CAN I SEE SOME IDENTIFICATION?

DETECTING AND PATCHING SSL SOURCE CODE VULNERABILITIES

by

JEREMY LIPPS

A THESIS

Presented to the Department of Computer and Information Science
and the Robert D. Clark Honors College
in partial fulfillment of the requirements for the degree of
Bachelor of Science

June 2015

An Abstract of the Thesis of

Jeremy Lipps for the degree of Bachelor of Science
in the Department of Computer and Information Science to be taken June 2015

Title: Can I See Some Identification?: Detecting and Patching Source Code Vulnerabilities

Approved: [illegible]

Boyana Norris

This paper reflects research with the goal of building source analysis of security vulnerabilities for poorly written or faulty code intended to connect two parties via online interaction. Today's world is becoming more inundated with technology and increased digital functionality through the use of the Internet, and as a result code libraries have been built to support these data transfers. However, these libraries still contain unsafe code and often lack the ability to inform developers of improper usages of the libraries' tools. In this proof of concept project, the research uses the C programming language and the ROSE compiler to search through the libcurl SSL source code library in an effort to locate such problems and warn the developer of them. The libcurl variable *insecure_ok* was found to be uninitialized, and so code was built in order to find it and other such variables, as well as warn programmers of its potential dangers. These represent the first steps for further research into other problems within SSL libraries and improvement of checks within the SSLChecker suite.

Acknowledgements

First and foremost, I would like to thank Professor Norris for the hours upon hours of aid she provided me throughout the completion of this research. I knew from having had previous classes with her how generous and patient she was with her students, and so it was with little hesitation I approached her about finding a topic I could work on for my thesis. Despite her busy schedule, she always made time for me and helped me with every step along the way, from finding me resources to aid my research to consoling me when I occasionally felt lost and disheartened. I consider this thesis a joint project because I could not have done this without her.

Second, thank you to my other committee members, Professors Mossberg and Li. Professor Mossberg led my thesis prospectus class and never stopped encouraging her students to be passionate in their work and always believe in themselves. I could always count on her optimism and love for life to lift my spirits and motivate my work. Professor Li was kind enough to come on as my Second Reader and apply his many years of expertise in working with security to improve my work and understanding of the topic. He helped me to understand the incredible importance of security in the software world and the significant impact that each bit of code can have.

Last, I want to express my eternal gratitude to all my friends and family. If not for them, I would not be who I am or where I am today, and though I will never be able to fully reciprocate what they have given me, I won't waste a chance to thank them for their ceaseless love and support.

Table of Contents

List of Acronyms

API:	Application Programming Interface
AST:	Abstract Syntax Tree
IDE:	Integrated Development Environment
MITM:	Man In The Middle
SSL:	Secure Sockets Layer
TCP/IP:	Transmission Control Protocol/Internet Protocol
TSL:	Transport Layer Security

List of Figures

Introduction

Technological Growth

The advancements made in technology in recent years have prompted unprecedented interaction between customers and digital products, especially within the realms of e-commerce and the Internet. Online shopping has become a trillion dollar world economy, and while it still only represents single digit percentages of the retail market, its rate of growth far exceeds that of in-store shopping (Jordan, “The tipping point”). Even though the United States represents roughly a fifth of the world’s e-commerce on its own, a huge boom in mobile phone shopping has occurred in the populous and rapidly developing Asian-Pacific countries. Thanks to smartphones, many people now have convenient and “on-the-go” access to massive digital stores like Amazon and eBay for their shopping needs, especially if people are located in more isolated or rural areas. These stores are accessed through what are commonly referred to as “apps,” or software that is designed to improve consumer’s interaction with the store and products. Digital programs like these help dictate nearly every action an average citizen participates in while accessing the Internet, from web browsers that navigate the world wide web to company-specific apps that make using their product easier.

Moreover, mobile financial applications have become a huge part of developing countries, allowing isolated people and communities access to Internet stores, online banking, and even a local form of currency. Kenya and Uganda, for example, have become part of the vanguard of mobile financial services, simply through necessity of progression. With our current technological status, cell phone coverage is not a prohibitively expensive operation, even in poorer nations. In fact, even in rural African countries, 30% of households own cell phones (Voigt). This connectivity has enabled online interaction like never before, growing local economy by significant margins, simply by allowing businesses to set up shop and use e-currency for their goods or giving farmers the opportunity to check where they can find the best prices for their livestock.

Data Transfer and Security

Almost everyone knows that they need Internet access in order to connect with the products they want, play their games, or browse the web, but what is the actual behind-the-scenes process that allows them to do so? The short and woefully incomplete answer is that our network-connecting devices follow a series of rules, the Transmission Control Protocol/Internet Protocol (TCP/IP), that tell them how and when to transmit and receive packets of data. Since there is rarely a direct line from one computer to another, the data is sent over the Internet from one terminal to another, which includes personal computers, each of which then forward it onto the next destination until it finally reaches its goal. This data comprises all the visible aspects of interacting with the Internet, such a web page, and a lot of hidden information, such a computer's digital fingerprint or the encryption of the data. The security of the data being transferred is oftentimes overlooked because while we can physically prevent other people from seeing our screens, we rely on our device's software to protect our information.

One of the most common, current methods for keeping data private and secure is o use TSL/SSL (Transport Layer Security/Secure Sockets Layer), so called for the layering of data with encryption and the connected servers acting as two ends of a transfer. This is a system that makes use of cryptography to provide a secure and verified connection between two computers, so that the data being transferred cannot be read and abused by an outside source. The host server, i.e., the computer being accessed by the consumer, has purchased an SSL certificate from a web services company that essentially confirms that the purchaser is who they say they are through research and reference checking. Once confirmed, the certificate is awarded, thus allowing a company to verify their online identity, use a hosting server, and uniquely encrypt any data going into or leaving their online service. This all comes into play when consumers attempt to connect to the service through their own network devices. Initially, the consumer's device makes a connection to the website, but before transferring any data they verify the service with the SSL certificate. If that checks out, a connection, or "handshake," is formed between the two computers, and data flows back and forth, scrambled on departure and descrambled on arrival according to the certificate's encryption. This sort of security is widely used in web browsing, e-mail, messaging, and e-commerce, among others, because they are all areas that have the potential to reveal a great deal of personal and important information.

Man In The Middle Attacks

Despite all the money, effort, and research that has gone into improving the security of our Internet connections, virtual identity theft, hacked accounts, and stolen financial information are not uncommon news stories. In addition, online privacy rights have been a large issue in developed countries since the Internet's inception and widespread popularization. Edward Snowden achieved infamy recently for his release of government documents detailing some questionable investigative activities the United States government has performed on its citizens. In interviews he still advocates that people avoid using potentially insecure products like Dropbox, Facebook, Google, and unencrypted text messages (Snowden). The underlying similarity between products like these is the concern for the security of transferred information. For example, Snowden cites Dropbox because it only encrypts data during transfer, rather than while it is still on the machine, allowing a vulnerability in the case that some error occurs during the transfer or prior to the encryption, allowing an opportunity for a Man-in-the-Middle (MITM) attack to occur.

A MITM attack is when a third party is able to gain access to the connection between two parties by intercepting the data being transferred. Recalling the TCP/IP process mentioned earlier, data must travel through multiple machines until it reaches its destination, which means that if anyone manages to insert themselves in the data's path, the hijacker has a chance at eavesdropping on the digital conversation. The best opportunity for a person to hack the connection is by knowing the location of the sender or the receiver. For this reason, most MITM attacks occur on unsecured WiFi connections, but hackers can also wait outside popular server addresses like those of online banks. Here, there is a brief window in which the person can employ a variety of methods to gain the sender's trust, which include stealing the server's identification key, mimicking the server's certificate authority, or taking advantage of the sender's lack of validation process. If any of these work, an exchange of public encryption keys occurs such that the middleman can read the data sent by either side, as well as send data to one party under the guise of the other. A simplified visual representation of a common financial MITM attack is shown below:

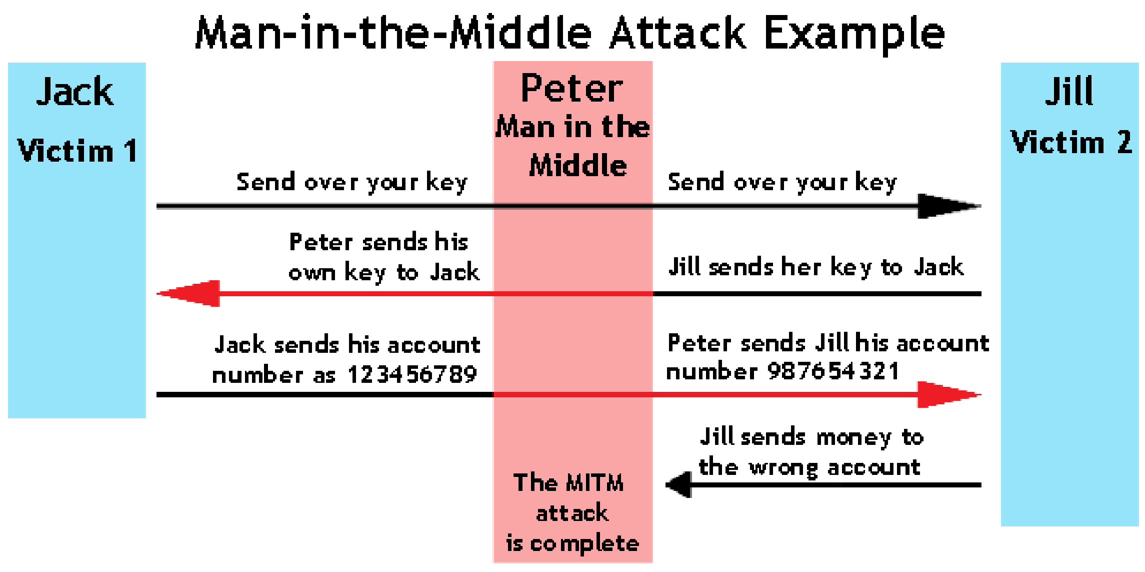

Man-in-the-Middle Attack Example
Jack
Victim 1
Peter
Man in the
Middle
Jill
Victim 2
Send over your key
Send over your key
Peter sends his own key to Jack
Jill sends her key to Jack
Jack sends his account number as 123456789
Peter sends Jill his account number 987654321
Jill sends money to the wrong account
The MITM attack is complete

Figure 1: MITM attack. veracode.com

MITM attacks are one of the greatest threats to personal information because if someone can get a hold of the digital key that unlocks the SSL encryption scheme, that person has free reign with the transferred data to decrypt it, read it, modify it, re-encrypt it with the other computer's key, and forward it. Although these attacks are often used by those with malicious motives, sometimes they are done to spy on and watch over others. Following Snowden's release on NSA surveillance and news coverage of strict government control over the Internet in other countries like China and North Korea, people are becoming more wary of George Orwell's *1984* Big Brother scenario turning from fiction to reality. However, those most at risk are the ones who use everyday mobile applications, often related to online shopping, as these programs are used the most often and have been found to have many susceptibilities. Studies have shown that many developers build broken code on purpose, or that after being notified of a vulnerability in their code they will choose not to fix it, or if they do then it often takes over a year (Bates et. al.). The report does not indicate why that is (a route for further research), but one possibility is simply a desire to save time, money, and energy by cutting corners, especially if the application works most of the time and is consequently easier to use. Considering our world's ever-growing e-market and pursuit of convenience through mobile devices, continued research in this field becomes proportionally more relevant and important in its efforts to sustain optimum security through the detection and prevention of vulnerabilities.

Research/Methods

Current Research

The current research was motivated by the report “Detecting and Patching Vulnerable SSL Source Code with ROSE” by Adam Bates, Braden Hollembaek, and Dave Tian of the University of Oregon. The report focuses on the inability of clients to accurately authenticate the server when presented with its public key certificate. Essentially, how do we as consumers know that the online service we are connecting to is legitimate? To give a real life example, imagine going through the checkout line at the supermarket and paying for your items with a credit card. There are a few under-the-surface assumptions being made that we tend not to think about during these interactions. One is that our money is being extracted in the correct amount from our bank account and that it is actually going to the store, rather than the cashier’s own account. Another is that our card’s information is not somehow being saved locally and abused at a later date. These are difficult for us to control or ensure, so we have to trust the system. Another assumption from the store’s end is that the person paying for the items is also the owner of the card. The cashier can ask, “Can I see some identification?” to validate that customer is the card’s owner, but that certainly doesn’t happen every time, which leaves an opportunity for a real-life MITM attack to use your information without your consent. Additionally, if the store doesn’t teach their employees how to verify a card’s owner, then it becomes impossible for the consumer to hope that in future interactions where the credit card is used, the owner will be validated.

The ultimate problem is that SSL certificate validation is broken in many critical software applications and libraries, primarily due to terrible design of the application programming interfaces, or APIs, to underlying SSL libraries. APIs are collections of code that provide similar services but allow a developer to implement those services according to their needs. An everyday example would be a deck of cards. Their general purpose is to be played with, but the execution is up to the user. A person can play any number of card games, either on their own or with others, with only their own rules to govern their actions. Once you own the cards, they simply provide a platform for your implementation. An example of an API on the Internet could be Facebook. If you have ever seen those Facebook buttons on websites asking you to like them or check out their page, that is a Facebook API. It is not Facebook itself, but rather a tool whose primary purpose is to link to Facebook, and which allows the website to customize its click destination, placement on the page, and so on. To understand the API problem as it relates to SSL, imagine that the Facebook button did not employ SSL checking properly, so when you were redirected and asked to sign into your account, someone else managed to steal your login information. Certificate validation in the initial handshake is critical to the success of a secure connection, so when code is created that ignores, breaks, or does not fully address this issue, man-in-the-middle attacks are much more likely to succeed. The research that's already been done on this issue has noticed that most of the fault lies with the developers themselves because the SSL libraries being used are mainly correct. Programmers simply don't use the provided code correctly, or they misunderstand the numerous options, parameters, and return values given by the library's working code. Thus, this research does not seek to improve existing libraries and SSL code, but rather identify when written code is failing to achieve the desired security result.

SSLChecker and ROSE

This research focuses on building an automatic and scalable program called SSLChecker for going through the original, or “source,” code of vulnerable SSL applications and making sure those validation checks are in place. This program makes use of ROSE, an open source compiler infrastructure developed at the Lawrence Livermore National Laboratory (Quinlan et. al.). Like a compiler, its job is to take in source code and convert it into different source or machine code so that the computer can read and understand the operations it is being asked to perform. In everyday life, this process could be compared to that of a grammar checker in a text editor. The editor does not actually understand what the sentence is saying or the context it is being written in. All it knows is that there are certain structural rules that clauses and sentences must follow in order to be considered valid, such as the necessary presence of a subject and a verb, or that two commas in a row does not make sense. The sentence is broken down to its component parts of speech and its punctuation, and from there a set of rules is consulted to make sure that they are all abiding by the grammatical laws of the language. So, after the compiler finishes the translation from source to machine code, it creates a new type of file called an object file that can be run and execute the actions dictated by the source code.

ROSE provides source code parsing for various programming languages, as well as a variety of program analysis and transformation tools. This is because as the code is being compiled and the component parts are being identified, ROSE builds a tree data structure, setting each part as its own “node” with a ROSE equivalent explanation of what the node’s function is. In effect, ROSE is making the code transformation processes available to the programmer in more a more readable and analyzable fashion, specifically through the creation of a traversable abstract syntax tree, or AST. An AST is an inverted tree-looking representation of source code’s syntactic structure detailing the component parts, their locations in the code, and their relationships to one another. Because ROSE has its own nomenclature for these parts, ROSE can implement a unique traversal method to visit every node and get its information, through a variation on the Visitor Pattern. Depending upon the programmer’s preference, ROSE can then display the information, change the node, or simply move to the next node. Below is a small example of an abstract syntax tree with its associated code, though the reader should keep in mind the number of nodes present for only five lines of simple code, as well the fact that ROSE’s complex infrastructure would add many additional layers for the same code:

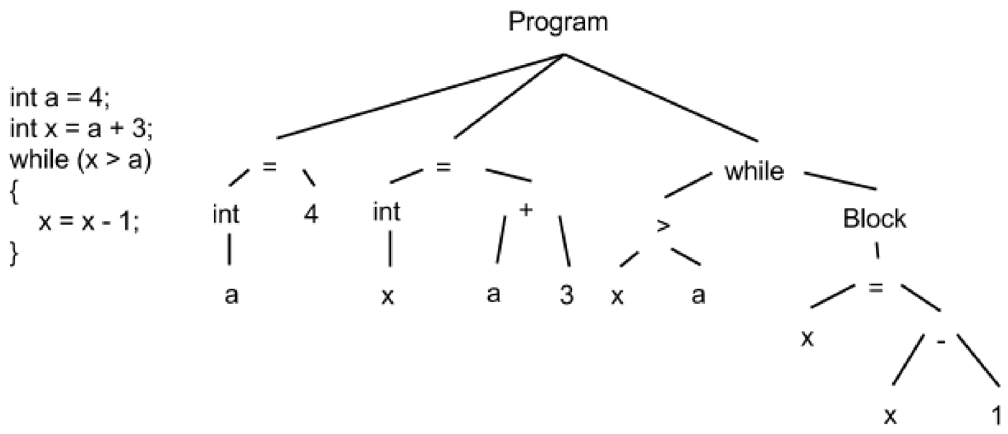


Figure 2: A Syntax Tree representation example for a small program.

Using ROSE to step through every piece of the source code, automatic modifications can be made to the source code if SSLChecker finds an absence of a critical verification method or mistake in the verification process. So, if the user fails to make the host server identify itself, that code will be added directly into the application. This result is designed to create an “idiot-proof” method for fixing, alerting, and optimizing code automatically, simply telling the developer what was wrong and how it was fixed, hopefully providing an incentive and means for easier code improvement. Although a prototype SSLChecker has already been built, the “Detecting and Patching Vulnerable SSL Source Code with ROSE” report details a great many ways that the software could be improved upon moving forward, and as of now this is something the research will focus on. Some examples include adding checks for non-existent but necessary methods for verification, analyzing the returned statuses of the validation functions, detecting which library is being used to build the application software so that only the appropriate checks are being run, and running the checker on unfamiliar software.

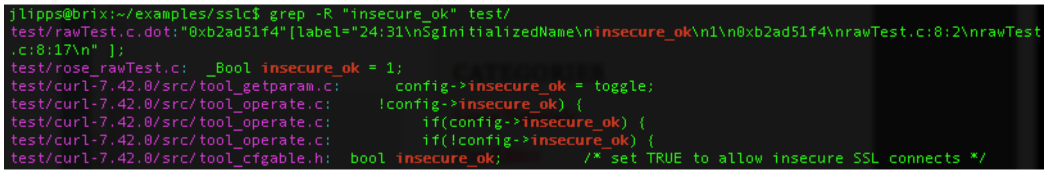
Process

Language and Library

The process began by identifying which programming language the team wanted to analyze with SSLChecker and ROSE, as they both have the capacity of working with C, C++, Java, Python, and PHP. The plan at the beginning of the project was to work with Java in order to make use of the Eclipse IDE as both a helpful programming tool and hopefully allowing us to incorporate any changes made into Eclipse's automated code checking itself. However, it was later decided that C would be the best language to begin in because it is the most stable environment for both SSLChecker and ROSE, as well as the fact that libcurl's SSL source code base could be analyzed. The libcurl library is a collection of URL transfer protocols written in C (hence the lib-c-url name) that purports to allow a programmer to establish secure online connections, including HTTPS and SSL certificates (Stenberg et. al.). Many large and well-known companies like Apple, Adobe, and Google use libcurl for their online platforms, so it is important that the library and its security-based connections are as safe as possible to prevent MITM attacks. A potential end goal for this project is to be able to run SSLChecker on a large source code library like libcurl, hone in on the uses of SSL certificate verification, and then find cases where either the correct SSL verification functions are absent, or where critical variables are set incorrectly.

Insecure_ok

Because ROSE is a complicated environment where many lines of code are needed to sift through ROSE's tree structure correctly, we decided to start by finding one instance of a variable that was created, or “declared”, unsafely and then writing code to find it. Thus, we searched through libcurl’s library and found a Boolean (true or false) variable *insecure_ok* that matched our requirements for faulty variables**.** Figure 1 below shows all the instances of *insecure_ok* in the libcurl library, and what is interesting to note is that when the variable was declared, it was not assigned a constant value of any sort, meaning that it was potentially uninitialized.



```
jlipps@brix:~/examples/sslc$ grep -R "insecure_ok" test/
test/rawTest.c.dot:"0xb2ad51f4"[label="24:31\nSgInitializedName\ninsecure_ok\n1\n0xb2ad51f4\nrawTest.c:8:2\nrawTest
.c:8:17\n" ];
test/rose_rawTest.c:  _Bool insecure_ok = 1;
test/curl-7.42.0/src/tool_getparam.c:      config->insecure_ok = toggle;
test/curl-7.42.0/src/tool_operate.c:     !config->insecure_ok) {
test/curl-7.42.0/src/tool_operate.c:         if(config->insecure_ok) {
test/curl-7.42.0/src/tool_operate.c:         if(!config->insecure_ok) {
test/curl-7.42.0/src/tool_cfgable.h:  bool insecure_ok;         /* set TRUE to allow insecure SSL connects */
```

Figure 3: The accumulated instances of *insecure_ok*. Personal screenshot.

An analogy for an uninitialized variable in the English language could be explained via the sentence, “Bob went to the store and it was good.” Whatever problems the sentence might have, let us focus on the use of the word *it*. When we say *it* was good, what does that mean? Was *it* the trip to the store, the store, the shopping experience, Bob himself, or something else entirely? In this instance, *it* represents our uninitialized variable and creates an issue because it was not explicit enough. Uninitialized variables are often a cause of bugs in programs because the language can set that variable to any value. Some programming languages like Java and Python have built in checks, such as not allowing the use of such variables, but C was designed for systems programming in which developers were aware of the dangers uninitialized variables posed to performance. In C, variables are allocated stack space, and a collection of these spaces make up a stack frame. An uninitialized variable can then be assigned the value of where the stack pointer is located, which is typically a virtual address within the computer. Thus, the problem is that *insecure_ok* has an arbitrary value when it is initialized, and if it is used before it is given a constant **true**/**false** value, like when `tool_operate.C` checks its value in Figure 1, an error can occur if the value is not a constant. Alternatively, a hacker might manage to force the system to give a certain value to the uninitialized variable, that would set *insecure_ok* to true, allow for insecure SSL connections to go through, and open the connection to a MITM attack.

Like the paper mentioned before, this research builds off of the previously constructed SSLChecker code to create additional checks for problems like the one mentioned above. The file `sslc_c.C` was part of that suite and checked C code using ROSE, so the team modified the `visitorTraversal` function of the program, which visits and identifies every component node, to run additional checks for this variable and others like it. In ROSE, this involves identifying all variable declarations that exist within the source code, then consecutively ensuring that an initializer and assignment initializer both exist. If the initializer exists, then we know that there is an equals sign indicating that the variable is being initialized to some value. If the assignment initializer is not empty or set to a **NULL** value, then we know that some value is being assigned by the developer to the variable. At this point, the right-hand-side operand can be checked for its type, such as an integer, Boolean, text string, or some other constant. If it is a static and unchanging constant, then we know the variable is properly initialized. This does not necessarily ensure that the constant being assigned is correct, but that is beyond the scope of this research. However, if it turns out that one variable is being assigned the value of another variable, then the cycle of safety checking begins anew with the new variable. It could be the case that when it is declared elsewhere in the library, perhaps even in another file or directory, it is uninitialized, given a bad value, or assigned to yet another variable. Hence, we conservatively check for assignment with a constant on the right-hand-side.

In addition, a check for global variables is included in order to provide greater analysis and understanding of the variable. Global variables are declared outside of any functions and can be used anywhere in the code. When a global variable's value is set or changed in one location, every instance of the variable is updated with the new value. This is distinguished from local variables which may be declared and used in a single function. While global variables can provide some usefulness in simplifying code by making it easily accessible to all the functions and classes in a program, it is common practice to avoid them for their potentially dangerous consequences. The nonlocality of globals makes testing them, constraining their program-wide influence, preventing simultaneous usage, and optimizing memory allocation difficult, and when these factors are not controlled, errors and bugs become more likely.

Specifically for this research, assigning the value of a global variable to another variable may make tracking the value to its source more straightforward, but it also leaves room for other variables of the same name as the global to change the value to an unallowed type or constant. The variable *insecure_ok* is a global variable, though as can be seen from Figure 1, the uses are limited and easily tracked. Searching for the declaration of *toggle*, the value given to *insecure_ok* after its declaration, in the libcurl library, its only usage is inside the `tool_getparam.c` file, and it is even initialized to **TRUE**. Upon closer examination, however, *toggle* can be switched from **FALSE** depending upon the input parameters. Thus, a user decides whether to allow for Boolean values to be used in their program by inputting their own argument at launch-time. Though this by itself does not cause an issue in the program, leaving an opportunity available for such uncontrolled and unrestricted input could be a source for error in the future.

Results and the Future

Though the newly added checks to the SSLChecker suite might not sound extensive, the importance of this first step should not be underestimated. The fact is that those simple but hard-fought for steps act as a proof of concept for greater changes to be made and improved upon within the pre-established suite. There already exists a problem in software development where programmers ignore warnings or suppress them entirely for the sake of creating and using workable code. However, the fact that they exist at all to be examined and fixed is the key to optimizing code, and so this research attempts to improve upon SSLChecker's functionality by adding those types of warnings when faulty code is found. Simply by warning any programmer that uses this software on a large SSL source code library that potentially dangerous uninitialized variables exist, or that critical functions for ensuring the security of data via the SSL process are missing or set incorrectly, previously unknown problems emerge for analysis and correction.

With software, it is important to understand that even small changes can have an enormous impact. If using the wrong type of variable can cost the European Space Agency $7.5 billion with the explosion of the Ariane 5 rocket, then it is not a stretch to say that any size of fix to SSL source code libraries could save people and their banks an enormous amount of time, stress, and money over the years. With identity fraud occurring once every two seconds, resulting in billions of dollars stolen each year, providing increased security and safer SSL libraries should be a priority. Especially as continuously more people enter into the digital methods of banking, shopping, and transferring money, it should be our duty as programmers to ensure that such widely used services provide the best possible experience for their users, including ourselves. That is why it is imperative that this research continues so that the checks established now can be expanded upon to work with different programming languages and SSL libraries. Additionally, these checks should be incorporated into the programming integrated development environments, or IDEs (where code is written), so that rather than run the third party SSLChecker on a library, the checks can be done automatically as the code is being written for the convenience of the developer. This way, the problems are addressed at the source, rather than after the software has been completed and distributed.

The digital revolution has brought about miraculous advancements to our world in such a short amount of time, and the evolution of technology still increases exponentially. Meanwhile, these same changes have also brought with them their fair share of problems, demonstrating that the results of this progress are merely tools that can be used for either good or bad. The Internet has undoubtedly changed the world, though being as imperfect as it is, much of its effect on society and people has been negative. Keeping this in mind, however, and understanding the role SSL connections play in netizen (net citizen) interactions with the web, are some of the most impactful ways that we can help alleviate these burdens for ourselves and all the netizens to come.

Bibliography

Bates, A., Hollembaek, B., and Tian, D. “Detecting and patching vulnerable SSL source code with ROSE”. Tech. Rep. Department of Computer and Information Science, University of Oregon, Eugene, OR, USA, May 2014.

Conti, M., Dragoni, N., and Gottardo, S. “MITHYS: Mind the hand you shake - protecting mobile devices from SSL usage vulnerabilities”. In *Security and Trust Management*, R. Accorsi and S. Ranise, Eds., vol. 8203 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 65-81.

Ducklin, Paul. "The TURKTRUST SSL certificate fiasco - What really happened, and what happens next?" Naked Security. SOPHOS, 08 Jan. 2013. Web. 21 Nov. 2014. <https://nakedsecurity.sophos.com/2013/01/08/the-turktrust-ssl-certificate-fiasco-what-happened-and-what-happens-next/>.

DuPaul, Neil. "Man in the middle (MITM) attack." Man in the Middle Attack: Tutorials & Examples. Veracode, n.d. Web. 2 Apr. 2015. <http://www.veracode.com/security/man-middle-attack>.

Ellis, Blake. "Identity fraud hits new victim every two seconds." *CNNMoney*. Cable News Network, 6 Feb. 2014. Web. 13 May 2015. <http://money.cnn.com/2014/02/06/pf/identity-fraud/>.

Fahl, S., Harbach, M., Perl, H., Koetter, M., and Smith, M. “Rethinking SSL development in an applied world”. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer; Communications Security* (New York, NY, USA, 2013), CCS '13, ACM, pp. 49-60.

Fisher, D. “Microsoft Revokes Trust in Five Diginotar Root Certs”. Wired. Sept. 2011. <http://threatpost.com/microsoft-revokes-trust-five-diginotar-root-certs-mozilla-drops-trust-staat-der-nederland-cert>.

Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., and Shmatikov, V. “The most dangerous code in the world: Validating SSL certificates in non-browser software”. In *Proceedings of the 2012 ACM conference on Computer and communications security* (Raleigh, NC, USA, 2012), CCS '12, ACM, pp. 38-49.

Jordan, Jeff. "The tipping point (e-commerce version)." Recode, 14 Jan. 2014. Web. 5 Feb. 2015. <https://recode.net/2014/01/14/the-tipping-point-e-commerce-version/>.

Marlinspike, M. “New tricks for defeating SSL in practice”. *BlackHat DC*, Feb. 2009.

Mills, E. “Comodo: Web attack broader than initially thought”. CNET, March 2011. <http://news.cnet.com/8301-27080_3-20048831-245.html?part=rss&tag=feed&subj=InSecurityComplex>.

Nichols, T., Bates, A., Pletcher, J., Hollembaek, B., Tian, D., Alkhelaifi, A., and Butler, K. R. “Talk certy to me feat. 2 Chainz: Securing SSL certificate verification through dynamic linking”. Tech. Rep. TR- 201405-01, Department of Computer and Information Science, University of Oregon, Eugene, OR, USA, May 2014.

Quinlan, Daniel J., Chunhua Liao, Justin Too, Robb P. Matzke, and Markus Schordan. *ROSE Compiler Infrastructure*. Lawrence Livermore National Laboratory, n.d. Web. 12 January 2015. <rosecompiler.org/>.

Snowden, Edward. "The Virtual Interview: Edward Snowden at the New Yorker Festival." Interview by Jane Mayer. The New Yorker, 11 Oct. 2014. Web. 17 Mar. 2015. <http://www.newyorker.com/new-yorker-festival/live-stream-edward-snowden>.

Sounthiraraj, D., Sahs, J., Greenwood, G., Lin, Z., and Khan, L. “SMV-HUNTER: Large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in android apps”. In *Proceedings of the 19th Network and Distributed System Security Symposium.* (2014).

Stenberg, Daniel, Dan Fandrich, and Yang Tse. “Libcurl - the multiprotocol file transfer library”. Computer software. Haxx AB, n.d. Web. 4 Apr. 2015. <http://curl.haxx.se/libcurl/>.

United States. Census Bureau. Dept. of Commerce. Economics and Statistics Administration. "E-Stats 2013: Measuring the electric economy." Washington: US Census Bureau, 28 May 2015. Web. 28 May 2015. <https://www.census.gov/content/dam/Census/library/publications/2015/econ/e13-estats.pdf>.

Voigt, Kevin. "Mobile phone: Weapon against global poverty." *CNN Tech*. CNN, 09 Oct. 2011. Web. 28 Oct. 2014. <http://www.cnn.com/2011/10/09/tech/mobile/mobile-phone-poverty/>.