WHITEPAPER

Six Degrees of Protocols

Stateful Fuzzing via Protocol Spidering™ Enables a Quantum Leap Forward in Software Testing — Discovers Subtle Flaws in Protocol Implementations

Today's businesses have become critically dependent on IP-based networks. This is not a news flash. What is not widely appreciated is that these networks are built from a [growing] set of increasingly complex protocols in an ever-expanding set of possible configurations. Worryingly, just as networks have become essential to our businesses, they are becoming more fragile. We see this practically every day in the news, where a new worm/virus outbreak is happening, or where hackers break in to some newsworthy network. These "attacks" cost real money. Millions of dollars per incident is not unheard of.

It is increasingly apparent that networks are becoming more and more fragile, which is evidence of the disparity between the increasingly rapid ability to develop ever more complex software compared to the essentially nonexistent capability to do negative testing (i.e., developers are able to create new features at a far faster rate than traditional test tools can evaluate them for quality). The pace of application development has left traditional testing methodologies in the dust, and networks have become increasingly application-aware – so we have a complex, fragile mess instead of a well-oiled machine.

The Mu-4000 analyzer puts 21st-century testing technology in the hands of developers and end-users to restore the balance between the pace of development and the ability to test complex systems.

686 W. Maude Avenue, Suite #104
Sunnyvale, CA 94085
866-276-4640 toll-free
408-329-6330 international
408-329-6317 fax

Complex Implies Fragile: Quality Suffers

Complexity is the enemy of robustness. Implications range from insufficient service availability all the way to security vulnerabilities and their exploits. The fragility we see is a direct result of the increasing complexity of the products and protocols that form the backbone of our emerging “converged” networks. This complexity undermines the ability of enterprise IT shops or service providers (e.g., triple-play providers, IP-over-cable and DSL operators, etc.) to deploy the mission-critical technology such that the network (and the applications that depend on it) is stable and can provide the level of service demanded by the users who are paying for the [reliable] service.

The industry is long past the point where device and application vendors could test their products thoroughly enough using labor-intensive traditional methods. QA is now able to complete only the minimal functional (positive) testing, limited to just the new features in a release, omitting testing how the new features interact with all the other features that are already in the product.

Moreover, even if QA at company X could do a complete job of testing a new release fully (they can’t), they cannot test their product’s interaction with all other compatible products in all possible customer network configurations. A customer’s network will have multiple vendors’ products, and will even have multiple versions of this vendor’s products, too. The vendor can still make significant progress on improving the quality of their code, but end users and vendors benefit from a platform that enables system-level testing in the field, using live configurations, and provides actionable feedback to the developers.

It is simply impossible for any vendor to verify that their product is robust in all possible customer deployments. The end-user must be able to do some very personalized testing of their product set in their particular configuration, but this only works when test tools are comprehensive, easy to use and highly automated. Most importantly, end users need tools that enable quick reproduction and remediation of issues by the vendor.

Layering Creates Complexity

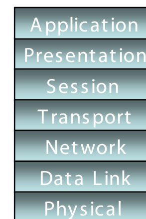
We have been using the term “protocol” and haven’t really defined it. It’s simple: Protocols form the basis of all networked communication. Over 5,000[1] IETF RFCs document in detail many of the protocols are used to move data across the Internet. These protocols enable us to surf the web, access hard drives attached to file servers, manage devices on a factory floor from a control room, deliver media streams over the network, send and receive e-mail and instant messages, and to do literally hundreds of other things. All of these applications involve at least one protocol.

Protocol: In computing, a protocol is a convention or standard that controls or enables the connection, communication, and data transfer between two computing endpoints. In its simplest form, a protocol can be defined as the rules governing the syntax, semantics, and synchronization of communication. Protocols may be implemented by hardware, software, or a combination of the two. At the lowest level, a protocol defines the behavior of a hardware connection. http://en.wikipedia.org/wiki/Protocol_%28computing%29

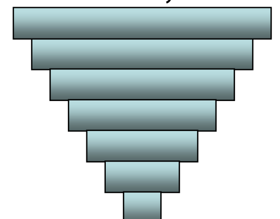
Protocol layering arose from a divide-and-conquer approach to protocol standardization, and relates to the well-known technique in computer programming of abstraction. Each layer in a protocol stack offers a well-defined set of services to the layer above, through a well-defined set of interfaces. Layering was a way to break down large standardization efforts into more manageable pieces, giving each working group the ability to focus on its part of the puzzle, without needing to worry about the big picture. There is a price for this degree of separation, in that because it is expected that protocols will depend on other protocols, there is a “spidering” effect in that a web of dependencies is created that is not only reflected in the standards, but also in the implementations of these protocols. These software dependencies make it easier for bugs to hide among the many interdependent pieces of code that must be assembled to make a real protocol implementation.

Layering Isn’t As Simple As It Looks

Protocol Stack



Vulnerability Stack

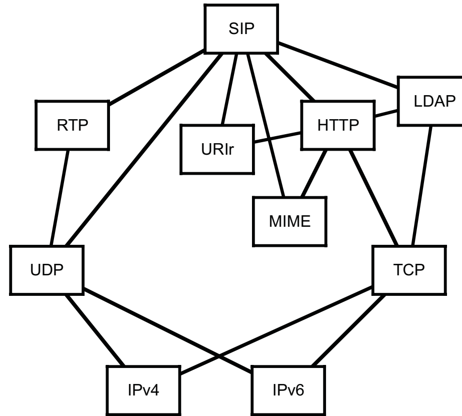


Layered protocols depend on other protocols. Implementations of standards-based protocols necessarily depend on implementations of existing lower-layer protocols. The power of understanding the implication(s) of the interdependencies within the protocol standards is the central idea that this paper presents.

The OSI Reference Model is a perfect place to start understanding protocol inter-dependencies: By definition, higher-layer protocols depend on lower-layer protocols, and thus inherit all the vulnerabilities of the entire protocol stack. Thus, the “vulnerability stack” is wider at the top, since higher layers inherit all the vulnerabilities of the protocols and interfaces of lower layers, while adding their own new vulnerabilities.

1. Not all RFCs define standards. Many are “informational” or “experimental” and contain useful real-world deployment or implementation experience. Even though they are not “normative” (standards-speak for “defining a standard”) they may still be important to read.

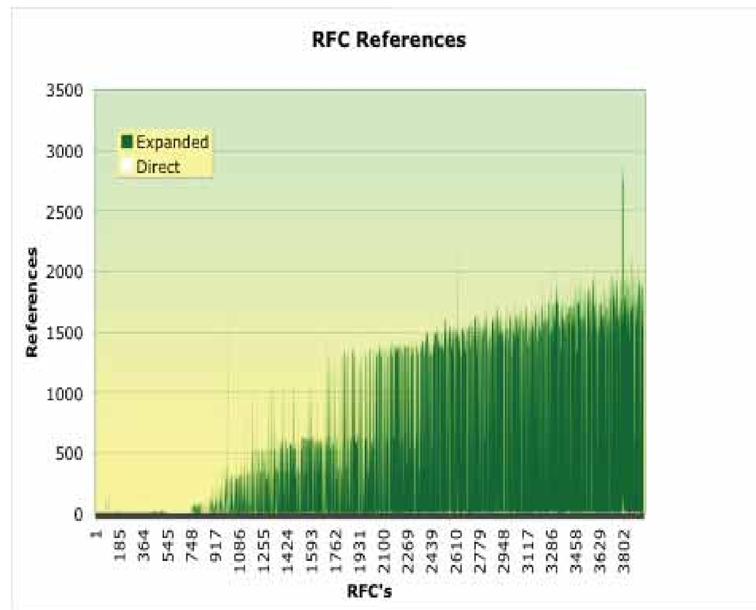
So far so good; further insight is gained by noticing that only trivial protocols may be represented as a strictly layered hierarchy – "interesting" protocols refer to multiple other lower-layer protocols, as in the following diagram which shows a tiny snippet of the structure on which SIP depends:



A layered approach taken by a standards body when designing a protocol has surprising implications for implementations and therefore for the security of layered protocols. Layered protocol specifications create nested implementations, not layered implementations! This insight is productively applied to negative testing in which test cases are designed to reflect the complex structure of the protocols and are therefore able to encompass enough of that structure (specification coverage) in order to achieve excellent code coverage of the target implementation across the whole set of test cases.

Protocol Spidering

The insight underlying Protocol Spidering is that protocols have many similarities with each other, primarily due to their deep interdependencies. This is an artifact of the way protocols are standardized: As new protocol standards are created, they tend to re-use existing standardsrather than reinvent the wheel. These dependencies grow over time, since older protocols have fewer other protocols on which they can depend, but newer ones have a far larger set of potential dependencies. The chart on the right illustrates this effect:



The graphic shows that the number of direct references for each RFC is typically less than about 16. Yet the indirect references to other RFCs are increasing to the point that newer protocols reference as many as half or more of the total number of extant RFCs at the time a given RFC is published!

Do developers really have to be familiar with the content of hundreds (or thousands?) of documents just to implement a new protocol? Repeated instances of the same mistakes indicate that broad familiarity with the letter (and spirit) of the underlying documents is lacking, however beneficial that would be. The reality is that developers should be able to focus on their deliverables and hone their expertise on a particular set of protocols, and rely on test tools to make sure that there are no bugs in code that they link into their application (or bugs in their own code!).

Returning to the concept of protocol [standard] interdependencies, we must observe that implementations of protocols must have a parallel set of interdependencies since the structure of the implementation must mirror the structure of the protocol standards tree to some extent. The fact that implementations are similarly structured begins captures the complexity for developers, but that's really only the beginning. These complex applications are built increasingly from outsourced, open-source and/or third party components, so there is decreasing visibility into these dependencies — even for the developers that are producing the application! It should be obvious that testing tools need to be applied against a finished product (since only the finished product fully includes all of its dependencies in "live" code). Source code analyzers can never capture the fully dynamic behavior of a running application; even binary code analyzers cannot fully exercise the code unless there is an active source of invalid inputs that is comprehensively exploring the protocol implementation's attack surface.

The critical attribute of negative test tools is that the test cases they generate must deeply probe the structure of a protocol implementation, and this structure is best derived from a deep understanding of the structure of protocol and the protocols on which it depends. Seeing past the recurring vulnerability patterns that recur across disparate protocols and into the underlying structures is what helps the researchers at Mu Dynamics to create high quality deeply penetrating test cases that provide excellent code coverage. Test cases, in order to be most effective (measured by code coverage and other hard metrics) need to leverage the deep structure of the protocol(s) that are implemented within a product. Protocol Spidering exposes these formerly hidden "family relationships" hidden within protocol implementations.

As with social groups, where the average chain length connecting any two people is smaller than you'd expect (e.g., "six degrees of separation[2]"), there are relationships among protocols and their implementations. As a result there is a strong likelihood that implementations of similar protocols will have similar vulnerabilities or failure modes, even in very dissimilar devices. However, as opposed to social networks – wherein the connections are usually not obvious – in families of standardized protocols, the interdependencies are open to analysis.

An Example of Spidering

RFC-3261 defines the Session Initiation Protocol (SIP) that is the enabling signaling protocol for Voice over IP (VoIP), and has over 1700 total references. RFC-3261, which defines how the SIP protocol should operate, only directly references the following other specifications:

- RFC-768 User Datagram Protocol
- RFC-1123 Requirements for Internet Hosts -- Application and Support
- RFC-1321 The MD5 Message-Digest Algorithm
- RFC-1750 Randomness Recommendations for Security
- RFC-1847 Security Multiparts for MIME
- RFC-1889 RTP: A Transport Protocol for Real-Time Applications
- RFC-2046 Multipurpose Internet Mail Extensions: Media Types
- RFC-2076 Common Internet Message Headers
- RFC-2119 Key words for use in RFCs to Indicate Requirement Levels
- RFC-2183 The Content-Disposition Header Field
- RFC-2234 Augmented BNF for Syntax Specifications: ABNF
- RFC-2277 IETF Policy on Character Sets and Languages
- RFC-2279 UTF-8, a transformation format of ISO 10646
- RFC-2326 Real Time Streaming Protocol (RTSP)
- RFC-2327 SDP: Session Description Protocol
- RFC-2368 The mailto URL scheme
- RFC-2396 Uniform Resource Identifiers (URI): Generic Syntax
- RFC-2401 Security Architecture for the Internet Protocol
- RFC-2543 SIP: Session Initiation Protocol
- RFC-2616 Hypertext Transfer Protocol -- HTTP/1.1
- RFC-2617 HTTP Authentication: Basic and Digest Access Authentication
- RFC-2630 Cryptographic Message Syntax
- RFC-2633 S/MIME Version 3 Message Specification
- RFC-2806 URLs for Telephone Calls
- RFC-2822 Internet Message Format
- RFC-2849 The LDAP Data Interchange Format (LDIF)
- RFC-2914 Congestion Control Principles
- RFC-2960 Stream Control Transmission Protocol
- RFC-2976 The SIP INFO Method
- RFC-3015 Megaco Protocol Version 1.0
- RFC-3204 MIME media types for ISUP and QSIG Objects
- RFC-3263 Session Initiation Protocol (SIP): Locating SIP Servers
- RFC-3264 An Offer/Answer Model with the Session Description Protocol
- RFC-3268 Advanced Encryption Standard (AES) Ciphersuites for TLS

In the above list of directly referenced RFCs, there is an unexpected reference to LDAP (Lightweight Directory Access Protocol). It's not really a bad thing that the IETF chose to re-use an existing directory standard rather than reinventing a new directory standard just for SIP. It is not surprising that the signaling protocol for a VoIP call would want to leverage any extant directories. The issue is that software developers implementing a dedicated IP phone are unlikely to be experts on both SIP and LDAP. A developer building a SIP application is more likely to be an expert on SIP, and will probably include LDAP support by linking to an open-source or commercially available LDAP library, which will have bugs that the SIP developer can't troubleshoot because even if s/he had access to the source code, it's highly unlikely that s/he would understand LDAP well enough to make correct changes that wouldn't cause other failures.

This is just one example that shows why developers, including SIP application developers are increasingly likely (more than that...they are required) to leverage a third party, open source, or outsourced libraries to provide essential functionality for their applications. However, the finished application then will have inherited the flaws that are present in those libraries, and the developer may not have the source code to those libraries (even if s/he did, they would probably lack the expertise to be able to fix the flaws). The only way a flaw can be discovered is through thorough positive and negative testing. Since positive testing cannot expose flaws (other than missing functionality), negative testing is increasingly important as development teams become more decentralized and as the vendors make more use of open-source and third-party libraries to help accelerate time-to-market.

2. http://en.wikipedia.org/wiki/Small_world_phenomenon

SIP is one interesting example of a larger issue. All newly standardized protocols necessarily and significantly leverage standards that already exist. As a result, implementations of these protocols share code to the maximum extent possible. The very structure of layered protocols and their inter-dependent specifications creates a situation in which vulnerabilities are multiplied as new protocols are implemented.

Finally: A New Way Forward

A new form of proactive security analysis, using negative test cases modeled on the very structure of the network protocols themselves, enables vulnerabilities (either outright crashes or decreases in service availability) to be discovered before the bug can affect anyone. The affect of a bug can be very slight, but it is also possible that the bug will result in a service outage that could result in revenue interruption or cause penalties to accrue, or could simply cost money when people are scrambling to isolate the problem and get it fixed. These costs all get in the way of the normal business operations of the network owner and may have secondary effects such as loss of reputation, increased customer churn, etc.

To be clear…software flaws are always going to be present, but large classes of bugs are now testable using a scientific, organized process. Protocol implementations are the part of the software that is almost by definition exposed to the outside of a system; protocol implementations are software that must be written to process unexpected inputs. This code must be able to interpret data from other implementations, and since it is coming across an unreliable network, packets can be reordered, dropped, corrupted, etc. Protocol mutations therefore are naturally occurring, but when they happen in the wild, they don't lend themselves to fault isolation. You may observe a crash or a device misbehaving, and have no idea what triggered it. On the other hand, protocol mutations derived from Protocol Spidering are able to "get inside" the implementation by exploring the deep nooks and crannies of the protocol, ensuring thorough code coverage as part of a robust fault isolation engine.

While a set of test cases that achieves 100% code coverage is the goal, that alone is not a guarantee that the code will be free of vulnerabilities. However, the converse is definitely true: If your testing has not achieved 100% code coverage, you must assume that the code has bugs. A well-known aphorism in the software development world is: "If code hasn't been tested, it's broken." Using targeted test cases (protocol mutations) generated by Protocol Spidering in the context of a robust fault isolation process enables issues to be discovered proactively, resulting in continuous improvement in the inherent security and availability of networked applications and devices as new fixes are progressively folded in to successive generations of the product. It is not sufficient for the code to be exercised once; thorough negative testing requires the code be executed as many times as it takes to cover all the states of the protocol.

The Emerging Security Analyzer

Security is a process, not a destination. One can always be more secure, but never absolutely secure. There has never been a way to be proactive about this process…until now.

The previously missing element is a tool that can exhaustively probe a device's "attack surface" based on the protocols it implements. Attack surface analysis is a powerful new technique for device vulnerability analysis enables a new type of proactive security analyzer that can examine the undiscovered structural vulnerabilities that are lurking within devices, awaiting exploit by the first hacker to discover them. Attack surface analysis is modeled on the inherent dependencies and patterns in protocols and their implementations.

Two decades after the Morris worm, we don't seem to have learned much about building secure software. With Aleph One's Phrack article[3], smashing the stack got easier. Open frameworks like Metasploit[4] make exploiting any vulnerability a piece of cake. The mechanics of exploitation have certainly gotten a great deal of attention in the recent past and we now have access to a broader class of specific attacks like format-string, buffer overflows, stack overflows, integer overflows and so on. In short, hackers have a variety of tools to exploit any vulnerability once it has been identified.

Gradually making protocols and implementations more secure would be an amazing improvement over the status quo. The last 20 years have shown that the computer and communications industries have no strategy for reducing vulnerabilities, despite the fact that everyone knows that they are present. It's not the fault of the vendors…they have never had a tool with which to scan the "vulnerability space" of any of their devices. Security analyzers enable a systematic approach to finding and fixing vulnerabilities. Armed with a security analyzer, you can continuously improve the security of your product or network.

Armed with a security analyzer, end-users can validate any IP-speaking device that they use, or plan to use, within their network. Network equipment manufacturers can leverage security analyzers to validate networked devices in their local configurations. The benefit to the end-user is that they can find, monitor, and shortcut the vendor remediation process for pre-zero-day vulnerabilities-without needing to be a security expert.

---

3. http://www.insecure.org/stf/smashstack.txt
4. http://www.metasploit.com/

Appendix: Why Do Bad Things Happen to Good Networks?

We know that networks are not getting more reliable, and the fact that they work at all is because of the talents of the dedicated people that make it happen. But the trend is not encouraging…at some point no amount of human effort will be able to keep these networks running. We need a new testing paradigm that can get to the root of the problem. These protocols are all related in deep (yet non-obvious) ways, and if we can use their structure against them, we can make progress in developing a meaningful testing regime that is automated and that can keep pace with the growth of the complexity in our networks.

Implementation flaws frequently result from improperly exercised exception handling code that was not well tested in QA, leading to poor service availability since key products are not individually robust enough to provide highly available service in the context of the wider network.

The lack of stability means that the network uptime is not good enough, and it's easy to destabilize the network by adding a new type of device, or by changing a configuration, or by installing a patch on an existing device. These kinds of changes happen all the time, and they are done without any visibility into the availability or security impact(s) of the change. More worrying is the fact that these non-robust devices are also more vulnerable to attack by people with bad intentions. Making the products more robust makes it easier to make promises about network availability, and it also removes weaknesses that could allow attackers to access the network.

Complexity is the Enemy of Security (and Availability)

Despite the well-known aphorism that complexity is the enemy of security, it is unfortunately true that networking products – especially those targeted at emerging markets like VoIP – are differentiated by new features, which gives rise to increasingly complex products. Vendors must add new features to maintain their competitive edge. Not surprisingly, security suffers. Especially in the case of emerging technologies, new features are being added before the collective intelligence of the market can truly assimilate the necessary experience to really design, develop, and deploy highly available services based on these features.

Even a moderate-sized network may be complex enough so no one understands its "normal" behavior, much less its behavior when under attack. This lack of perfection is not the fault of the developers, who undoubtedly intend to write the best possible code. Any sufficiently complex device has way too many possible configurations such that it is impractical for a vendor to test them all before the product is shipped. Until now, the end-user has had no way to "close the vendor loop," to ensure that the product has no vulnerabilities in their unique configuration.

Complexity can arise from inherently complex protocols or applications, exacerbated by complicated configurations, deployed in ways that may be unique to a particular network. Within any implementation of a standard (or any general specification), we observe that interdependencies multiply vulnerabilities. However, the good news is that we can use the structure underlying the complexity to give order and structure to our mutations, effectively using the complexity against itself.

Defense-in-Depth Can Never Be Deep Enough

Until now, security has been provided by layered defense-in-depth perimeter security strategies that have at their core a reactive operational stance: In order to protect against some particular flaw, someone somewhere must have become a victim of this flaw. Our current defenses are built on the hope that that "someone" won't be us! Security or availability events damage our ability to meet our business goals of, for example "five nine's" of availability (i.e., 99.999%). Whatever your SLA commitments, and however you are measured against them, downtime is bad for business.

Layered defenses, consisting of independent purpose-built networking and security devices, allow customers to "mix-and-match" to suit their requirements, ostensibly to independently select best-of-breed equipment in multiple categories (i.e., routers, firewalls, intrusion detection and prevention, proxies, etc.). However, every new or upgraded device added to a network introduces yet another type of vulnerability – the defense-in-depth system is itself complex. Each of these disparate security enforcement devices is complex and non-trivial to configure, and moreover it is challenging to keep their configurations aligned in a meaningful way so they work together appropriately.

It is essential that defense-in-depth security be applied around a network that has highly robust components…a belt-and-suspenders approach. It is insufficient to use a perimeter defense (no matter how strong) around a fundamentally weak network infrastructure because that weak infrastructure will also have lower availability than desired, and no amount of defense-in-depth can fix that. It is also incorrect to assume that one can draw a line around your network at which defenses can be deployed. People (your employees, partners, contractors, etc.) walk right past your defenses every day carrying active electronic components – laptops, PDAs, smart phones, etc. The center of your network needs to be as strong as possible, regardless of the capabilities deployed within your perimeter defenses. It is still important to be able to manage the flow of traffic across the perimeter, but we need to treat security and availability holistically and extend our defense-in-depth strategy across our whole network infrastructure.

The most important insight here is that the defenses are as complex on their own as the rest of the network. We can't know if our defenses are working unless we can test them. If the devices are found to have flaws, we need actionable remediation aids to help the vendor fix the specific problem(s) we are reporting.

Customers need tools to validate their chosen network and security hardware in their intended configuration(s) before they deploy or change them. Without such tools, deployed systems continue to be fragile and fail to meet service goals at increasing rates, resulting in increased downtime and associated increased staff costs to deal with seemingly inevitable emergency situations.

Constant emergencies have become standard operating procedure. Does this make sense? Shouldn't these emergencies be the exception rather than the rule?

About Mu Dynamics

Mu Dynamics is a technology innovator that has created a new class of security analysis system. The company's mission is to widely deploy security analysis and reduce product and application vulnerabilities. The security analysis process and the Mu-4000 analyzer platform provide a rigorous and streamlined methodology for verifying and improving the service availability and security readiness of any IP-based product or application.

Mu Dynamics enables enterprises and service providers to evaluate new products and software updates for known and previously undetected security vulnerabilities. In so doing, the company:

- Introduces security readiness as a metric for end-users' product purchase and deployment decisions;
- Allows development teams to efficiently identify security flaws in their products before release;
- Significantly decreases the number of security events in production networks through a proactive methodology that detects security flaws before systems and applications are deployed.

Mu Dynamics was founded in 2005 by experts in intrusion detection and prevention, ethical hacking and network management. The company is headquartered in Sunnyvale, California, and is backed by preeminent venture capital firms, including Accel Partners, Benchmark Capital and DAG Ventures.

Copyright © 2008 Mu Dynamics. All rights reserved. Mu Dynamics, Mu-4000 and the Mu Dynamics logo are trademarks of Mu Dynamics.