AFRL-RI-RS-TR-2013-188

VULNERABILITY ASSESSMENT OF OPEN SOURCE WIRESHARK AND CHROME BROWSER

UNIVERSITY OF WISCONSIN

AUGUST 2013

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2013-188 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /

JAMES L. SIDORAN
Work Unit Manager

/ S /

WARREN H. DEBANY, JR
Technical Advisor, Information
Exploitation and Operations Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
AUG 2013	FINAL TECHNICAL REPORT	NOV 2009 – JAN 2013

4. TITLE AND SUBTITLE

VULNERABILITY ASSESSMENT OF OPEN SOURCE WIRESHARK AND CHROME BROWSER

5a. CONTRACT NUMBER
FA8750-10-2-0030

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
69199F

6. AUTHOR(S)
Barton P. Miller

5d. PROJECT NUMBER
HS33

5e. TASK NUMBER
TM

5f. WORK UNIT NUMBER
03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Wisconsin
1210 W. Dayton Street
Madison, WI 53706-1685

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2013-188

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The objective of this effort was to conduct an in depth vulnerability assessment of the Wireshark network protocol monitoring system. An in-depth assessment using First Principles Vulnerability Assessment (FPVA) methodology was performed that produced architectural, resource, privilege and trust analyses of the code, which, in turn, identified several verified security vulnerabilities. In addition, a similar analysis on the Google Chrome/Chromium web browser was performed, producing similar products and a vulnerability report. Other accomplishments included new work on tools to speed the task of analyst-driven vulnerability assessment of code, new techniques for statically analyzing source code for defects, and useful collaborations with industry and academia.

15. SUBJECT TERMS

First principles vulnerability assessment, in-depth code analysis, Wireshark, Google Chrome

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**JAMES L. SIDORAN**
U	U	U	UL	19	19b. TELEPONE NUMBER (*Include area code*) **N/A**

Contents

LIST OF FIGURES

1.0 SUMMARY

The primary task under this funding was to conduct an in depth vulnerability assessment of the Wireshark network protocol monitoring system. In this task, we successfully performed an in-depth assessment using our First Principles Vulnerability Assessment (FPVA) methodology, producing architectural, resource, privilege and trust analyses of the code, and resulting in reporting several verified security vulnerabilities. In addition, we performed a similar analysis on the Google Chrome/Chromium web browser, producing similar products and a vulnerability report.

Other accomplishments under this funding include new work on tools to speed the task of analyst-driven vulnerability assessment of code, new techniques for statically analyzing source code for defects, significant collaborations with industry and academia, and technical and tutorial presentations at workshops, conferences, companies, and government meetings.

2.0 INTRODUCTION

The research performed under this contract has included the following agenda:

1. Wireshark assessment: We performed an in-depth vulnerability assessment of the open source Wireshark protocol monitor and analyzer. This assessment followed our FPVA methodology and produced vulnerability reports and suggested remediations. As part of this assessment process, we demonstrated a new threat model for Wireshark.
2. Dissemination and analyst training: We continued to develop our tutorial materials. We teach analysts how to use FPVA and software developers how to avoid programming practices that can cause vulnerabilities. We have taught these tutorials in a wide variety of venues.
3. Vulnerability characterization and automation algorithms: Our past assessment activities have found serious vulnerabilities that could not be found with the best commercial tools. We used our database of vulnerabilities, along with those collected by other organizations, to identify common characteristics and formally describe these characteristics. We have been using these characteristics to automated analysis algorithms to find vulnerabilities that share these characteristics.
4. System analysis tools: For the part of the assessment process that cannot be automated, we must provide tools that improve the productivity of the analyst. To this end, we have furthered our work on tools that help automate the identification of the structural characteristics of a software system and the resources and privilege levels used by each component.
5. Technology transfer to industry: The code analysis industry is constantly trying to improve their products and is open to academic collaborations. We have had discussions with major vendors in the software analysis field, giving them access to our database of vulnerabilities.

6. Additional software assessments: In addition to our assessment of Wireshark, we also worked with Google to performance an in depth assessment of the widely used Chrome/Chromium web browser. This design of Chrome and the web environment in general led to new challenges for our FPVA methodology. We were able to make good progress on this assessment, including reporting a serious, exploitable vulnerability to Google.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Wireshark Assessment

The primary task under this funding was to conduct an in depth vulnerability assessment of the Wireshark network protocol monitoring system. Wireshark is used worldwide as a diagnostic, security, and forensics tool, including heavily used by U.S. federal and state law enforcement agencies. As Wireshark is an open source system with hundreds of authors from dozens of countries, its reliability, security, and veracity as an investigative tool might be called into question. We applied our assessment methodology to increase our confidence that it cannot be reasonably exploited or manipulated in producing its results. Interestingly, while we did not find any signs of intentional manipulations of the Wireshark code, we did find two vulnerabilities, one where malicious traffic from the system being monitored could be hidden and the other where Wireshark itself could be used to transmit malicious data out of the system being monitored.

We developed an analyst centric vulnerability assessment methodology, called *First Principles Vulnerability Assessment* (FPVA). This technique allows us to evaluate the security of a system in depth. We assume access to the source code, documentation, and, when available, the developers. While FPVA is certainly a labor-intensive approach to vulnerability assessment, it is effective in real systems, finding serious vulnerabilities that cannot discovered through the use of the best automated tools of today.

Rather than working from known vulnerabilities, the starting point for FPVA is to identify *high value assets* in a system, i.e., those components (for example, processes or parts of processes that run with high privilege) and resources (for example, configuration files, databases, connections, devices) whose exploitation offer the greatest potential for damage by an intruder. From these components and resources, we work outward to discover execution paths through the code that might exploit them. This approach has a couple of advantages. First, it allows us to find new vulnerabilities, not just exploits based on those that were previously discovered. Second, when vulnerability is discovered, it is likely to be a serious one whose remediation is of high priority.

FPVA starts with an architectural analysis of the code, identifying the key components in a system. It then goes on to identify the resources associated with each component, the privilege level of each component, the value of each resource, how the components interact, and how trust is delegated. The results of these steps are documented in clear diagrams that provide a roadmap

for the last stage of the analysis, the manual code inspection. In addition, the results of this step can also form the basis for a risk assessment of the system, identifying which parts of the system are most immediately in need of evaluation. After these steps, we then use code inspection techniques on the critical parts of the code. Our analysis strategy targets the high value assets in a system and focuses attention on the parts of the system that are vulnerable to not only unauthorized entry, but unauthorized entry that can be exploited.

Our first step was to understand the structure, including the architectures, resources, and use of privilege and trust in Wireshark. Wireshark consists of **almost 2.5 million lines of code**, so this was a significant task. However, the overall structure of Wireshark turns out to be fairly simple, with approximately 1.7 million lines due to the more than 800 protocol dissectors. These dissectors allow Wireshark to interpret a huge variety of network protocols, including many rarely used and proprietary ones. Figure 1 is the basic diagram from this early stage of our analysis. The value of this diagram is that it captures a clear and simple view of this very large system.

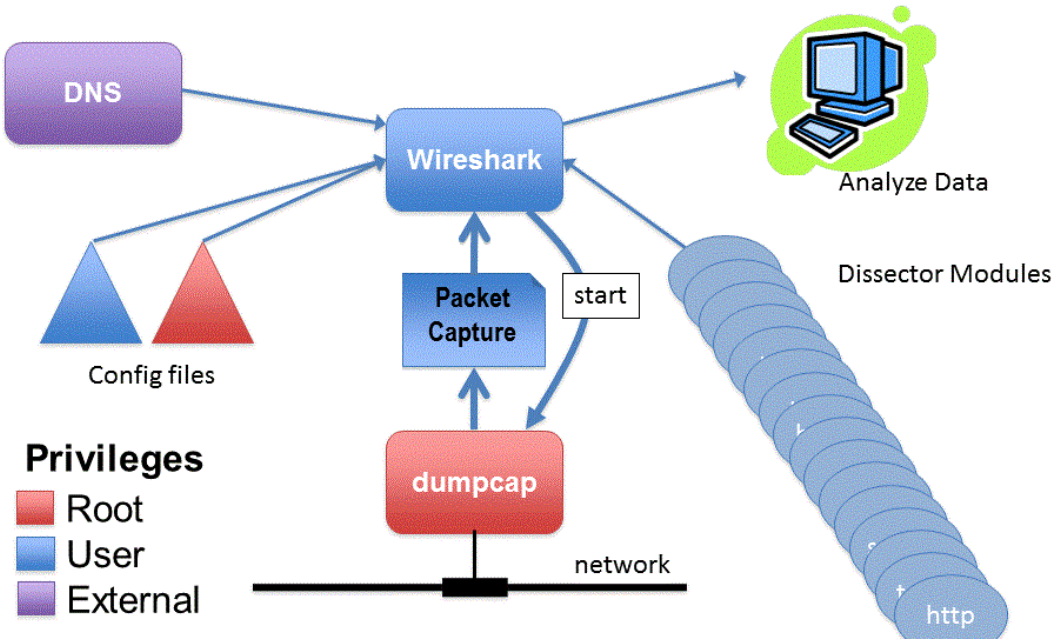

Figure 1: Wireshark Architecture and Resources

Once we had completed these early stages of FPVA analysis of Wireshark, we then moved on to the detailed assessment of the component code, with our attention focused by the previous stages. The results of our assessment of Wireshark appear in Section 4.0.

3.2 Chrome Assessment

Following the assessment of Wireshark, we began assessment of the Google's Chrome web browser. As Google has no corporate independent assessment strategy, they were quite interested in our FPVA. We had full access to all source code and the developers at Google. Chrome's core

source code totals about 2.4 million lines of code, mostly written in C++. There are an additional 7.7 million lines of third party code (not including standard libraries) used as part of Chrome. Our assessment was focused "only" on the core Chrome code.

We performed First Principles Vulnerability Assessment (FPVA) of the Chromium browser, the open source variant of Google Chrome. Problems that we discover in Chromium would likely apply also to the commercial Chrome browser (and, in fact, this proved to be the case). As a web browser, Chromium runs on a single machine at a time and with a single user's privilege. It does, however, interact heavily with the untrusted network, and it must enforce a wide variety of security policies to keep information from one site from flowing to another, potentially malicious site. Chromium is a heavily multi-process and multi-threaded application, and it employs various sandboxing techniques to drop privilege in those threads and processes that process web pages or run third-party extension code.

Architectural analysis was performed by reading the Chromium design documents and publications and by running traces on a Chromium build corresponding to the current dev channel release of Google Chrome. These traces identified portions of Chromium that are currently under-documented or wrongly documented. We constructed a first draft architectural diagram showing how Chromium's threads and processes interact with each other and with system resources (such as files, DNS servers, and web servers).

Chromium's sandboxing prevents renderers, the components responsible for displaying web pages and running JavaScript, from causing damage directly if, for instance, the WebKit layout engine or V8 JavaScript interpreter were to be compromised. This makes it critical, however, that the inter-process communication interface that surrounds the sandbox is implemented correctly, and that it makes no assumptions about what values the sandboxed code returns.

Chromium also allows for browser extensions consisting of segregated components: a main JavaScript body that cannot access pages directly, other pieces of JavaScript that can only access individual pages and communicate with the main component, and, optionally, a native code binary that also receives information from the main component. Unrestricted third-party native code is thus doubly isolated from interacting with web content, but it is critical that this isolation is actually implemented correctly.

Finally, these native code components of browser extensions, along with conventional plugins that display various file formats within web pages, could conceivably be vulnerable to buffer overflows or other attacks that would thus allow an attacker to control a process running with the same privilege as the browser. Sandboxing would prevent such a process from simply, for example, attaching to the page renderers via ptrace and exploiting them directly, but it is unclear whether an attacker could make progress in this situation by taking control of Chromium's local configuration and data files.

We spent much of the initial time learning about the logical model that modern HTML5 web browsers support, including how users interact with the browser, how external data enters (network protocols used, the types of objects allowed and their format), what local data and metadata the browser stores for efficient and correct operation (including displayed pages, history, cached copies of data, cookies, site local data storage), Javascript scripting and what restrictions are placed on it prevent security issues to prevent one site from manipulating another without the site's or user's knowledge.

The architecture diagram appears in Figure 2, emphasizing code structure. And an alternative view, emphasizing process structure appears in Figure 3. The results of our Chrome/Chromium assessment appear in Section 4.0.

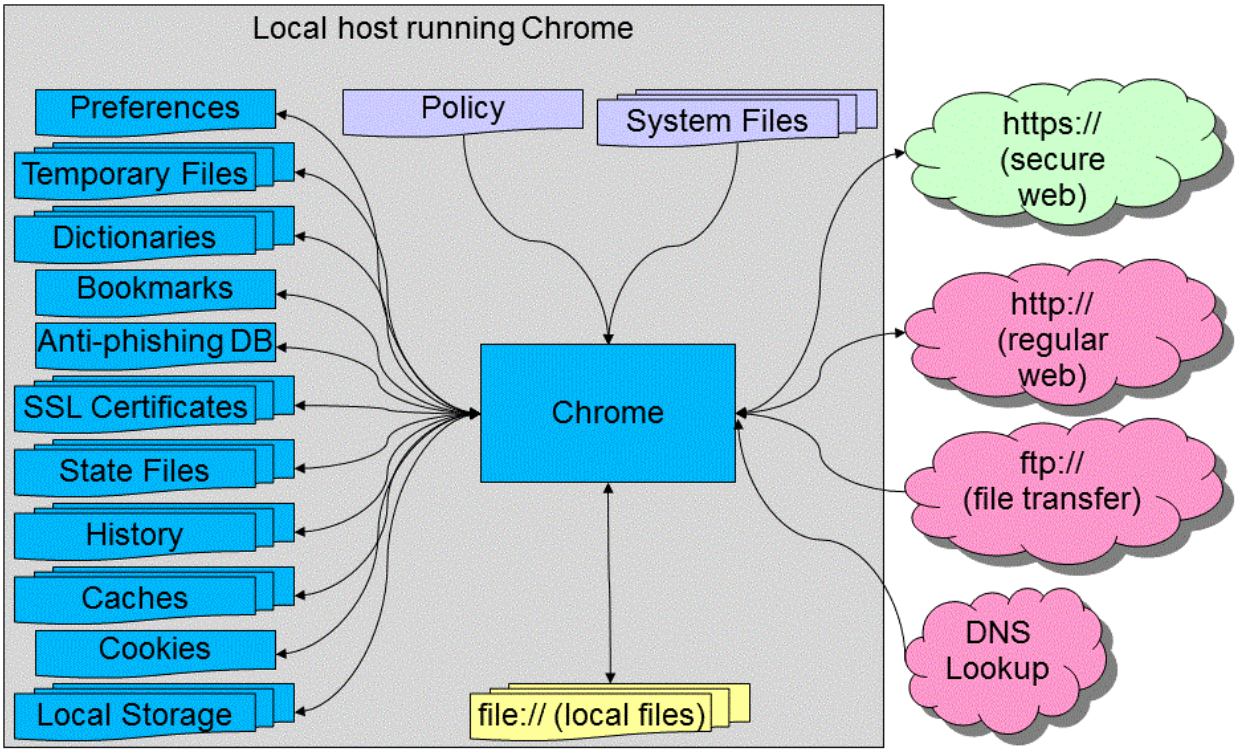

Figure 2: Chrome Architecture

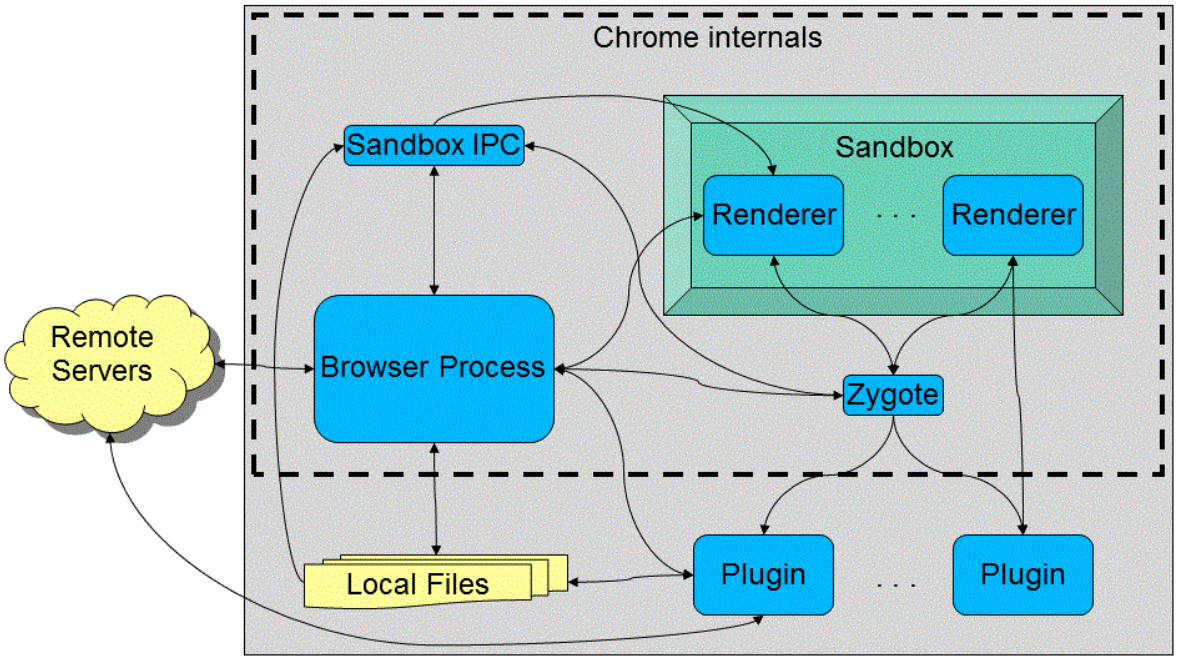

Figure 3. Process Communication in Chrome

3.3 Assessment Tool Technology – Static Code Analysis

We worked to develop a static source code dataflow analysis tool. This tool was motivated by our earlier study[1] that showed that our FPVA analyst-driven methodology was finding significant vulnerabilities not being found by the best commercial tools. We set about to study the characteristics of these difficult-to-find vulnerabilities and try to design new analyses to detect them.

We built our prototype on the Rose Compiler framework (www.rosecompiler.org) from Lawrence Livermore National Lab. The Rose Compiler had several problems with its interprocedural analysis implementation and we improved it to a stage wherein our algorithms could get us more complete results. We are now collaborating with Dan Quinlan of the Rose team through e-mail and phone calls to add these changes back into the main Rose builds and get feedback on any additional future changes.

As the first vulnerabilities to target with our tool, we chose those flow vulnerabilities that involve improperly checked string data flowing from an untrusted source to a trusting sink, with that string data being transformed by transforming functions (concatenation, substring, etc.) as well as manual iterations over string contents. A literature review showed two common tactics for dealing generally with information flow in strings.

[1] James A. Kupsch and Barton P. Miller, "Manual vs. Automated Vulnerability Assessment: A Case Study", *The First International Workshop on Managing Insider Security Threats*, West Lafayette, IN, June 2009.

First, static taint checking labels information manipulated by the outside world as tainted, while enforcing that certain functions take untainted arguments. This approach has shown promise in some real-world situations – for instance, detecting format string vulnerabilities – but it is necessarily quite coarse grained: analyses must either trust that every manipulation by the program in question is sanitizing or run the risk of overwhelming numbers of false positives.

Second, strings can be approximated by regular languages – i.e., finite automata or regular expressions – being a popular choice owing to decidability requirements. Previous work of this nature has focused on languages in which strings are completely captured by a library abstraction; the systems programs that are our focus present additional complications. Additionally, formal languages do not, on their own, capture any notion of degree of trusted.

From this we concluded that our tool should operate using an arbitrary lattice for the interpretation of strings, so that we can take advantage of both taint checking and regular languages (both of which naturally form lattices) and ideally combine the two. For now we have settled on a flow (i.e., program point) sensitive but path insensitive approach, tracking abstract values of string variables at each program point but not differentiating between different control flow paths. We may adjust this decision as efficiency concerns dictate.

3.4 Assessment Tool Technology – Dynamic Tools to Aid the Analyst

In-depth assessment of software security is typically an analyst-driven task, and visualization of the program's structure and security characteristics is the intrinsic part of this assessment task. For example, two widely used assessment methodologies, Microsoft Threat Modeling (MTM) and the UW/UAB First Principles Vulnerability Assessment (FPVA) both require developers or analysts to construct diagrams of software systems to represent key architectural components of the system, interactions between these components, the privilege levels of each component, delegation of privilege, and how components interact with high-value resources such as files, databases and external services. These diagrams are used as a road map for later in depth (often manual) inspection of the code. However, constructing such diagrams requires extensive manual effort, causing the analyst to conduct a careful reading of source code looking for key operations such as those that operate on files and sockets or modify privilege levels. These manual static inspections of the source code are often augmented by trace data from a variety of tools such as lsof or strace. Data is collected and synthesized from a variety of sources, in different formats, to provide the basis for the analyst to manually draw the various structural diagrams. To help address this problem, we are developing an approach to dynamically collect the key information for such diagrams and automatically produce the necessary diagrams to support the first steps of MTM or FPVA. This approach is embodied in a tool called the SecSTAR.

SecSTAR operates in two steps, data collection and visualization. We leverage a flexible and efficient dynamic binary instrumentation technique, self-propelled instrumentation, to collect trace data from production systems. Using self-propelled instrumentation, SecSTAR produces trace data for analyst-specified events in unmodified production systems during runtime. We

monitor and trace key events, such as those involved with process creation and destruction, socket creation and connection, privilege level changes, and file I/O operations. The trace data also contains temporal information, so that the later analysis and visualization steps can analyze for time-based risks such as race conditions and animate the visualizations of the program structure. The second step of SecSTAR is a postmortem operation that visualizes the trace data generated from the data collection step. In the visualization, the trace data is used to generate SVG diagrams using Graphviz dot layout, then these diagrams can be interactively viewed and explored in a web browser. The SecSTAR visualizer allows the analyst to navigate the diagrams through time and at different levels of detail.

The results of our SecSTAR efforts are reported in Section 4.4.

3.5 Vulnerability Reference Set

We have begun the effort to produce code representative of several simplified vulnerabilities previously found by our group. These simplified vulnerabilities can be used to test results from our tool or compare results with other analysis tools. Part of the goal is to identify how much a vulnerability representation can be reduced without trivializing its identification in an analysis tool. We will be able to easily distribute these simplified test cases to tool developers to add to their evaluation. In addition, tool users can use these test cases to help evaluate the market space of tools. This work is being done in collaboration with the NIST SAMATE project, under the direction of Dr. Paul Black.

4.0 RESULTS AND DISCUSSIONS

4.1 Wireshark Assessment Results

We initially directed our attention to the protocol dissectors, concentrating on the rarely used protocols as those appeared less likely to be well tested. The structure in which a dissector is written leaves little room for mishandling data or buffers, and the Wireshark coding standards further reduce the opportunity for programming mistakes. As a result, we found no vulnerabilities in the (many) dissectors that we examined.

However, our analysis of the structure of Wireshark led to the discovery of two interesting vulnerabilities, which are briefly described below. Details of both of these reports were delivered to the Wireshark developers for study and resolution.

The first of these, WIRESHARK-2011-0001, allows an attacker to use TCP to send a stream of bytes to another host where Wireshark decodes the values of the byte stream to be arbitrarily different from that which was actually sent. The result is that an exploited system can continue to communicate with an outside agent in a way that its communication stream would be invisible to the analyst using Wireshark. This technique is based on early research by one of Miller's Ph.D.

students, Shai Rubin[2], where they were able to send malicious packet streams through an intrusion detection or prevention system (IDS/IPS) in way that was undetected. In Figure 4, we illustrate this scenario:

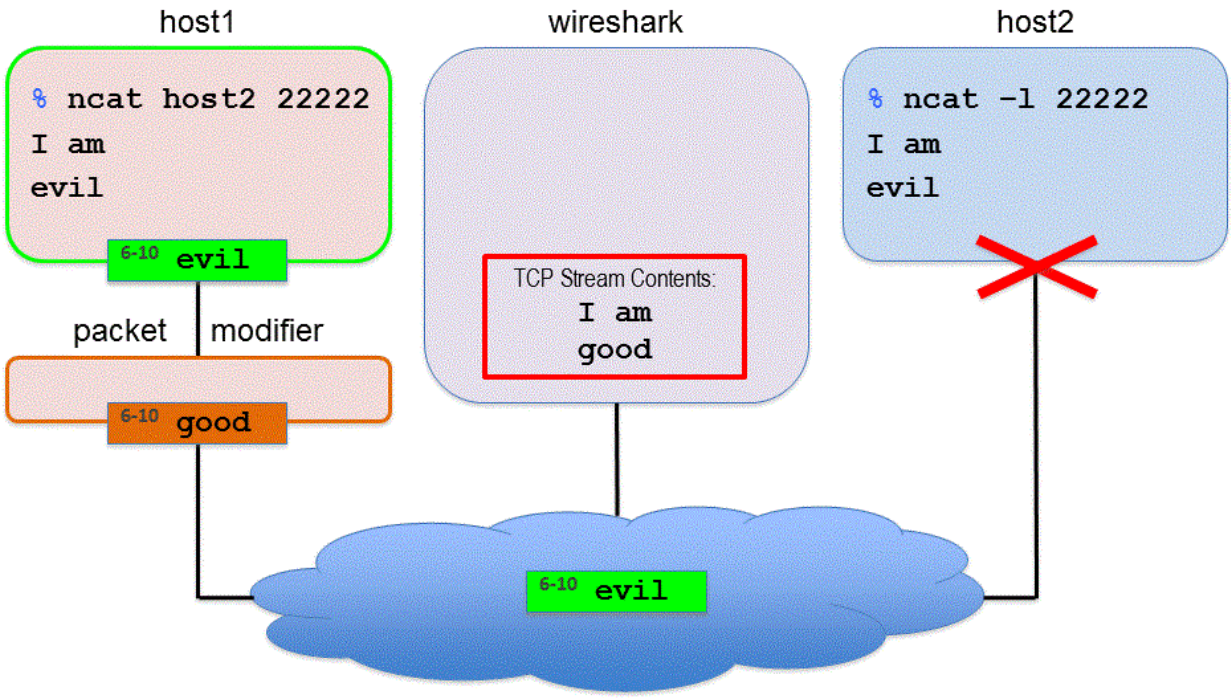

Figure 4. Hiding Packet Streams from Wireshark

The second vulnerability, WIRESHARK-2011-0002, makes it possible for a person or group responsible for exploiting a system detect if they are under surveillance. This detection occurs when the exploited system causes Wireshark to act as a beacon. If the exploited system creates a network packet that contains an IP address for a domain name for which they control the DNS server, and if reverse name translation (RARP) is enabled in Wireshark (the default setting), then this created network packet will cause Wireshark to act as a beacon or communication conduit for the exploiting code. Thus, even if the exploited system is isolated from the Internet, the analyst's computer may act as a unintentional communication path.

4.2 Chrome Assessment Results

While the Chrome code contains many minor coding errors, as you would expect in any software system, the overall architecture, including features like *canvases*, limit the impact of any of these errors. As a result, Chrome had relatively few serious vulnerabilities.

[2] Published as: S. Rubin, S. Jha, and B.P. Miller, "Automatic Generation and Analysis of NIDS Attacks", *Annual Computer Security Applications Conference (ACSAC)*, December 6-10, 2004.

We did find one vulnerability that made Chrome vulnerable to cross-site scripting type problems, identified as CHROME-2012-0001. We briefly describe the vulnerability here.

The vulnerability is a result of an unintended interaction between the new Content Security Policy (CSP) headers (which are supposed to protect users of that site against cross-site scripting) and the new powerful an all-encompassing HTML5 standard. This vulnerability was reported to Google and caused fixes in the WebKit sub-system (see https://code.google.com/p/chromium/issues/detail?id=140544).

4.3 Assessment Tool Technology – Static Code Analysis – Results

For our work on static analysis tools, as of the end of this contract, we have an initial working prototype and are working to evaluate its ability to operate on real code. We note that Rose is showing serious limitations in performance whole-program analysis, so we have also started an evaluation of using LLVM as the foundation for our tools.

4.4 Assessment Tool Technology – Dynamic Tools to Aid the Analyst – Results

To evaluate the effectiveness of SecSTAR dynamic analysis tool, we conducted a case study by using it to automatically produce FPVA analysis diagrams for a real world, widely used, and complex distributed middleware system: the Condor high-throughput scheduling system. We had previously conducted an in depth, manual FPVA assessment on Condor. This assessment was conducted over a nine month period by an experienced analyst, and resulted in finding several serious vulnerabilities that were not able to be found using current source code analysis tools. As part of the FPVA assessment, the analyst manually produced the associated analysis diagrams. Our use of SecSTAR allowed us to compress these initial steps of FPVA to a brief and interactive task, significantly improving the productivity of security analysts.

We have written a paper on SecSTAR [1], which can be found at http://research.cs.wisc.edu/mist/papers/Wenbin12SecSTAR.pdf and appears in the *VizSec 2012 (Symposium on Visualization for Cyber Security)*. A demo of SecSTAR's interface is available online at : http://research.cs.wisc.edu/mist/projects/SecSTAR/

4.5 Safefile I/O Library

We had previously developed a library to allow programmers to open files in a way that is resistant to attack. This library was based on our discovery of frequent risky coding practices in this area and the extreme difficulties of getting these details correct. Under this funding, we made minor corrections and improved comments in the safefile library. We worked around minor issues were uncovered by the Condor group in non-standard POSIX behavior in new versions of Solaris.

4.6 Collaboration with Microsoft

We have had ongoing discussions with Microsoft on ways to improve their Threat Modeling methodology (their name for vulnerability assessment) based on the work from our project. We

have been investigating ways to speed up their initial analysis of software based on the binary-analysis-based Self-Propelled Instrumentation. As a result of our discussions, these new directions have been placed in Microsoft's latest Three Year Development Plan. This work is ongoing and will continue through the summer with students working at Microsoft.

4.7 Transitions

A major component of this research project is to interact with groups in government, industry, and academia, with the goal of disseminating our results, gaining insights to their problem spaces, and adopting their research results, when applicable. Below is a summary of some of the interactions that took place under this funding:

- We maintained our ongoing relationship with the Condor team members about various security issues including using the current safefile library in Condor, assessment of potential security bugs and minor feature changes to the privilege separation code in Condor, and changes to Condor's use of gLExec.

  As a consulting courtesy to our colleagues in the Center for High Throughput Computing (directed by Prof. Miron Livny), we reviewed a small code change in an authentication module that Condor uses for security problems. The change did not introduce any new problems, but we noticed a rare case where it would be possible for an attacker to impersonate another user that existed in both implementation. Wrote up a vulnerability report and talked to Condor development team. A fix will require a new authentication type, as additional information must be transmitted by the protocol.
- We have also been in discussions with Michael Bender and Richard (Dickie) George at the NSA R Division about our approach to vulnerability assessment and evaluation. They asked us to work with one of their contractors, TASC Inc., to help them apply our techniques and experience in evaluating software assessment tools.

  We worked with Stephen Sutton and Michael Frank of Northrop Grumman Advisory Services Division (TASC, now an independent company), contracted by the NSA to study the effectiveness of automated assessment tools. We provided them with a set of known vulnerabilities in the Condor code base and helped them get an environment setup to build and work with the code, answering questions about the code and our earlier study, and reviewing questionable findings of the tools. This study extends our earlier work by including additional static analysis tools including Ounce Labs, Veracode, and a newer version of Fortify. Their work corroborated the results that we have gathered in our previous study. In addition, it helped to change the NSA approach to such evaluations.
- We maintained our ongoing relationship with the security group at the Autonomous University of Barcelona (under the direction of Prof. Elisa Heymann). We reviewed and provided feedback to their staff research Manuel Brugnoli about his architecture, resource, and trust and privilege diagrams for the VOMS (Virtual Organization Management System)

vulnerability assessment. Also reviewed and provided feedback for a report he wrote about a denial of service vulnerability he discovered.

- Corresponded with Prof. Tevfik Kosar, SUNY-Buffalo, the new maintainer of the Stork project, about old vulnerabilities we discovered in Stork. The vulnerabilities are now fixed.
- We continue our discussions and collaborations with Dr. Mine Altunay, Security Officer Open Science Grid. She wants to start working with the MIST group and is interested in trying to formalize some of the things in the earlier stages of FPVA such as trust relationships and security policies.
- We started discussions with people from NIST at SwA including Paul Black, Tim Boland, Barbara Guttman about including our reference vulnerability set in the NIST SAMATE standard collection. Paul runs the SAMATE project and is interested in collaborating with our group. They previously had run several static analysis tools on an older version of Wireshark and have shared the results with us.
- We had several discussions with Coverity, having extended calls with Andy Chou (CTO) and his technical staff. They are interested in our reference set of vulnerabilities, especially the ones that Coverity Prevent cannot identify currently. They were also interested in our framing of the problem as one of "flow vulnerabilities". We continue to share our growing reference set and to identify ways to migrate our new techniques into their software framework.
- We started discussions with Stephen Ewell, Executive Director of J-6 at, EUCOM and liaison to NATO. We have been discussing potential new cyber threat scenarios related to international shipping. Plans for a follow up discussion at EUCOM and NATO are in the works.
- We have had email discussions with Robert Seacord, CERT, Secure Coding Project about problems discovered in the compliant solution presented in the CERT C Secure Programming topic FIO15-C: Ensure that file operations are performed in a secure directory, Worked with the CERT staff to correct the immediate problems, and working a describing a more comprehensive solution as implemented in our safefile library.

5.0 CONCULSIONS

Overall, we are proud of the results of the research that we conducted under this funding. We have accomplishment vulnerability assessments of two major, high impact software systems (Wireshark and Chrome), developed new tool technology both for static and dynamic tools, widely disseminated our results, influenced the direction of government projects, and trained new practitioners in the field. One area of less success is that of making concrete transitions to our local law enforcement agencies. While we have made good connections with both the local FBI office and the Fusion Center, those connections have not yet produced reportable results.

6.0 REFERENCES

1. Wenbin Fang, James A. Kupsch, and Barton P. Miller, "Automated Tracing and Visualization of Software Security Structure and Properties", *9th International Symposium on Visualization for Cyber Security (VizSec),* Seattle, October 2012.
2. Andrew R. Bernat and Barton P. Miller, "Structured Binary Editing with a CFG Transformation Algebra", *19th Working Conference on Reverse Engineering (WCRE),* Kingston, Ontario, October 2012.
3. Guifre Ruiz, Elisa Heymann, Eduardo Cesar and Barton P. Miller, "Automating Threat Modeling through the Software Development Life-Cycle", *XXIII Jornadas de Paralelismo (JP2012),* Elche, Spain, September 2012.
4. Nathan E. Rosenblum, Xiaojin (Jerry) Zhu and Barton P. Miller, "Who Wrote This Code? Identifying the Authors of Program Binaries", *2011 European Symposium on Research in Computer Security (ESORICS),* Leuven, Belgium, September 2011.
5. Emily R. Jacobson, Nathan E. Rosenblum and Barton P. Miller, "Labeling Library Functions in Stripped Binaries", *10th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)*, Szeged, Hungary, September 2011.
6. Andrew R. Bernat, Kevin A. Roundy and Barton P. Miller, "Efficient, Sensitivity Resistant Binary Instrumentation", *International Symposium on Software Testing and Analysis (ISSTA),* Toronto, Canada, July 2011.
7. Nathan E. Rosenblum, Barton P. Miller and Xiaojin (Jerry) Zhu, "Recovering the Toolchain Provenance of Binary Code", *International Symposium on Software Testing and Analysis (ISSTA),* Toronto, Canada, July 2011. ACM SIGSOFT Distinguished Paper Award.
8. Jairo D. Serrano, Elisa Heymann, Eduardo Cesar and Barton P. Miller, "Vulnerability Assessment Enhancement for Middleware", *Computing and Informatics* **31**, February 2012, pp. 1001-1016. Also appears in the *5th Iberian Grid Infrastructure Conference (IBERGRID),* Santander, Spain, June 2011.
9. James A. Kupsch, Barton P. Miller, Eduardo César, and Elisa Heymann, "First Principles Vulnerability Assessment", *2010 ACM Cloud Computing Security Workshop,* Chicago, IL, October 2010.
10. Kevin A. Roundy and Barton P. Miller, "Hybrid Analysis and Control of Malware Binaries", *13th International Symposium on Recent Advances in Intrusion Detection (RAID2010)*, Ottawa, Ontario, Canada, September 2010.

LIST OF ACRONYMS

CERT	Computer Incidence Response Team
CSP	Content Security Policy
DNS	Domain name server
EUCOM	European Command
FPVA	First Principles Vulnerability Assessment
IP	Internet Protocol
MTM	Microsoft Threat Modeling
NSA	National Security Agency
NIST	National Institutes of Standards
VOMS	Virtual Organization Management System