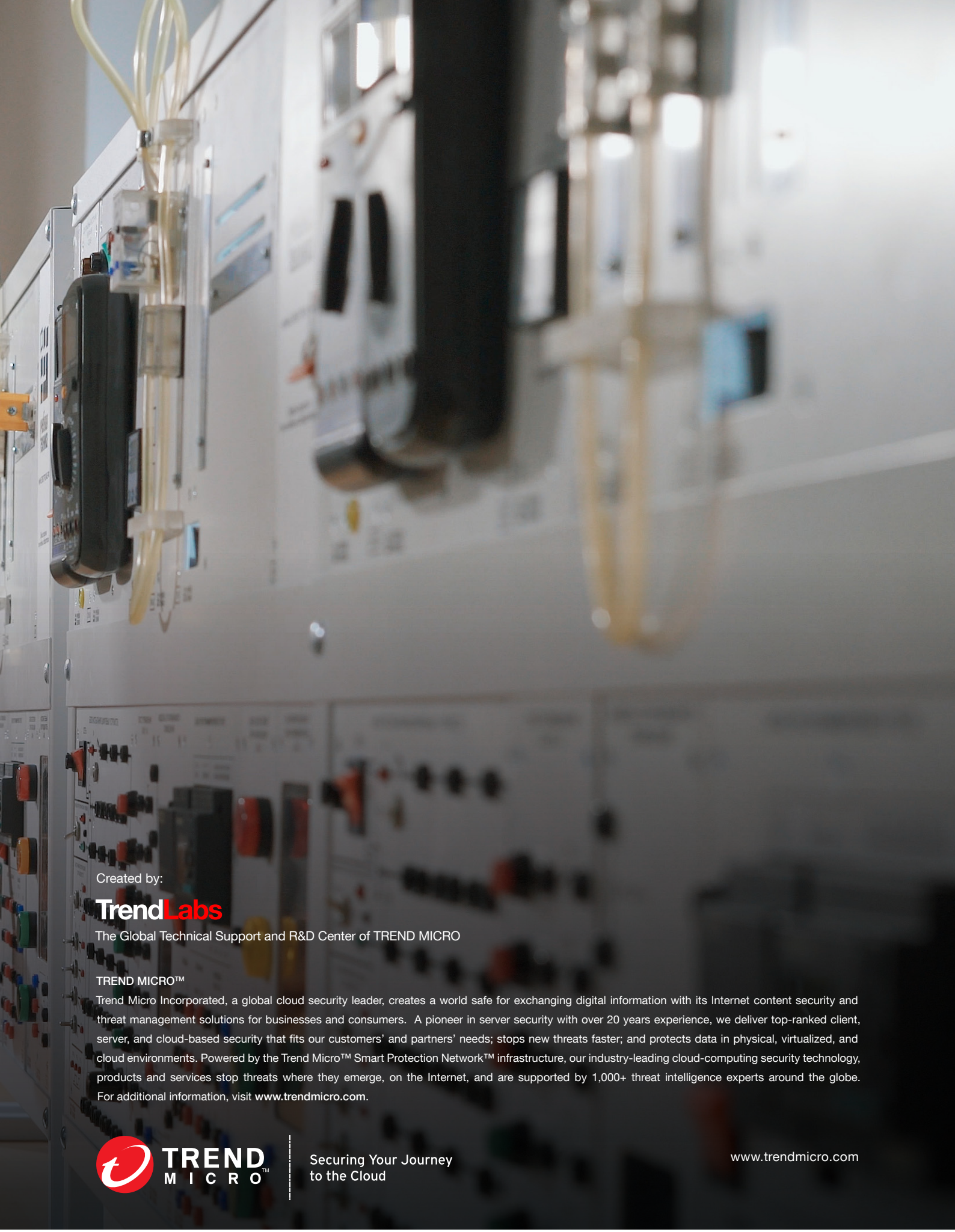
18. Advantech Co. Ltd. (1983–2017). *Advantech*. “Advantech WebAccess.” Last accessed on 23 February 2017, http://www.advantech.com/industrial-automation/webaccess.

19. Tim Rains. (2 August 2012). *Microsoft Secure Blog*. “Microsoft’s Free Security Tools—Attack Surface Analyzer.” Last accessed on 23 February 2017, https://blogs.microsoft.com/microsoftsecure/2012/08/02/microsofts-free-security-tools-attack-surface-analyzer/.

20. Microsoft. (2017). *Microsoft*. “What Is the Security Development Lifecycle?” Last accessed on 23 February 2017, https://www.microsoft.com/en-us/sdl/default.aspx.

21. US-CERT. (1 July 2014). *ICS-CERT*. “Advisory (ICSA-14-178-01): ICS Focused Malware.” Last accessed on 23 February 2017, https://ics-cert.us-cert.gov/advisories/ICSA-14-178-01.

22. Trend Micro Incorporated. (2017). *TippingPoint Zero Day Initiative*. “Program Benefits.” Last accessed on 23 February 2017, http://www.zerodayinitiative.com/about/benefits/.



Created by:

TrendLabs

The Global Technical Support and R&D Center of TREND MICRO

TREND MICRO™

Trend Micro Incorporated, a global cloud security leader, creates a world safe for exchanging digital information with its Internet content security and threat management solutions for businesses and consumers. A pioneer in server security with over 20 years experience, we deliver top-ranked client, server, and cloud-based security that fits our customers' and partners' needs; stops new threats faster; and protects data in physical, virtualized, and cloud environments. Powered by the Trend Micro™ Smart Protection Network™ infrastructure, our industry-leading cloud-computing security technology, products and services stop threats where they emerge, on the Internet, and are supported by 1,000+ threat intelligence experts around the globe. For additional information, visit **www.trendmicro.com**.

TREND MICRO™

Securing Your Journey to the Cloud

www.trendmicro.com